From Decidability to Undecidability by Considering Regular Sets of Instances*

Petra Wolf

Universität Trier, Fachbereich 4 - Abteilung Informatikwissenschaften,
D-54286 Trier, Germany
wolfp@informatik.uni-trier.de, https://www.wolfp.net/

Abstract. We are lifting classical problems from single instances to regular sets of instances. The task of finding a positive instance of the combinatorial problem P in a potentially infinite given regular set is equivalent to the so called int_{Reg}-problem of P, which asks for a given DFA A, whether the intersection of P with $\mathcal{L}(A)$ is non-empty. The int_{Reg}-problem generalizes the idea of considering multiple instances at once and connects classical combinatorial problems with the field of automata theory. While the question of the decidability of the int_{Reg}-problem has been answered positively for several NP and even PSPACE-complete problems, we are presenting some natural problems even from L with an undecidable int_{Reg}-problem. We also discuss alphabet sizes and different encoding-schemes elaborating the boundary between problem-variants with a decidable respectively undecidable int_{Reg}-problem.

Keywords: Deterministic finite automaton · Regular intersection emptiness problem · Undecidability

1 Introduction and Motivation

In many fields multiple problem instances are considered all at once and they are accepted if there is at least one positive instance among them. The instances are described through a strongly compressed representation. For instance, in *graph modification problems*[1] a graph G together with several graph editing operations is given and one asks whether G can be transformed into a graph G' with a certain property using up to n editing operations [2,6,15]. Here, the graph G represents the finite set of graphs which can be generated by G using up to n editing operations. Another example are problems with *uncertainty* in the instance [3,9] where some parameters of the instance are unknown and therefore stand for a variety of values. Finding a positive instance among plenty of candidates is also a task in synthesis problems [4]. In [4] the authors generate a finite set of candidate Petri nets among which they search for a solution. The

 The author was partially supported by DFG (FE 560 / 9-1).

[1] A Dagstuhl seminar on "Graph Modification Problems" was held in 2014 [2]

synthesis of an object with a certain property can be seen as the search for an object with this property among several candidates.

A natural generalization of the task of finding a positive instance in a finite set of instances is to search in an *infinite* set of instances. A well studied class of potentially infinite languages are the regular languages which are also in a compressed way represented by finite automata or regular expressions. We call $A = (Q, \Sigma, \delta, q_0, F)$ a deterministic finite automaton (DFA for short) if Q is a finite set of states, Σ a finite alphabet, $\delta\colon Q \times \Sigma \to Q$ a total transition function, $q_0 \in Q$ the start state, and $F \subseteq Q$ the set of final states. We generalize δ to words in the usual way. We denote the language accepted by A with $\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$. Asking whether the accepted language of a DFA A contains a positive instance of a problem P is equivalent to asking whether the intersection $P \cap \mathcal{L}(A)$ is non-empty. This question was introduced in [8] as the int_{Reg}-problem of P or $int_{\mathrm{Reg}}(P)$ for a fixed problem P.

Definition 1 ($int_{\mathrm{Reg}}(P)$)**.** *Given: DFA A. Question: Is $\mathcal{L}(A) \cap P \neq \emptyset$?*

In [1,12,13], $int_{\mathrm{Reg}}(L)$ was studied for languages L with low computational complexity, but which describe structural word-properties that have high relevance for combinatorics on words and formal language theory (e.g., set of primitive words, palindromes, etc.). There, (efficient) decision procedures are obtained.

The int_{Reg}-problem has been studied independently under the name *regular realizability problem* $RR(L)$, where the *filter language* L plays the role of problem P above, i. e., $RR(L) = int_{\mathrm{Reg}}(L)$ (see [1,16,17,20,21,22,23]), motivated by computational complexity questions. The aim was to present with the RR-problem 'a specific class of algorithmic problems that represents complexities of all known complexity classes [. . .] in a unified way' [22]. It turned out that RR-problems are universal in the sense that for any problem P, there exists an RR-problem $RR(L)$ with the same complexity (note that P and L are different languages). In [23] the authors focused on context-free filter languages and presented languages L for which $RR(L)$ is either P-complete, NL-complete or has an intermediate complexity. In [20] the decidability of the RR-problem with filter languages over permutations of binary words was studied.

In contrast, the line of research in [8,24,26,27] aims to use the int_{Reg}-problem as a tool to get insights into classes of hard problem as for instance the class of NP and PSPACE-complete problems. While the decidability of $int_{\mathrm{Reg}}(P)$ for hard problems P such as SAT [8], ILP [26], Vertex Cover [27] and TQBF [8] is known, we present in this work problems, with a complexity ranging from contained in L to PSPACE-completeness, with an undecidable int_{Reg}-problem. These results indicate that the decidability of the int_{Reg}-problem of a language does not directly coincide with its computational complexity. This study rises the natural question what, for instance, NP-complete problems with a decidable int_{Reg}problem have in common that separates them from NP-complete problems with an undecidable int_{Reg}-problem. We also examine for some problems the size of the input alphabet and the encoding scheme resulting in different decidability results of the considered int_{Reg}-problem.

This paper is structured as follows. First, we discuss machine languages for several complexity classes. Then, we consider the problems of bounded and corridor tiling, followed by bounded PCP. We will show that all of these problems have an undecidable int_{Reg}-problem. Next, we investigate the PSPACE-complete problem of Equivalence of Regular Expressions and prove that the problem in a shuffled encoding has an undecidable int_{Reg} problem. As the proof only uses the concatenation operator of regular expressions, we get the undecidability of int_{Reg} of the so called String Equivalence Modulo Padding problem in a shuffled encoding, which lies in L. For this problem, we will discuss different alphabet sizes and encoding schemes and show that all other considered variants of this problem have a decidable int_{Reg}-problem. Finally, we present a graph problem on directed multi-hyper-graphs with an undecidable int_{Reg}-problem. This contrasts the results in [27] where classes of graph problems with a decidable int_{Reg}-problem are identified.

We expect the reader to be familiar with regular languages and their description through finite automata and regular expressions. The reader should also be familiar with the complexity classes L, NL, NP, and PSPACE. We refer to the textbooks [7] and [10] for details. Missing proofs can be found in the long version of this work [25].

