Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Master Thesis Computer Science

Decidability of the Regular Intersection Emptiness Problem

Petra Wolf

October 12, 2018

Reviewers

Prof. Dr. Klaus-Jörn Lange
(Theoretische Informatik)
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Prof. Dr. Michael Kaufmann
(Algorithmik)
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Wolf, Petra:
Decidability of the Regular Intersection Emptiness Problem
Master Thesis Computer Science
Eberhard Karls Universität Tübingen
Thesis period: 04.2018 - 10.2018

Abstract

We determine the decidability of the int_{Reg}-problem for several languages. The int_{Reg}-problem of a fixed language L asks whether the intersection of L with a given regular language R is empty or not. We prove undecidability of the int_{Reg}-problem for variations of the Machine Language, Bounded Tiling, Corridor Tiling, Bounded PCP, Equivalence of Regular Expressions in a shuffled encoding, and String Equivalence Modulo Padding in a shuffled encoding. We prove decidability of the int_{Reg}-problem for String Equivalence Modulo Padding in a sequential encoding as well as over a unary alphabet in both encodings, SAT, k-TQBF, True Quantified Boolean Formula, Integer Linear Programming, Vertex Cover, Independent Set, Knapsack, and Integer Knapsack. In the most cases we show the decidability of the int_{Reg}-problem by constructing a condensed automaton condensed(A) from a given DFA A. In L(condensed(A)) only finitely many words have to be tested for membership in the intersection (defined by int_{Reg}) under which there is one element in the intersection if and only there is one in the original regular language described by A. We develop the three techniques *merging*, *separating*, and *replacing* of constructing the automaton condensed(A) and demonstrate them on the problems Vertex Cover, Independent Set, and Knapsack.

Zusammenfassung

Wir präsentieren Entscheidbarkeitsresultate für das Schnittleerheitsproblem mit regulären Sprachen (kurz int_{Reg}) für diverse Sprachen. Das so genannte int_{Reg}-Problem für eine bestimmte Sprache L fragt, ob der Schnitt von L mit einer gegebenen regulären Sprache R leer ist oder nicht. Wir zeigen die Unentscheidbarkeit des int_{Reg}-Problems für folgende Sprachen: Verschiedene Varianten der Maschinen-Sprache, BOUNDED TILING, CORRIDOR TILING, BOUNDED PCP, EQUIVALENCE OF REGULAR EXPRESSIONS in einer geshuffelten Codierung und STRING EQUIVALENCE MODULO PADDING ebenfalls in einer geshuffelten Codierung. Dahingegen werden wir die Entscheidbarkeit des int_{Reg}-Problems für STRING EQUIVALENCE MODULO PADDING in einer sequentiellen Codierung, sowie über einem einelementigen Alphabet in beiden Codierungsvarianten, wie auch für SAT, k-TQBF, TRUE QUANTIFIED BOOLEAN FORMULA, INTEGER LINEAR PROGRAMMING, VERTEX COVER, INDEPENDENT SET, KNAPSACK und INTEGER KNAPSACK zeigen. In den meisten Fällen zeigen wir die Entscheidbarkeit des int_{Reg}-Problems durch die Konstruktion eines komprimierten Automaten condensed(A) aus einem gegebenen deterministischen endlichen Automaten A. Für das int_{Reg}-Problem der Sprache L und den aus A konstruierten Automaten condensed(A) gilt dann, dass ein Element im Schnitt von L und $L(A)$ genau dann liegt, wenn wir ein Element in $L(\text{condensed}(A)) \cap L$ unter endlich vielen Kandidaten aus $L(\text{condensed}(A))$ finden. Für die Konstruktion von condensed(A) präsentieren wir mit dem *Zusammenfassen*, *Separieren* und *Ersetzen* drei Techniken, die wir an den Problemen VERTEX COVER, INDEPENDENT SET und KNAPSACK demonstrieren.

Acknowledgements

I thank Prof. Dr. Klaus-Jörn Lange for giving me the opportunity to be a member of his team for several years. He enabled me to work on interesting research topics and aroused my enthusiasm for the topic of theoretical computer science.
I also thank Demen Güler for the joint work on "*Deciding Regular Intersection Emptiness of Complete Problems for PSPACE and the Polynomial Hierarchy*" [GKLW18].

I thank *Vector Informatik GmbH* in Stuttgart for sponsoring my scholarship Deutschlandstipendium.

Last but not least, I thank friends and family for their support over the last years. It was a pleasure to study together with you guys.

Contents

List of Figures

List of Algorithms

List of Tables

List of Abbreviations

3-CNF	Conjunctive Normal-Form with Three Literals
BPCP	BOUNDED POST'S CORRESPONDENCE PROBLEM (*problem*)
CFL	Context-Free Language
DFA	Deterministic Finite Automaton
EXP	Deterministic Exponential Runtime
EXPSPACE	Exponential Space
ILP	INTEGER LINEAR PROGRAMMING (*problem*)
int_{Reg}	Regular Intersection Emptiness
IS	INDEPENDENT SET (*problem*)
KS	KNAPSACK (*problem*)
NL	Nondeterministic Logarithmic Space
NP	Nondeterministic Polynomial Runtime
NTM	Nondeterministic Turing Machine
P	Deterministic Polynomial Runtime
PCP	POST'S CORRESPONDENCE PROBLEM (*problem*)
PH	Polynomial Hierarchy
PSPACE	Polynomial Space
QBF	Quantified Boolean Formula
RegEx	Regular Expression
SAT	SATISFIABILITY (*problem*)
SEQUENTIAL$\equiv_{\text{REGEX}}$	EQUIVALENCE OF REGULAR EXPRESSIONS IN A SEQUENTIAL ENCODING (*problem*)

SEQUENTIAL$\equiv_{\text{STRING}^{\epsilon}}$	SEQUENTIAL STRING EQUIVALENCE MODULO PADDING (*problem*)
SHUFFLED$\equiv_{\text{REGEX}}$	EQUIVALENCE OF REGULAR EXPRESSIONS IN A SHUFFLED ENCODING (*problem*)
SHUFFLED$\equiv_{\text{STRING}^{\epsilon}}$	SHUFFLED STRING EQUIVALENCE MODULO PADDING (*problem*)
TM	Turing Machine
TQBF	TRUE QUANTIFIED BOOLEAN FORMULA (*problem*)
UNARY-SEQUENTIAL$\equiv_{\text{REGEX}}$	EQUIVALENCE OF REGULAR EXPRESSIONS OVER UNARY ALPHABET IN A SEQUENTIAL ENCODING (*problem*)
UNARY-SEQUENTIAL$\equiv_{\text{STRING}^{\epsilon}}$	SEQUENTIAL STRING EQUIVALENCE OVER UNARY ALPHABET MODULO PADDING (*problem*)
UNARY-SHUFFLED$\equiv_{\text{REGEX}}$	EQUIVALENCE OF REGULAR EXPRESSIONS OVER UNARY ALPHABET IN A SHUFFLED ENCODING (*problem*)
UNARY-SHUFFLED$\equiv_{\text{STRING}^{\epsilon}}$	SHUFFLED STRING EQUIVALENCE OVER UNARY ALPHABET MODULO PADDING (*problem*)
VC	VERTEX COVER (*problem*)

Chapter 1

Introduction

1.1 Problem Definition and Related Work

This work examines the decidability of the so called *Regular Intersection Emptiness Problem*, to which we will refer as int_{Reg}. The problem is considered regarding a fixed language L and asks whether the intersection of L with a given regular language is non-empty.

Definition 1 ($int_{\mathrm{Reg}}(L)$)**.** Let R be a regular language. The question whether $L \cap R \neq \emptyset$ is defined as the int_{Reg}-problem of L or short $int_{\mathrm{Reg}}(L)$.

The problem primordially occurred in the task of distinguishing *families of formal languages*, like regular or context-free languages, from *complexity classes*. While for formal language classes we have pumping lemmas and plenty decidabilities, the complexity classes lack those arguments. This distinctions are emphasized by the existence of families of formal languages being densely complete in the classes NP, SAC^1 and $\mathrm{NSPACE}(\log n)$ [KL12, KLL15], but the differentiation of formal language classes from complexity classes currently relies only on examples [Lan96]. In order to find a criterion which generally separates the two kinds of language classes, we investigated the int_{Reg}-problem. Since families of formal language classes, like the regular and context-free languages and their subclasses, are closed under intersection with regular languages and have a decidable emptiness problem, a decidable int_{Reg}-problem seemed to be a good criterion candidate. But this criterion was voided by proving the decidability of the int_{Reg}-problem for True Quantified Boolean Formula [GKLW18], a problem which is PSPACE-complete [Pap03]. Since all known families of formal languages are contained in NP (even the largest known one in the OI-hierarchy [Dam82]) finding a problem (most likely) outside of NP with a decidable regular intersection emptiness problem, disqualifies a decidable int_{Reg}-problem as a criterion. Further work in this field has developed a technique called "*hiding*" which alters a decidable language L by expanding the words of L with redundant information and adding all *misshaped* words to L [Gü19]. The resulting language L' will have a decidable int_{Reg}-problem. This approach is still in work. While the technique of hiding heavily changes the notation of the language, in this work we are only interested in the decidability of *natural encoded* languages.

If we focus on the class NP, it turned out quickly that with SAT there are problems in NP with a decidable int_{Reg}-problem and with The Machine Language For NP there are problems inside NP with an undecidable int_{Reg}-problem. This thesis was motivated by the leading question what problems with a decidable int_{Reg}-problem have in common and what distinguishes them from problems with an undecidable int_{Reg}-problem. Therefore, several problems with different complexities are presented and the decidability of their int_{Reg}-problem is investigated. The aim was to find a list of conditions for the regarded language, inspired by the theorem of Greibach [Gre68], which lead to a decidable or undecidable int_{Reg}-problem. This goal could not be fulfilled, but we were able to develop three different techniques with which int_{Reg}-decidability can be shown. Those techniques also give an insight in the nature of problems and suggest a problem classification according to the composition of items in a problem instance. The techniques deal with *merging*, *separating* and *replacing* problem items in order to simplify a problem instance.

The search for a criterion for decidability of the int_{Reg}-problem is also a search for limits of decidability, as they where discovered for other questions. For example, language equivalence is undecidable for context-free languages in general, but becomes decidable for CFL's over unary alphabets [HU69]. The problem is also decidable if we consider right-linear context-free grammars, but undecidable for linear or arbitrary context-free grammars [HIRS76]. Just like Hunt, we are trying to identify "simple underlying properties" of the investigated languages which determine the decidability of their int_{Reg}-problem. We will show that the int_{Reg}-problem for string equivalence modulo padding is undecidable in a shuffled encoding but becomes decidable in a sequential encoding. The problem is also decidable for both encodings if we restrict the string to contain only one non-padding type of symbols.

This thesis is structured as follows: We begin in Chapter 2 with Turing machine related languages with an undecidable int_{Reg}-problem. Then, in Chapter 3 we investigate the more natural problem of Equivalence of Regular Expressions and prove undecidability of its int_{Reg}-problem. We differentiate here between different encodings and alphabet sizes uncovering some limits of decidability. In Chapter 4 we review the article "*Deciding Regular Intersection Emptiness of Complete Problems for PSPACE and the Polynomial Hierarchy*" [GKLW18] and introduce the construction of the finite automaton condensed(A) based on a given DFA A. With condensed(A) we will be able to solve the int_{Reg}-problem, since only finitely many elements in condensed(A) have to be considered in order to find a word in $L \cap L(A)$ if there exists one. The proof-schema of construction a condensed automaton condensed(A) in order to show decidability of the int_{Reg}-problem will stretch through the following chapters. In Chapter 5 we will adapt those ideas to show decidability of the int_{Reg}-problem for INTEGER LINEAR PROGRAMMING. Finally, in Chapter 6 we differentiate the techniques used in Chapter 4 and 5 and demonstrate each of them by proving decidability of the int_{Reg}-problem for VERTEX COVER, INDEPENDENT SET, and KNAPSACK. In Chapter 7 we discuss the four types of problems, which are induced by our developed techniques and the decidability of their int_{Reg}-problem. We also give an overview over the achieved int_{Reg}-decidability results spanning from problems in L to PSPACE-complete problems.

1.2 Preliminaries

We assume the reader to be familiar with common families of formal languages [HU69] and complexity classes [Pap03]. We demand a good understanding of regular languages, regular expressions, deterministic finite automata, and the Pumping Lemma for regular languages. We sometimes use regular expressions in the description of a language for readability reasons. Such a regular expression e is meant to be replaced by a variable string w and the condition $w \in L(e)$, where $L(e)$ denotes the languages described by e. The considered problems as well as their encodings are introduced in the referring chapters.

Chapter 2

Undecidable Regular Intersection Emptiness

2.1 Introduction

In this chapter we will focus on NP-hard languages with an undecidable int_{Reg}-problem. The presented problems are restricted version of well-known undecidable problems. In all three cases, the intersection with a regular language will remove the restriction and therefore show undecidability of the regular intersection emptiness problems.

2.2 Machine Languages

For several complexity classes, we can define a *machine language* which is complete for this complexity class. We will show that the following machine languages have an undecidable int_{Reg}-problem.

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

Definition 3 (Machine language for NP).
Given: Encoded nondeterministic Turing-Machine $\langle M \rangle$, input-word x, and string a^n with $n \in \mathbb{N}$.
Question: Does M accept x in n steps?
Encoding: $L_{\mathrm{NP}} = \{\langle M \rangle, x, a^n \mid M \text{ is an NTM accepting } x \text{ in } n \text{ steps}\}$.

Definition 4 (Machine language for PSPACE).
Given: Encoded deterministic Turing-Machine $\langle M \rangle$, input-word x, and string

a^n with $n \in \mathbb{N}$.
Question: Does M accept x visiting only n different tape-positions?
Encoding: $L_{\mathrm{PSPACE}} = \{\langle M\rangle, x, a^n \mid M$ is an TM accepting x in n space$\}$.

Theorem 1. *Let R be a regular language. It is undecidable whether the intersection of R with*

- *the machine language for NL*
- *the machine language for NP*
- *the machine language for PSPACE*

is empty or not, i.e. $int_{\mathrm{Reg}}(L_{NL})$, $int_{\mathrm{Reg}}(L_{NP})$, and $int_{\mathrm{Reg}}(L_{PSPACE})$ are undecidable.

Proof. Let $\langle M_{\text{fixed}}\rangle$ be an arbitrary but fixed encoded Turing-Machine with the input alphabet Σ. We define the regular language

$$R = \{\langle M_{\text{fixed}}\rangle, x, a^n \mid x \in \Sigma^*, n \geq 0\}.$$

Then,

- $R \cap L_{\mathrm{NL}} = \emptyset \Leftrightarrow L(M_{\text{fixed}}) = \emptyset$.
- $R \cap L_{\mathrm{NP}} = \emptyset \Leftrightarrow L(M_{\text{fixed}}) = \emptyset$.
- $R \cap L_{\mathrm{PSPACE}} = \emptyset \Leftrightarrow L(M_{\text{fixed}}) = \emptyset$.

Since the emptiness-problem for recursive enumerable sets is undecidable [HU69], the undecidability of the problems $int_{\mathrm{Reg}}(L_{\mathrm{NL}})$, $int_{\mathrm{Reg}}(L_{\mathrm{NP}})$, and $int_{\mathrm{Reg}}(L_{\mathrm{PSPACE}})$ follows. □

2.3 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 tile a plane with the given tiles under some conditions. While the problem for an infinite plane is undecidable [Str], it becomes NP-complete if we restrict the plane to an $n \times n$-square with a given edge coloring, and PSPACE-complete if we only restrict the width of the plane by a given edge coloring and ask for a finite height, such that the plane can be tiled [vEB$^+$97].

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 the emptiness-problem for Turing-Machines 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. Tiles can be regarded as instances of tile types. A tile must not be rotated or reflected. In the following problem, we will 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 preset the edge color of tiles at the border of the square grid. 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 $\{1, 2, \ldots, n\}$.

Definition 5 (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 while fulfilling the following condition, where the indexed function $f(i, j)_k$ references the color of the edge in position k of the tile $f(i, j)$.

$$f(i, j)_w = f(i + 1, j)_e, f(i, j)_n = f(i, j + 1)_s \tag{2.1}$$

Meaning adjacent edges of the tiles have the same color.
Encoding: Bounded Tiling := $\{\langle T\rangle, l\$t\$r\$b \mid$ there is a tiling f: $[n] \times [n] \to T$ fulfilling condition (2.1) and extending edge coloring $l\$t\$r\$b.\}$ Where $l = l_1\#l_2\#\ldots\#l_n$, $t = t_1\#t_2\#\ldots\#t_n$, $r = r_1\#r_2\#\ldots\#r_n$, and $b = b_1\#b_2\#\ldots\#b_n$ with $l_i, t_i, r_i, b_i \in C$.

Howard Straubing gives in his article “Tiling Problems” [Str] a reduction from the complement of the halting problem to the problem of tiling an infinite plane. Therefore, he gives an algorithms “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 [vEB$^+$97] 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 labellings are transformed into a set of tile types. The input word, padded with blank symbols, is encoded in the button 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. *Let R be a regular language. It is undecidable whether $R \cap$ Bounded Tiling $\neq \emptyset$, i.e. the problem int_{Reg}(Bounded Tiling) is undecidable.*

Proof. We will give a reduction from the undecidable problem $L_{\neq\emptyset} := \{\langle M\rangle \mid M \text{ is a nondeterministic TM with } L(M) \neq \emptyset\}$. Let $\langle M\rangle \in L_{\neq\emptyset}$. We will construct a regular language R which contains a solvable 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 [Str] and van Emde Boas [vEB$^+$97], 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 filed 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 signs, 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

$$R = \{\langle T_N\rangle, \diamond^* \ \$ \ q_f\square^* \ \$ \ \diamond^* \ \$ \ C_\Sigma{}^*\square^*\}.$$

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.

$$R \cap \text{Bounded Tiling} \neq \emptyset \Leftrightarrow L(N) \neq \emptyset$$

"$\Rightarrow$": The tile set of R was constructed from N in a way that valid tilings correspond to successive listed configurations of the NTM N. This means that, if $R \cap$ Bounded Tiling $\neq \emptyset$, R contains an edge colored square field which is large enough, such that there is a sequence of successive configurations fitting in the square field starting with the initial configuration containing the input word and ending in the single accepting configuration of N. Therefore, there is a finite sequence of configurations of the TM N, which proves that N accepts the input word and therefore $L(N) \neq \emptyset$.

"$\Leftarrow$": If on the other hand N accepts at least one word w in t steps using s space units, we can construct a tiling from the set of tile types of N. This tiling will fill a plane of size $t \times s$ of which the bottom row will contain the input word w and the top row will contain the single accepting configuration of length s. We can enlarge this tiling using white border tiles and the $\square$-sign tiles representing empty tape locations to a square tiling of size $\max(t, s) \times \max(t, s)$. If we take the edge coloring of this square tiling, we get an instance of Bounded Tiling which is obviously solvable. Since R contains every combination of input word and field size referring to N, it also contains the just described instance of Bounded Tiling and therefore $R \cap$ Bounded Tiling $\neq \emptyset$. $\square$

With the same argument, we can show that the PSPACE-complete problem Corridor Tiling [vEB+97] also has an undecidable int_{Reg}-problem.

Definition 6 (Corridor Tiling)**.**
Given: Finite set T of tile types with colors from a finite color set C, a top edge coloring t, and a bottom edge coloring b, both of length n.
Question: Is there a finite height h and a tiling function $f\colon [n] \times [h] \to T$ that tiles V extending the edge coloring while fulfilling the tiling Condition 2.1.

Theorem 3. *Let R be a regular language. It is undecidable whether $R \cap$ CORRIDOR TILING $\neq \emptyset$, i.e. the problem int$_{\text{Reg}}$(CORRIDOR TILING) is undecidable.*

Proof. The proof works analogously to the proof of Theorem 2 with a reduction from the emptiness problem of Type 0 languages, the only difference is that R only encodes the bottom and top borders and no left and right borders. Every rectangle can be embedded in a larger square and vice versa, so if there is an accepting run of the considered Turing machine, we will find a large enough square to hold the corresponding tiling as we have shown in the proof of Theorem 2. The upper and lower border of this square gives us the analogue corridor tiling instance in R. □

2.4 Bounded PCP

An other undecidable problem, which becomes decidable if we restrict the size of the potential solution, is the POST'S CORRESPONDENCE PROBLEM (short PCP). We will show that the NP-complete version BOUNDED POST CORRESPONDENCE PROBLEM [GJ79] (short BPCP) has an undecidable *int*$_{\text{Reg}}$-problem by a reduction from the unrestricted undecidable PCP problem [HU69]. The problem PCP will be studied in detail in chapter 3.

Definition 7 (BPCP).
Given: Finite alphabet Σ, two sequences $a = (a_1, a_2, \dots, a_n)$ and $b = (b_1, b_2, \dots, b_n)$ of strings from Σ^*, and a positive integer $K \leq n$.
Question: Is there a sequence $i_1, i_2, \dots, i_k$ of $k \leq K$ (not necessarily distinct) positive integers, each between 1 and n, such that the two strings $a_{i_1} a_{i_2} \dots a_{i_k}$ and $b_{i_1} b_{i_2} \dots b_{i_k}$ are identical?
Encoding: $L_{BPCP} := \{a_1\#a_2\#\dots\#a_n\$b_1\#b_2\#\dots\#b_n\$\,\text{bin}(K) \mid K \leq n \wedge a = (a_1, a_2, \dots, a_n), b = (b_1, b_2, \dots, b_n)$ is a PCP instance with a solution $\leq K\}$.

Theorem 4. *Let R be a regular language. It is undecidable whether $R \cap$ BPCP $\neq \emptyset$, i.e. the problem int$_{\text{Reg}}$(BPCP) is undecidable.*

Proof. We will give a reduction PCP $\leq$ *int*$_{\text{Reg}}$(BPCP). Let $a = (a_1, a_2, \dots, a_n)$ and $b = (b_1, b_2, \dots, b_n)$ 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 $a_{i_1} a_{i_2} \dots a_{i_m} = b_{i_1} b_{i_2} \dots b_{i_m}$.

"$\Rightarrow$": Let $w = a_1 \# a_2 \# \dots \# a_n (\# a_n)^l \$ b_1 \# b_2 \# \dots \# b_n (\# b_n)^l \$ \operatorname{bin}(K) \in R \cap \text{BPCP}$. Since w is a valid BPCP instance, the length of lists a and b are equal and $K \leq l + n$. The fact that $w \in \text{BPCP}$ means that there is a sequence of indexes $i_1, i_2, \dots, i_k$ such that $a_{i_1} a_{i_2} \dots a_{i_k} = b_{i_1} b_{i_2} \dots b_{i_k}$.

"$\Leftarrow$": Let $i_1, i_2, \dots, i_m$ be a sequence of indexes such that $a_{i_1} a_{i_2} \dots a_{i_m} = b_{i_1} b_{i_2} \dots b_{i_m}$. By the construction of R, there is a word $w \in R$ with $w = a_1 \# a_2 \# \dots \# a_n (\# a_n)^{m-n} \$ b_1 \# b_2 \# \dots \# b_n (\# b_n)^{m-n} \$ \operatorname{bin}(m) \in R \cap$. The lists a and b are padded with elements form the lists itself, until the input word is long enough, such that the length of the solution is less than the length of the input word and thereby can be bounded by K. If we set $K = k = m$, we have $w \in \text{BPCP}$.

$\square$

Chapter 3

Equivalence of Regular Expressions

3.1 Introduction

In this chapter we will show that the problem of EQUIVALENCE OF REGULAR EXPRESSIONS 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 union 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.

3.2 Regular Expressions in a Shuffled Encoding

We first want to define the problem of EQUIVALENCE OF REGULAR EXPRESSIONS (adapted from [GJ79]). For a regular expression E we denote with $L(E)$ the regular language described by E. We use concatenation implicitly and omit the operator sign.

Definition 8 (SHUFFLED$\equiv_{\text{RegEx}}$)**.**
Given: A word $w = e_{1,1}e_{2,1}e_{1,2}e_{2,2}e_{1,3}e_{2,3}\dots e_{1,n}e_{2,n}$ over the alphabet $\Sigma \cup \{\emptyset, \epsilon, (,), |, *\}$ such that $E_1 = e_{1,1}e_{1,2}e_{1,3}\dots e_{1,n}$ and $E_2 = e_{2,1}e_{2,2}e_{2,3}\dots e_{2,n}$ are regular expressions over the alphabet Σ using the operators *union*, *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 $L(E_1) = L(E_2)$?

The problem of equivalence of the regular expressions is well known to be PSPACE-complete [SM73]. Since we can change the encoding of an EQUIVALENCE OF REGULAR EXPRESSIONS instance from shuffled to sequential and vice versa in quadratic time, the shuffled version of this problem is also PSPACE-complete.

For readability reasons, we will refer to words $w \in \text{SHUFFLED}{\equiv}_{\text{REGEX}}$ as

$$w = \begin{matrix} e_{1,1} \\ e_{2,1} \end{matrix} \cdots \begin{matrix} e_{1,n} \\ e_{2,n} \end{matrix}.$$

We will show that $int_{\text{Reg}}(\text{SHUFFLED}{\equiv}_{\text{REGEX}})$ is undecidable by a reduction from the POST'S CORRESPONDENCE PROBLEM [HU79] short PCP.

Definition 9 (PCP).
Given: Two lists $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots, b_k$ with $k \in \mathbb{N}$ consisting of words over some alphabet Σ.
Question: Is there a sequence of integers $i_1, i_2, \ldots, i_n$ with $1 \leq i_j \leq k$ such that interpreted as indexes for strings, it holds that $a_{i_1}a_{i_2}\ldots a_{i_n} = b_{i_1}b_{i_2}\ldots b_{i_n}$.

Fact 1. *The* POST'S CORRESPONDENCE PROBLEM *(*PCP*) is undecidable [HU79].*

Reduction

From a given PCP instance we will construct a regular language L_{Reg} which words will describe possible solutions of the PCP instance. The words will consist of two shuffled regular expression 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 feasible solution.

Theorem 5. *Let R be a regular language. It is undecidable whether* $\text{SHUFFLED}{\equiv}_{\text{REGEX}} \cap R \neq \emptyset$*, i.e. the problem* $int_{\text{Reg}}(\text{SHUFFLED}{\equiv}_{\text{REGEX}})$ *is undecidable.*

Proof. We give a reduction $\text{PCP} \leq int_{\text{Reg}}(\text{SHUFFLED} \equiv_{\text{RegEx}})$ 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 highlighted. 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*

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

Let $\genfrac{}{}{0pt}{}{{a_i}'}{{b_i}'} = \genfrac{}{}{0pt}{}{{a_{i_1}}'}{{b_{i_1}}'}\genfrac{}{}{0pt}{}{{a_{i_2}}'}{{b_{i_2}}'} \ldots \genfrac{}{}{0pt}{}{{a_{i_l}}'}{{b_{i_l}}'}$ with $l := \max(|a_i|, |b_i|)$ where

$$\genfrac{}{}{0pt}{}{{a_{i_j}}'}{{b_{i_j}}'} = \begin{cases} \genfrac{}{}{0pt}{}{a_{i_j}}{b_{i_j}} & \text{if } |a_i| \geq j \wedge |b_i| \geq j, \\ \genfrac{}{}{0pt}{}{a_{i_j}}{\epsilon} & \text{if } |a_i| \geq j \wedge |b_i| < j, \\ \genfrac{}{}{0pt}{}{\epsilon}{b_{i_j}} & \text{if } |a_i| < j \wedge |b_i| \geq j. \end{cases}$$

The ϵ sign is here used as an alphabet symbol of the language L_{Reg}. Intuitively spoken, the string $\genfrac{}{}{0pt}{}{{a_i}'}{{b_i}'}$ consists of the two shuffled strings a_i, b_i where the shorter string is padded with ϵ-signs at the end until both stings have the same length. 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-signs 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 $L_{\text{Reg}} \cap$ SHUFFLED$\equiv_{\text{REGEX}}$ $\neq \emptyset$. More precisely, the intersection will contain all solutions of the PCP instance, if they exist.

"$\Rightarrow$": 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 $\genfrac{}{}{0pt}{}{{a_1}'}{{b_1}'}, \ldots, \genfrac{}{}{0pt}{}{{a_k}'}{{b_k}'}$ 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

$$w = \genfrac{}{}{0pt}{}{{a_{i_1}}'}{{b_{i_1}}'}\genfrac{}{}{0pt}{}{{a_{i_2}}'}{{b_{i_2}}'} \cdots \genfrac{}{}{0pt}{}{{a_{i_n}}'}{{b_{i_n}}'} .$$

The word w consists of the two shuffled regular expressions $E_1 = {a_{i_1}}'{a_{i_2}}' \ldots {a_{i_k}}'$ and $E_2 = {b_{i_1}}'{b_{i_2}}' \ldots {b_{i_k}}'$. Since they are both nonempty strings with padded ϵ's their described language is a singleton set. By construction, we have $L(E_1) = \{a_{i_1}a_{i_2}\ldots a_{i_k}\}$ and $L(E_2) = \{b_{i_1}b_{i_2}\ldots b_{i_k}\}$. 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 $L(E_1) = L(E_2)$ and $w \in L_{Reg} \cap$

Shuffled$\equiv_{\text{RegEx}}$.

"$\Leftarrow$": Assume $L_{\text{Reg}} \cap$ Shuffled$\equiv_{\text{RegEx}} \neq \emptyset$. Let $w = \genfrac{}{}{0pt}{}{a_{i_1}'}{b_{i_1}'} \genfrac{}{}{0pt}{}{a_{i_2}'}{b_{i_2}'} \dots \genfrac{}{}{0pt}{}{a_{i_n}'}{b_{i_n}'} \in L_{\text{Reg}} \cap$ Shuffled$\equiv_{\text{RegEx}}$ consists of the two shuffled regular expressions $E_1 = a_{i_1}'a_{i_2}' \dots a_{i_k}'$ and $E_2 = b_{i_1}'b_{i_2}' \dots b_{i_k}'$. By assumption is $L(E_1) = L(E_2)$. Since $L(E_1)$ and $L(E_2)$ each contain only one element, from which the describing regular expressions differ only by padded ϵ-signs, it holds by construction that $a_{i_1}a_{i_2} \dots a_{i_n} = b_{i_1}b_{i_2} \dots b_{i_n}$. Therefore, $i_1, i_2, \dots, i_n$ is a solution of the PCP instance. $\square$

3.3 String Equivalence

To show the undecidability of the int_{Reg}(Shuffled$\equiv_{\text{RegEx}}$) problem we have made use of only one operator of regular expressions, namely the concatenation. If we restrict the Shuffled$\equiv_{\text{RegEx}}$ problem to regular expressions using only letters from Σ, the ϵ-sign and the concatenation, we get the much easier problem of Shuffled String Equivalence Modulo Padding, short 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 all can be solved deterministically using two pointers.

Definition 10 (Shuffled$\equiv_{\text{String}^\epsilon}$).
Given: A word $w = s_{1,1}s_{2,1}s_{1,2}s_{2,2}s_{1,3}s_{2,3} \dots s_{1,n}s_{2,n}$ such that $s_i \in \Sigma \cup \{\epsilon\}$.
Question: Is $h(s_{1,1}s_{1,2}s_{1,3} \dots s_{1,n}) = h(s_{2,1}s_{2,2}s_{2,3} \dots s_{2,n})$ where $h : \Sigma \cup \{\epsilon\} \to \Sigma$ is an erasing homomorphism which leaves all signs in Σ unchanged and deletes the ϵ-signs.

Theorem 6. *Let R be a regular language. It is undecidable whether* Shuffled$\equiv_{\text{String}^\epsilon}$ $\cap R \neq \emptyset$*, i.e. the problem int_{Reg}(*Shuffled$\equiv_{\text{String}^\epsilon}$*) is undecidable.*

Proof. In the proof of Theorem 5 we have constructed a *regular language* of shuffled regular expressions which described singleton sets by using concatenation and padding with ϵ-signs. So, the two regular expressions describe the same language if and only if the regular expressions themselves yield the same string under deleting the ϵ-signs. Therefore, the proof of Theorem 5 also works for Theorem 6. $\square$

If we restrict the Problem Shuffled$\equiv_{\text{String}^\epsilon}$ to unary alphabets Σ, the int_{Reg} problem becomes decidable.

Definition 11 (Unary-Shuffled$\equiv_{\text{String}^\epsilon}$).
Given: A word $w = s_{1,1}s_{2,1}s_{1,2}s_{2,2}s_{1,3}s_{2,3} \dots s_{1,n}s_{2,n}$ such that $s_i \in \Sigma \cup \{\epsilon\}$ with

$|\Sigma| = 1$.
Question: Is $h(s_{1,1}s_{1,2}s_{1,3}\dots s_{1,n}) = h(s_{2,1}s_{2,2}s_{2,3}\dots s_{2,n})$ where $h : \Sigma \cup \{\epsilon\} \to \Sigma$ is an erasing homomorphism which leaves all signs in Σ unchanged and deletes the ϵ-signs.

Theorem 7. *Let R be a regular language. It is decidable whether* UNARY-SHUFFLED$\equiv_{\text{STRING}^\epsilon}$ $\cap R \neq \emptyset$*, i.e. the problem* $\mathit{int}_{\text{Reg}}$(UNARY-SHUFFLED$\equiv_{\text{STRING}^\epsilon}$) *is decidable.*

Proof. The language UNARY-SHUFFLED$\equiv_{\text{STRING}^\epsilon}$ is context-free, since for a given word we only have to count the number of letters unequal to ϵ at the even and at the odd positions in the word. If those numbers are equal, the word is in UNARY-SHUFFLED$\equiv_{\text{STRING}^\epsilon}$. This property can be checked by a deterministic pushdown automaton and hence the language is context-free. Since the context-free languages are closed under intersection with regular languages and have a decidable emptiness problem [HU69], the problem $\mathit{int}_{\text{Reg}}$(UNARY-SHUFFLED$\equiv_{\text{STRING}^\epsilon}$) is decidable, too. □

The $\mathit{int}_{\text{Reg}}$ problem becomes also decidable if we get rid of the shuffled encoding. The following two problems have a decidable $\mathit{int}_{\text{Reg}}$ problem as well.

Definition 12 (SEQUENTIAL$\equiv_{\text{STRING}^\epsilon}$)**.**
Given: A word $w = s_1s_2\dots s_n\$t_1t_2\dots t_{n'}$ such that $s_i, t_i \in \Sigma \cup \{\epsilon\}$.
Question: Is $h(s_1s_2\dots s_n) = h(t_1t_2\dots t_{n'})$ where $h : (\Sigma \cup \{\epsilon, \$\})^* \to (\Sigma \cup \{\$\})^*$ is an erasing homomorphism which leaves all signs in $\Sigma \cup \{\$\}$ unchanged and deletes the ϵ-signs.

Theorem 8. *Let R be a regular language. It is decidable whether* SEQUENTIAL$\equiv_{\text{STRING}^\epsilon}$ $\cap R \neq \emptyset$*, i.e. the problem* $\mathit{int}_{\text{Reg}}$(SEQUENTIAL$\equiv_{\text{STRING}^\epsilon}$) *is decidable.*

Proof. Let R be a regular language. Without loss of generality, we may assume that $R \subseteq L\left((\Sigma \cup \{\epsilon\})^*\ \$\ (\Sigma \cup \{\epsilon\})^*\right)$ [1]. Let $A = (Q, \Sigma \cup \{\epsilon, \$\}, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) = R$. We will provide a decision procedure for the question, whether R contains a word from SEQUENTIAL$\equiv_{\text{STRING}^\epsilon}$ by splitting the automaton A in sub-automata accepting every sub-word before the symbol \$ and sub-automata accepting every sub-word after the symbol \$. We will then use the fact, that the class of regular languages is closed under erasing-homomorphisms [HU69], to delete the ϵ-signs from the words of the sub-automata and obtain regular languages, for which we then perform pairwise intersection emptiness tests.

[1]The language SEQUENTIAL$\equiv_{\text{STRING}^\epsilon}$ is a subset of the regular language $L\left((\Sigma \cup \{\epsilon\})^*\ \$\ (\Sigma \cup \{\epsilon\})^*\right)$, hence we can narrow R to the intersection of R with $L\left((\Sigma \cup \{\epsilon\})^*\ \$\ (\Sigma \cup \{\epsilon\})^*\right)$.

We define for every pair of states of the automaton A the set of sub-words, which can be read before the \$-sign, between which the \$-sign can be read, and which can be read after the \$-sign.

$$\begin{aligned}
&S_{q_0,q} := \{w \in (\Sigma \cup \{\epsilon\})^* \mid \delta(q_0, w) = q \wedge \exists q' \in Q : \delta(q, \$) = q'\} \\
&\$_{q,q'} := \{\$ \mid \delta(q, \$) = q'\} \\
&T_{q,q_f} := \{w \in (\Sigma \cup \{\epsilon\})^* \mid \delta(q, w) = q_f \wedge q_f \in F \wedge \exists q' \in Q : \delta(q', \$) = q\}
\end{aligned}$$

Therefore, R can be written as $R = \bigcup_{q,q' \in Q,\, q_f \in F} S_{q_0,q}\$_{q,q'}T_{q',q_f}$. Since A is a deterministic finite automaton, there are only finitely many sets $S_{q_0,q}$, $\$_{q,q'}$, and T_{q,q_f} and all of them are regular, since we easily can alter the automaton A to obtain finite automata for each of those languages.

Let $h\colon (\Sigma \cup \{\epsilon, \$\})^* \to (\Sigma \cup \{\$\})^*$ be a homomorphism mapping every symbol form $\Sigma \cup \{\$\}$ to itself and deleting the ϵ-signs. For every pair of states, the languages $h\left(S_{q_0,q}\right)$, $h\left(\$_{q,q'}\right)$, and $h\left(T_{q,q_f}\right)$ are regular. For two regular languages the intersection emptiness problem is decidable, i.e. it is decidable, whether both languages contain a common word [HU69].

We will now show that $\textsc{Sequential}_{\equiv_{\textsc{String}^\epsilon}} \cap R \neq \emptyset$ if and only if there exists states $q, q', q_f \in Q$ and a word $v \in \Sigma^*$ such that $v \in h\left(S_{q_0,q}\right) \wedge \$ \in h\left(\$_{q,q'}\right) \wedge v \in h\left(T_{q',q_f}\right)$. Since there are only finitely many states in Q, and the membership and intersection-emptiness problems for regular languages are decidable, we can simply test the right-hand condition for every combination of states to decide whether there exists a word $v \in \Sigma^*$ fulfilling the condition.

"$\Rightarrow$": Let $w \in \textsc{Sequential}_{\equiv_{\textsc{String}^\epsilon}} \cap R \neq \emptyset$, then w is the label of an accepting path in A. By definition, w is of the form $w = u\$u'$ where the stings u and u' are equal if we delete the ϵ-signs in both strings. This means that $h(u) = h(u')$. Since w labels an accepting path in A, there are states q, q', q_f such that $\delta(q_0, u) = q \wedge \delta(q, \$) = q' \wedge \delta(q', u') = q_f \wedge q_f \in F$. Hence, there exists sets $S_{q_0,q}$, $\$_{q,q'}$, and T_{q,q_f} with $u \in S_{q_0,q}$, $\$ \in \$_{q,q'}$, and $u' \in T_{q,q_f}$. Let $v = h(u)$, then $v \in h\left(S_{q_0,q}\right) \wedge \$ \in h\left(\$_{q,q'}\right) \wedge v \in h\left(T_{q',q_f}\right)$ which was to be proven.

"$\Leftarrow$": Let $v \in \Sigma^*$ be a word, such that there exist states $q, q', q_f \in Q$ with $v \in h\left(S_{q_0,q}\right) \wedge \$ \in h\left(\$_{q,q'}\right) \wedge v \in h\left(T_{q',q_f}\right)$. Let $u \in \left(S_{q_0,q} \cap h^{-1}(v)\right)$, $\$ \in \left(\$_{q,q'} \cap h^{-1}(\$)\right)$, and $u' \in \left(T_{q',q_f} \cap h^{-1}(v)\right)$. All three strings $u, \$, u'$ exist and are non-empty by assumption. Therefore, $\delta(q_0, u) = q \wedge \delta(q, \$) = q' \wedge \delta(q', u') = q_f \wedge q_f \in F$ and $u\$u'$ is the label of an accepting path in A. Since u and u' are equal when we delete the ϵ-signs, the word $u\$u'$ is also in $\textsc{Sequential}_{\equiv_{\textsc{String}^\epsilon}}$. $\square$

Definition 13 (UNARY-SEQUENTIAL$\equiv_{\text{STRING}^\epsilon}$).
Given: A word $w = s_1 s_2 \dots s_n \$ t_1 t_2 \dots t_{n'}$ such that $s_i, t_i \in \Sigma \cup \{\epsilon\}$ with $|\Sigma| = 1$.
Question: Is $h(s_1 s_2 \dots s_n) = h(t_1 t_2 \dots t_{n'})$ where $h : (\Sigma \cup \{\epsilon, \$\})^* \to (\Sigma \cup \{\$\})^*$ is an erasing homomorphism which leaves all signs in $\Sigma \cup \{\$\}$ unchanged and deletes the ϵ-signs.

Corollary 1. *Let R be a regular language. It is decidable whether* UNARY-SEQUENTIAL$\equiv_{\text{STRING}^\epsilon} \cap R \neq \emptyset$, *i.e. the problem* int_{Reg}(UNARY-SEQUENTIAL$\equiv_{\text{STRING}^\epsilon}$) *is decidable.*

Proof. Since the problem is already decidable for an arbitrary alphabet, it is also decidable for a unary alphabet. □

3.4 Open Questions

We have shown that the problem of SHUFFLED$\equiv_{\text{REGEX}}$ has an undecidable int_{Reg}-problem. We do not know the decidability of the int_{Reg}-problem for the following problems, which are defined similarly to the variations of the $\equiv_{\text{String}^\epsilon}$-problems.

- SEQUENTIAL$\equiv_{\text{REGEX}}$
- UNARY-SHUFFLED$\equiv_{\text{REGEX}}$
- UNARY-SEQUENTIAL$\equiv_{\text{REGEX}}$

For the problem of SEQUENTIAL$\equiv_{\text{REGEX}}$ the reduction from PCP fails because we can not describe the set of all eventual solutions of the PCP-instance by a *regular set* of regular expressions. The corresponding list-items in one possible solution are arbitrarily far apart from each other, because the two regular expressions are not in a shuffled but sequential encoding. Therefore, the relation between the corresponding list elements can no longer be generated by a *regular expression*, even if we allow padding ϵ-signs.

For the problems UNARY-SHUFFLED$\equiv_{\text{REGEX}}$ and UNARY-SEQUENTIAL$\equiv_{\text{REGEX}}$ the reduction from PCP fails to prove undecidability of the int_{Reg}-problem because the PCP-problem over unary alphabets is decidable [RW03].

At the attempt to prove decidability of the int_{Reg}-problem for the above listed problems, we have to deal with the fact that we can not restrict the given regular language to a regular language, which only contains correctly encoded problem instances, as we can do in all cases of proven int_{Reg} decidability. Allowing operators like union and star in the regular expressions automatically brings brackets to the regular expressions and therefore the language of all correctly encoded regular expressions is no longer a regular set. The fact, that we can not restrict the given regular language R to a regular language of a known shape, containing the interesting subset of R, makes using pumping arguments, like presented in the next chapters, difficult.

Finally, in Table 3.1 we sum up the above proven decidability results of the int_{Reg}-problem for several variations of the EQUIVALENCE OF REGULAR EXPRESSIONS-problem.

Table 3.1: Decidability of the int_{Reg}-problem for regular expression and string equivalence.

	$\equiv_{\text{RegEx}}$		$\equiv_{\text{String}^\epsilon}$	
Σ	Shuffled	Sequential	Shuffled	Sequential
N-ary	undecidable	?	undecidable	decidable
Unary	?	?	decidable	decidable

Chapter 4

True Quantified Boolean Formula

This chapter includes the article "*Deciding Regular Intersection Emptiness of Complete Problems for PSPACE and the Polynomial Hierarchy*" [GKLW18], which was a collaboration of the author together with Demen Güler, Andreas Krebs, and Klaus-Jörn Lange. The presented construction of a so called automaton condensed(A) is here given in detail and will be adapted to other problems in the following chapters. The present chapter differs from the original article by additional notes and comments.

In this chapter we investigate the decidability of the int_{Reg}-problem for different languages of true quantified Boolean formulae. Arbitrary true quantified Boolean formulae generate a PSPACE-complete language, where constraining quantification depth and order yields complete-languages for the classes of the polynomial hierarchy. We show for Σ_k^P-, Π_k^P- and respectively PSPACE-complete languages L_{Σ_k}, L_{Π_k} and L_{TQBF} that $int_{\text{Reg}}(L_{\Sigma_k})$, $int_{\text{Reg}}(L_{\Pi_k})$ and $int_{\text{Reg}}(L_{\text{TQBF}})$ are decidable. We adopt an encoding for QBFs that only differentiates from the commonly used notation by omitting explicit quantifiers. Instead, literals carry information about their quantification depth as a unary string.

A converse viewpoint of this problem is to consider a regular set of encoded quantified Boolean formulae and to decide whether at least one does evaluate to true. For finite sets this problem is, from a decidability perspective, trivial, since each quantified Boolean formula can be evaluated by a Turing machine with a polynomial space bound. On the contrary, the question whether an arbitrary (not necessarily regular) infinite set contains a true quantified Boolean formula is not per se decidable. In our proof we make use of the finiteness of the state set of finite automata which assure us the repetitions of certain sub-words in words accepted by the automaton. This way, we are able to reduce the set of possibilities to a finite number of candidates, when we search for true quantified Boolean formulae in infinite regular sets.

The chapter is structured as follows: We start with preliminaries and define the complete problems we need. The main results are listed in Section 4.2, where a short overview of the proof ideas are given. The detailed elaboration of the proofs is content of Section 4.3 and 4.4. Finally, in the last section we discuss our results and list some open problems.

4.1 Preliminaries

We use the common notation for Boolean formulae with 0 and 1 as truth values. For readability reasons, we extend regular expressions by operations $E^{\leq n} := (E|E^2|\cdots|E^n)$ and $E^{\geq n} := E^nE^*$ for a fixed $n \in \mathbb{N}$ and E a regular expression. In particular, we write $E^{\geq 1}$ instead of E^+ because signs are part of the alphabet we use. For $w \in \Gamma^*$ and $\gamma \in \Gamma$ let $\#_\gamma(w)$ denote the number of γs in w. Existentially and universally quantifying a propositional formula with Boolean values yields a *quantified Boolean formula*: $\mathcal{Q}_1\vec{x}_1\mathcal{Q}_2\vec{x}_2\dots\mathcal{Q}_k\vec{x}_k\ \phi(\vec{x}_1,\vec{x}_2,\dots,\vec{x}_k)$, where $\mathcal{Q}_i \in \{\exists,\forall\}$ with $\mathcal{Q}_i \neq \mathcal{Q}_{i+1}$ and $\vec{x}_i$ are finite vectors of Boolean variables and ϕ is a propositional Boolean formula with $\sum_{i=1}^{k}|\vec{x}_i|$ free variables.

Wrathall [Wra76] showed that true quantified Boolean formulae with k alternating quantifiers and suitable propositional structure yield complete languages for the k-th level of the polynomial hierarchy. The shape of the propositional formula depends on the innermost quantifier. If the innermost quantifier is existential (universal), it is in conjunctive (disjunctive) normal form. The set of true quantified Boolean formulae with k alternating quantifiers, where $\mathcal{Q}_1 = \exists$ ($\mathcal{Q}_1 = \forall$), is complete for Σ_k^P (Π_k^P). If the propositional formula is kept in CNF regardless of the innermost quantifier, then for each odd (even) k and $\mathcal{Q}_1 = \exists$ ($\mathcal{Q}_1 = \forall$) the true quantified formulae with k alternating quantifiers yield Σ_k^P (Π_k^P)-complete language. If the number of quantifier alternations in formulae is not bounded the set of true quantified Boolean formulae yields the PSPACE-complete set TQBF [SM73].

Thoughout the article we will describe regular sets of quantified Boolean formulae. The typical representation of quantified Boolean formulae is not regular. We will use a natural encoding (similar to Stockmeyer [Sto76]) of quantified Boolean formulae, where literals $\pm b^*a^*$ contain three kinds of information: A sign will indicate whether the variable is negated or not, followed by a factor b^* which indicates the quantification depth and the name or index of the variable used in this literal as suffix in a^*.

Odd length quantification levels denote that variables are existentially quantified and analogously even levels that they are universally quantified. Also, we will assume the propositional structure to be in 3-CNF.