2 Machine Languages

For several complexity classes, we can define *machine languages* which are complete for their complexity classes. We will show that the following machine languages have an undecidable int_{Reg}-problem. The int_{Reg}-problem of the machine language for NP was already discussed in [8] and is listed here for the sake of completeness.

Definition 2 (Machine Language for NL).
Given: Encoded nondeterministic Turing machine $\langle M \rangle$*, input-word* x*, and a string* a^n *with* $n \in \mathbb{N}$.
Question: Does M *accept* x *visiting only* $\log(n)$ *different tape-positions?*
Encoding: $L_{NL} = \{\langle M \rangle \$ x \$ a^n \mid M \text{ is an NTM accepting } x \text{ in } \log(n) \text{ space}\}$.

The language L_{NL} is complete for the class NL. Every language in NL can be accepted by a non-deterministic Turing machine which is space-bounded by a function $f \in \Theta(\log)$. Since f is logspace-constructible, there exists a deterministic TM M_f which computes $f(n)$ on the input 0^n in logarithmic space. Hence, every fixed problem in NL can be reduced to L_{NL} by hard-wiring the NTM M which decides the problem and is space-bounded by f, followed by the input word w and a unary string of size $2^{f(|w|)}$. Note that $f(|w|)$ is logarithmically smaller than $|w|$ and hence can be stored using $\log(|w|)$ many cells. A logarithmically space-bounded TM can compute an output string which is exponentially in the size of its used memory. As can be easily verified $L_{\text{NL}} \in \text{NL}$.

The machine language for NP, in short L_{NP}, is defined analogously demanding that x is accepted in n steps, while the machine language for PSPACE, in short

L_{PSPACE}, demands x to be accepted in n space. With similar arguments L_{NP} is complete for NP and L_{PSPACE} is complete for PSPACE.

Theorem 1. *$int_{\text{Reg}}(L_{NL})$, $int_{\text{Reg}}(L_{NP})$, and $int_{\text{Reg}}(L_{PSPACE})$ are undecidable.*

Proof. We give a reduction from the undecidable non-emptiness-problem for recursively enumerable sets [10] defined as $L_{\neq\emptyset} := \{\langle M\rangle \mid M \text{ is a nondeterministic TM with } \mathcal{L}(M) \neq \emptyset\}$. Let $\langle M\rangle$ be an arbitrary encoded Turing machine with the input alphabet Σ. We define the regular language $f(\langle M\rangle) := R := \{\langle M\rangle\$x\$a^n \mid x \in \Sigma^*, n \geq 0\}$. Then, $f(\langle M\rangle) \in int_{\text{Reg}}(L_{\text{NL}}) \Leftrightarrow R \cap L_{\text{NL}} \neq \emptyset \Leftrightarrow \mathcal{L}(M) \neq \emptyset \Leftrightarrow \langle M\rangle \in L_{\neq\emptyset}$. The same holds for L_{NP} and L_{PSPACE}. Since the emptiness-problem for recursive enumerable sets is undecidable, the undecidability of the problems $int_{\text{Reg}}(L_{\text{NL}})$, $int_{\text{Reg}}(L_{\text{NP}})$, and $int_{\text{Reg}}(L_{\text{PSPACE}})$ follows. □

3 Bounded and Corridor Tiling

The next problem we want to investigate is about the *tiling of the plane*. For a given set of tile types and a fixed corner tile, the question is to fill a plane with the given tiles under some conditions. While the problem for an infinite plane is undecidable [14,19], it becomes NP-complete if we restrict the plane to an $n \times n$-square and preset the tiles on the edges of the square; it becomes PSPACE-complete if we only restrict the width with preset tiles and ask for a finite height, such that the plane can be tiled according to the preset tiles [5].

First, we will give a formal definition of the problem Bounded Tiling. Then, we will show that this problem has an undecidable int_{Reg}-problem by reducing $L_{\neq\emptyset}$ to the problem int_{Reg}(Bounded Tiling).

A *tile* is a square unit where each of the edges is labeled with a color from a finite set C of colors. The color assignment is described by *tile types*. A tile type is a sequence $t = (w, n, e, s)$ with $w, n, e, s \in C$ of four symbols representing the coloring of the left, top, right, and bottom edge color. We denote with t_w, respectively t_n, t_e, t_s, the first, respectively second, third, and forth entry of the tuple t. Tiles can be regarded as instances of tile types. A tile must not be rotated or reflected. In the following problem, we give a finite set of tile types as input. From every tile type arbitrary many tiles can be placed. The tiles have to cover up a square grid region such that adjacent edges have to have the same color. The grid comes with an *edge coloring* which contains for each border of the square grind a sequence of colors presetting the adjacent color of tiles resting on the edge. A *tiling* is a mapping from the square grid region to a set of tile types. With $\langle T\rangle$ we denote a proper encoding of the tile type set T and with $[n]$ we denote the set $\{1, 2, \ldots, n\}$. We call $f\colon [n] \times [n] \to T$ a *tiling function* if for all $i, j \in [n]$, it holds that $f(i, j)_e = f(i + 1, j)_w$ for $i < n$ and $f(i, j)_n = f(i, j + 1)_s$ for $j < n$ meaning that adjacent edges of the tiles have the same color. Here, the bottom left square of a grid region is indexed by $(1, 1)$.

Definition 3 (Bounded Tiling).
Given: Finite set T of tile types with colors from a finite color set C and an $n \times n$

square grid region V with a given edge coloring.
Question: Is there a tiling function $f\colon [n] \times [n] \to T$ that tiles V extending the edge coloring?
Encoding: $\langle T \rangle$ followed by an edge coloring $\$l\$t\$r\$b$, $l = l_1\#l_2\#\ldots\#l_n$, with $t = t_1\#t_2\#\ldots\#t_n$, $r = r_1\#r_2\#\ldots\#r_n$, $b = b_1\#b_2\#\ldots\#b_n$ with $l_i, t_i, r_i, b_i \in C$.

Howard Straubing gives in his article "Tiling Problems" [19] a reduction from the complement of the halting problem to the problem of tiling an infinite plane. Therefore, he gives an algorithm "that takes input $\langle M \rangle$ and produces the associated $\langle T, c \rangle$" (where c is the given corner tile in the unrestricted case of the problem). The tiles represent every possible transition of the Turing machine and are constructed in a way that correctly tiled rows correspond to configurations of the given Turing machine. The four colors of the tiles also ensure that two adjacent rows represent two consecutive configurations. Therefore, the infinite plane can only be tiled if and only if the Turing machine runs forever.

Peter van Emde Boas [5] uses a similar construction to simulate Turing machines and shows that the Bounded Tiling problem is NP-complete. For a given nondeterministic Turing machine, the possible transitions and tape cell labelings are transformed into a set of tile types. The input word, padded with blank symbols, is encoded in the bottom edge coloring b and a distinguished accepting configuration is encoded in the top edge coloring t. The left and right borders are colored with the fixed color *white* which is a color only occurring on vertical edges and which do not represent any state or alphabet letter of the Turing machine. So, white can be seen as a neutral border color. Blank symbols are trailed to the input word to enlarge the size of the square field to the exact time bound of the Turing machine. The Turing machine is altered in a way that it accepts with one distinguished accepting configuration. The tile types are constructed in a way that this accepting configuration can be repeated over several adjacent rows. Therefore, the constructed edge colored square region can be correctly tiled matching the edge coloring if and only if the given Turing machine accepts the input word within its time bound.

With that construction in mind, we will now prove that the int_{Reg}-problem for Bounded Tiling is undecidable.

Theorem 2. *The problem int_{Reg}(Bounded Tiling) is undecidable.*

Proof. We give a reduction from the undecidable problem $L_{\neq\emptyset}$. Let $\langle M \rangle$ be the encoding of an arbitrary NTM. We construct a regular language R which contains a positive Bounded Tiling instance if and only if M accepts at least one word. We alter the machine M to an NTM N which behaves like M except having only one distinguished accepting configuration, i.e., an empty tape with the head on the first position of the former input word. According to Straubing [19] and van Emde Boas [5], there is an algorithm which, given a TM N, produces the corresponding set of tile types T such that a correct extending tiling of a given edge colored square field corresponds to a sequence of successive configurations of the given machine, starting on an input word represented through the coloring of the bottom border.

Let T_N be the corresponding tile type set for the NTM N and let C_N be the set of colors appearing in T_N. Let $C_\Sigma \subseteq C_N$ be the subset of colors representing input alphabet symbols, let $\diamond \in C_N$ be the *white* color representing a white vertical border edge of the square grid, and let $\square \in C_N$ be the color representing an empty tape cell. Finally let $q_f \in C_N$ be the color representing the accepting state of the Turing machine. We define the regular set R as $R = \mathcal{L}\left(\{\langle T_N\rangle \ \$ \diamond^* \ \$ \ q_f\square^* \ \$ \diamond^* \ \$ \ {C_\Sigma}^*\square^*\}\right)$. The set R consists of the set of tile types for the NTM N together with edge colorings for every possible input word and every possible size of the field V. The top row will always contain the accepting configuration of N padded with arbitrary many blank symbols. The left and right borders of the field V can consist of arbitrary many white edges, while the bottom row can encode every possible input word with arbitrary many added blank symbols allowing an arbitrary time bound for the Turing machine. Note that the edge coloring does not have to define a square, but the square shape is also contained in the set R for every input word and every number of padding symbols. Therefore, for every input word w, the set R contains every size of squared fields with w encoded in the bottom edge coloring. The tile type set of R is constructed in a way that in a valid tiling adjacent rows will represent successive configurations of the Turing machine. So, for every number of steps the TM makes on the input word, there is a square field, with the input word encoded, in the set R preventing enough space for the configurations of the TM. This brings us to our main claim, $R \cap$ Bounded Tiling $\neq \emptyset \Leftrightarrow \mathcal{L}(N) \neq \emptyset$. □

With the same argument, we can show that the PSPACE-complete problem Corridor Tiling [5] also has an undecidable int_{Reg}-problem.

4 Bounded PCP

Another undecidable problem, which becomes decidable if we restrict the size of the potential solution, is the Post's Correspondence Problem (in short PCP). We show that the NP-complete version Bounded Post Correspondence Problem [7] (in short BPCP) has an undecidable int_{Reg}-problem by a reduction from the unrestricted undecidable PCP problem [10].

Definition 4 (BPCP).
Given: Finite alphabet Σ, two sequences $A = (a_1, a_2, \ldots, a_n)$, $B = (b_1, b_2, \ldots, b_n)$ of strings from Σ^*, and a positive integer $K \leq n$.
Question: Is there a sequence $i_1, i_2, \ldots, i_k$ of $k \leq K$ (not necessarily distinct) positive integers $i_j \in [n]$ such that $a_{i_1}a_{i_2}\ldots a_{i_k} = b_{i_1}b_{i_2}\ldots b_{i_k}$?
Encoding: $L_{BPCP} := \{a_1\#a_2\#\ldots\#a_n\$b_1\#b_2\#\ldots\#b_n\$\,\text{bin}(K) \mid K \leq n \wedge A = (a_1, a_2, \ldots, a_n), B = (b_1, b_2, \ldots, b_n)$ is a PCP instance with a solution $\leq K\}$.

The problem PCP is defined analogously but does not contain a bound K.

Theorem 3. *The problem int_{Reg}(BPCP) is undecidable.*

Proof. We give a reduction PCP $\leq int_{\text{Reg}}$(BPCP). Let $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ be a PCP instance. We construct a regular language R

consisting of the given PCP instance combined with every possible solution bound K. Since K is bounded by the length of list A and B, we will pump those lists up by repeating the last list element of both lists arbitrarily often. Because the same element can be picked multiple times, adding elements already appearing in the given lists does not change the solvability of the instance. We define R as $R = \{a_1\#a_2\#\dots\#a_n(\#a_n)^*\$b_1\#b_2\#\dots\#b_n(\#b_n)^*\$\{0,1\}^*\}$. It holds that $R \cap \text{BPCP} \neq \emptyset$ if and only if there is a sequence of indexes $i_1, i_2, \dots, i_m$ such that $a_{i_1}a_{i_2}\dots a_{i_m} = b_{i_1}b_{i_2}\dots b_{i_m}$. □

5 Regular Expressions in a Shuffled Encoding

In this section we show that the problem of Equivalence of Regular Expressions (in short $\equiv_{\text{RegEx}}$) over a binary alphabet in a shuffled encoding has an undecidable regular intersection emptiness problem. It turns out, that the problem is already undecidable if the regular expressions do not use alternation or the Kleene star. Thus, also the problem of String Equivalence Modulo Padding over a binary alphabet in a shuffled encoding has an undecidable int_{Reg}-problem. When we consider the String Equivalence Modulo Padding problem over a unary alphabet or in a sequential encoding, the problem becomes decidable. We first define the problem of Equivalence of Regular Expressions (adapted from [7]). For a regular expression E, we denote with $\mathcal{L}(E)$ the regular language described by E. We use concatenation implicitly and omit the operator symbol. The alternation is represented by |-symbols.