Example 1. *Consider following quantified Boolean formula ψ_1 over the variable set $\vec{x}_1 = (x_{1,1}, x_{1,2})$, $\vec{x}_2 = (x_{2,1}, x_{2,2})$ and $\vec{x}_3 = (x_{3,1}, x_{3,2})$.*

$$\psi_1 = \exists\vec{x}_1\forall\vec{x}_2\exists\vec{x}_3\ (x_{1,1} \vee \neg x_{1,2} \vee \neg x_{2,1}) \wedge (x_{3,1} \vee x_{2,2} \vee x_{1,2}) \wedge (\neg x_{3,2} \vee x_{1,1} \vee \neg x_{2,1})$$

The encoding of ψ_1 then reads

$$\langle +ba \vee -baa \vee -bba\rangle \wedge \langle +bbba \vee +bbaa \vee +baa\rangle \wedge \langle -bbbaa \vee +ba \vee -bba\rangle\ .$$

The Boolean vectors of variables are not necessarily indexed by successive numbers.

$$\psi_2 = \exists\vec{x}_1\forall\vec{x}_4\exists\vec{x}_5\ (x_{1,1} \vee \neg x_{1,2} \vee \neg x_{4,1}) \wedge (x_{5,1} \vee x_{4,2} \vee x_{1,2}) \wedge (\neg x_{5,2} \vee x_{1,1} \vee \neg x_{2,1})\ ,$$

where the Boolean vectors are $\vec{x}_1 = (x_{1,1}, x_{1,2})$, $\vec{x}_4 = (x_{4,1}, x_{4,2})$, $\vec{x}_5 = (x_{5,1}, x_{5,2})$ and thus, $\psi_1 \simeq \psi_2$ since ψ_2 is a (partial) renaming of ψ_1. The encoding of ψ_2 then is

$$\langle +ba \vee -baa \vee -b^4a\rangle \wedge \langle +b^5a \vee +b^4aa \vee +baa\rangle \wedge \langle -b^5aa \vee +ba \vee -bba\rangle\ .$$

The Boolean vectors of variables are not necessarily indexed by successive numbers. If two consecutive vectors are both indexed odd/even they will be quantified identically. Let ψ with range over $\vec{x}_1 = (x_{1,1})$, $\vec{x}_3 = (x_{3,2}, x_{3,4})$, $\vec{x}_4 = (x_{4,2})$ and $\vec{x}_8 = (x_{8,1}, x_{8,2})$ be

$$\exists\vec{x}_1\exists\vec{x}_3\forall\vec{x}_4\forall\vec{x}_8(x_{1,1} \vee x_{8,1} \vee x_{3,2}) \wedge (x_{3,4} \vee x_{4,2} \vee \neg x_{4,2}) \wedge (\neg x_{3,4} \vee x_{8,2} \vee x_{3,4})$$

Then the encoding of ψ reads

$$\langle +ba \vee +b^8a \vee +b^3aa\rangle \wedge \langle +b^3a^4 \vee +b^4aa \vee -b^4aa\rangle \wedge \langle -b^3a^4 \vee +b^8aa \vee +b^3a^4\rangle$$

Definition 14. Let $\Gamma = \{a, b, \langle, \rangle, \wedge, \vee, +, -\}$ and $\pm$ be the regular expression of $\{+, -\}$. As the regular set of sequential encoded quantified Boolean formulae in 3-CNF we define $L_{k\text{-QBF}} :=$

$$\left\{\langle \pm b^{\leq k}a^{\geq 1} \vee \pm b^{\leq k}a^{\geq 1} \vee \pm b^{\leq k}a^{\geq 1}\rangle \left(\wedge\langle \pm b^{\leq k}a^{\geq 1} \vee \pm b^{\leq k}a^{\geq 1} \vee \pm b^{\leq k}a^{\geq 1}\rangle\right)^*\right\}$$

Furthermore, let $L_{\text{QBF}} := \bigcup_{k\geq 1} L_{k\text{-QBF}}$ be the set of encoded quantified Boolean formulae without bound of quantifier alternation depth.

Remark 1. *The results presented also work for some slightly modified encodings. For example, variables and quantification depth could be encoded in binary (as Stockmeyer [Sto76] does) and arbitrary but fixed clause sizes could be allowed. We could also get rid of the brackets or encode signs inside variable names.*

Definition 15. Let L_{Σ_k} $(L_{\Pi_k}) \subseteq L_{k\text{-QBF}}$ be the set of all true quantified Boolean formulae in sequential encoding and 3-CNF where the first quantifier is existential (universal).

Let $L_{k\text{-TQBF}} := L_{\Sigma_k} \cup L_{\Pi_k}$ and $L_{\text{TQBF}} := \bigcup_{k\geq 1} L_{k\text{-TQBF}}$.

Fact 2. *For odd (even) k, the language L_{Σ_k} (L_{Π_k}) is Σ_k^P $\left(\Pi_k^P\right)$-complete [Wra76]. The set $L_{TQBF} \subseteq L_{QBF}$ of encoded true quantified Boolean formulae in 3-CNF is PSPACE-complete [SM73].*

4.2 Results

In this section we present our two main theorems for the Σ_k^P-complete language L_{Σ_k}, the Π_k^P-complete language L_{Π_k} and the PSPACE-complete language L_{TQBF}.

Theorem 9. *Let R be a regular language. For each $k \in \mathbb{N}$ it is decidable whether $L_{\Sigma_k} \cap R = \emptyset$ and $L_{\Pi_k} \cap R = \emptyset$, i.e. $int_{\text{Reg}}\left(L_{\Sigma_k}\right)$ and $int_{\text{Reg}}\left(L_{\Pi_k}\right)$ are decidable.*

The next paragraph sketches the proof. The formal proof of Theorem 9 is content of Section 4.3.

Let A be a DFA recognizing R. We can use the structure of A to identify the finitely many "most promising" words in R under which a true quantified Boolean formula is contained, if and only if there is one in the language R. To do so, for every pair of states in A we compute the (possibly empty) regular set of end-to-end literals that can be read in-between them. Each such literal set is then assigned a finite set of *representing literals* in a way that universally quantified literals are referencing the same variable, while existentially quantified literals are referencing different variables. We define an automaton condense(A) based on the finitely many representatives and show that condense(A) recognizes a true quantified Boolean formula if and only if A accepts one. Finally, we show that for each $k \in \mathbb{N}$ the emptiness of condense(A) $\cap L_{\Sigma_k}$ and condense(A) $\cap L_{\Pi_k}$ is decidable, which in total proves Theorem 9.

Theorem 10. *Let R be a regular language. It is decidable whether $L_{TQBF} \cap R = \emptyset$, i.e. $int_{\text{Reg}}\left(L_{TQBF}\right)$ is decidable.*

Again, we will give a brief overview of the proof idea here and prove Theorem 10 in Section 4.4.

Let R be given as a DFA A. Formulae recognized by A can be unbounded in their quantification depth. Using pumping arguments, we can find a constant d limiting the quantification depth of "interesting formulae", where d is only dependent on the size of A. We construct a new automaton *restrict*(A) which only accepts formulae of quantification depth up to d and we show that A accepts a *true* quantified Boolean formula if and only if *restrict*(A) accepts one. Following Theorem 9 it is decidable whether $L(\mathit{restrict}(A))$ contains a true quantified Boolean formula with at most d alternating quantifiers and thus $L_{\mathrm{TQBF}} \cap R = \emptyset$ is decidable, too.

4.3 Decidability of $\mathit{int}_{\mathrm{Reg}}$ for Problems in the Polynomial Hierarchy

Let $R \subseteq L_{k\text{-QBF}}$ be a regular language. If R is finite, checking whether R contains at least one true quantified Boolean formula can be achieved by decoding and evaluating every single word in R.

For infinite R this procedure is obviously not possible. Instead, we will show that we only have to test finitely many quantified Boolean formulae in R for each (infinite) regular language to decide the intersection emptiness with L_{Σ_k}/L_{Π_k}.

The idea is to extract from R a finite subset $\{w_1, \dots, w_n\} \subseteq R$ such that $R \cap L_{\Sigma_k} \neq \emptyset$ if and only if $w_i \in L_{\Sigma_k}$ for some i. In order to do so, we look in a word $x \wedge \langle l_1 \vee l_2 \vee l_3 \rangle \wedge y \in R$ for literals l_i which are existentially quantified and make following case distinction. If the finite automaton A accepting R has a loop while reading l_i, we can single out a uniquely referenced variable for l_i. Thus, l_i can be existentially satisfied without effecting other literals/clauses. Otherwise, if A has no loop while reading l_i leading from some state q to some state q', we conclude that there can only exist finitely many different literals l' leading from q to q'. All of these are put (after some massage) in a set $\mathit{rep}(\Lambda_{q,q'})$ for joint treatment of all $\mathit{rep}(\Lambda_{q,q'})$. The case of universally quantified literals needs more care. For every pair (q, q') of states in A we compute a set $\Lambda_{q,q'}$ of literals leading from q to q'. From the powerset of all Λ sets we identify combinations of Λ sets, sharing a common variable, which is chosen as a representative. Therefore, wherever possible the same universally quantified variable can be referenced, reducing the total number of different universally quantified variables in the formula.

4.3.1 Construction of the condensed automaton

We will first define for every pair of states the set of literals which can be read in between those states.

Definition 16. Let $A = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) \subseteq L_{k\text{-QBF}}$. For every pair of states $q, q' \in Q$, $s \in \{+, -\}$ and

$1 \leq d \leq k$ we define

$$\Lambda^{d,s}_{q,q'} := \left\{ w \in L\left((\langle | \varepsilon) s b^d a^{\geq 1} (\vee | \rangle) \right) \mid \delta^*(q,w) = q' \right\} \quad ,$$

the *literal transition set* from q to q' with sign s and quantifier depth d. Furthermore, let

$$\Lambda_{q,q'} := \bigcup_{s \in \{+,-\}, 1 \leq d \leq k} \Lambda^{d,s}_{q,q'}$$

be the union of all literal transition sets from q to q'. For easier readability, we will sometimes refer to $\Lambda^{d,s}_{q,q'}$ as Λ if d, s and q, q' are understood.

Each Λ is recognized by a *sub-automaton* of A and therefore is a regular set.

Since we have to treat existentially and universally quantified literals differently, we put the universally ones in an extra set.

Definition 17. Let $A = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) \subseteq L_{k\text{-QBF}}$. Let $\Upsilon^d := \left\{ \Lambda^{d,s}_{q,q'} \mid q, q' \in Q, s \in \{+,-\} \right\}$ be the set containing all universally quantified literal transition sets with quantifier depth d, for all even d with $1 \leq d \leq k$.

Definition 18. Let *trunc*: $\Gamma = \{a, b, \langle, \rangle, +, -, \vee, \wedge\}^* \to \{a, b\}^*$ be an homomorphism with

$$\mathit{trunc}(\gamma) := \begin{cases} \gamma & \text{if } \gamma \in \{a, b\}, \\ \varepsilon & \text{otherwise.} \end{cases}$$

Define the operation *extend* such that $\mathit{extend}(w, \Lambda) := \mathit{trunc}^{-1}(w) \cap \Lambda$ for $w \in \mathit{trunc}(\Lambda)$ and language Λ .

Intuitively, function *trunc* returns the variable referenced by a literal forgetting its sign and its position in a clause, while *extend* provides us with all possible occurrences of a variable as a literal leading from state q to state q'.

Definition 19. Let $P^d := \left\{ p \in \mathcal{P}\left(\Upsilon^d\right) \mid \cap_{\Lambda \in p} \mathit{trunc}(\Lambda) \neq \emptyset \right\}$ for all even d with $1 \leq d \leq k$ be the subset of the powerset of all literal transition sets with quantifier depth d which only contains sets of languages with a common variable.

The sets P^d consists of all combinations of universally quantified literal transition sets with quantifier depth d which share a common variable. This means that for all corresponding pairs of states we can read literals in between the two states referencing the same variable. Note that for every $p \in P^d$ all subsets of p are also contained in P^d.

In the following, we construct for each Λ a finite set $\mathit{rep}(\Lambda) \subseteq \Lambda$ of representatives which will carry the essential information whether the language contains a true quantified Boolean formula.

Definition 20. For a language $L \subseteq \Gamma^*$ let $\min_{\text{lex}}(L)$ denote the lexicographically minimal element of L.

Since an existentially quantified variable which only occurs once in a quantified Boolean formula can always be satisfied, we try to *separate* all existentially quantified variables when assigning representatives to each literal transition set. Conversely, having many different universally quantified variables makes a quantified Boolean formula harder to be true. The elements in P^d state which literal transition sets contain universally quantified literals, which can reference the same variable. Intuitively, the set of representatives $\mathit{rep}(\Lambda)$ enumerate the finitely many "interesting" literals of Λ.

Definition 21. For every $\Lambda^{d,s}_{q,q'}$ we define a finite set of representatives $\mathit{rep}(\Lambda^{d,s}_{q,q'})$ with the following case distinction:

1. If d is even (universally quantified) use Algorithm 1 to compute the representatives of $\Lambda^{d,s}_{q,q'}$.

Algorithm 1 Computation of $\mathit{rep}(\Lambda)$ for even d.

```
for all $\Lambda \in \Upsilon^d$ do
    $\mathit{rep}(\Lambda) \leftarrow \emptyset$
end for
for all $p \in P^d$ do
    $\mathit{label}(p) \leftarrow \min_{\text{lex}}\left(\bigcap_{\Lambda \in p} \mathit{trunc}(\Lambda)\right)$
    for all $\Lambda \in p$ do
        $\mathit{rep}(\Lambda) \leftarrow \mathit{rep}(\Lambda) \cup \mathit{extend}(\mathit{label}(p), \Lambda)$
    end for
end for
```

2. If d is odd (existentially quantified) and $|\Lambda^{d,s}_{q,q'}| < \infty$, then $\mathit{rep}(\Lambda^{d,s}_{q,q'}) = \Lambda^{d,s}_{q,q'}$.

3. If d is odd (existentially quantified) and $|\Lambda^{d,s}_{q,q'}| = \infty$, then $\mathit{rep}(\Lambda^{d,s}_{q,q'}) = \mathit{extend}(b^d a^l, \Lambda^{d,s}_{q,q'})$, such that the truncated $b^d a^l$ is in no other set of representatives for any $\Lambda^{d',s'}_{p,p'}$, with $p, p' \in Q$, quantifier depth d' and sign s'.

In the first loop of Algorithm 1 we initialize all sets of representatives with the empty set. In the second loop we choose for every combination of literals transition sets sharing a variable the lexicographically smallest common variable and add for every involved Λ all literals in Λ which reference the picked variable to the set of representatives of Λ. Since we do so for every combination of Λ

in P^d the set of representatives of Λ can contain multiple but finitely many elements. Since for all $\Lambda \in \Upsilon$ the set $\{\Lambda\}$ is in P^d, every literal transition set will get at least one representative.

Remark 2. *The lexicographic minimal element of each $p \in P^d$ is chosen to make the algorithm deterministic. Any other element of $\bigcap_{\Lambda \in p} trunc(\Lambda)$ could be picked as possible label.*

Let us consider an example computation for literal transition sets and representatives.

Example 2. *Let A_{sub} be a subgraph of an automaton recognizing a regular language $R \subseteq L_{3\text{-}QBF}$ illustrated as extended automaton [Pin10] shown in Figure 4.1. We give an example computation of the literal transition sets*

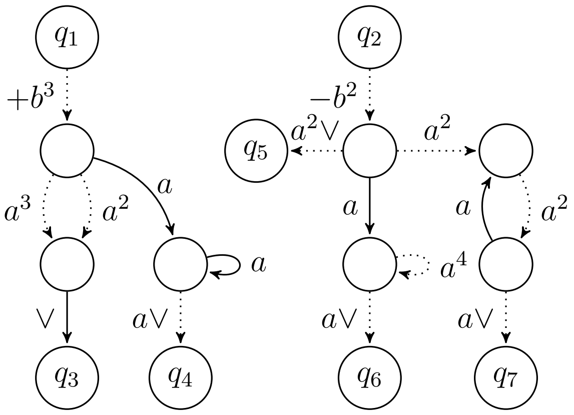

Figure 4.1: Subgraph A_{sub} of a finite automaton recognizing a factor of a regular language $R \subseteq L_{3\text{-QBF}}$. Dotted transitions, which are labeled with words over Γ, use extra states that are omitted here.

(Definition 16) and respective representatives (Definition 21) for pairs of states $\{q_1, q_2\} \times \{q_3, q_4, q_5, q_6, q_7\}$ in A_{sub}, as all other pairs yield empty literal transition sets and representatives. The non-empty literal transition sets of A_{sub} are:

$$\begin{aligned}
\Lambda^{2,-}_{q_2,q_5} &= \{-b^2a^2\vee\}\\
\Lambda^{2,-}_{q_2,q_6} &= \{-b^2a(a^4)^i a\vee \mid i \geq 0\}\\
\Lambda^{2,-}_{q_2,q_7} &= \{-b^2a^4(a^3)^i a\vee \mid i \geq 0\}\\
\Lambda^{3,+}_{q_1,q_3} &= \{+b^3a^2\vee, +b^3a^3\vee\}\\
\Lambda^{3,+}_{q_1,q_4} &= \{+b^3aa^ia\vee \mid i \geq 0\}
\end{aligned}$$

The only universal quantification level for $R \subseteq L_{3\text{-}QBF}$ is $d = 2$. Table 4.1 shows the elements of P^2 and their respective labels. This yields representatives for $\Lambda^{2,-}_{q_2,q_5}$, $\Lambda^{2,-}_{q_2,q_6}$ and $\Lambda^{2,-}_{q_2,q_7}$. The literal transition set $\Lambda^{3,+}_{q_1,q_3}$ has an odd quantification

Table 4.1: Elements of P^2 with their corresponding sets of labels for A_{sub}.

$p_i \in P^2$	Contained Λs	$label(p_i)$
p_1	$\{\Lambda^{2,-}_{q_2,q_5}\}$	b^2a^2
p_2	$\{\Lambda^{2,-}_{q_2,q_6}\}$	b^2a^2
p_3	$\{\Lambda^{2,-}_{q_2,q_7}\}$	b^2a^5
p_4	$\{\Lambda^{2,-}_{q_2,q_5}, \Lambda^{2,-}_{q_2,q_6}\}$	b^2a^2
p_5	$\{\Lambda^{2,-}_{q_2,q_6}, \Lambda^{2,-}_{q_2,q_7}\}$	b^2a^{14}

level and is finite. Thus, $rep(\Lambda^{3,+}_{q_1,q_3}) = \Lambda^{3,+}_{q_1,q_3} = \{+b^3a^3\vee, +b^3a^2\vee\}$*. The only infinite* Λ *with odd quantification level is* $\Lambda^{3,+}_{q_1,q_4}$*, for which we pick an arbitrary unique* $e \in \Lambda^{3,+}_{q_1,q_4}$ *as representative. Table 4.2 lists all transitioning sets with their representatives.*

Table 4.2: Literal transition sets of A_{sub} with corresponding sets of representatives.

Literal transitioning	$rep(\Lambda)$
$\Lambda^{2,-}_{q_2,q_5} = \{-b^2a^2\vee\}$	$\{-b^2a^2\vee\}$
$\Lambda^{2,-}_{q_2,q_6} = \{-b^2a(a^4)^ia\vee \mid i \geq 0\}$	$\{-b^2a^2\vee, -b^2a^{14}\vee\}$
$\Lambda^{2,-}_{q_2,q_7} = \{-b^2a^4(a^3)^ia\vee \mid i \geq 0\}$	$\{-b^2a^5\,or, -b^2a^{14}\vee\}$
$\Lambda^{3,+}_{q_1,q_3} = \{+b^3a^3\vee, +b^3a^2\vee\}$	$\{+b^3a^3\vee, +b^3a^2\vee\}$
$\Lambda^{3,+}_{q_1,q_4} = \{+b^3aa^ia\vee \mid i \geq 0\}$	$\{+b^3a^{42}\vee\}$

Lemma 1. *For all* $q, q' \in Q$*, quantifier depth* $d \leq k$ *and* $s \in \{+,-\}$ *the set* $rep(\Lambda^{d,s}_{q,q'})$ *is finite.*

Proof. For odd d the claim is easily verified. So, assume d to be even. The set Υ^d is finite since d is bounded by k, Q is finite and s can only assume two values. Hence, the powerset $\mathcal{P}\left(\Upsilon^d\right)$ is finite and therefore P^d is finite, too. Each $p \in P^d$ has exactly one label. So, for finitely many p, finitely many elements are added to $rep(\Lambda)$ for any Λ. Hence, $rep(\Lambda)$ is finite for any $\Lambda \in \Upsilon^d$. □

We now want to construct the new automaton condense(A) based on the sets of representatives of the original automaton A. We will then have to test only finitely many words accepted by condense(A), under which a true quantified Boolean formula will be contained, if and only if there is one in the original language R.

Definition 22. For $q, q' \in Q$ let

$$rep(\Lambda_{q,q'}) := \bigcup_{s\in\{+,-\}, d\leq k} rep\left(\Lambda^{d,s}_{q,q'}\right)$$

be the set of representatives from q to q' of arbitrary quantification and sign s.

Definition 23. Based on literal transition sets (see Definition 16), we define for every $q, q' \in Q$ *clause transition sets*

$$C_{q,q'} := \bigcup_{q_1,q_2 \in Q, \Lambda_{q,q_1} \neq \emptyset, \Lambda_{q_1,q_2} \neq \emptyset, \Lambda_{q_2,q'} \neq \emptyset} \mathit{rep}(\Lambda_{q,q_1}) \cdot \mathit{rep}(\Lambda_{q_1,q_2}) \cdot \mathit{rep}(\Lambda_{q_2 q'}) \ ,$$

where we require that all words in Λ_{q,q_1} start with $\langle$, the words in Λ_{q_1,q_2} contain neither $\langle$ nor $\rangle$, and the words in $\Lambda_{q_2,q'}$ end with $\rangle$.

Lemma 2. *For every $q, q' \in Q$ the clause transition set $C_{q,q'}$ is finite.*

Proof. For every $q, q' \in Q$ the set of representatives $\mathit{rep}(\Lambda_{q,q'})$ is finite, and hence finite concatenation yields again a finite set. □

Definition 24. Let $R \subseteq L_{k\text{-QBF}}$ be a regular language and $A = (Q, \Gamma, \delta, q_0, F)$ be a DFA with $L(A) = R$. Define the *condensed automaton* $\text{condense}(A) = (Q, \Gamma', \delta', q_0, F)$ where

$$\begin{aligned} \Gamma' = & \bigcup_{q,q' \in Q} C_{q,q'} \cup \{\wedge\} \\ \text{and} \quad & \forall q, q' \in Q : (q, w, q') \in \delta' \text{ if } w \in C_{q,q'} \\ & \forall q, q' \in Q : (q, \wedge, q') \in \delta' \text{ if } (q, \wedge, q') \in \delta \ . \end{aligned}$$

The automaton condense(A) accepts a subset of $L(A)$ such that the words in $L(\text{condense}(A))$ are built over a finite set of clauses.
The next section shows that $L(A) \cap L_{\Sigma_k} \neq \emptyset \iff L(\text{condense}(A)) \cap L_{\Sigma_k} \neq \emptyset$. Thus, we can decide the emptiness of the intersection by inspecting the finitely many elements which are accepted by loop-free paths in condense(A) one-by-one.

4.3.2 Condensation preserves regular intersection non-emptiness

Lemma 3. *Let $R \subseteq L_{k\text{-}QBF}$ be a regular language and A be a DFA with $L(A) = R$. If $L(\text{condense}(A)) \cap L_{k\text{-}TQBF} \neq \emptyset$ then also $R \cap L_{k\text{-}TQBF} \neq \emptyset$.*

Proof. The condensed automaton condense(A) recognizes a subset of R. In particular, every recognized true quantified Boolean formula w, is also in R. □

Lemma 4. *Let $R \subseteq L_{k\text{-}QBF}$ be a regular language and A be a DFA with $L(A) = R$. If $R \cap L_{k\text{-}TQBF} \neq \emptyset$ then also $L(\text{condense}(A)) \cap L_{k\text{-}TQBF} \neq \emptyset$.*

Proof. The idea of the proof is the following: Let $w \in R$ evaluate to true. Every literal of w is part of some Λ. Each one can be substituted by a representative in $rep(\Lambda)$, making a case distinction between existential and universal quantification, yielding a word in $L(\text{condese}(A))$ which is also true. Now, we will go through the proof in detail:
Let $w \in R$ be a true quantified Boolean formula ($w \in L_{k\text{-TQBF}}$). Then w can be split up into factors of the form

$$w = w_0 w_1 w_2 \wedge w_3 w_4 w_5 \wedge \cdots \wedge w_{n-2} w_{n-1} w_n$$

with the properties $w_i \in \Lambda_{q_i,q_{i+1}}$ for $i \not\equiv 0 \mod 3$, $w_i \in \Lambda_{q_i',q_{i+1}}$ and $\delta(q_i, \wedge) = q_i'$ for $i \equiv 0 \mod 3, i > 0$ for all $i < n$. Furthermore, $w_0 \in \Lambda_{q_0,q_1}$ and $w_n \in \Lambda_{q_n,q_f}$ for some final state $q_f \in F$.

Then, there is a $w' \in L(\text{condense}(A))$ which can be partitioned with analogous properties into $w' = w_0' w_1' w_2' \wedge w_3' w_4' w_5' \wedge \cdots \wedge w_{n-2}' w_{n-1}' w_n'$ and $w_i \in \Lambda_{q_i,q_{i+1}} \Rightarrow w_i' \in rep(\Lambda_{q_i,q_{i+1}})$. To show that there is a true quantified w' of such a form we will do a case differentiation of the type of $\Lambda_{q_i,q_{i+1}}$.

1. If $\Lambda_{q_i,q_{i+1}}$ is finite and existentially quantified, then $rep(\Lambda_{q_i,q_{i+1}}) = \Lambda_{q_i,q_{i+1}}$. So, we can pick $w_i' = w_i \in rep(\Lambda_{q_i,q_{i+1}})$.

2. If $\Lambda_{q_i,q_{i+1}}$ is infinite and existentially quantified, then $w_i' \in \Lambda_{q_i,q_{i+1}}$ can be picked such that its variable is not referenced by any other literal in w'. Since it is independent and existentially quantified w_i' is equivalent to the Boolean constant 1. Thus, the whole clause w_i' can be viewed as constant 1.

3. If $\Lambda_{q_i,q_{i+1}}$ is universally quantified we have to consider which other literals $w_j \in \Lambda_{q_j,q_{j+1}}$ references the same variable as w_i. Let $J := \{1 \leq j \leq n \mid trunc(w_j) = trunc(w_i)\}$ be the index set of all literals encoding the same variable as w_i. The collection of these literal transition sets $p_i := \{\Lambda_{q_j,q_{j+1}} \mid j \in J\}$ is in $P^{\#_b(w_i)}$ because the intersection of the literal sets is, as observed, non-empty. By construction, $rep(\Lambda_{q_i,q_{i+1}})$ and all other $rep(\Lambda_{q_j,q_{j+1}})$ for $j \in J$ received a label from p_i. Picking w_i' and w_j' for all $j \in J$ as the label given by p_i will yield a *consistent renaming* of the literal w_i and thus not change the evaluation of w'.

Thus, substituting each w_i by the respective w_i' yields a true quantified Boolean formula $w' \in L(\text{condense}(A))$ which proves this lemma. □

4.3.3 Deciding $int_{\text{Reg}}(L_{\Sigma_k})$ and $int_{\text{Reg}}(L_{\Pi_k})$

We now show that it is decidable whether $\text{condense}(A)$ accepts a true quantified Boolean formula.

Lemma 5. *Let A be a DFA with $L(A) \subseteq L_{k\text{-}QBF}$ and $w \in L(\text{condense}(A))$ where w is the labeling of an accepting path p in* condense(A) *containing at least one loop. Let w' be w without the factors read in the loops of p. If w is a true quantified Boolean formula, then so is w'.*

Proof. The formula w is in conjunctive form $w = c_1 \wedge \cdots \wedge c_n$ with some clauses c_i which all evaluate to true. Loops only contain whole clauses and thus w' consists of a subset of the clauses of w. Thus, w' also evaluates to true. □

This means that words recognized through simple accepting paths in the condensed automaton are the only ones that need consideration when looking for true quantified Boolean formulae.

Lemma 6. *Let R be a regular language and A be a DFA with $L(A) = R$. It is decidable whether $L(\text{condense}(A)) \cap L_{\Sigma^k} \neq \emptyset$ and $L(\text{condense}(A)) \cap L_{\Pi^k} \neq \emptyset$.*

Proof. Let $R \subseteq L_{k\text{-QBF}}$ w.l.o.g. enumerate all words $w_1, w_2, \ldots, w_n$ which are recognized through (the finitely many) simple accepting paths in condense(A). For $i = 1, \ldots, n$ test if w_i is a true quantified Boolean formula. Following Lemma 5, if all w_i evaluate to false, no other $w \in L(\text{condense}(A))$ can evaluate to true. Thus, the intersection $L(\text{condense}(A)) \cap L_{\Sigma^k}$ is non-empty if and only if at least one w_i evaluates to true and the first quantifier in w_i is existential. Analogously, $L(\text{condense}(A)) \cap L_{\Pi^k} \neq \emptyset$ if and only if at least one w_i evaluates to true and the first quantifier in w_i is universal. □

In total, this yields Theorem 9.

Theorem 1. *Let R be a regular language. For each $k \in \mathbb{N}$ it is decidable whether $L_{\Sigma_k} \cap R = \emptyset$ and $L_{\Pi_k} \cap R = \emptyset$, i.e. $int_{\text{Reg}}(L_{\Sigma_k})$ and $int_{\text{Reg}}(L_{\Pi_k})$ are decidable.*

Proof. Assume w.l.o.g. that $R \subseteq L_{k\text{-QBF}}$. Let A be a DFA recognizing R. Lemma 6 states that condense$(A) \cap L_{\Sigma_k} \neq \emptyset$ and condense$(A) \cap L_{\Pi_k} \neq \emptyset$ is decidable. Following Lemma 3 and Lemma 4 $L(\text{condense}(A)) \cap L_{k\text{-TQBF}} \neq \emptyset \iff R \cap L_{k\text{-TQBF}} \neq \emptyset$. Thus, both $L_{\Sigma_k} \cap R = \emptyset$ and $L_{\Pi_k} \cap R = \emptyset$ is decidable. □

4.4 Decidability of $int_{\text{Reg}}(\text{TQBF})$

In this section we prove Theorem 10, stating that $int_{\text{Reg}}(L_{\text{TQBF}})$ is decidable. That is, one can test whether a regular language encoding quantified Boolean formulae with unbounded quantifier alternation contains at least one *true* quantified Boolean formula. Observe that the automaton condense(A) constructed in the last subsection to test for (non)emptiness of $R \cap L_{\Sigma_k}$ is dependent in k.

We now give a construction which allows us to use the condense(A)-automaton despite the presence of unboundedly nested quantifiers.

Fact 3. *Let $\vec{x}_1$ and $\vec{x}_2$ be vectors of Boolean variables and ϕ be a propositional Boolean formula with $|\vec{x}_1|+|\vec{x}_2|$ free variables. Then, the following implications holds:*

1. $\exists\vec{x}_1\forall\vec{x}_2\phi(\vec{x}_1,\vec{x}_2) \Rightarrow \forall\vec{x}_2\exists\vec{x}_1\phi(\vec{x}_1,\vec{x}_2)$
2. $\forall\vec{x}_1\phi(\vec{x}_1) \Rightarrow \exists\vec{x}_1\phi(\vec{x}_1)$
3. *Let $\mathcal{Q}$ be fixed as $\exists$ or $\forall$. Then* $\mathcal{Q}\vec{x}_1\mathcal{Q}\vec{x}_2\phi(\vec{x}_1,\vec{x}_2) \Rightarrow \mathcal{Q}\vec{x}_2\mathcal{Q}\vec{x}_1\phi(\vec{x}_1,\vec{x}_2)$

Fact 3 intuitively speaking states that we can "pull out" universal quantifiers. Furthermore, we can substitute universal quantifiers by existential ones and permute consecutive vectors quantified by the same type of quantifier without falsifying the formula.

4.4.1 Defining the reduction

For a given DFA A we construct a new automaton *restrict*(A) which accepts a true quantified Boolean formula if and only if A does. Depending on the size of A there is constant d such that formulae in $L(\textit{restrict}(A))$ have at most $d+2$ (and therefore finitely many) quantifier alternations. Words of $L(\textit{restrict}(A))$ have the property that quantifier levels of literals above a threshold d are summed up in a sequence of universal quantifier levels, followed by a non-overlapping sequence of existentially quantified levels. This is done by pumping the b-strings of literals in a way that existential quantifiers are shifted over universal ones, separating the two kinds of quantifiers in two blocks of quantification depth $d+1$ and $d+2$.

Definition 25. For a DFA $A=(Q,\Gamma,\delta,q_0,F)$ and $s\in\{+,-\}$ let

$$G^s_{q,q'} := \{n\in\mathbb{N} \mid \exists q_1,q_2\in Q : \delta(q_1,s)=q \wedge \delta^*(q,b^n)=q' \wedge \delta(q',a)=q_2\}$$

be the set of quantification depths we can use for s-signed literals starting in q and *passing* q' on the way.

If we are passing a loop while reading a quantification depth in $G^s_{q,q'}$, we can use pumping to read infinitely many different quantification depths between q and q'.

Lemma 7. *Let $A=(Q,\Gamma,\delta,q_0,F)$ be a DFA. For all $q,q'\in Q$ and $s\in\{+,-\}$ the following statement holds: If there exists $n\in G^s_{q,q'}$ with $n\geq|Q|$ and $n\equiv 1 \mod 2$, then $|G^s_{q,q'}\cap\{i \mid i\geq|Q| \text{ and } i\equiv 1 \mod 2\}|=\infty$.*

Proof. Assume there exists such an $n \geq |Q|$. This means $\delta^*(q, b^n) = q'$. Following the pumping lemma for regular languages there is an m such that for all $i \in \mathbb{N} : n + mi \in G^s_{q,q'}$. This does in particular hold for even i, represented as $2i'$. If $n \equiv 1 \mod 2$, we have that $n + mi \bmod 2 = n + 2mi' \bmod 2 = n \bmod 2$. This means there are infinitely many odd numbers in $G^s_{q,q'}$. □

If the same even quantification level n with $n \geq 2|Q|^{|Q|}$ can be read in different locations of an DFA A, a smaller, also even, quantification level n' with $n' \leq 2|Q|^{|Q|}$ can be read in the same locations.

Lemma 8. *Let $A = (Q, \Gamma, \delta, q_0, F)$ be a DFA and $Z \subseteq Q \times Q$. For all $(q, q') \in Z$ and $s \in \{+, -\}$ the following statement holds: If $\exists n \in \bigcap_{(q,q') \in Z} G^s_{q,q'}$ with $n \geq 2|Q|^{|Q|}$ and $n \equiv 0 \mod 2$, then $\exists n' \in \bigcap_{(q,q') \in Z} G^s_{q,q'}$ with $n' \leq 2|Q|^{|Q|}$ and $n' \equiv 0 \mod 2$.*

Proof. Assume there exists $n \in \bigcap_{(q,q') \in Z} G^s_{q,q'}$ with $n \geq 2|Q|^{|Q|}$ and $n \equiv 0 \mod 2$. This means for every pair $(q, q') \in Z$ holds $\delta^*(q, b^n) = q'$. Since $n \geq 2|Q|^{|Q|} \geq |Q|$ there exists a $q'' \in Q$ for every pair $(q, q') \in Z$ with $\delta^*(q, b^{i_{q,q'}}) = q'', \delta^*(q'', b^{j_{q,q'}}) = q'', \delta^*(q'', b^{k_{q,q'}}) = q'$, $i_{q,q'} + j_{q,q'} + k_{q,q'} = n$ and $j_{q,q'} \leq |Q|$. So, the state q'' is visited multiple times between q and q'. Following the pumping lemma for regular languages $\delta^*(q, b^{i_{q,q'} + m \cdot j_{q,q'} + k_{q,q'}}) = q'$ for every pair $(q, q') \in Z$ and integer m. This holds in particular for $m = J := \prod_{(q_1,q_2) \in Z, (q_1,q_2) \neq (q,q')} j_{q_1,q_2}$. So, every multiple of $\mathcal{J} := j_{q,q'} \cdot J$ is the length of a loop over q''. Since $\mathcal{J} \leq |Q|^{|Q|}$ we can omit loops labeled by $b^{2\mathcal{J}}$ over q'' until we get a $n' \leq 2|Q|^{|Q|}$ for which for all $(q, q') \in Z$ holds $\delta^*(q, b^{n'}) = q'$. For every pair (q, q') we are omitting loops of the same even length, hence n' is even and $n' \in G^s_{q,q'}$ for all pairs $(q, q') \in Z$. □

Definition 26. For a DFA $A = (Q, \Gamma, \delta, q_0, F)$, let $G^{\triangle,s}_{q,q'} := G^s_{q,q'} \cap \{i \mid i \geq |Q|\}$ be the set of quantifier levels higher then $|Q|$. We will define the *level* of $G^{\triangle,s}_{q,q'}$ algorithmically. We set `counter` $\leftarrow 2|Q|^{|Q|} + 1$ in the beginning and compute for each $q, q' \in Q$ and $s \in \{+, -\}$ the *level* of a $G^{\triangle,s}_{q,q'}$ in arbitrary, but fixed order with Algorithm 2.

Algorithm 2 selects for every set $G^{\triangle,s}_{q,q'}$ of quantification depths higher than $|Q|$ representative levels of the set. If there is any existential quantification level in the set, then by Lemma 7 there are infinitely many and we can pick one level above the threshold $2|Q|^{|Q|} + 1$ as a representative. If there are no existential quantification levels in the set, we have to pick all universal quantification levels up to our threshold as representatives. Lemma 8 states that we can pump every universal quantification level below the threshold. Algorithm 2 prefers assigning existential quantification levels as representatives and shifting them above the universal quantification levels according to Fact 3.

Algorithm 2 Computing the level for $G^{\triangle,s}_{q,q'}$.

```
if $G^{\triangle,s}_{q,q'} \cap \{i \mid i \equiv 1 \mod 2\} \neq \emptyset$ then
    $level(G^{\triangle,s}_{q,q'}) \leftarrow \min\left(G^{\triangle,s}_{q,q'} \cap \{i \mid i \text{ is odd and } i \geq \texttt{counter}\}\right)$
    $\texttt{counter} \leftarrow level(G^{\triangle,s}_{q,q'}) + 2$
else
    $level(G^{\triangle,s}_{q,q'}) \leftarrow G^{\triangle,s}_{q,q'} \cap \{i \mid i \equiv 0 \mod 2 \text{ and } |Q| \leq i \leq 2|Q|^{|Q|}\}$
end if
```

Definition 27. For a DFA $A = (Q, \Gamma, \delta, q_0, F)$ define *restrict*(A) as the automaton $A' = (Q', \Gamma, \delta', q_0, F)$ with δ' being the following modification of δ. For each $p, p' \in Q$ remove each transition of the form $\delta(p, b) = p'$. Instead, we will introduce a path $\delta^*(q, b^n) = q'$ for each $n \in G^s_{q,q'}$ with $n < |Q|$ using $n-1$ supplementary states we add to Q'. Moreover, for each $G^{\triangle,s}_{q,q'}$ we add additional paths $\delta^*(q, b^n) = q'$ for each $n \in level(G^{\triangle,s}_{q,q'})$.

In the restricting process we remove all b-transitions to get rid of b-loops. Loop-free b-transitions are reintroduced in such a way, that the emptiness of the intersection with L_{TQBF} is not changed.

4.4.2 Correctness of the reduction

Lemma 9. *Let $A = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton. Then, $L(A) \cap L_{TQBF} \neq \emptyset$ if and only if $L(restrict(A)) \cap L_{TQBF} \neq \emptyset$.*

Proof. "$\Rightarrow$": The idea is to use the pumping property to substitute universal quantifiers by existential ones and shift the existential quantifiers to the end, while we conflate the unbounded universal quantifiers to a finite block. Renaming the variables yields a word in $L(restrict(A))$.

Assume $L(A) \cap L_{\text{TQBF}} \neq \emptyset$. Then, there is some $w \in L(A)$ with $w \in L_{\text{TQBF}}$. Removing parenthesis and logical operators from w we are left with a sequence of $\pm b^{\geq 1} a^{\geq 1}$ blocks $s_1 b^{\beta_1} a^{\alpha_1} \dots s_{3n} b^{\beta_{3n}} a^{\alpha_{3n}}$. For each literal $lit_i := s_i b^{\beta_i} a^{\beta_i}$ with $\beta_i \geq |Q|$ consider the states $q_i, q'_i \in Q$ in which between b^{β_i} is read.

1. For odd β_i there is, following Lemma 7, an odd β'_i larger then any β_j for $j = 1, \dots, 3n$. We substitute lit_i by $s_i b^{\beta'_i} a^{\alpha_i}$ in w.

2. If β_i is even and $G^{\triangle,s}_{q,q'}$ contains an odd number $o \in \{|Q|, \dots, 2|Q|\}$, we use the same lemma as before and *pump* up the b-sequence to an odd $\beta'_i \in G^s_{q,q'}$ larger then any previous β_j (or β'_j) for $j = 1, \dots, 3n$. By this we change the quantification of lit_i from universal to existential and make it the last quantified literal.

3. If there is no such odd number o and $\beta_i \leq 2|Q|^{|Q|}$, we leave $s_i b^{\beta_i} a^{\beta_i}$ as it is.

4. If there is no such odd number o and $\beta_i > 2|Q|^{|Q|}$, we identify all pairs $(q, q') \in Q \times Q$ in between β_i can be read and obtain by Lemma 8 a $\beta'_i \leq 2|Q|^{|Q|}$ with $\delta^*(q, b^{\beta'_i}) = q'$ for all such pairs (q, q'). We substitute lit_i by $s_i b^{\beta'_i} a^{\alpha_i}$ in w.

After doing these substitutions, we are left with a new word w' which is still in $L(A)$ and, following Fact 3, still is a true quantified Boolean formula. In essence, we use the pumping property to substitute universal quantifiers by existential ones and shift the existential quantifiers to the end, while we conflate the unbounded universal quantifiers to a finite block, yielding the same structure the words in $L(\mathit{restrict}(A))$ have. By renaming these variables into the scheme shown in Definition 26 we obtain a word in $L(\mathit{restrict}(A))$.

"$\Leftarrow$": By construction $L(\mathit{restrict}(A))$ is a proper subset of $L(A)$. Thus, any w encoding a true quantified Boolean formula is also in $L(A)$. □

4.4.3 Deciding $\mathit{int}_{\mathrm{Reg}}(L_{\mathbf{TQBF}})$

Theorem 2. *Let R be a regular language. Then $R \cap L_{TQBF} = \emptyset$ is decidable.*

Proof. Let A be the finite automaton recognizing R and let k be the length of the longest b-factor along an accepting path of $\mathit{restrict}(A)$. Following Theorem 9 it is decidable whether $L(\mathit{restrict}(A)) \cap L_{k\text{-TQBF}} = \emptyset$. Lemma 9 states that $L(\mathit{restrict}(A))$ contains an encoded true quantified Boolean formula if and only if $L(A)$ does. Hence, $L(A) \cap L_{\mathrm{TQBF}} = \emptyset$ if and only if $L(\mathit{restrict}(A)) \cap L_{\Sigma^k} = \emptyset$ and $L(\mathit{restrict}(A)) \cap L_{\Pi^k} = \emptyset$. Since the intersection is non-empty if and only if $\mathit{restrict}(A)$ accepts a true quantified formula, the latter is also decidable. □

Chapter 5

Integer Linear Programming

In this chapter we will investigate the NP-complete problem Integer Linear Programming (short ILP). We will adapt the techniques from the previous chapter to show that the emptiness of the intersection of ILP with a regular language is decidable. This chapter is divided into three parts. In the first section we will construct a condensed automaton condensed(A) for a given finite automaton A. In the next section we will show that $L(\text{condensed}(A))$ contains a solvable ILP if and only if $L(A)$ does. In the last section we will show that it is decidable whether $L(\text{condensed}(A))$ contains a solvable ILP.

We first want to define the problem of integer linear programming. While the standard-form of ILP varies in the fields of science, we refer to the definition in *Computational Complexity* from K. Wagner and G. Wechsung [WW89] where this problem is called LIQ. We will refer to the described problem as ILP.

Definition 28 (ILP).
Given: Finite set $\mathcal{A}$ of pairs $(\vec{\alpha}, \beta)$ where $\vec{\alpha}$ is an m-tuple of integers and β is an integer.
Question: Is there an m-tuple $\vec{x}$ of integers such that $\vec{\alpha} \cdot \vec{x} \leq \beta$ for all $(\vec{\alpha}, \beta) \in \mathcal{A}$?

Fact 4. *The problem* ILP *is NP-complete if we ask for solutions in* $\mathbb{Z}$ *[WW89].*

The problem will be encoded in the following way. The whole set $\mathcal{A}$ will be encoded in one word. For each pair $(\vec{\alpha}, \beta)$ the elements of $\vec{\alpha}$ are unary encoded over the symbol a while β will be unary encoded over b. Each positive integer will be preceded with a $+$ while each negative integer will be preceded with a $-$. The integers of α will be separated form β by a $\leq$ symbol.
So, every encoded instance of ILP will be a member of the language L_{enc} defined as follows.

Definition 29. We define the set of all encoded ILPs, regardless if they are solvable or not, as the set L_{enc}.

$$L_{\text{enc}} := L\left(\left(([+|-]a^*)^* \leq [+|-]b^*\right)^*\right)$$

The following example illustrates the encoding.

Example 3. *Consider the following integer linear program.*

$$\mathcal{A} := \{((5,1,0,7),15),((0,-8,1,-1),-4),((0,0,0,1),-1),((-1,0,3,0),6)\}$$

The encoding of $\mathcal{A}$ reads as follows.

$$+a^5+a++a^7 \le +b^{15}+-a^8+a-a \le -b^4++++a \le -b-a++a^3+ \le +b^6$$

The question we want to investigate is whether the intersection of ILP, encoded in the above described way, with a regular language, given by an automaton, is empty.

Definition 30 ($int_{\text{Reg}}(\text{ILP})$)**.**
Given: Deterministic finite automaton A.
Question: Is $L(A) \cap \text{ILP}_{\text{enc}} \neq \emptyset$?. In other words, does A accept a solvable integer linear programming instance?

Where ILP_{enc} is ILP encoded in the described way.
Since L_{enc} is defined by a regular expression it is a regular language. Therefore, we can assume $L(A)$ to be a subset of L_{enc}.[1]

5.1 Construction

We will follow the ideas presented in the previous chapter of investigating what kinds of loops can occur in the automaton without violating the encoding format, namely loops inside a coefficient, loops over whole coefficients, and loops over whole inequations. This will lead us to the definition of coefficient transition sets, for which we will find representatives. Over these representatives we will define inequation transition sets which will describe all inequations which can be read in between two states. We will again find representatives for these sets and construct a new automaton from these representatives similar to the one in chapter 4.

Definition 31 (Transition set.)**.** Let $A = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton. We define for all $q, q' \in Q$ the coefficient transition set $\Lambda^s_{q,q'}$ as

$$\Lambda^+_{q,q'} = \{+a^i \mid \exists q_1, q_2 : \delta(q,+) = q_1 \wedge \delta^*(q_1, a^i) = q' \wedge \\ \delta(q', \sigma) = q_2 \text{ with } \sigma \in \Sigma \backslash \{a\}\}$$

[1]Otherwise we will consider the intersection of $L(A) \cap L_{\text{enc}}$, which is also regular.

if s is a plus and as

$$\Lambda^{-}_{q,q'} = \{-a^i \mid \exists q_1, q_2 : \delta(q,-) = q_1 \wedge \delta^*(q_1, a^i) = q' \wedge \\ \delta(q', \sigma) = q_2 \text{ with } \sigma \in \Sigma \backslash \{a\}\}$$

if s is a minus. Therefore, $\Lambda^s_{q,q'}$ contains all coefficients which can be completely read in between the states q and q'. For the β-values, we analogously define transition sets $B^s_{q,q'}$ for every pair of states $q, q' \in Q$ as

$$B^{+}_{q,q'} = \{+b^i \mid \exists q_1, q_2 : \delta(q,+) = q_1 \wedge \delta^*(q_1, b^i) = q' \wedge \\ (\delta(q', \sigma) = q_2 \text{ with } \sigma \in \Sigma \backslash \{a\} \vee q' \in F)\}$$

if s is a plus and as

$$B^{-}_{q,q'} = \{-b^i \mid \exists q_1, q_2 : \delta(q,-) = q_1 \wedge \delta^*(q_1, b^i) = q' \wedge \\ (\delta(q', \sigma) = q_2 \text{ with } \sigma \in \Sigma \backslash \{a\} \vee q' \in F)\}$$

if s is a minus. When the context is clear, we will often refer to $\Lambda^s_{q,q'}$ as Λ and to $B^s_{q,q'}$ as B.