Definition 5 (Shuffled$\equiv_{\text{RegEx}}$).
Given: A word $w = e_1f_1e_2f_2e_3f_3\dots e_nf_n$ *over the alphabet* $\Sigma \cup \{\emptyset, \epsilon, (,), |, *\}$ *such that* $E = e_1e_2e_3\dots e_n$ *and* $F = f_1f_2f_3\dots f_n$ *are regular expressions over the alphabet* Σ *using the operators* alternation*,* concatenation*, and* Kleene star*. Note that one regular expression can be padded with* ϵ *or* $\emptyset$ *if the regular expression are of unequal length.*
Question: Is $\mathcal{L}(E) = \mathcal{L}(F)$*?*

The problem of equivalence of the regular expressions is well known to be PSPACE-complete [18]. Since we can change the encoding of an $\equiv_{\text{RegEx}}$ instance from shuffled to sequential and vice versa in quadratic time, the shuffled version of this problem is also PSPACE-complete. We will show that int_{Reg}(Shuffled$\equiv_{\text{RegEx}}$) is undecidable by a reduction from the PCP problem [11]. For readability reasons, we will refer to words $w \in$ Shuffled$\equiv_{\text{RegEx}}$ as $w = \frac{e_1}{f_1} \dots \frac{e_n}{f_n}$. From a given PCP instance we will construct a regular language L_{Reg}, the words of which will describe all possible solutions of the PCP instance. The words will consist of two shuffled regular expressions using only the concatenation as an operator. By construction, the first regular expression will be a concatenation of strings from the A list of the PCP instance while the second regular expression will consists of the concatenated corresponding strings from the B list. Since the regular expressions only use concatenation, languages described by them only contain one element each. The language L_{Reg} will contain two shuffled regular

expressions describing the same language if and only if the PCP instance has a valid solution.

Theorem 4. *The problem* $int_{\mathrm{Reg}}(\textsc{Shuffled}{\equiv_{\textsc{RegEx}}})$ *is undecidable.*

Proof. We give a reduction PCP $\leq int_{\mathrm{Reg}}(\textsc{Shuffled} \equiv_{\mathrm{RegEx}})$ and translate a given PCP instance into a regular language L_{Reg}. We emphasize references to the *regular expression* defining the language L_{Reg}, while references to the regular expressions encoded in the words of L_{Reg} are not emphasized. We also emphasize references to the *regular language* of shuffled regular expressions.

Let $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots, b_k$ be a PCP instance. We define a *regular expression*, describing a *regular language* L_{Reg} of shuffled regular expressions describing concatenations of list elements. Let L_{Reg} be defined through the *regular expression*

$$\left(\begin{array}{c|c|c|c} {a_1}' & {a_2}' & \ldots & {a_k}' \\ {b_1}' & {b_2}' & & {b_k}' \end{array}\right)^+$$

where the string $\begin{smallmatrix} {a_i}' \\ {b_i}' \end{smallmatrix}$ consists of the two shuffled strings a_i, b_i where the shorter string is padded with ϵ-symbols at the end until both stings have the same length. The ϵ-symbol is here used as an alphabet symbol of the language L_{Reg} and refers to the regular expression ϵ which will be interpreted as $\{\epsilon\}$ and not to the empty word itself. Therefore, L_{Reg} consists of all possible pairwise concatenations of elements of the lists A and B where the concatenated strings are padded with ϵ-symbols to have the same length.

For every PCP instance, the described *regular expression* of the language L_{Reg} can be computed by a computable total function. It remains to show that the PCP instance A, B has a solution if and only if $L_{\mathrm{Reg}} \cap \textsc{Shuffled}{\equiv_{\textsc{RegEx}}} \neq \emptyset$. More precisely, the intersection will contain all solutions of the PCP instance.

First, consider the only if direction. Let $i_1, i_2, \ldots, i_n$ be a solution of the PCP instance A, B such that $a_{i_1} a_{i_2} \ldots a_{i_n} = b_{i_1} b_{i_2} \ldots b_{i_n}$. By construction, the *regular language* L_{Reg} contains all possible concatenations of the strings $\begin{smallmatrix} {a_1}' \\ {b_1}' \end{smallmatrix}, \ldots, \begin{smallmatrix} {a_k}' \\ {b_k}' \end{smallmatrix}$ corresponding to the pairs $(a_1, b_1), \ldots (a_k, b_k)$ of the strings of the lists A and B. Therefore, L_{Reg} also contains the word $w = \begin{smallmatrix} {a_{i_1}}' \\ {b_{i_1}}' \end{smallmatrix} \begin{smallmatrix} {a_{i_2}}' \\ {b_{i_2}}' \end{smallmatrix} \ldots \begin{smallmatrix} {a_{i_n}}' \\ {b_{i_n}}' \end{smallmatrix}$. The word w consists of the two shuffled regular expressions $E = {a_{i_1}}' {a_{i_2}}' \ldots {a_{i_k}}'$ and $F = {b_{i_1}}' {b_{i_2}}' \ldots {b_{i_k}}'$. Since they are both nonempty strings with padded ϵ's their described language is a singleton set. By construction, we have $\mathcal{L}(E) = \{a_{i_1} a_{i_2} \ldots a_{i_k}\}$ and $\mathcal{L}(F) = \{b_{i_1} b_{i_2} \ldots b_{i_k}\}$. By assumption is $a_{i_1} a_{i_2} \ldots a_{i_n} = b_{i_1} b_{i_2} \ldots b_{i_n}$, therefore we have $\mathcal{L}(E) = \mathcal{L}(F)$ and $w \in L_{Reg} \cap \textsc{Shuffled}{\equiv_{\textsc{RegEx}}}$.