We now want to find a set of representatives reps(Λ) for each coefficient transition set Λ. The set reps(Λ) will contain all relevant[2] coefficients from the set Λ. Since all inequations are of the form $\alpha_1 x_1 + \cdots + \alpha_m x_m \leq \beta$, increasing a positive summand makes the inequation system harder to be solved, while decreasing a positive summand may only enlarge the set of solutions. Similarly, decreasing a negative summand enlarges the set of solutions, while increasing it may shrink the set of solutions. So, we only have to consider the largest and smallest coefficient contained in the coefficient transition set. The largest [smallest] coefficient will correspond, in combination with a negative [positive] x_i value, to the smallest negative [positive] summand. If a coefficient transition set is infinite, coefficients with an arbitrary large magnitude are contained in the set. To focus on the main idea, we will choose the meta-characters $+\infty$ and $-\infty$ in this case as representatives indicating that we can replace them with a large enough coefficient. If a β-transition set is infinite, we can find arbitrary large β-values in the set what we will indicate by choosing the $+\infty$-sign as a representative. We will later ignore all inequations with an $+\infty$ representative for a β-transition set. We refer to Definition 21 and leave it to the reader to convince himself that there is an algorithm replacing the ∞-sign by actual coefficients.

[2]A coefficient is considered to be irrelevant if it can be replaced by a smaller or larger coefficient without making the ILP unsolvable.

Definition 32. We define for every coefficient transition set $\Lambda^s_{q,q'}$ the set $\text{reps}(\Lambda^s_{q,q'})$ and for the β-transition sets $B^s_{q,q'}$ the set $\text{reps}(B^s_{q,q'})$ as:

$$\text{reps}(\Lambda^+_{q,q'}) := \begin{cases} \{\min_{\text{val}}(\Lambda^+_{q,q'}), +\infty\}, & \text{if } |\Lambda^+_{q,q'}| = \infty \\ \Lambda^+_{q,q'}, & \text{otherwise,} \end{cases}$$

$$\text{reps}(\Lambda^-_{q,q'}) := \begin{cases} \{-\infty, \max_{\text{val}}(\Lambda^-_{q,q'})\}, & \text{if } |\Lambda^-_{q,q'}| = \infty \\ \Lambda^-_{q,q'}, & \text{otherwise,} \end{cases}$$

$$\text{reps}(B^+_{q,q'}) := \begin{cases} +\infty, & \text{if } |B^+_{q,q'}| = \infty \\ \{\max_{\text{val}}(B^+_{q,q'})\}, & \text{otherwise,} \end{cases}$$

$$\text{reps}(B^-_{q,q'}) := \{\max_{\text{val}}(B^-_{q,q'})\}.$$

Where the functions $\min_{\text{val}}$ and $\max_{\text{val}}$ return the element which encodes the minimal, respectively maximal, value of all elements in the set.

The next step is to identify all inequations which can be read in between two states. Those inequations should only contain relevant coefficients for what reason, the inequation sets are defined over the sets $\text{reps}(\Lambda)$ and $\text{reps}(B)$.

Definition 33. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) \subseteq L_{\text{enc}}$. For every states $q, q' \in Q$ we define the inequation transition set $\Xi_{q,q'}$ as:

$$\begin{aligned} \Xi_{q,q'} = \{s_1 i_1 s_2 i_2 \dots s_k i_k \leq s_b j \mid k \in \mathbb{N}, \exists q_1, q_2, \dots q_{k+2} : s_1 i_1 \in \text{reps}(\Lambda^{s_1}_{q,q_1}) \wedge \\ s_2 i_2 \in \text{reps}(\Lambda^{s_2}_{q_1,q_2}) \wedge \dots \wedge s_k i_k \in \text{reps}(\Lambda^{s_k}_{q_{k-1},q_k}) \\ \wedge\, \delta(q_k, \leq) = q_{k+1} \wedge s_b j \in \text{reps}(B^{s_b}_{q_{k+1},q_{k+2}})\} \end{aligned}$$

The set $\Xi_{q,q'}$ contains therewith all inequations, which can be read in between the states q and q' and consists only of important coefficients.

Now we want to pick finitely many representatives for every inequation transition set $\Xi_{q,q'}$. An inequation transition set $\Xi_{q,q'}$, which contains for every possible solution vector $\vec{x}$ an inequation which is satisfied by $\vec{x}$, is assigned with the representative ϵ indicating that inequations read between q and q' can be removed from the ILP. This can be done because if the other inequations of the ILP have a common solution, then there is an inequation which can be read in between q and q' which is satisfied by this solution. Two types of inequation transition sets have this property. If $\Xi_{q,q'}$ contains an inequations with an ∞-sign as β-value, an inequation with an arbitrary high actual β-value can be read in between q and q'. So, for every value of the left side of the inequation we can read an even bigger right side. Therefore, we assign those inequation transition sets with an ϵ representative.

The other type of $\Xi_{q,q'}$ sets are those which contain an arbitrary high indexed coefficient. As we will see in Lemma 11, we can simultaneously pick inequations from all of these sets, such that in each inequation the coefficient with the highest index will dominate the sum in a way that the inequation is satisfied. We will identify those inequations transition as the sets which are infinite after we removed all inequation ending with more than n consecutive signs[3].
If the modified inequation sets are finite, we simply pick the hole set as the set of representatives. Inequations with more then $|Q| = n$ consecutive sign after the last non-zero coefficient can also be ignored, because there is an equivalent inequation with less then n sign in the inequation set. With this considerations in mind, we define for all states $q, q' \in Q$ a set of representatives $\mathrm{reps}(\Xi_{q,q'})$ for the inequation transition set $\Xi_{q,q'}$.

Definition 34. Let $L_{\text{Trash}} := L(([+|-][a^*|\infty])^* \, [+|-]^{>n} \leq [+|-][b^*|\infty])$ be the set of all inequations where the left side ends with more than $|Q| = n$ consecutive signs.
We define for every inequation transition set $\Xi_{q,q'}$ a set of representatives $\mathrm{reps}(\Xi_{q,q'})$ as

$$\mathrm{reps}(\Xi_{q,q'}) := \begin{cases} \{\epsilon\}, & \text{if } \exists w \in \Xi_{q,q'} \text{ which ends with } \infty, \\ \{\epsilon\}, & \text{if } |\Xi q, q' \backslash L_{\text{Trash}}| = \infty, \\ \Xi q, q' \backslash L_{\text{Trash}}, & \text{otherwise.} \end{cases}$$

Note that there are only finitely many sets $\mathrm{reps}(\Xi_{q,q'})$ which are by construction all of a finite size.

We will now construct a shrunken automaton which will have the finitely many inequations, chosen as a representative, as it's alphabet.

Definition 35. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) \subseteq L_{\text{enc}}$. We define $\mathrm{shrink}(A) := (Q, \Sigma', \delta', q_0, F)$ with

$$\Sigma' = \bigcup_{q,q' \in Q} \mathrm{reps}(\Xi_{q,q'})$$

$$\delta' = \{(q, \xi, q') \mid \xi \in \mathrm{reps}(\Xi_{q,q'})\}$$

where we interpret all members of the sets of representatives as atomic letters.

Lemma 13 will show that we only have to consider simple paths in the shrunken automaton.

Definition 36. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton and $\mathrm{shrink}(A)$ be the shrunken automaton corresponding to A. We define $\mathrm{condensed}(A)$ as the automaton containing only the simple paths in $\mathrm{shrink}(A)$.

[3]By removing more than n consecutive signs from the end of the sum, we ensure that there is a non-zero coefficient under the last n coefficient. If the set is still infinite we can find inequations with arbitrary high non-zero coefficients.

5.2 Correctness

We will now present a bunch of lemmas which in the end will proof that $L(A) \cap \mathrm{ILP}_{\mathrm{enc}} \neq \emptyset \Leftrightarrow L'(\mathrm{condensed}(A)) \cap \mathrm{ILP}_{\mathrm{enc}} \neq \emptyset$. With $L'(\mathrm{condensed}(A))$ $[L'(\mathrm{shrink}(A))]$ we refer to the language $L(\mathrm{condensed}(A))$ $[L(\mathrm{shrink}(A))]$ where the wild-cards ∞ are replaced by actual coefficients.

First, we will show that it is sufficient to only consider the largest and smallest coefficient which can be read in between two states.

Lemma 10. *If $w \in L(A) \cap \mathrm{ILP}_{enc}$, $\vec{x}$ is a solution of w, a_{ij} is the j-th coefficient of the i-th inequation of w, and a_{ij} can be partitioned into substrings $a_{ij_1}, a_{ij_2}, a_{ij_3}$ with $a_{ij} = a_{ij_1}a_{ij_2}a_{ij_3}$, $a_{ij_2} \neq \epsilon$, and the state in which A is after reading w up to the end of a_{ij_1} is the same state in which A is after reading w up to a_{ij_2}*[4]*, then the following holds:*

1. *If $x_j \geq 0$ and a_{ij} has a $+$ sign, then $w' = w$ where a_{ij} is replaced by $a_{ij_1}a_{ij_3}$ is also in $L(A) \cap \mathrm{ILP}_{enc}$ and $\vec{x}$ is a solution for w'.*

2. *If $x_j \geq 0$ and a_{ij} has a $-$ sign, then $w' = w$ where a_{ij} is replaced by $a_{ij_1}a_{ij_2}^2a_{ij_3}$ is also in $L(A) \cap \mathrm{ILP}_{enc}$ and $\vec{x}$ is a solution for w'.*

3. *If $x_j \leq 0$ and a_{ij} has a $+$ sign, then $w' = w$ where a_{ij} is replaced by $a_{ij_1}a_{ij_2}^2a_{ij_3}$ is also in $L(A) \cap \mathrm{ILP}_{enc}$ and $\vec{x}$ is a solution for w'.*

4. *If $x_j \leq 0$ and a_{ij} has a $-$ sign, then $w' = w$ where a_{ij} is replaced by $a_{ij_1}a_{ij_3}$ is also in $L(A) \cap \mathrm{ILP}_{enc}$ and $\vec{x}$ is a solution for w'.*

Proof. Note that all four statements affect only one coefficient of one inequation in the linear program. The fact that w' is accepted by A follows from the definition of a_{ij}, which states that A is in the same state before and after reading the sub-word a_{ij_2}. Therefore, a_{ij_2} can be pumped.
Since we know that $\vec{x}$ is already a solution for w we have to show that it remains a solution if we change one inequation in the described way. Note that all considered inequations are of the form $\vec{\alpha} \cdot \vec{x} \leq \beta$.
In 1) x_i, as well as its coefficient, are non negative integers satisfying the inequation. In w' the value of the coefficient a_{ij} has decreased by the size of a_{ij_2} (which is not zero) decreasing also the value of the summand $|a_{ij_1}a_{ij_3}| \cdot x_i$, hence decreasing the sum of all summands in the inequation which has already been smaller than β. Thus, the altered inequation of w' is also satisfied and since the other inequations have not been altered, the ILP w' is also satisfied by the solution $\vec{x}$.
In 2) the value of the summand $|a_{ij}| \cdot x_i$ is negative, hence increasing the size of a_{ij} decreases the value of the summand and therefore the value of the left side

[4]This means that a_{ij} is read on a loop and the sub-word a_{ij_2} can be pumped.

of the inequation. We do so by pumping a_{ij_2}, therefore w' is also satisfied by $\vec{x}$. Case 3) is analog to 2) and case 4) is analog to case 1).
This concludes the proof. □

Next, we will focus on whole inequations. We will show that by restricting the inequations which can appear in a word of R to inequations from the above defined sets of representatives we are not altering the existence of a solvable ILP in R. As we already explained before Definition 34, for every solution vector $\vec{x}$ we can replace the inequations with an ∞-sign as β-value by inequations with actual β-values, which are satisfied by $\vec{x}$, and which can be read in between the same states in the automaton. Next, we want to focus on inequation transition sets containing inequations with arbitrary high non-zero coefficients. We will show that for every solution vector $\vec{x}$ we can simultaneously replace inequations from those sets in a way that the replacements are satisfied by an extension of $\vec{x}$. So, if the ILP is solvable without inequations from $\Xi_{q,q'}$ sets which are represented by ϵ-signs, then we can enlarge the ILP and the solution to include those inequations.

For the next lemma, we want to distinguish the finite inequation transition sets from the infinite ones, after removing some unimportant inequations.

Definition 37. Let $L_{\beta=\infty} := L(([+|-][a^*|\infty])^* \leq +\infty)$ be the set of all inequations with an annotated unbounded β-value. Let

$$\mathrm{Fin}_\Xi := \{\Xi_{q,q'} \mid q, q' \in Q \wedge |\Xi q, q' \backslash L_{\mathrm{Trash}}| < \infty \wedge \Xi_{q,q'} \cap L_{\beta=\infty} = \emptyset\}$$

be the set of all finite inequation transition sets without an unbounded β-value and

$$\mathrm{Inf}_\Xi := \{\Xi_{q,q'} \mid q, q' \in Q \wedge |\Xi q, q' \backslash L_{\mathrm{Trash}}| = \infty \wedge \Xi_{q,q'} \cap L_{\beta=\infty} = \emptyset\}$$

be the set of all infinite ones.

We will now find representatives for every set in Inf_Ξ such that the following holds. If an ILP consisting of inequations from the sets of Fin_Ξ has a solution, then we can extend the ILP with any combination of representatives of the sets in Inf_Ξ such that we can extend the solution to a solution of the extended ILP. Therefore, we can ignore inequations from the sets in Inf_Ξ, as we did by assigning ϵ as a representative in Definition 34.

Definition 38. Let $\sigma\colon [|\mathrm{Inf}_\Xi|] \to \mathrm{Inf}_\Xi$ be an arbitrary but fixed ordering of the sets in Inf_Ξ. We define alternative representatives *arep* for the inequation transition sets in Inf_Ξ with algorithm 3. Let $n := |Q|$ and $\#_\pm(w)$ denote the number of signs in w.

Algorithm 3 Computation of alternative representatives for $\Xi \in \mathrm{Inf}_\Xi$.

```
for $i \leftarrow 1$ to $|\mathrm{Inf}_\Xi|$ do
    $arep(\sigma(i)) \leftarrow \min_{\mathrm{lex}}(\{w \in \sigma(i) \mid i \cdot n^2 < \#_\pm(w) \leq (i+1) \cdot n^2\})$
end for
```

The idea is to pick inequations as alternative representatives, which form together a matrix in row echelon form. We can assign a value to the variable of the leading coefficient in a way, that this summand dominates the other summands, such that the inequation is satisfied.

The inequations in Ξ are defined over representatives of coefficient transition sets, which are by definition the sign $\pm\infty$ or labels of loop-fee paths. An inequation in the sets of Fin_Ξ can only consist of up to $n = |Q|$ different coefficients, each of size up to n. Therefore, every inequation in the sets of Fin_Ξ is of length at most n^2. The definition of $arep(\Xi)$ ensures that the representatives of $\Xi \in \mathrm{Inf}_\Xi$ contain more coefficients than any representative of the finite inequation transition sets. It also ensures that the number of coefficients contained in the representing inequation is strictly monotonously rising with the order σ. Especially, the index of the highest non-zero coefficient is strictly monotonously rising with σ.

Lemma 11. *Let w be a solvable* ILP *consisting only of inequations from sets in Fin_Ξ. Let $\vec{x}$ be a valid solution of w. Then, for every* ILP *w' consisting of w and additional inequations from $\{arep(\Xi) \mid \Xi \in Inf_\Xi\}$ the vector $\vec{x}$ can be extended to a solution $\vec{x}'$ of w'.*

Proof. Let $\vec{x} = (x_1, x_2, \ldots, x_i)$ and let m be the number of variables in w'. Let $\mathrm{var\_set}(\xi)$ be a function returning the variables appearing in the inequation ξ with a non-zero coefficient. We denote with $\mathrm{coeff}(\xi, y_i)$ the coefficient of variable y_i in the inequation ξ. With $\mathrm{value}(y_j)$ we denote the assigned value x_j of the variable y_j and $\beta(\xi)$ refers to the right side β of the inequation ξ. Algorithm 4 assigns values to the variables $y_{i+1}, y_{i+2}, \ldots, y_m$ such that $\vec{x}' = (x_1, \ldots, x_i, x_{i+1}, \ldots, x_m)$ is a solution of the ILP w'.

The algorithm works as follows. We go through the inequations appearing in w' which have been chosen as representatives for the sets in Inf_Ξ in the same order as in Algorithm 3 when we assigned the representatives. Therefore, the number of appearing variables per inequation is rising. In every inequation we consider, there is at least one variable which has not appeared in the previously considered inequations. We assign the new variables with a zero value, except for the variable with the highest index. This variable (called `MaxVar`) gets a value which compensates all the other summands in the inequation satisfying the inequation. We can choose the value of `MaxVar` freely, because the variable has not appeared in any other inequation we considered earlier and if it appears in any later considered inequation, there will always be at least one

Algorithm 4 Extending the solution $\vec{x}$ of the ILP w to a solution $\vec{x}'$ of w'.

```
$\texttt{AssignedVars} \leftarrow \{1, \dots, i\}$
for $j \leftarrow 1$ to $|\mathrm{Inf}_\Xi|$ do
    $\texttt{CurIneq} \leftarrow \mathit{arep}(\sigma(j))$
    if $\texttt{CurIneq}$ appears in $w'$ then
        $\texttt{ToAssign} \leftarrow \mathrm{var\_set}(\texttt{CurIneq}) \setminus \texttt{AssignedVars}$
        $\texttt{MaxVar} \leftarrow x_k \in \texttt{ToAssign}$ with highest index $k$
        $\texttt{ToAssign} \leftarrow \texttt{ToAssign} \setminus \{\texttt{MaxVar}\}$
        for all $y \in \texttt{ToAssign}$ do
            $\mathrm{value}(y) \leftarrow 0$
        end for
        $\texttt{AssignedVars} \leftarrow \texttt{AssignedVars} \cup \texttt{ToAssign}$
        $\texttt{ToAssign} \leftarrow \emptyset$

        $b \leftarrow \beta(\texttt{CurIneq})$
        $\texttt{SumOthCoeff} \leftarrow \sum_{x_l \in \{\mathrm{var\_set}(\texttt{CurIneq}) \setminus \{\texttt{MaxVar}\}\}} \mathrm{coeff}(\texttt{CurIneq}, x_l) \cdot \mathrm{value}(x_l)$
        $\texttt{CoeffMaxVar} \leftarrow \mathrm{coeff}(\texttt{CurIneq}, \texttt{MaxVar})$

        $\mathrm{value}(\texttt{MaxVar}) \leftarrow \frac{b - \texttt{SumOthCoeff}}{\texttt{CoeffMaxVar}}$
        $\texttt{AssignedVars} \leftarrow \texttt{AssignedVars} \cup \{\texttt{MaxVar}\}$
    end if
end for
```

new variable in the inequation which has not appeared earlier itself, and which can again compensate every other summand. It is easy to see that inequation CurIneq is satisfied by the chosen variable assignment. Therefore, $\vec{x}' = (x_1, \dots, x_i, \mathrm{value}(y_{i+1}), \dots, \mathrm{value}(y_m))$ is a solution of the ILP w'. $\square$

We will now show that, if there is a solvable ILP in $L(\mathrm{shrink}(A))$, we can replace any ϵ signs in this ILP by actual inequations, resulting in a solvable ILP in $L(A)$.

Lemma 12. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) \subseteq L_{enc}$. Let $w \in L(A)$ and let $S = \{\xi \in w \mid \exists\ \Xi_{q,q'}$ with $\xi \in \Xi_{q,q'}$, the subword ξ in w is read between q and q', and $rep(\Xi_{q,q'}) = \{\epsilon\}\}$ be the sets of inequations in w for which the representative ϵ could be read on a similar path in* shrink(A)*. Then, for $w' \in L'(\mathrm{shrink}(A))$ with w' being w with inequations in S being replaced by ϵ, it holds that if $w' \in \mathrm{ILP}_{enc}$, then either $w \in \mathrm{ILP}_{enc}$ or there exists a word $w'' \in L(A)$ with w' is a sub-word of w'' and $w'' \in \mathrm{ILP}_{enc}$[5].*

[5]Where the ϵ-signs in w' have been interpreted as the empty word.

Proof. It holds that w' only consists of inequations from sets in Fin_Ξ or the ϵ-sign. If w' is already a member of $L(A)$, the claim follows. Assume $w' \notin L(A)$. Since $w' \in L'(\text{shrink}(A))$, there is a path p in shrink(A) corresponding to w'. This path includes at least one transition $\delta(q,\epsilon) = q'$. By Definition 34, the set $\Xi_{q,q'}$ either contains an inequation with an ∞-sign as β-value or is in Inf_Ξ. Let $\vec{x}$ be a solution of w'. In the first case, we can replace the ϵ-sign by an inequation $(\vec{\alpha},\beta)$ from $\Xi_{q,q'}$ with $\vec{\alpha}\vec{x} \leq \beta$ since there are inequations with arbitrary high β-values in $\Xi_{q,q'}$. Let's consider the second case. Definition 38 defines representatives which can be read between q and q' in the original automaton instead of ϵ. Following Lemma 11 w' can be enlarged with inequations from $\{arep(\Xi) \mid \Xi \in \text{Inf}_\Xi\}$ without making the ILP represented by w' unsolvable. So, if we replace all the ϵ-transitions in the path p by paths in A corresponding to the alternative representatives form Definition 38, we get an ILP $w'' \in L(A)$ with $w'' \in \text{ILP}_{\text{enc}}$. □

Only simple paths in shrink(A) must be considered.

Lemma 13. *Let $w, w' \in L_{enc}$ and w' be w without an arbitrary inequation ξ from w. (So, w' is w with one inequation less.) If $w \in \text{ILP}_{enc}$ then $w' \in \text{ILP}_{enc}$.*

Proof. Adding an inequation can only decrease the set of solutions of the ILP. So, removing an inequation can not make the ILP unsolvable. □

If there exists a solvable ILP in $L(A)$, there exists a solvable ILP in $L'(\text{condensed}(A))$.

Lemma 14. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) \subseteq L_{enc}$. If $L(A) \cap \text{ILP}_{enc} \neq \emptyset$ then $L'(\text{condensed}(A)) \cap \text{ILP}_{enc} \neq \emptyset$.*

Proof. Let $w \in L(A) \cap \text{ILP}_{\text{enc}}$ be the label of an accepting path in A representing a solvable ILP. Lemma 10 states that by pumping the coefficients in w in the described way we obtain an ILP w' which is still solvable. Therefore, we used ∞-signs as a wild-card indicating that we can replace the coefficient with an arbitrary high coefficient. Note that we do not have to predict which value the corresponding variable will get, we just add all of the finitely many coefficient-variants to the set of representatives. So, by restricting the coefficient transition sets of A to the sets or representatives defined in Definition 32 we will not lose any solvable ILP. Therefore, we can focus only on inequations over the representatives of the coefficient transition sets.

By assigning representatives to the inequation transition sets in Definition 34, we kept every inequation in the finite transition sets and deleted the inequations in the infinite transition sets. Lemma 13 states that by removing inequations from w' we will not lose the solvability of the ILP. Therefore, if there is a solvable ILP in $L(A)$ there is a solvable ILP in $L(\text{shrink}(A))$, too.