For the other direction, assume $L_{\mathrm{Reg}} \cap \textsc{Shuffled}{\equiv_{\textsc{RegEx}}} \neq \emptyset$. Let $w = \begin{smallmatrix} {a_{i_1}}' \\ {b_{i_1}}' \end{smallmatrix} \begin{smallmatrix} {a_{i_2}}' \\ {b_{i_2}}' \end{smallmatrix} \ldots \begin{smallmatrix} {a_{i_n}}' \\ {b_{i_n}}' \end{smallmatrix} \in L_{\mathrm{Reg}} \cap \textsc{Shuffled}{\equiv_{\textsc{RegEx}}}$ consists of the two shuffled regular expressions $E = {a_{i_1}}' {a_{i_2}}' \ldots {a_{i_k}}'$ and $F = {b_{i_1}}' {b_{i_2}}' \ldots {b_{i_k}}'$. By assumption is $\mathcal{L}(E) = \mathcal{L}(F)$. Since $\mathcal{L}(E)$ and $\mathcal{L}(F)$ each contain only one element, from which the describing regular expressions differ only by padded ϵ-symbols, it holds by construction that $a_{i_1} a_{i_2} \ldots a_{i_n} = b_{i_1} b_{i_2} \ldots b_{i_n}$. Therefore, $i_1, i_2, \ldots, i_n$ is a solution of the PCP instance. □

To show undecidability of the $int_{\text{Reg}}(\text{SHUFFLED}{\equiv_{\text{REGEX}}})$ problem we have made use of only one operator of regular expressions, namely the concatenation. If we restrict the $\text{SHUFFLED}{\equiv_{\text{REGEX}}}$ problem to regular expressions using only letters from Σ, the ϵ-symbol and the concatenation, we get the much easier problem of SHUFFLED STRING EQUIVALENCE MODULO PADDING, in short $\text{SHUFFLED}{\equiv_{\text{STRING}^\epsilon}}$. Since we are only using the associative operation of concatenation, we can get rid of brackets. All of the following problems are in the complexity class L, since they can be solved deterministically using two pointers.

Definition 6 ($\text{SHUFFLED}{\equiv_{\text{STRING}^\epsilon}}$).
Given: A word $w = s_1t_1s_2t_2s_3t_3 \dots s_nt_n$ such that $s_i, t_i \in \Sigma \cup \{\epsilon\}$. Question: Is $h(s_1s_2s_3 \dots s_n) = h(t_1t_2t_3 \dots t_n)$ where $h : (\Sigma \cup \{\epsilon\})^* \to \Sigma^*$ is an erasing homomorphism which leaves all symbols in Σ unchanged and deletes the ϵ-symbols.

Corollary 1. *The problem $int_{\text{Reg}}(\text{SHUFFLED}{\equiv_{\text{STRING}^\epsilon}})$ is undecidable.*

If we restrict the alphabet Σ to singleton sets, the $\text{SHUFFLED}{\equiv_{\text{STRING}^\epsilon}}$ becomes decidable as this problem, considered as a language, is a context-free language. Alternatively, if we refrain from the shuffled encoding and consider instead a sequential encoding, the problem also becomes decidable. Here, we identify sub-automata which accept prefixes up to the symbol \$ and sub-automata which accept suffixes starting after the symbol \$. We use a homomorphism h to erase the padding symbol ϵ and check for each pair of prefix and suffix sub-automata A_P and A_S whether $h(\mathcal{L}(A_P)) \cap h(\mathcal{L}(A_S)) \neq \emptyset$. Details on the above discussed variations can be found in the long version of this work [25].

6 An Undecidable int_{Reg} Problem About Graphs

In this section we consider a graph problem with an undecidable int_{Reg}-problem which contrasts the decidability results for graph problems in [27].

For a word $w \in \Sigma^*$ and a letter $\sigma \in \Sigma$, we denote with $w_{|\sigma}$ the number of σ's in w. We consider directed hyper-multi-graphs with self loops and 2 to 4 vertices per edge. More formally, we consider graphs of the form $G = (V, E)$, where V is a set of vertices and E a set of edges together with the function $f_E \colon E \to V^{[2..4]}$ which assigns each edge with a tuple consisting of 2 to 4 vertices incident to this edge. An edge is called a *loop* if all of its incident vertices are identical. We encode G by listing its edges separated by \$-signs. Vertices appearing in edges are encoded by strings of a's separated by #'s. To extract the encoded graph, we define the following decoding function. For $m \in \mathbb{N}$, $1 \leq k_i \leq 3, i \leq m$, let

$$\text{dec}_{\text{dir\_hyp\_mul}}\left(\prod_{i=1}^{m}\left(\mathsf{a}^{p_i}\prod_{j=1}^{k_i}(\#\mathsf{a}^{q_{i,j}})\,\$\right)\right) = G\,,$$

where $G = (V, E)$ with $V = \{v_{p_i} \mid i \in [m]\} \cup \bigcup\{v_{q_{i,j}} \mid j \leq k_i\}$, $E = \{e_1, e_2, \dots, e_m\}$, $f_E \colon e_i \mapsto (v_{p_i}, v_{q_{i,1}}, ..., v_{q_{i,k_i}})$. We present a graph-problem over

this class of graphs for which its int_{Reg}-problem is undecidable by encoding the sets of derivation-trees of two given context-free grammars in Chomsky-normal-form (CNF for short) into a regular set of directed hyper-multi-graphs. The languages of the two grammars will share a common word w if and only if the intersection of the constructed regular language with the graph-problem is non-empty and contains the two derivations of w. We call $\mathcal{G} = (V, T, P, S)$ a context free grammar in CNF, if V is a finite set of variables, T a finite set of terminal, P a set of derivation rules of the from $A \to BC$ or $A \to a$ with $A, B, C \in V$, $a \in T$, and $S \in V$ the start variable. We first give the construction and then define the graph problem Embedded Derivation Trees, EDT for short.