Finally, Lemma 13 also tells us that if there is a solvable ILP in $L(\text{shrink}(A))$ it can be read on a simple path because adding more inequation makes the ILP harder to be true. Since in Definition 36 the automaton $\text{condensed}(A)$ is constructed by restring the automaton $\text{shrink}(A)$ to its simple paths it holds that $L(A) \cap \text{ILP}_{\text{enc}} \neq \emptyset$, then $L'(\text{condensed}(A)) \cap \text{ILP}_{\text{enc}} \neq \emptyset$. □

If there exists a solvable ILP in $L'(\text{condensed}(A))$, there exists a solvable ILP in $L(A)$

Lemma 15. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) \subseteq L_{enc}$. If $L'(\text{condensed}(A)) \cap \text{ILP}_{enc} \neq \emptyset$ then $L(A) \cap \text{ILP}_{enc} \neq \emptyset$.*

Proof. Let $w \in L(\text{condensed}(A))$ be the label of an accepting path in $\text{condensed}(A)$. In finite time, we can pick explicit coefficients for the wild-card-signs ∞ in w with values above $|Q|^{|Q|}$, such that their summand dominates the inequation in which it appears. Let w_c be w where the wild-cards are replaced by actual coefficients. We will show that if there is a $w \in L(\text{condensed}(A))$ such that $w_c \in \text{ILP}_{\text{enc}}$, then there is a word $x \in L(A) \cap \text{ILP}_{\text{enc}}$. Since $L'(\text{condensed}(A)) \subseteq L'(\text{shrink}(A))$ the word w_c is also in $L'(\text{shrink}(A))$ and by Lemma 12 we can extend the word w_c to a word $w_c{}' \in L(A)$, such that $w_c \in \text{ILP}_{\text{enc}} \Rightarrow w_c{}' \in \text{ILP}_{\text{enc}}$. If we set x to $w_c{}'$, the claim follows. □

5.3 Decidability

Theorem 11. *Let R be a regular language. It is decidable whether $R \cap \text{ILP}_{enc} \neq \emptyset$, i.e. the problem $int_{\text{Reg}}(\text{ILP})$ is decidable.*

Proof. Since L_{enc} is regular, we can restrict R to the regular language $R' = R \cap L_{\text{enc}}$. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) = R'$. For the automaton A, the Definitions 31 and 32 describe the construction of coefficient transition sets and assigning their representatives. In Definition 33 inequation transition sets are constructed based on those representatives. These inequation transition sets get representatives themselves in Definition 34. In Definition 35 a new automaton $\text{shrink}(A)$ is defined, based on the representatives for the inequation transition sets. Finally, in Definition 36 the automaton $\text{condensed}(A)$ is defined as the loop-free version of $\text{shrink}(A)$. All those constructions are constructive and can be computed by an algorithm. Lemma 14, together with Lemma 15, state that $L(A) \cap \text{ILP}_{\text{enc}} \neq \emptyset \Leftrightarrow L'(\text{condensed}(A)) \cap \text{ILP}_{\text{enc}} \neq \emptyset$. Since $L'(\text{condensed}(A))$ is by construction finite and testing a given ILP for solvability can be done in finite time, we can test all words in $L'(\text{condensed}(A))$ for membership in ILP_{enc} in finite time. Therefore, $R \cap \text{ILP}_{\text{enc}} \neq \emptyset$ is decidable. □

Chapter 6

Merge, Separate, and Replace

In this chapter we will elucidate the three main techniques of *merging*, *separating*, and *replacing*, which have been used to show decidability of the int_{Reg}-problem. In chapter 4 we used pumping arguments to separate the existentially quantified variables from each other, while we wanted to merge the references to universally quantified variables wherever possible. In chapter 5 we replaced coefficients with other coefficients which simplified the inequation system. We will discuss the techniques of merging, separating, and replacing by using them to show decidability of the int_{Reg}-problem for Vertex Cover, Independent Set, and Knapsack. Each of this proofs will focus on one of the techniques and illustrates it's applicability.

6.1 Merging - Vertex Cover

First, we want to investigate the problem Vertex Cover (short VC) and demonstrate the method of *merging* while proving decidability of $int_{\mathrm{Reg}}(\mathrm{VC})$. We begin with a formal definition of the NP-complete problem [CFK$^+$15].

Definition 39 (VC).
Given: Graph $G = (V, E)$ and a positive integer k.
Question: Is there a vertex cover for G of size k or less, i.e., a subset $V' \subseteq V$ with $|V'| \leq k$ such that for each edge $\{u, v\} \in E$ it holds that $u \in V' \vee v \in V'$?

We will encode the problem in the following way. First, the integer k is given in an unary encoding. Then, the edges in E are listed, separated by \$-signs, where the vertexes are separated by #-signs and given in a unary encoding as well. Vertexes, which are not part of any edge, are listed in an unary encoding at the end of the word, separated by \$-sings. The set of all encoded VC-instances, regardless if they are solvable or not, is a subset of the following set.

Definition 40. Every encoded VC-instance is a member of the set $\mathrm{Enc}_{\mathrm{VC}}$ defined by the regular expression

$$a^*\$(a^*\# a^*\$)^*(a^*\$)^*$$

Note that the set $\mathrm{Enc}_{\mathrm{VC}}$ is regular. The following example illustrates the encoding.

Example 4. *Let $G = (V, E)$ be a graph with*

$$V = \{1, 3, 4, 8, 10, 11\},$$

$$E = \{\{1, 4\}, \{1, 11\}, \{4, 8\}, \{4, 11\}, \{8, 10\}\{10, 4\}\}.$$

The VC*-instance $G, k = 3$ is encoded as:*

$$a^3\$a\#a^4\$a\#a^{11}\$a^4\#a^8\$a^4\#a^{11}\$a^8\#a^{10}\$a^{10}\#a^4\$a^3\$$$

The sequence of indexes of the vertexes does not need to be succeeding, nor increasing.

We want to show decidability of the $\mathit{int}_{\mathrm{Reg}}(\mathrm{VC})$ problem. Therefore, we will start with a regular language accepted by a deterministic finite automaton and construct a new automaton condensed(A). It will hold that there is a solvable VC-instance under finitely many candidates in L(condensed(A)) if and only if there is one in the original language. To do so, we have to go through some definitions and lemmas first.

6.1.1 Construction

Every word w in $\mathrm{Enc}_{\mathrm{VC}}$ consists of three parts. The first string of a's determines the bound k. If this part can be read over a loop, the string of a's can be pumped to an arbitrary high number. Therefore, we can enlarge the bound k until it is larger than $|V|$, in which case we know that $R \cap \mathrm{VC} \neq \emptyset$.

The second part of w describes the set E. Adding an additional edge to a graph will never make the graph more likely to contain a vertex cover of a fixed size. Therefore, pumping whole edges will not yield to a word which is in VC if the original one was not. But, changing already contained edges such that they both share a vertex, might let the graph contain a vertex cover of the demanded size. To collect all the possibilities of edges sharing a vertex, we will define vertex transition sets. Then, we will look at every nonempty intersection of those sets like in Definition 19 and assign representatives to the vertex transitions sets similar to Algorithm 1.

The third part of w lists all vertexes of the graph, which are not part of any edge. Adding, removing, or replacing those vertexes do not influence whether the graph contains a vertex cover, therefore loops over those parts can be ignored.

Definition 41. Let $A = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) \subseteq \mathrm{Enc}_{\mathrm{VC}}$. For every pair of states $q, q' \in Q$ we define the two sets

$$\Lambda^1_{q,q'} := \{x \in L(a^*\#) \mid \exists q_1 \in Q\colon \delta(q_1, \$) = q \wedge \delta^*(q, x) = q'\},$$
$$\Lambda^2_{q,q'} := \{x \in L(a^*\$) \mid \exists q_1 \in Q\colon \delta(q_1, \#) = q \wedge \delta^*(q, x) = q'\},$$

as the *vertex transition sets* from q to q', where $\Lambda^1_{q,q'}$ $[\Lambda^2_{q,q'}]$ contains vertexes being the first [second] part of an edge. Furthermore, let

$$\Lambda_{q,q'} := \Lambda^1_{q,q'} \cup \Lambda^2_{q,q'}$$

be the union of all vertex transition sets from q to q'. For readability reasons, we will sometimes refer to $\Lambda^i_{q,q'}$ as Λ if i and q, q' are understood.

Each Λ is a regular set because it is recognized by a sub-automaton of A.

We define the operations trunc_{VC} and extend_{VC} according to the operations *trunc* and *extend* in Definition 18.

Definition 42. Let $\mathit{trunc}_{VC}\colon \Gamma = \{a, \$, \#\}^* \to \{a\}^*$ be an homomorphism with

$$\mathit{trunc}_{VC}(\gamma) := \begin{cases} a & \text{if } \gamma = a, \\ \varepsilon & \text{otherwise.} \end{cases}$$

We define the operation extend_{VC} such that $\mathit{extend}_{VC}(w, \Lambda) := \mathit{trunc}^{-1}_{VC}(w) \cap \Lambda$ for $w \in \mathit{trunc}_{VC}(\Lambda)$ and language Λ .

Intuitively, function trunc_{VC} returns the vertex forgetting its position in an edge, while extend_{VC} provides us with all possible occurrences of a vertex in an edge leading from state q to state q'.

Based on Definition 19 we define the set of all combinations of vertex transition sets sharing at least one vertex. This means that for all corresponding pairs of states we can read the same vertex and the involved edges share this vertex.

Definition 43. Let $P := \{p \in \mathcal{P}(\{\Lambda_{q,q'} \mid q, q' \in Q\}) \mid \cap_{\Lambda \in p} \mathit{trunc}_{VC}(\Lambda) \neq \emptyset\}$ be the subset of the powerset of all vertex transition sets, which only contains sets of languages with a common vertex.

Note that for every $p \in P$ all non-empty subsets of p are also contained in P.

We will now pick a finite set of representatives $\mathit{rep}(\Lambda) \subseteq \Lambda$ for every vertex transition set Λ. We determine these sets with Algorithm 5 based on Algorithm 1.

Definition 44. For every vertex transition set $\Lambda^i_{q,q'}$ we define a finite set of representatives $rep(\Lambda^i_{q,q'})$ with Algorithm 5.

Algorithm 5 Computation of $rep(\Lambda)$ for vertex transition sets.

```
for all $\Lambda \in \{\Lambda^i_{q,q'} \mid q, q' \in Q, i \in \{1,2\}\}$ do
    $rep(\Lambda) \leftarrow \emptyset$
end for
for all $p \in P$ do
    $label(p) \leftarrow \min_{\text{lex}} \left(\bigcap_{\Lambda \in p} trunc_{VC}(\Lambda)\right)$
    for all $\Lambda \in p$ do
        $rep(\Lambda) \leftarrow rep(\Lambda) \cup extend_{VC}(label(p), \Lambda)$
    end for
end for
```

The sets $rep(\Lambda)$ are finite because the set $\{\Lambda^i_{q,q'} \mid q, q' \in Q, i \in \{1,2\}\}$ is finite and for every element of this set only one representative is picked.

Next, we will define transition sets for the threshold k and for every single vertex not appearing in an edge. If the transition set for k is infinite we will always find a word in R which represents a solvable VC instance because we can set the boundary k to a larger number than the number of vertexes. Hence, if the transition set of k is infinite, we set a string larger than $2^{2^{2|Q|^2}}$ as a representative[1].

Definition 45. Let $A = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) \subseteq \text{Enc}_{\text{VC}}$. For all $q \in Q$ let $\mathcal{K}_{q_0,q} := \{w \in L(a^*\$) \mid \delta^*(q_0, w) = q\}$ be the transition set of k from the initial state q_0 to the state q.

Definition 46. For every transition set $\mathcal{K}_{q_0,q}$ of k we define a finite set of representatives $rep(\mathcal{K}_{q_0,q})$ as

$$rep(\mathcal{K}_{q_0,q}) := \begin{cases} \{\min_{\text{lex}}(\mathcal{K}_{q_0,q} \cap L\left(a^{\geq 2^{2^{2|Q|^2}}}\$\right))\} & \text{if } |\mathcal{K}_{q_0,q}| = \infty, \\ \mathcal{K}_{q_0,q} & \text{otherwise.} \end{cases}$$

Since the single vertexes do not contribute to the problem VC we will assign, in the case of infinite transition sets, the lexicographical minimal element of their transition sets, which does not overlap with any previously assign representative, as a representative. If the transition set is finite, we chose the whole set as the set of representatives.

[1]There are $2|Q|^2$ many vertex transition sets Λ. The power set of the set of all Λ has $2^{2|Q|^2}$ elements. Therefore, there is a total of $2^{2|Q|^2}$ different representatives for vertexes appearing in edges. So, there can be up to $2^{2^{2|Q|^2}}$ different edges in a word from $L(\text{condensed}(A))$.

Definition 47. Let $A = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) \subseteq \mathrm{Enc}_{\mathrm{VC}}$. For all $q, q' \in Q$ we define the transition set for single vertexes not appearing in an edge as

$$\mathcal{S}_{q,q'} := \{w \in L\left(a^*\$\right) \mid \exists q_1 \in Q \colon \delta(q_1, \$) = q \wedge \delta^*(q, w) = q'\}.$$

Definition 48. For every transition set $\mathcal{S}_{q,q'}$ we define a finite set of representatives $\mathit{rep}(\mathcal{S}_{q,q'})$ as

$$\mathit{rep}(\mathcal{S}_{q,q'}) := \begin{cases} \{\min\limits_{\text{lex}}\big(\mathcal{S}_{q,q'} \backslash \{\mathit{extend}_{VC}(\mathit{trunc}_{VC}(\gamma), \mathcal{S}_{q,q'}) \mid \\ \qquad\qquad \gamma \in \bigcup_{q,q' \in Q,\, i \in \{1,2\}} \mathit{rep}(\Lambda^i_{q,q'})\}\big)\} & \text{if } |\mathcal{S}_{q,q'}| = \infty, \\ \mathcal{S}_{q,q'} & \text{otherwise.} \end{cases}$$

So, by pumping, the isolated vertexes do not fall together with vertexes contained in edges.

Now we are able to define a condensed automaton based on the chosen representatives according to Definition 24.

Definition 49. Let $R \subseteq \mathrm{Enc}_{\mathrm{VC}}$ be a regular language and $A = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) = R$. We define the *condensed automaton* $\mathrm{condensed}(A) = (Q, \Gamma', \delta', q_0, F)$ with

$$\begin{aligned} \Gamma' = \bigcup_{q \in Q} \mathit{rep}(\mathcal{K}_{q_0,q}) \cup \bigcup_{q,q' \in Q} \mathit{rep}(\Lambda_{q,q'}) \cup \bigcup_{q,q' \in Q} \mathit{rep}(\mathcal{S}_{q,q'}) \\ \text{and} \quad \forall q \in Q : (q_0, w, q) \in \delta' \text{ if } w \in \mathit{rep}(\mathcal{K}_{q_0,q}) \\ \forall q, q' \in Q : (q, w, q') \in \delta' \text{ if } w \in \mathit{rep}(\Lambda_{q,q'}) \\ \forall q, q' \in Q : (q, w, q') \in \delta' \text{ if } w \in \mathit{rep}(\mathcal{S}_{q,q'}) \ . \end{aligned}$$

6.1.2 Correctness

In this section we will show for a given DFA A that $L(A) \cap \mathrm{VC} \neq \emptyset \Leftrightarrow L(\mathrm{condensed}(A)(A)) \cap \mathrm{VC} \neq \emptyset$.

Lemma 16. *Let $R \subseteq \mathit{Enc}_{VC}$ be a regular language, defined through the deterministic finite automaton $A = (Q, \Gamma, \delta, q_0, F)$. If $L(\mathrm{condensed}(A)) \cap \mathrm{VC} \neq \emptyset$ then also $R \cap \mathrm{VC} \neq \emptyset$.*

Proof. By construction $L(\mathrm{condensed}(A))$ is a subset of $L(A)$. Therefore, every solvable VC instance in $L(\mathrm{condensed}(A))$ also appears in R. □

Lemma 17. *Let $R \subseteq \mathit{Enc}_{VC}$ be a regular language, defined through the deterministic finite automaton $A = (Q, \Gamma, \delta, q_0, F)$. If $R \cap \mathrm{VC} \neq \emptyset$ then also $L(\mathrm{condensed}(A)) \cap \mathrm{VC} \neq \emptyset$.*

Proof. Let $w \in R \cap \mathrm{VC}$. Then w can be split up into factors

$$w = w_k w_{e_1^1} w_{e_1^2} w_{e_2^1} w_{e_2^2} \dots w_{e_m^1} w_{e_m^2} w_{v_1} w_{v_2} \dots w_{v_n}$$

where $w_k \in \mathcal{K}_{q_0, q_{e_1^1}}$, $w_{e_i^1} \in \Lambda^1_{q_{e_i^1}, q_{e_i^2}}$, $w_{e_i^2} \in \Lambda^2_{q_{e_i^2}, q_{e_{i+1}^1}}$, and $w_{v_j} \in \mathcal{S}_{q_{v_j}, q_{v_{j+1}}}$. Then, there is a word $w' \in \mathrm{condensed}(A)$ which can be analogously partitioned into

$$w' = {w_k}' {w_{e_1^1}}' {w_{e_1^2}}' {w_{e_2^1}}' {w_{e_2^2}}' \dots {w_{e_m^1}}' {w_{e_m^2}}' {w_{v_1}}' {w_{v_2}}' \dots {w_{v_n}}'.$$

We will replace the sub-words in $\Lambda^1_{q,q'}$ with sub-words in $rep(\Lambda^1_{q,q'})$ and the sub-words in $\Lambda^2_{q,q'}$ with sub-words in $rep(\Lambda^2_{q,q'})$. By construction, there are at most $2^{2|Q|^2}$ words in all sets $rep(\Lambda)$ together. Therefore, there can be at most $2^{2|Q|^{2^2}}$ different edges in a word in $L(\mathrm{condensed}(A))$. So, w_k is either already contained in $rep(\mathcal{K}_{q_0, q_{e_1}})$, and we set $w'_k = w_k$, or we set $w'_k = a^{2^{2|Q|^{2^2}}} \in rep(\mathcal{K}_{q_0, q_{e_1}})$ and hence k is larger then $|E|$. Since additional vertexes, which do not appear in any edge, don't have to be considered while finding a vertex cover, we can just replace the sub-words $w_{v_j} \in \mathcal{S}_{q_{v_j}, q_{v_{j+1}}}$ with their representatives in $rep(\mathcal{S}_{q_{v_j}, q_{v_{j+1}}})$. For the sub-words $w_{e_i^1} \in \Lambda^1_{q_{e_i^1}, q_{e_i^2}}$ and $w_{e_i^2} \in \Lambda^2_{q_{e_i^2}, q_{e_{i+1}^1}}$ we have to consider which other sub-words $w_{e_h} \in \Lambda_{q,q'}$ reference the same vertex. For a sub-word $w_e \in E_w := \{w_{e_1^1}, w_{e_1^2}, w_{e_2^1}, w_{e_2^2}, \dots, w_{e_m^1}, w_{e_m^2}\}$ let $H_1 := \{h_1 \in \{e_1^1, \dots e_m^1\} \mid \mathit{trunc}_{VC}(w_{h_1}) = \mathit{trunc}_{VC}(w_e)\}$, $H_2 := \{h_2 \in h_2 \in \{e_1^2, \dots e_m^2\} \mid \mathit{trunc}_{VC}(w_{h_2}) = \mathit{trunc}_{VC}(w_e)\}$. Let $H := H_1 \cup H_2$ be the index set of all sub-words encoding the same vertex as w_e. The collection of the corresponding vertex transition sets $p_e := \Big(\{\Lambda_{q_{h_1}, q_{h_2}} \mid h_1 \in H\} \cup \{\Lambda_{q_{h_2}, q_{(h+1)_1}} \mid h_2 \in H\}\Big)$ is in P, because the intersection of the vertex transition set is by definition not empty. By construction, for all $h_1, h_2 \in H$ the set $rep(\Lambda_{h_1,h_2})$ respectively $rep(\Lambda_{h_2, h+1_1})$ received a label from p_e. We pick ${w_h}'$ for all $h \in H$ as the label given by p_e. This will yield a consistent renaming of the vertex w_e.

The process of renaming will not disconnect any vertexes which has been previously connected, but it might lead to merging of distinct vertexes. If for example, there are several occurrences of vertexes in the set $\Lambda_{q,q'}$, which all appear in one single edge each, then the renaming process will replace all of them with the same vertex, since they are all only in the set $p := \{\Lambda_{q,q'}\}$. Lemma 18 will proof that merging vertexes in this way will maintain the existence of a vertex cover of the demanded size. So, we constructed a word $w' \in \mathrm{condensed}(A)$ which is also in VC. Together with Lemma 18 this concludes the proof. $\square$

Lemma 18. *Let $G = (V, E)$ be a graph containing a vertex cover V' of size k. Let $v, v' \in V$ be two distinct vertexes. Then, $G^\star = (V^\star, E^\star)$ being G, where v' and v have been merged by replacing v' by v, also contains a vertex cover V'' of size at most k.*

Proof. Since G has a vertex cover, for each edge there has to be one vertex in V'. We make a case distinction of the vertexes contained in V'.

- If $v \notin V'$ and $v' \notin V'$, then for all $\{v, u\}, \{v', u'\} \in E$ the vertexes u and u' has to be in V'. If v' is replaced by v, all edges $\{v, u\} \in E^\star$ are also covered by $V'' = V'$.
- If $v \notin V'$ and $v' \in V'$, then for all $\{v, u\} \in E$ it holds $u \in V'$. If v' is replaced by v, all edges $\{v, u\} \in E^\star$ are also covered by $V'' = (V' \backslash \{v'\}) \cup \{v\}$. Edges $\{v', u'\} \in E$ are replaced by edges $\{v, u'\} \in E^\star$ and hence covered by $v \in V''$.
- If $v \in V'$ and $v' \notin V'$, then for all $\{v', u'\} \in E$ it holds $u' \in V'$. If v' is replaced by v, all edges $\{v, u\} \in E^\star$ are also covered by $V'' = V'$. Edges $\{v', u'\} \in E$ are replaced by edges $\{v, u'\} \in E^\star$ and hence covered by $v \in V''$.
- If $v \in V'$, $v' \in V'$, and v' is replaced by v, all edges $\{v, u\} \in E^\star$ are also covered by $V'' = V' \backslash \{v'\}$. Edges $\{v', u'\} \in E$ are replaced by edges $\{v, u'\} \in E^\star$ and hence covered by $v \in V''$.

Edges not containing v or v' have not been changed and hence V'' is a vertex cover for $G^\star$ of size at most k. □

6.1.3 Decidability

We now show that it is decidable whether $L(\text{condensed}(A))$ contains a solvable Vertex Cover instance. Therefore, we will proof that it is sufficient to consider only simple paths in condensed(A). We will give a lemma similar to Lemma 5.

Lemma 19. *Let A be a deterministic finite automaton with $L(A) \subseteq Enc_{\mathrm{VC}}$ and $w \in L(\text{condensed}(A))$ be the labeling of an accepting path p in* condensed(A) *containing at least one loop. Let w' be w without the factors read in the loops of p. If w is a solvable* Vertex Cover *instance, so is w'.*

Proof. The structure of the words in Enc_{VC} only allows loops in the listing of edges or in the listing of single vertexes. Adding additional edges to a graph introduces more edges, which have to be covered, and therefore never makes the graph more likely to contain a vertex cover. Therefore, removing edges from the graph, which are by assumption already covered by a vertex cover, yields a graph which is also covered by the same vertex cover. Since not connected vertexes doesn't need to be considered while finding a vertex cover, adding or removing them do not influence the existence of a vertex cover. Therefore, w' is covered by the same vertex cover then w. □

This means that we only have to consider simple paths in condensed(A) when looking for solvable VC instances.

Lemma 20. *Let $R \subseteq Enc_{VC}$ be a regular language accepted by the deterministic finite automaton A. It is decidable whether $L(\text{condensed}(A)) \cap \text{VC} \neq \emptyset$.*

Proof. To decide whether the intersection is empty or not, we enumerate all finitely many words $w_1, w_2, \ldots, w_n$ which are labels of accepting paths in condensed(A). Since the problem Vertex Cover is in NP [GJ79], we can test for each of the words w_i in finite time whether $w_i \in \text{VC}$. According to Lemma 19, if for all words w_1 to w_n this is not the case, no other word $w \in \text{condensed}(A)$ is in VC. Therefore, the intersection $L(\text{condensed}(A)) \cap \text{VC}$ is not empty if and only if at least one $w_i \in \text{VC}$ which can be tested in finite time. □

Summarizing the previous lemmas yields a proof for the decidability of $int_{\text{Reg}}(\text{VC})$.

Theorem 12. *Let R be a regular language. It is decidable whether $R \cap$ Vertex Cover $\neq \emptyset$, i.e. int_{Reg}(Vertex Cover) is decidable.*

Proof. Since R and Enc_{VC} are regular languages, we can assume w.l.o.g. that $R \subseteq \text{Enc}_{\text{VC}}$. Let A be a deterministic finite automaton recognizing R. Definitions 41 to 49 describe the computable construction of the automaton condensed(A). Lemma 16 together with Lemma 17 prove that $L(\text{condensed}(A)) \cap \text{VC} \neq \emptyset \Leftrightarrow R \cap \text{VC} \neq \emptyset$. Following Lemma 20 the question $L(\text{condensed}(A)) \cap \text{VC} \neq \emptyset$ is decidable. Hence, we can decide whether $R \cap$ Vertex Cover $\neq \emptyset$ by construction the automaton condensed(A) and testing whether $L(\text{condensed}(A)) \cap \text{VC} \neq \emptyset$. □

6.2 Separating - Independent Set

While for the problem Vertex Cover merging vertexes of a graph made the graph more likely to contain a vertex cover, for the problem Independent Set we want to separate the vertexes from each other. The idea of *separating* is to reference wherever possible a uniquely occurring vertex. A uniquely occurring vertex has a degree of at most one and for a vertex of degree one, we can either put the vertex itself or its neighbor in an independent set. We begin with a formal definition of the NP-complete problem Independent Set (short IS) [CFK+15].