Theorem 5. *int_{Reg}(EDT) is undecidable.*

Proof. Let $\mathcal{G}_1 = (V_1, T, P_1, S'), \mathcal{G}_2 = (V_2, T, P_2, S'')$ be two context free grammars in CNF. We alter them, by introducing two new variables S_1 and S_2, to the grammars $\mathcal{G}_1 = (V_1 \cup \{S_1\}, T, P_1 \cup \{S_1 \to S'\}, S_1)$, $\mathcal{G}_2 = (V_2 \cup \{S_2\}, T, P_2 \cup \{S_2 \to S''\}, S_2)$. From now on we will identify $V_1 \cup \{S_1\}$ as V_1 and $V_2 \cup \{S_2\}$ as V_2. W.l.o.g. assume that the sets V_1, V_2 are disjoint and both grammars share the terminal alphabet T. Note that the following problem is undecidable: Is there a word $w \in T^*$ such that w can be derived from $\mathcal{G}_1$ and from $\mathcal{G}_2$?

Let $m_1 = |V_1|$, $m_2 = |V_2|$, $t = \frac{|T|(|T|+1)}{2}$, $n = m_1 + m_2 + 2t + 2$. We construct a regular set $R = R_1 \cdot R_2$, where R_1 (respectively R_2) is defined as follows: We fix an order on the elements in V_1, V_2, T such that S_1 is the first element in V_1 and S_2 is the first element in V_2. We refer to the i'th element of a set $\mathcal{S}$ as $\mathcal{S}[i]$. Let $V_1[s'] = S'$, $V_2[s''] = S''$ for integers s' and s''. For the derivation rule $S_1 \to S'$ in $\mathcal{G}_1$, we define the regular expression $r_{s_1} = \mathtt{a}^0\#\mathtt{a}^1\#\mathtt{a}^0\#\mathtt{a}^{s'}\$$ and for the derivation rule $S_2 \to S''$ in $\mathcal{G}_2$ we define $r_{s_2} = \mathtt{a}^{n-1}\#\mathtt{a}^{m_1+1}\#\mathtt{a}^{n-1}\#\mathtt{a}^{m_1+s''}\$$ For every derivation rule $p_1 = V_1[i] \to V_1[j]V_1[k]$, $i, j, k \leq m_1$ in P_1, we define the regular expression $r_{p_1} = \mathtt{a}^0\#\mathtt{a}^i(\mathtt{a}^n)^*\#\mathtt{a}^j(\mathtt{a}^n)^*\#\mathtt{a}^k(\mathtt{a}^n)^*\$$. For a derivation rule $p_2 = V_2[i] \to V_2[j]V_2[k]$, $i, j, k \leq m_2$ in P_2, we define $r_{p_2} = \mathtt{a}^{n-1}\#\mathtt{a}^{m_1+i}(\mathtt{a}^n)^*\#\mathtt{a}^{m_1+j}(\mathtt{a}^n)^*\#\mathtt{a}^{m_1+k}(\mathtt{a}^n)^*\$$. We define for each $j \in [|T|]$ and $b \in \{1, 2\}$ a regular expression which encodes a cycle of length j consisting of binary edges. We call these cycles *leave-cycle* later.

$$r_{lc}^{jb} = \prod_{k=m_1+m_2+(b-1)t+\frac{j(j+1)}{2}}^{m_1+m_2+(b-1)t+\frac{(j+1)(j+2)}{2}-2} \left(\mathtt{a}^k(\mathtt{a}^n)^*\#\mathtt{a}^{k+1}(\mathtt{a}^n)^*\$\right) \mathtt{a}^{m_1+m_2+(b-1)t+\frac{(j+1)(j+2)}{2}-1}(\mathtt{a}^n)^*\#\mathtt{a}^{m_1+m_2+(b-1)t+\frac{j(j+1)}{2}}(\mathtt{a}^n)^*\$$$

For derivation rules $q_1 = V_1[i] \to T[j]$ in P_1 and $q_2 = V_2[i] \to T[j]$ in P_2, we define: $r_{q_1} = \mathtt{a}^0\#\mathtt{a}^i(\mathtt{a}^n)^*\#\mathtt{a}^{m_1+m_2+\frac{j(j+1)}{2}}(\mathtt{a}^n)^*\$r_{lc}^{j1}$, and $r_{q_2} = \mathtt{a}^{n-1}\#\mathtt{a}^{m_1+i}(\mathtt{a}^n)^*\#\mathtt{a}^{m_1+m_2+t+\frac{j(j+1)}{2}}(\mathtt{a}^n)^*\$r_{lc}^{j2}$. We are now ready to define R_1 and R_2. We set $R_i = r_{s_i}\left(\bigcup_{p_i \in P_i} r_{p_i} \bigcup_{q_i \in P_i} r_{q_i}\right)^*$ for $i \in \{1, 2\}$.

We now define our graph property such that it filters out the encoded graphs in the regular set which consists of two derivation trees, one from $\mathcal{G}_1$ and one from $\mathcal{G}_2$, which derivate the same word. It is helpful to consider Figure [1] in the long version while reading though the following arguments.

Definition 7 (Multi-Graph Embedding ϕ).
Let $G = (V, E)$ be a directed hyper-multi-graph such that each edge contains two, three, or four vertices. The multi-graph embedding ϕ maps G onto a multi-graph $\phi(G) = G_m = (V_m, E_m)$ with $E_m \subseteq V_m \times V_m$ in the following way: $V_m = V$, for a 4-nary edge $(a, b, c, d) \in E$ we add the edges $(a, c), (b, c), (b, d)$ to E_m. For a ternary edge $(a, b, c) \in E$, we add the edges $(a, c), (b, c)$ to E_m. Binary edges are simply added to E_m.

Definition 8 (EDT).
Input: *A directed hyper-multi-graph $G = (V, E)$ with $f_E \colon E \to V^{[2..4]}$.*
Question: *Does the multi-graph embedding $\phi(G)$ consists fo two connected components G_{1m} and G_{2m} such that the following holds. G_{1m} contains a vertex v_1 and G_{2m} contains a vertex v_2, such that $G_{1m} \backslash \{v_1\}$ and $G_{2m} \backslash \{v_2\}$ are (directed) binary trees (where edges are pointing from parents to children) in which the leave layer consists of directed cycles (called* leave-cycles*). Exactly the root-node and the parents of leave-cycles have out-degree one, all other nodes which are not part of a leave-cycle have out-degree 2. The node v_1 is connected to exactly one child of each parent node in $G_{1m} \backslash \{v_1\}$ except for the root as here v_1 is pointing to the root and not to the child. The only node pointing towards v_1 is the root of $G_{1m} \backslash \{v_1\}$. The node v_1 has no further connections. The same holds for v_2 with respect to $G_{2m} \backslash \{v_2\}$. If G_{1m} and G_{2m} are drawn such that v_1 and v_2 always point to the left child, then the sequence of lengths of the leave-cycles of G_{1m} and G_{2m} (read from left to right) must coincide. Both graphs must not contain multi-edges and loops are only allowed as a leave-cycle. The leave-cycles are not connected to each other.*

We will first argue that for any $w \in R$ with $\mathrm{dec}_{\mathrm{dir\_hyp\_mul}}(w)$ being a positive instance of EDT the sub-graphs G_{1m} and G_{2m} of $\phi(\mathrm{dec}_{\mathrm{dir\_hyp\_mul}}(w))$ must correspond to two derivation trees, one for $\mathcal{G}_1$ and one for $\mathcal{G}_2$.

Note that for $i \equiv 0 \pmod n$ and $j \equiv -1 \pmod n$ the only vertex labels $\mathtt{a}^i(\#|\$)$ and $\mathtt{a}^j(\#|\$)$ which can be a factor of a word in R are $\mathtt{a}^0\#$ and $\mathtt{a}^{n-1}\#$. Especially, there are no factors of the form $(\mathtt{a}^n)^k(\#|\$)$ or $(\mathtt{a}^{n-1})(\mathtt{a}^n)^k(\#|\$)$ with $k > 0$ and the vertex $\mathtt{a}^0\#$ is only appearing in sub-graphs corresponding to derivation rules of $\mathcal{G}_1$ whereas $\mathtt{a}^{n-1}\#$ is only appearing in sub-graphs corresponding to derivation rules of $\mathcal{G}_2$. Hence, for a graph $G = \phi(\mathrm{dec}_{\mathrm{dir\_hyp\_mul}}(w))$ with $w \in R$ in order to consist of two disjoint graph G_{1m} and G_{2m} one of them must contain a vertex encoded by $\mathtt{a}^0\#$ and hence be constructed by $\mathcal{G}_1$ and the other one must contain a vertex encoded by $\mathtt{a}^{n-1}\#$ and be constructed by $\mathcal{G}_2$ as one of this elements is part of any regular expression $r_{s_i}, r_{p_i}, r_{q_i}$.

Note that by the definition of R_1 and R_2 each string $w \in R$ contains exactly one factor encoding the derivation rule $S_1 \to S'$ and one factor encoding the derivation rule $S_2 \to S''$. There are no other derivation rules in which S_1 or S_2 appear. Hence, there will be no factor $\mathtt{a}^1(\mathtt{a}^n)^k\#$ and $\mathtt{a}^{m_1+1}(\mathtt{a}^n)^k\#$ in w with $k > 0$. As r_{s_1} and r_{s_2} are the only regular expressions allowing to create an edge, the multi-edge embedding of which creates an arc pointing towards v_1 (encoded

by $\mathtt{a}^0\#$) and v_2 (encoded by $\mathtt{a}^{n-1}\#$), the root of the tree[2] $G_{m1}\backslash\{v_1\}$ necessarily be S_1 with the child S' and the root of the tree $G_{m2}\backslash\{v_2\}$ needs to be S_2 with child S''. The sub-tree starting in S' will then correspond to a derivation tree of $\mathcal{G}_1$, when the leave-cycle of length i is interpreted as the i-th letter of T, and the sub-tree starting in S'' will correspond to a derivation tree of $\mathcal{G}_2$.

W.l.o.g. we will focus on G_{m1}. By the definition of EDT G_{1m} must be a binary tree with leave-cycles and by previous arguments have the root S_1 with the single child S', all other internal parent-nodes have out-degree two. As G_{1m} does not contain any multi-edges and loops are only allowed as leave-cycles, every inner node $V_{1m}[i]$ has exactly two appearances of its encoding $\mathtt{a}^k\#$ in w namely in the 4-nary hyper-edge where it appears as one child and in the 4-nary hyper-edge where it appears as the parent node. The first edge corresponds to a derivation where the variable represented by $V_{1m}[i]$ is 'created' and the second edge to a derivation where this variable is 'consumed'. Hence, following the inner nodes gives us a derivation tree for derivations of the form $A \to BC$ in P_1.

Every member of a cycle of length k has its own domain of representatives which is disjoint with the domains of the other elements of the cycle. It is also disjoint with the domains of elements of cycles with different lengths. We show that if $w \in R$ encodes a positive instance of EDT, then for each leave-cycle C_i of length k which is the child of a node $V_{1m}[j']$ encoding the variable $V_1[j]$, the description of C_i is created by the regular expression r_{q_1} which encodes the derivation $V_1[j] \to T[k]$. Let $C_i[1]$ be the first node in the cycle C_i, i.e., the child of $V_{1m}[j']$. Then, there is a factor uv in w encoding the edge $(V_{1m}[j'], C_i[1])$ where u encodes the node $V_{1m}[j']$ and v encodes the node $C_i[1]$. We know that $u_{|\mathtt{a}} \equiv j \pmod n$ and $v_{|\mathtt{a}} \equiv m_1 + m_2 + \frac{k(k+1)}{2} \pmod n$. By the definition of R the node $C_i[1]$ can only point to a node $C_i[2]$ encoded by a factor x for which $x_{|\mathtt{a}} \equiv m_1 + m_2 + \frac{k(k+1)}{2} + 1 \pmod n$ for $k \geq 2$, or to itself for $k = 1$, but the only regular expressions which allow to create such a factor correspond to derivation rules $A \to T[k]$. Indeed single edges between the same type of nodes (where the number of $\mathtt{a}$'s have the same remainder modulo n) can be exchanged between different derivation rules with the same letter $T[k]$ on the right side but the number of nodes in a cycle can not be altered that way without losing a cycle structure or introducing forbidden multi-edges. Hence, we can assume that C_i is created by a regular expression encoding the derivation rule $V_1[j] \to T[k]$.

Replacing cycles of length k by the corresponding letters $T[k]$ completes our derivation tree constructed by the inner nodes with derivations of terminals. As the trees G_{1m} and G_{2m} have the same sequence of cycle lengths in the leave-level if the child pointing to v_1, respectively v_2 is drawn as the left child, the constructed derivations derive the same word.