Definition 50 (IS).
Given: A graph $G = (V, E)$ and a positive integer k.
Question: Does G have an independent set V' of size at least k, i.e. is there

a set $V' \subseteq V$ with $|V'| \geq k$ such that no two vertexes in V' are joined by an edge?

We encode this problem in the same way we encoded VERTEX COVER.

Definition 51. Every encoded IS-instance is a member of the regular set $\mathrm{Enc}_{\mathrm{IS}}$ defined by the regular expression $a^*\$(a^* \# a^*\$)^*(a^*\$)^*$.

We refer to Example 4 for an illustration of the encoding.

6.2.1 Construction

Again, we will construct from a given automaton A, an automaton condensed(A) based on sets of representatives. In the accepted language $L(\text{condensed}(A))$ only finitely many words have to be considered, under which there is a solvable IS-instance, if and only if there is one in the given language $L(A)$. In the last section we tried to merge vertexes and therefore joined edges such that fewer vertexes can cover more edges. This time we are looking for preferably large sets of vertexes which are not connected by edges. Our approach is to disconnect edges by reading uniquely occurring vertex labels in the automaton wherever possible. Therewith, we are reducing the degree of vertexes in the graph, while we are increasing the number of vertexes appearing in the graph, which makes it easier to find an independent set of the demanded size.

Since IS uses the same encoding as VC we will make use of the transition sets defined in Definition 41, 45, and 47 but we will assign representatives in a different way.

Since the bound k is a lower bound, we can just set it to the smallest value possible for each group of words. Because the language is given by an automaton, there is always a value for k smaller than $|Q|$ meaning that we have to look for independent sets with a size of at least $|Q|$. We will use this fact and assign every vertex transition set $\Lambda_{q,q'}$ with a domain of size $(|Q|+1) \cdot |Q|$ and define its set of representatives as the set of all words in this domain. Therewith, we make sure that we can read at least $|Q|$ different vertexes between q and q' which can not be read between any other pair of states. For the transition sets $\mathcal{S}_{q,q'}$ we will proceed in a similar way, assigning them with domains of equal size above the domains for the vertex transition sets. We will later see that these sets of representatives are large enough such that $L(\text{condensed}(A))$ contains a solvable IS-instance if and only if $L(A)$ does.

We will now define sets of representatives for the transition sets $\Lambda_{q,q'}$ (Definition 41), $\mathcal{K}_{q,q'}$ (Definition 45), and $\mathcal{S}_{q,q'}$ (Definition 47).

Definition 52. For every transition set $\mathcal{K}_{q_0,q}$ we define a finite set of representatives $\mathit{rep}(\mathcal{K}_{q_0,q})$ as

$$\mathit{rep}(\mathcal{K}_{q_0,q}) = \left\{ \min_{\text{lex}}(\mathcal{K}_{q_0,q}) \right\}.$$

Note that $|\min_{\text{lex}}(\mathcal{K}_{q_0,q})| \leq |Q|$.

Since we have to look for independent sets of size at most $|Q|$ it is enough to assign every infinite vertex transition set $\Lambda_{q,q'}$ with $|Q|+1$ unique representatives. We need more than one representative, because vertexes read in between q and q' can be part of a loop and every time the loop is taken we want to be able to read a uniquely occurring vertex. We need at most $|Q|$ uniquely occurring vertexes, read in the same side of an edge, to be able to construct an independent set of size at least $|Q|$, namely the set of all those vertexes. The additional "+1" vertex is needed in case the word is mention to have more than $|Q|$ occurrences of a label of a path between q and q'. In order to avoid loosing the uniqueness of the other $|Q|$ labels we add an additional lexicographical largest label to the set of representatives, which should be read when more than $|Q|$ labels are needed. This element acts like a *garbage reference* which will likely not be part of an independent set.

Since single loops in between q and q' are at most of length $|Q|$, the elements in $\Lambda_{q,q'}$ have a lexicographical distance of at most $|Q|$. Therefore, a domain of $(|Q|+1)\cdot|Q|$ will contain at least $|Q|+1$ different words from $\Lambda_{q,q'}$.

If the set $\Lambda_{q,q'}$ is finite we just assign all its member to the set of representatives.

Definition 53. We define for all vertex transition sets $\Lambda_{q,q'}$ sets of representatives in the following way:
Order the sets $\Lambda_{q,q'}$ in an arbitrary but fixed way, such that $\Lambda^{\sigma(i)}$ is the i-th Λ set starting with $i = 1$. Then, we set

$$rep(\Lambda^{\sigma(i)}) := \begin{cases} \Lambda^{\sigma(i)} & \\ & \text{if } |\Lambda^{\sigma(i)}| < \infty, \\ \Lambda^{\sigma(i)} \cap domain^1(i\cdot(|Q|^2+|Q|), (i+1)\cdot(|Q|^2+|Q|)-1) & \\ & \text{if } \Lambda^{\sigma(i)} = \Lambda^1_{q,q'}, \\ \Lambda^{\sigma(i)} \cap domain^2(i\cdot(|Q|^2+|Q|), (i+1)\cdot(|Q|^2+|Q|)-1) & \\ & \text{if } \Lambda^{\sigma(i)} = \Lambda^2_{q,q'}. \end{cases}$$

Where $domain^1(x,y) := \{a^x\#, a^{(x+1)}\#, \ldots, a^{(y-1)}\#, a^y\#\}$ and $domain^2(x,y) := \{a^x\$, a^{(x+1)}\$, \ldots, a^{(y-1)}\$, a^y\$\}$ being the set of all vertexes with values between x and y which can be read in the first, respectively second, position of an edge.

With the same argument, we assign the infinite single vertex transition sets $\mathcal{S}_{q,q'}$ with at least $|Q|$ unique representatives. Here, we do not need the extra *garbage* representative "+1" because reading the same non-connected vertex twice adds no extra edge to the graph. We set the domains for $rep(\mathcal{S}_{q,q'})$ above the domains of $rep(\Lambda)$ which is bounded by $|Q|^2\cdot(|Q|+1)\cdot|Q| = |Q|^4+|Q|^3$.

Definition 54. We define for all unconnected vertex transition sets $\mathcal{S}_{q,q'}$ sets of representatives in the following way:
Order the sets $\mathcal{S}q, q'$ in an arbitrary but fixed way, such that $\mathcal{S}^{\sigma(i)}$ is the i-th $\mathcal{S}$ set starting with $i = 1$. Then, we set

$$rep(\mathcal{S}^{\sigma(i)}) := \begin{cases} \mathcal{S}^{\sigma(i)} & \text{if } |\mathcal{S}^{\sigma(i)}| < \infty, \\ \mathcal{S}^{\sigma(i)} \cap domain^2(|Q|^4 + |Q|^3 + i \cdot |Q|^2, & \\ \qquad |Q|^4 + |Q|^3 + (i+1) \cdot |Q|^2 - 1) & \text{otherwise.} \end{cases}$$

Now we are able to construct the condensed automaton condensed(A) based on the sets of representatives.

Definition 55. Let $R \subseteq \text{Enc}_{\text{IS}}$ be a regular language and $A = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) = R$. We define the *condensed automaton* condensed$(A) = (Q, \Gamma', \delta', q_0, F)$ with

$$\begin{aligned} \Gamma' = & \bigcup_{q \in Q} rep(\mathcal{K}_{q_0,q}) \cup \bigcup_{q,q' \in Q} rep(\Lambda_{q,q'}) \cup \bigcup_{q,q' \in Q} rep(\mathcal{S}_{q,q'}) \\ \text{and} \quad & \forall q \in Q : (q_0, w, q) \in \delta' \text{ if } w \in rep(\mathcal{K}_{q_0,q}) \\ & \forall q, q' \in Q : (q, w, q') \in \delta' \text{ if } w \in rep(\Lambda_{q,q'}) \\ & \forall q, q' \in Q : (q, w, q') \in \delta' \text{ if } w \in rep(\mathcal{S}_{q,q'}) \ . \end{aligned}$$

6.2.2 Correctness

In this section we will show for a given DFA A that there is a solvable Independent Set instance in $L(A)$ if and only if there is one in $L(\text{condensed}(A))$. The proof is pretty similar to the VC case.

Lemma 21. *Let $R \subseteq Enc_{IS}$ be a regular language defined through the deterministic finite automaton $A = (Q, \Gamma, \delta, q_0, F)$. If $L(\text{condensed}(A)) \cap \text{IS} \neq \emptyset$, then also $R \cap \text{IS} \neq \emptyset$.*

Proof. By construction $L(\text{condensed}(A))$ is a subset of $L(A)$. Therefore, every solvable IS instance in $L(\text{condensed}(A))$ is also in R. □

Lemma 22. *Let $R \subseteq Enc_{IS}$ be a regular language defined through the deterministic finite automaton $A = (Q, \Gamma, \delta, q_0, F)$. If $R \cap \text{IS} \neq \emptyset$, then also $L(\text{condensed}(A)) \cap \text{IS} \neq \emptyset$.*

Proof. Let $w \in R \cap \text{VC}$. Then, w can be split up into factors

$$w = w_k w_{e_1^1} w_{e_1^2} w_{e_2^1} w_{e_2^2} \dots w_{e_m^1} w_{e_m^2} w_{v_1} w_{v_2} \dots w_{v_n}$$

where $w_k \in \mathcal{K}_{q_0,q_{e_1^1}}$, $w_{e_i^1} \in \Lambda^1_{q_{e_i^1},q_{e_i^2}}$, $w_{e_i^2} \in \Lambda^2_{q_{e_i^2},q_{e_{i+1}^1}}$, and $w_{v_j} \in \mathcal{S}_{q_{v_j},q_{v_{j+1}}}$. We will show that there is a word $w' \in L(\text{condensed}(A))$ which can be analogously partitioned into

$$w' = w_k{}'w_{e_1^1}{}'w_{e_1^2}{}'w_{e_2^1}{}'w_{e_2^2}{}' \dots w_{e_m^1}{}'w_{e_m^2}{}'w_{v_1}{}'w_{v_2}{}' \dots w_{v_n}{}'$$

by replacing the sub-words of w by elements of the sets of representatives of the transition sets they appear in. Let G_w be the graph encoded in w. By assumption, G contains an independent set of size at least k where k is the number encoded in w_k. Since $rep(\mathcal{K}_{q_0,q_{e_1^1}})$ is the smallest element in $\mathcal{K}_{q_0,q_{e_1^1}}$ it holds for the single element $w_k{}' \in rep(\mathcal{K}_{q_0,q_{e_1^1}})$ that $k' \leq k$. Therefore, G also contains an independent set of size at least k'. The sub-words $w_{e_1^1}{}', w_{e_1^2}{}', w_{e_2^1}{}', w_{e_2^2}{}', \dots, w_{e_m^1}{}', w_{e_m^2}{}'$ are chosen from the sets of representatives $rep(\Lambda^1_{q_{e_i^1},q_{e_i^2}})$ and $rep(\Lambda^2_{q_{e_i^2},q_{e_{i+1}^1}})$ according to Algorithm 6.

Algorithm 6 Assignation of sub-words $w_e{}'$ to sub-words w_e.

```
OccuringΛ ← List($\Lambda^{l'}_{q,q'}$ | $w_{e_l^{l'}} \in \Lambda_{q,q'}$ for some $l, l'$)
for $j \leftarrow 1$ to |Occuring_Λ| do
    Counter $i_j \leftarrow 0$
end for
for all $w_e \in E_w := \{w_{e_1^1}, w_{e_1^2}, w_{e_2^1}, w_{e_2^2}, \dots, w_{e_m^1}, w_{e_m^2}\}$ do
    $j \leftarrow$ Number in OccuringΛ of $\Lambda_j$ in which $w_e$ occurs.
    if $|\Lambda_j| < \infty$ then
        $w_e' \leftarrow w_e$
    else
        if $i_j < |rep(\Lambda_j)|$ then
            $i_j \leftarrow i_j + 1$
            $w_e' \leftarrow i_j$-th element of $rep(\Lambda_j)$
        else
            $w_e' \leftarrow \max_{\text{lex}}(rep(\Lambda_j))$
        end if
    end if
end for
```

So, we sequentially replace the sub-words occurring in an infinite transition set Λ through uniquely occurring representatives, while we keep track of the already assigned words through the counters i_j. If in w there are more sub-words from Λ then representatives in $rep(\Lambda)$ we replace the surplus sub-words all by the lexicographical maximal representative.

The sub-words $w_{v_j} \in \mathcal{S}_{q_{v_j},q_{v_{j+1}}}$ are replaced by representatives $w_{v_j}{}' \in rep(\mathcal{S}_{q_{v_j},q_{v_{j+1}}})$ in a similar way with Algorithm 6. The difference is, that we form the list *OccuringS* over the transition sets $\mathcal{S}$ and iterate over $V_w := \{w_{v_1}w_{v_2}\dots w_{v_n}\}$ instead of E_w. By this renaming we can lose some unconnected vertexes if we have more then $|Q|$ sub-words representing unconnected vertexes in the same transition set $\mathcal{S}$. But, since $k' \leq k$ with $k' \leq |Q|$ there are still enough unconnected vertexes in w' left which form an independent set of size at least k'.

It remains to show that, when G_w contained an independent set of size at least k, replacing the vertexes which are part of edges maintains the existence of an independent set of size at least k'. We will prove this statement separately in Lemma 23.

All together, beginning with a word $w \in L(A) \cap \mathrm{IS}$ we have constructed a word $w' \in L(\text{condensed}(A)) \cap \mathrm{IS}$. □

Lemma 23. *Let $G = (V, E)$ be a graph containing a maximal independent set V' of size k. Let $v \in V$ be a vertex of degree d. Let $G^\star = (V^\star, E^\star)$ be G where v has been replaced in every edge containing v through one of d new vertexes $v_1, \dots, v_d$ such that $v_1, \dots, v_d$ have only one appearance in the description of $G^\star$. Then, $G^\star$ contains an independent set of size at least k.*

Proof. Since G has a maximal independent set V', either v or at least one of its neighbors is contained in V'. We make a case distinction on the degree of v and its membership in V'.

- If $d = 1$ and $v \in V'$, then the only appearance of v will be replaced by v_1 which also has a degree of 1. Therefore, $V'' = (V' \backslash \{v\}) \cup \{v_1\}$ is an independent set of $G^\star$ of size k.

- If $d = 1$, $v \notin V'$, and $\{u, v\} \in E$ is the only edge in which v appears. Then, v will be replaced by v_1 and $\{u, v\}$ will be replaced by $\{u, v_1\}$ such that v_1 has also a degree of 1. Since $v \notin V'$ and the degree of v is 1, we have $u \in V'$ and therefore $V'' = V'$ is an independent set of $G^\star$ of size k.

- If $d > 1$, $v \in V'$, and v is part of the edges $\{u_1, v\}, \dots \{u_d, v\} \in E$. Then, v will be replaced by $v_1, \dots, v_d$ with $v_i \neq v_j$ such that the edges containing v are replaced by $\{u_1, v_1\}, \{u_2, v_2\}, \dots, \{u_d, v_d\}$. Since $v \in V'$ we know that $u_1, \dots, u_d \notin V'$ and therefore $V'' = (V' \backslash \{v\}) \cup \{v_1, \dots, v_d\}$ is an independent set of $G^\star$ of size $k + (d - 1)$.

- If $d > 1$, $v \notin V'$, and v is part of the edges $\{u_1, v\}, \dots \{u_d, v\} \in E$. Then, v will be replaced by $v_1, \dots, v_d$ with $v_i \neq v_j$ such that the edges containing v are replaced by $\{u_1, v_1\}, \{u_2, v_2\}, \dots, \{u_d, v_d\}$. Since $v \notin V'$ the set $V'' = V'$ is an independent set of $G^\star$ of size k.

□

6.2.3 Decidability

We will now prove that it is decidable whether $L(\text{condensed}(A)) \cap \text{IS}$ is empty or not. This means that we can decide whether $L(\text{condensed}(A))$ contains a solvable Independent Set instance. Therefore, we will show that it is sufficient to consider only paths with a length up to $l = 3|Q|^5 + 3|Q|^4$ in the automaton condensed(A). First, we will elucidate the value of the bound l.

We have up to $|Q|^2$ different transition sets $\mathcal{K}$ with up to $|Q|$ different representatives all together. The up to $2|Q|^2$ transition sets Λ have sets of representatives with each up to $|Q|^2 + |Q|$ elements (in the infinite case and $|Q|^2$ elements in the finite case). The up to $|Q|^2$ unconnected vertex transition sets have sets of representatives with each up to $|Q|^2$ elements. If we want to read all the different above described representative on one single path, we might have to read up to $|Q|$ other characters in order to read one new representative.

Definition 56. A path containing all above described representatives might have to have a length of at least $l :=$

$$\begin{aligned} |Q| \cdot \big([|Q|^2 \cdot |Q|] + [(2|Q|^4 + 2|Q|^3)] + [|Q|^4]\big) \\ = |Q| \cdot \big(3|Q|^4 + 3|Q|^3\big) \\ = 3|Q|^5 + 3|Q|^4 \end{aligned}$$

in the automaton condensed$(A) = (Q, \Gamma', \delta', q_0, F)$ consisting of letters from Γ' (see Definition 55).

Lemma 24. *Let A be a deterministic finite automaton with $L(A) \subseteq Enc_{IS}$ and $w \in L(\text{condensed}(A))$ with $|w| > l$. If w is a solvable* Independent Set *instance, then there is a word $w' \in L(\text{condensed}(A))$ with $w' \leq l$ and w' is a solvable* Independent Set *instance.*

Proof. We just explained that l is an upper bound on the length of a path needed to read all representatives of all sets of representatives in one long word. If a word w is longer than this threshold l, then w must contain multiple copies of at least one sub-word which is already contained in w and does not bring new vertexes or edges to the graph. Therefore, the word w' without those redundant copies describes the same graph as w and so w' with $|w'| \leq l$ is a solvable IS instance if w is. □

This means that it is sufficient to consider only paths of length up to l in the automaton condensed(A).

Lemma 25. *Let $R \subseteq Enc_{IS}$ be a regular language accepted by the DFA A. It is decidable whether $L(\text{condensed}(A)) \cap$* Independent Set *$\neq \emptyset$.*

Proof. Lemma 24 states that, if there is a word in $L(\text{condensed}(A)) \cap$ Independent Set, there is also a word in $L(\text{condensed}(A)) \cap$ Independent Set with size at most l. Therefore, we can decide the emptiness of the intersection by enumerating all finitely many words of length up to l and test each of them for membership in IS. Since IS is in NP [GJ79] this can be done in finite time. If none of those words is contained in IS the inversion of Lemma 24 states that no other word in $L(\text{condensed}(A))$ is contained in IS. $\square$

We are now able to state the main result of this section.

Theorem 13. *Let R be a regular language. It is decidable whether $R \cap$* Independent Set $\neq \emptyset$*, i.e. the problem int_{Reg}*(Independent Set) *is decidable.*

Proof. Since R and Enc_{IS} are regular languages, we can assume w.l.o.g. that $R \subseteq \text{Enc}_{\text{IS}}$. Let A be a DFA recognizing R. The Definitions 41, 45, 47, and 45 to 55 describe constructively the construction of the automaton condensed(A). Therefore, condensed(A) can be computed by an algorithm on input A. Lemma 21 in conjunction with Lemma 22 show that $L(A) \cap \text{IS} \neq \emptyset \Leftrightarrow L(\text{condensed}(A)) \cap \text{IS} \neq \emptyset$. Therefore, we can decide the problem based on the automaton condensed(A). Finally, Lemma 25 states that $L(\text{condensed}(A)) \cap \text{IS} \neq \emptyset$ is decidable by checking finitely many words. This yields a deciding procedure for the problem int_{Reg}(Independent Set). $\square$

While we can decide the int_{Reg}-problem for Independent Set, we have no decidability results for the complementary problem Clique.

6.3 Replacing - Knapsack

Beside merging and separating, we want to present the technique of *replacing* in order to show decidability of the int_{Reg}-problem. The idea of *replacing* is to replace elements of a problem instance through the easiest elements we can read in this position. The difference to the previous techniques is that we do not focus on whether the replacing elements are identical or disjunct. We will demonstrate this approach by proving the decidability of the int_{Reg}-problem for Knapsack.

We begin by a formal definition of the NP-complete problem [GJ79].

Definition 57 (Knapsack).
Given: A set U of items with a weight $w(u) \in \mathbb{Z}^+$ and a value $v(u) \in \mathbb{Z}^+$ for every item $u \in U$, and positive integers W and V.
Question: Is there a subset $U' \subseteq U$ such that $\sum_{u \in U'} w(u) \leq W$ and $\sum_{u \in U'} v(u) \geq V$.

The given construction will also work for the problem Integer Knapsack [GJ79] where the question is whether there exists an assignment of non-negative integers $c(u)$ to the elements $u \in U$ with $\sum_{u\in U} c(u)w(u) \leq W$ and $\sum_{u\in U} c(u)v(u) \geq V$. For this problem it will even be enough to consider simple paths in the automaton condensed(A) defined in Definition 61.

We will encode the problem Knapsack in the following way. First, the weight bound W is encoded as a unary string of a's separated by a \$-sign from the following string of a's encoding the value bound V. Then, the items of U are listed, separated through \$ signs. An element u is represented by a string of a's encoding its weight $w(u)$, followed by # and a string of a's encoding its value $v(u)$. We will refer to the set of all correctly encoded Knapsack instances, regardless if they are solvable or not, as Enc_{KS}.

Definition 58. Every encoded Knapsack instance is part of the regular set Enc_{KS} defined through the regular expression

$$a^*\$a^*\$\,(a^*\#a^*\$)^* \enspace .$$

When we refer to the set Knapsack, we regard the problem encoded in the described way. Example 5 illustrates the given encoding.

Example 5. *Consider the following* Knapsack *instance.*

$$\begin{aligned}
&W := 15\\
&V := 32\\
&U := \{u_1, \ldots, u_5\}\\
&w(u_1) = 12,\; w(u_2) = 1,\; w(u_3) = 3,\; w(u_4) = 8,\; w(u_5) = 4\\
&v(u_1) = 2,\; v(u_2) = 5,\; v(u_3) = 17,\; v(u_4) = 8,\; v(u_5) = 2
\end{aligned}$$

The corresponding encoding $w \in Enc_{KS}$ is

$$w := a^{15}\$a^{32}\$a^{12}\#a^{2}\$a^{1}\#a^{5}\$a^{3}\#a^{17}\$a^{8}\#a^{8}\$a^{4}\#a^{2}\$.$$

6.3.1 Construction

Following the previous pattern, we will give a construction of the automaton condensed(A) based on a given DFA A. The automaton condensed(A) will have the property $L(\text{condensed}(A)) \cap \text{Knapsack} \neq \emptyset \Leftrightarrow L(A) \cap \text{Knapsack} \neq \emptyset$ and only finitely many words in condensed(A) have to be considered in order to find a word in $L(\text{condensed}(A)) \cap \text{Knapsack}$ if one exists. We will define transition sets for each of the sub-words W and V, as well as for the item values v and weights w.

Definition 59. Let $A = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) \subseteq \mathrm{Enc}_{\mathrm{KS}}$. For every pair of states $q_0, q \in Q$ consisting of the initial state q_0 and an arbitrary state q, we define the transition set for W as

$$\mathcal{W}_{q_0,q} := \{x \in L(a^*\$) \mid \delta^*(q_0, x) = q\}$$

consisting of every value for W which can be read in between the states q_0 and q. Further, for every pair of states $q, q' \in Q$ we define the following sets

$$\begin{aligned}
\mathcal{V}_{q,q'} &:= \{x \in L(a^*\$) \mid \exists q_1 \in Q\colon \delta(q_1, \$) = q \wedge \delta^*(q, x) = q'\}\,, \\
\Lambda^w_{q,q'} &:= \{x \in L(a^*\#) \mid \exists q_1 \in Q\colon \delta(q_1, \$) = q \wedge \delta^*(q, x) = q'\}\,, \\
\Lambda^v_{q,q'} &:= \{x \in L(a^*\$) \mid \exists q_1 \in Q\colon \delta(q_1, \#) = q \wedge \delta^*(q, x) = q'\}\,.
\end{aligned}$$

Where $\mathcal{V}_{q,q'}$ is the transition set for V consisting of every value for V which can be read in between q and q'. The transition sets $\Lambda^w_{q,q'}$ and $\Lambda^v_{q,q'}$ describe the sets of item weights and values which can be read in between the two states.