For the other direction, we can construct a word $w \in R$ for which the graph $\mathrm{dec}_{\mathrm{dir\_hyp\_mul}}(w)$ is a Yes-instance from the two derivations of a common word $x \in \mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2)$ as follows. We go through the derivation trees from top to bottom, left to right and increase an index k with every new considered

[2] We interpret the cycles in the leave-layer as leaves and hence interpret the graph as a tree despite the fact that it contains cycles.

appearance of a variable. We encode each derivation according to the above defined regular expressions, s.t. for the encoding $\mathtt{a}^e$ of a variable appearance $e \div n = k$. Details on constructing w can be found in the long version [25]. □

References

1. Anderson, T., Loftus, J., Rampersad, N., Santean, N., Shallit, J.: Detecting Palindromes, Patterns and Borders in Regular Languages. Information and Computation **207**(11), 1096–1118 (2009)
2. Bodlaender, H., Heggernes, P., Lokshtanov, D.: Graph Modification Problems (Dagstuhl Seminar 14071) (2014)
3. Chaari, T., Chaabane, S., Aissani, N., Trentesaux, D.: Scheduling Under Uncertainty: Survey and Research Directions. In: International Conference on Advanced Logistics and Transport, ICALT 2014, Hammamet, Tunisia, May 1-3, 2014. pp. 229–234. IEEE (2014)
4. Desel, J., Reisig, W.: The Synthesis Problem of Petri Nets. Acta Informatica **33**(4), 297–315 (1996)
5. van Emde Boas, P.: The Convenience of Tilings. Lecture Notes in Pure and Applied Mathematics pp. 331–363 (1997)
6. Gao, X., Xiao, B., Tao, D., Li, X.: A Survey of Graph Edit Distance. Pattern Analysis and Applications **13**(1), 113–129 (2010)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
8. Güler, D., Krebs, A., Lange, K.J., Wolf, P.: Deciding Regular Intersection Emptiness of Complete Problems for PSPACE and the Polynomial Hierarchy. In: Klein, S.T., Martín-Vide, C., Shapira, D. (eds.) Language and Automata Theory and Applications - 12th International Conference, LATA 2018. Lecture Notes in Computer Science, vol. 10792, pp. 156–168. Springer (2018)
9. Hladík, M.: Interval Linear Programming: A Survey. In: Linear Programming – New Frontiers in Theory and Applications, chap. 2, pp. 85–120. Nova Science Publishers, New York (2012)
10. Hopcroft, J.E., Ullman, J.D.: Formal Languages and their Relation to Automata. Addison-Wesley series in computer science and information processing, Addison-Wesley (1969)
11. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory Languages and Computation. Addison-Wesley Publishing Company, Inc. (1979)
12. Horváth, S., Karhumäki, J., Kleijn, J.: Results Concerning Palindromicity. Elektronische Informationsverarbeitung und Kybernetik **23**(8/9), 441–451 (1987)
13. Ito, M., Katsura, M., Shyr, H.J., Yu, S.S.: Automata Accepting Primitive Words. Semigroup Forum **37**(1), 45–52 (1988)
14. Kari, J.: On the Undecidability of the Tiling Problem. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008: Theory and Practice of Computer Science, 34th Conference on Current Trends in Theory and Practice of Computer Science. Lecture Notes in Computer Science, vol. 4910, pp. 74–82. Springer (2008)
15. Liu, Y., Wang, J., Guo, J.: An Overview of Kernelization Algorithms for Graph Modification Problems. Tsinghua Science and Technology **19**(4), 346–357 (2014)
16. Rubtsov, A.A.: Regular Realizability Problems and Regular Languages. CoRR **abs/1503.05879** (2015)

17. Rubtsov, A.A., Vyalyi, M.N.: Regular Realizability Problems and Models of a Generalized Nondeterminism. CoRR **abs/1105.5894** (2011)
18. Stockmeyer, L.J., Meyer, A.R.: Word Problems Requiring Exponential Time (Preliminary Report). In: Aho, A.V., Borodin, A., Constable, R.L., Floyd, R.W., Harrison, M.A., Karp, R.M., Strong, H.R. (eds.) Proceedings of the 5th Annual ACM Symposium on Theory of Computing. pp. 1–9. ACM (1973)
19. Straubing, H.: Tiling Problems. `http://www.cs.bc.edu/~straubin/cs385-07/tiling`, accessed: 2018-09-03
20. Tarasov, S.P., Vyalyi, M.N.: Orbits of Linear Maps and Regular Languages. In: Kulikov, A.S., Vereshchagin, N.K. (eds.) Computer Science - Theory and Applications - 6th International Computer Science Symposium in Russia, CSR 2011. Lecture Notes in Computer Science, vol. 6651, pp. 305–316. Springer (2011)
21. Vyalyi, M.N.: On Regular Realizability Problems. Problems of Information Transmission **47**(4), 342–352 (2011)
22. Vyalyi, M.N.: On Expressive Power of Regular Realizability Problems. Problems of Information Transmission **49**(3), 276–291 (2013)
23. Vyalyi, M.N., Rubtsov, A.A.: On Regular Realizability Problems for Context-Free Languages. Problems of Information Transmission **51**(4), 349–360 (2015)
24. Wolf, P.: Decidability of the Regular Intersection Emptiness Problem. Master's thesis, Wilhelm Schickhard Institut für Informatik, Universität Tübingen, Sand 13, D 72076 Tübingen, Germany (2018)
25. Wolf, P.: From Decidability to Undecidability by Considering Regular Sets of Instances. CoRR **abs/1906.08027** (2019), `http://arxiv.org/abs/1906.08027`
26. Wolf, P.: On the Decidability of Finding a Positive ILP-Instance in a Regular Set of ILP-Instances. In: Hospodár, M., Jirásková, G., Konstantinidis, S. (eds.) Descriptional Complexity of Formal Systems - 21st IFIP WG 1.02 International Conference, DCFS 2019, Košice, Slovakia, July 17-19, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11612, pp. 272–284. Springer (2019)
27. Wolf, P., Fernau, H.: Regular Intersection Emptiness of Graph Problems: Finding a Needle in a Haystack of Graphs with the Help of Automata. CoRR **abs/2003.05826** (2020)