Next, we want to assign the transition sets with sets of representatives. Therefore, observe that a Knapsack instance is easier to solve, if the bound V is as low as possible, while W is large enough. If we are free to pick a Knapsack item from a set of items, we are looking for an element with the smallest possible weight and a value that is best above V. Therefore, we will pick the minimal elements of $\mathcal{V}_{q,q'}$ and $\Lambda^w_{q,q'}$ as representatives. Since the transition sets are defined by sub-automaton of A, there is a minimal element of size at most $|Q|$ in every transition set. So, we only have to reach a target value $V \leq |Q|$ with items with weights below $|Q|$. Even if those item have only an item value of 1, picking $|Q|$ of them will be enough to hit the target value. Therefore, a weight bound of $|Q|^2$ will be enough to allow the selection of enough items to hit the target value. Since we know that V is bounded by $|Q|$, any item value above $|Q|$ is enough to solve the Knapsack instance, if this item can be picked. These considerations lead us to the definition of representatives for the transition sets. In contrast to the previous constructions, the sets of representatives will all be singleton sets and hence we will define single representatives (instead of sets) for each transition set.

Definition 60. We assign representatives for the transition sets $\mathcal{W}_{q_0,q}$, $\mathcal{V}_{q,q'}$,

$\Lambda^w_{q,q'}$, and $\Lambda^v_{q,q'}$ for all $q, q' \in Q$ in the following way.

$$\begin{aligned}
\mathit{rep}(\mathcal{V}_{q,q'}) &:= \min_{\text{lex}}(\mathcal{V}_{q,q'}),\\
\mathit{rep}(\Lambda^w_{q,q'}) &:= \min_{\text{lex}}(\Lambda^w_{q,q'}),\\
\mathit{rep}(\mathcal{W}_{q_0,q}) &:= \begin{cases} \max_{\text{lex}}(\mathcal{W}_{q_0,q}) & \text{if } |\mathcal{W}_{q_0,q}| < \infty,\\ \min_{\text{lex}}(\{x \in \mathcal{W}_{q_0,q} \mid |x| > |Q|^2\}) & \text{otherwise,} \end{cases}\\
\mathit{rep}(\Lambda^v_{q,q'}) &:= \begin{cases} \max_{\text{lex}}(\Lambda^v_{q,q'}) & \text{if } |\Lambda^v_{q,q'}| < \infty,\\ \min_{\text{lex}}(\{x \in \Lambda^v_{q,q'} \mid |x| > |Q|\}) & \text{otherwise.} \end{cases}
\end{aligned}$$

Now we are able to build the automaton condensed(A) based on the assigned representatives.

Definition 61. Let $R \subseteq \mathrm{Enc}_{\mathrm{KS}}$ be a regular language and $A = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) = R$. We define the *condensed automaton* condensed(A) $= (Q, \Gamma', \delta', q_0, F)$ with

$$\begin{aligned}
\Gamma' = \bigcup_{q\in Q} \{\mathit{rep}(\mathcal{W}_{q_0,q})\} \cup \bigcup_{q,q'\in Q} \{\mathit{rep}(\mathcal{V}_{q,q'})\} \cup\\
\bigcup_{q,q'\in Q} \{\mathit{rep}(\Lambda^w_{q,q'})\} \cup \bigcup_{q,q'\in Q} \{\mathit{rep}(\Lambda^v_{q,q'})\}
\end{aligned}$$

$$\begin{aligned}
\text{and}\quad &\forall q \in Q : (q_0, x, q) \in \delta' \text{ if } x = \mathit{rep}(\mathcal{K}_{q_0,q})\\
&\forall q, q' \in Q : (q, x, q') \in \delta' \text{ if } x = \mathit{rep}(\mathcal{V}_{q,q'})\\
&\forall q, q' \in Q : (q, x, q') \in \delta' \text{ if } x = \mathit{rep}(\Lambda^w_{q,q'})\\
&\forall q, q' \in Q : (q, x, q') \in \delta' \text{ if } x = \mathit{rep}(\Lambda^v_{q,q'})\ .
\end{aligned}$$

6.3.2 Correctness

The automaton condensed(A), constructed from the DFA A, accepts a solvable KNAPSACK instance if and only if A does.

Lemma 26. *Let $R \subseteq Enc_{KS}$ be a regular language, defined through the deterministic finite automaton $A = (Q, \Gamma, \delta, q_0, F)$. If L(condensed(A)) $\cap$ KNAPSACK $\neq \emptyset$ then also $R \cap$ KNAPSACK $\neq \emptyset$.*

Proof. By construction, L(condensed(A)) is a subset of $L(A)$. Therefore, every solvable KNAPSACK instance in L(condensed(A)) is also in R. □

Lemma 27. *Let $R \subseteq Enc_{KS}$ be a regular language, defined through the deterministic finite automaton $A = (Q, \Gamma, \delta, q_0, F)$. If $R \cap$ KNAPSACK $\neq \emptyset$, then also L(condensed(A)) $\cap$ KNAPSACK $\neq \emptyset$.*

Proof. Again, we pick a word $x \in R \cap$ Knapsack and split it up into factors

$$x = x_W x_V x_{w_1} x_{v_1} x_{w_2} x_{v_2} \dots x_{w_n} x_{v_n}$$

where $x_W \in \mathcal{W}_{q_0,q_V}$, $x_V \in \mathcal{V}_{q_V,q_{w_1}}$, $x_{w_i} \in \Lambda^w_{q_{w_i},q_{v_i}}$, and $x_{v_i} \in \Lambda^v_{q_{v_i},q_{w_{i+1}}}$. We will replace the sub-words in x with sub-words $x_i{}'$ such that the word

$$x' = x_W{}' x_V{}' x_{w_1}{}' x_{v_1}{}' x_{w_2}{}' x_{v_2}{}' \dots x_{w_n}{}' x_{v_n}{}'$$

is in $L(\text{condensed}(A)) \cap$ Knapsack.

Let V be the value bound encoded in x_V. Let V' be the single value bound encoded in $\mathit{rep}(\mathcal{V}_{q_V,q_{w_1}})$, by definition $V' \leq V$, hence if there is a collection of items U' from x with a summed value above V, this sum is also above V'. So we set $x_V{}' := \mathit{rep}(\mathcal{V}_{q_V,q_{w_1}})$.

Let w_i encoded in x_{w_i} be the weight of the i-th element u_i from x. If $u_i \in U'$ then $(U' \backslash \{u_i\}) \cup \{u_i{}'\}$ with $u_i{}'$, defined through $v(u_i{}') := v(u_i)$, and $w(u_i{}') := \mathit{rep}(\Lambda^w_{q_{w_i},q_{v_i}})$ also fulfills the knapsack conditions. If $u_i \notin U'$ then U' remains unchanged by replacing w_i by $w_i{}'$ encoded in $\mathit{rep}(\Lambda^w_{q_{w_i},q_{v_i}})$. Therefore, we set $x'_{w_i} := \mathit{rep}(\Lambda^w_{q_{w_i},q_{v_i}})$.

Let v_i encoded in x_{v_i} be the value of the i-th element u_i from x and let $v_i{}'$ be the value of element $u_i{}'$ encoded in $\mathit{rep}(\Lambda^v_{q_{v_i},q_{w_{i+1}}})$. If $v_i > v_i{}'$ and $u_i \in U'$, then $U'' := (U' \backslash \{u_i\}) \cup \{u_i{}'\}$ is a solution of the knapsack instance x' when we set $x_{v_i{}'} := \mathit{rep}(\Lambda^v_{q_{v_i},q_{w_{i+1}}})$ since $v_i{}' > |Q|$ and $V' \leq |Q|$. If $u_i \notin U'$, then U' remains unchanged by replacing v_i by $v_i{}'$.

Let W be the weight bound encoded in x_W. We replaced V and every item weight w_i by a value smaller then $|Q|$. Since x has a solution U' we can assume w.l.o.g. that U' contains no element with a zero item value. Let W' be the weight bound encoded in $\mathit{rep}(\mathcal{W}_{q_0,q_V})$. If $W > W'$, then $W' > |Q|^2$ and hence large enough so that $\sum_{u \in U''} v(u) \geq V' \leq |Q|$ and $\sum_{u \in U''} w(u) \leq W'$ where U'' contains the replacements of up to $|Q|$ elements of U', which all have a non-zero value and a weight below $|Q|$. So, we set $x_W{}' := \mathit{rep}(\mathcal{W}_{q_0,q_V})$.

Therefore, we have found a word $x' \in L(\text{condensed}(A))$ which also encodes a solvable Knapsack instance. □

6.3.3 Decidability

We will now show that we can decide whether $L(\text{condensed}(A))$ contains a solvable Knapsack instance. Therefore, we will prove that we only have to consider finitely many words in $L(\text{condensed}(A))$.

Lemma 28. *Let A be a DFA with $L(A) \subseteq \mathit{Enc}_{KS}$ and $x \in L(\text{condensed}(A))$ with $|x| > |Q|^2$. If $x \in$ Knapsack, then there is a word $x' \in L(\text{condensed}(A)) \cap$ Knapsack with $x' \leq |Q|^2$.*

Proof. Since for every word in $L(\text{condensed}(A))$ the value bound V is at most $|Q|$ we have to pick at most $|Q|$ copies of an item to be able to reach V. In order to read an item twice, we might need to read up to $|Q|$ other letters until we reach the state again from which we can read the item. Hence, a path of length up to $|Q|^2$ is long enough to read up to $|Q|$ copies of any element, if possible. Adding more then $|Q|$ redundant items to the KNAPSACK instance in total does not help finding a solution. Therefore, if we set U'' to the first $|Q|$ elements of U' with a non-zero item value, removing all labels over loops in x which do not contain a word in U'' results in a KNAPSACK instance $x' \in L(\text{condensed}(A))$ with a solution U'' and $x' \leq |Q|^2$. □

This means that only words with a length up to $|Q|^2$ have to be considered in $L(\text{condensed}(A))$. For the problem INTEGER KNAPSACK we even have to consider only words with a length of up to $|Q|$ since we can pick one item multiple times.

Lemma 29. *Let $R \subseteq Enc_{KS}$ be a regular language accepted by the DFA A. It is decidable whether* $L(\text{condensed}(A)) \cap \text{KNAPSACK} \neq \emptyset$.

Proof. Lemma 28 states that, if there is a solvable KNAPSACK instance in $L(\text{condensed}(A))$, there is one with a size up to $|Q|^2$. Therefore, we can enumerate the finitely many words with a size below that threshold under which there is a word $w \in \text{KNAPSACK}$ if there is one in $L(A)$. Since the problem KNAPSACK is in NP [GJ79], we can test for every word, whether it is contained in KNAPSACK in finite time. Therefore, we have found a deciding procedure for the question whether $L(\text{condensed}(A)) \cap \text{KNAPSACK} \neq \emptyset$. □

This leads us to the main result of this section.

Theorem 14. *Let R be a regular language. It is decidable whether $R \cap$ KNAPSACK $\neq \emptyset$, i.e. the problem* $int_{\text{Reg}}(\text{KNAPSACK})$ *is decidable.*

Proof. W.l.o.g. we may assume that $R \subseteq \text{Enc}_{\text{KS}}$ since both language are regular. Let A be a DFA recognizing A. There is an algorithm which given the DFA A computes the finite automaton condensed(A) as described by the Definitions 59 to 61. Lemma 26 and 27 together state that $L(A) \cap \text{KNAPSACK} \neq \emptyset \Leftrightarrow L(\text{condensed}(A)) \cap \text{KNAPSACK} \neq \emptyset$. The question whether $L(\text{condensed}(A)) \cap \text{KNAPSACK} \neq \emptyset$ is decidable following Lemma 29. Hence, we can decide whether $L(A) \cap \text{KNAPSACK} \neq \emptyset$ by constructing condensed(A) and test whether $L(\text{condensed}(A)) \cap \text{KNAPSACK} \neq \emptyset$. □

Corollary 2. *Let R be a regular language. It is decidable whether $R \cap$ INTEGER KNAPSACK $\neq \emptyset$, i.e. the problem* $int_{\text{Reg}}(\text{INTEGER KNAPSACK})$ *is decidable.*

Proof. The same construction works for the problem of INTEGER KNAPSACK, where items can be picked multiple times. □

Chapter 7

Discussion

We have proven that the listed problems on the left side of Table 7.1 have a decidable regular intersection emptiness problem, while the problems on the right side have an undecidable int_{Reg}-problem.

Table 7.1: List of problems with a decidable, respectively undecidable int_{Reg}-problem.

Decidable int_{Reg}	Undecidable int_{Reg}
UNARY-SHUFFLED$\equiv_{\text{STRING}^\epsilon}$	SHUFFLED$\equiv_{\text{REGEX}}$
SEQUENTIAL$\equiv_{\text{STRING}^\epsilon}$	SHUFFLED$\equiv_{\text{STRING}^\epsilon}$
UNARY-SEQUENTIAL$\equiv_{\text{STRING}^\epsilon}$	MACHINE LANGUAGE
SAT	BOUNDED TILING
k-TQBF	CORRIDOR TILING
TRUE QUANTIFIED BOOLEAN FORMULA	BOUNDED PCP
INTEGER LINEAR PROGRAMMING	
VERTEX COVER	
INDEPENDENT SET	
KNAPSACK	
INTEGER KNAPSACK	

Table 7.2 orders the problems from Table 7.1 according to their complexity classes.

Table 7.2: Problems with a proven decidable (undecidable) int_{Reg}-problem, ordered by their complexity.

Complexity-Class	Problem	int_{Reg}
L	SHUFFLED$\equiv_{\text{STRING}^\epsilon}$	undecidable
	UNARY-SHUFFLED$\equiv_{\text{STRING}^\epsilon}$	decidable
	SEQUENTIAL$\equiv_{\text{STRING}^\epsilon}$	decidable
	UNARY-SEQUENTIAL$\equiv_{\text{STRING}^\epsilon}$	decidable
NL	MACHINE LANGUAGE FOR NL	undecidable
NP	MACHINE LANGUAGE FOR NP	undecidable
	BOUNDED TILING	undecidable
	BOUNDED PCP	undecidable
	SAT	decidable
	INTEGER LINEAR PROGRAMMING	decidable
	VERTEX COVER	decidable
	INDEPENDENT SET	decidable
	KNAPSACK	decidable
	INTEGER KNAPSACK	decidable
Polynomial Hierarchy	k-TQBF	decidable
PSPACE	MACHINE LANGUAGE FOR PSPACE	undecidable
	CORRIDOR TILING	undecidable
	SHUFFLED$\equiv_{\text{REGEX}}$	undecidable
	TRUE QUANTIFIED BOOLEAN FORMULA	decidable

To prove undecidability of the int_{Reg}-problem for variations of the MACHINE LANGUAGE, for BOUNDED TILING, and for BOUNDED PCP we used the fact that all of these problems are restricted versions of undecidable problems. The problems are made decidable by adding an artificial limit to the input, which restricts the possibly infinite set of potential solutions to a finite set. In order to show int_{Reg}-undecidability, we created a regular language R based on a fixed instance of the undecidable problem variant, containing the instance together with every possible limit-value. If R intersected with the restricted problem has a nonempty intersection, we have found a limit to the solution and hence the instance is in the undecidable problem variant. Therefore, we have found a reduction from the undecidable problem variant to the int_{Reg}-problem of the restricted decidable problem variant by constructing a regular language which contains every limit-value.

In the case of SHUFFLED$\equiv_{\text{REGEX}}$ we have used the semantics of the empty word symbol ϵ to construct regular expression which describe possible solutions of a PCP instance. We used ϵ as a padding character in a shuffled encoding of words created from the elements of the two PCP lists. The evaluation of the regular expressions naturally removed the ϵ's from these words, such that two regular expressions described the same language if and only if their language encoded a solution of the PCP instance. Since we didn't make use of the union and star operation of regular expression, the same proof also worked for string equivalence modulo a padding character, described in the problem SHUFFLED$\equiv_{\text{STRING}^\epsilon}$. We have seen that the undecidability of this problem strongly relied on the shuffled encoding and depends on the size of the alphabet over which the strings are build. We showed that the problem becomes decidable for an un-shuffled encoding as well as for an alphabet of size one (plus padding symbol).

In chapter 4 we presented decidability proofs for the int_{Reg}-problem of SAT, $L_{k\text{-TQBF}}$ (which is the problem of TRUE QUANTIFIED BOOLEAN FORMULA with up to k alternating quantifier blocks), and TQBF which has been published in the article *Deciding regular intersection emptiness of complete problems for PSPACE and the polynomial hierarchy* [GKLW18] by Demen Güler, Andreas Krebs, Klaus-Jörn Lange, and Petra Wolf. In the proof an automaton condensed(A) is constructed from a given DFA A, such that only finitely many words in L(condensed(A)) have to be considered in order to find a true formula in $L(A)$. In a way L(condensed(A)) contained the finitely many *most promising* words in $L(A)$ under which a true formula is contained if and only there is one in the whole language $L(A)$. To construct condensed(A) we defined so-called *transition sets* for every pair of states of A which described the set of all literals which can be read in-between those two states. Then, we picked for every transition set finitely many representatives such that existentially quantified variables are preferred to occur uniquely in the formula, while references to universally quantified variable are preferred to point to the same variable. Finally, we constructed the automaton condensed(A) based on the finitely many representatives for all finitely many transition sets.

We adapted this construction to prove decidability of the int_{Reg}-problem for INTEGER LINEAR PROGRAMMING in chapter 5. Here, we iterated the procedure of finding representative twice. We first considered transition sets for coefficients and assigned representatives to them. Then, we regarded transition sets for whole inequations and again assigned representatives. Based on the finitely many representatives for inequations, we constructed the automaton condensed(A) and showed that we only have to consider finitely many words in L(condensed(A)) under which there is a solvable ILP if and only if there is one in the original automaton.

In chapter 6 we elaborated the three techniques *merging*, *separating*, and *replacing* which has been used in the previous constructions to assign representatives. In the *merging* technique one representative for every combination of transition sets sharing at least one element is chosen. So, wherever the same problem-item can be read in the given automaton, there will be a word in $L(\text{condensed}(A))$ with the same reference in all of these positions. The technique was demonstrated on the problem VERTEX COVER. It pointed out, that under the chosen encoding, a VERTEX COVER instance becomes easier to solve, if vertex references point to the same vertex, resulting in merged edges.

The representatives in the *separating* techniques have been chosen in a way that problem items occur at one position in the instance only. The technique was demonstrated on the problem INDEPENDENT SET where separating vertex references led to disconnected edges. A separated vertex reference replaced the original vertex by a number of copies corresponding to the old degree of the vertex. Every vertex copy appeared in one edge previously connected to the old vertex. Therefore, one vertex with n edges was replaced by n vertexes of degree one. Hence, separating vertexes made an INDEPENDENT SET instance easier to be solved.

In the *replacing* techniques, the problem items were replaced by items which made the instance predictably easier. Here, it did not matter whether several transition sets shared representatives or not, so the replacement of an item was independent from the other items of the instance. A construction for the problem KNAPSACK was provided as an example for this technique. A KNAPSACK instance is easier to be solved if it has a preferably huge weight-bound and huge item-values while the value-bound and the item-weights should be low. Altering those values in the desired direction made the KNAPSACK instance predictably easier.

The three presented techniques give an insight in the nature of NP-complete problems in a way that some problems are getting *predictably* easier if we have the possibility to exchange items of the problem. Problems like VERTEX COVER are getting easier if we merge item references to the same problem item, while problems like INDEPENDENT SET are getting easier if we separate the item references. While we can predict how the solvability of instances of INDEPENDENT SET changes by replacing items, we have no results for the complementary problem of CLIQUE. We expect a decidability result of its int_{Reg}-problem by using the *merging* techniques but the construction does not seem to be easy. The problem KNAPSACK forms a third group of problems where items can be replaced by predictably easier items independent of other problem items. The presented problems with an undecidable int_{Reg}-problem form a fourth group of problems, where replacing problem items does not simplify the problem in a predictable way. This means for example, that we can not exchange tiles in BOUNDED TILING or list elements in BOUNDED PCP in a constructive way to make the problem easier.

The described four groups of NP-complete problems and their corresponding techniques seem to give an interesting insight in the different nature of NP-complete problems. Examining more NP-complete problem in order to prove or disprove the decidability of their int_{Reg}-problem could lead to an interesting segmentation of the class of NP-complete problems. It could also lead to a finer differentiation of this class along the four groups of problems. Finding more techniques to prove decidability or undecidability of the int_{Reg}-problem would also help understanding the different nature of NP-complete problems, while they are all similar regarding their complexity.

The described techniques can also help in the task of finding a solvable problem instance in a given infinite set of instances when there is some structure in the set. We suggest that our techniques are not limited to sets of regular languages. For instance our constructions could also work for non-regular transition sets. For example if the transition sets for a language of KNAPSACK instances are not regular, we should still be able to find finite sets of representatives as long as the transition sets themselves are connected in some "regular way".

An other task for further research is to develop more techniques to prove undecidability of the int_{Reg}-problem. So far, the only undecidability result for int_{Reg} for "non-artificial" problems is obtained by using a shuffled encoding of the equivalence problem for regular expressions. It would be desirable to have undecidability results for sequentially encoded "natural" problems. This circumstance brings us to the question of encoding. A threshold of decidability seems to be whether the problem is encoded sequentially or shuffled. This aspect could be probed for the problems with a proven decidable int_{Reg}-problem in a sequential encoding. For some int_{Reg}-problems, an other threshold of decidability is the size of the alphabet as it is also the case for issues in other areas like the equivalence of two context-free languages given by grammars. While the equivalence of context-free languages in general is undecidable, every CFL over an unary alphabet is already regular and hence the equivalence problem for those kinds of CFL's is decidable [HU69]. Finally, we hope that the investigation of the decidability of the int_{Reg}-problem will lead to a generic criteria of problems which determines the decidability of the corresponding int_{Reg}-problem. This criteria might as well define a structure on the set of NP-complete languages graduating this class into subclasses.

Bibliography

[CFK$^+$15] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*, volume 3. Springer, 2015.

[Dam82] Werner Damm. The IO-and OI-Hierarchies. *Theoretical Computer Science*, 20(2):95–207, 1982.

[GJ79] Michael R Garey and David S Johnson. *Computers and Intractability.* wh freeman New York, 1979.

[GKLW18] Demen Güler, Andreas Krebs, Klaus-Jörn Lange, and Petra Wolf. Deciding Regular Intersection Emptiness of Complete Problems for PSPACE and the Polynomial Hierarchy. In *International Conference on Language and Automata Theory and Applications*, pages 156–168. Springer, 2018.

[Gre68] Sheila Greibach. A Note on Undecidable Properties of Formal Languages. *Mathematical systems theory*, 2(1):1–6, 1968.

[Gü19] Güler, Demen. *What is (not) a Family of Formal Languages?* PhD thesis, Eberhard Karls Universität Tübingen, 2019. Forthcoming.

[HIRS76] Harry B Hunt III, Daniel J Rosenkrantz, and Thomas G Szymanski. On the Equivalence, Containment, and Covering Problems for the Regular and Context-Free Languages. *Journal of Computer and System Sciences*, 12(2):222–268, 1976.

[HU69] John E Hopcroft and Jeffrey D Ullman. Formal Languages and their Relation to Automata. 1969.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory Languages and Computation.* Addison-Wesley Publishing Company, Inc., 1979.

[KL12] Andreas Krebs and Klaus-Jörn Lange. Dense Completeness. In *Developments in Language Theory*, pages 178–189. Springer, 2012.

[KLL15] Andreas Krebs, Klaus-Jörn Lange, and Michael Ludwig. On Distinguishing NC^1 and NL. In *International Conference on Developments in Language Theory*, pages 340–351. Springer, 2015.

[Lan96] Klaus-Jörn Lange. Complexity and Structure in Formal Language Theory. *Fundamenta Informaticae*, 25(3, 4):327–352, 1996.

[Pap03] Christos H Papadimitriou. *Computational Complexity.* John Wiley and Sons Ltd., 2003.

[Pin10] Jean-Éric Pin. Mathematical Foundations of Automata Theory. *Lecture notes LIAFA, Université Paris*, 7, 2010.

[RW03] Piotr Rudnicki and Gerhard J Woeginger. The Post Correspondence Problem over a Unary Alphabet. *Applied mathematics letters*, 16(5):723–727, 2003.

[SM73] Larry J Stockmeyer and Albert R Meyer. Word Problems Requiring Exponential Time (Preliminary Report). In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 1–9. ACM, 1973.

[Sto76] Larry J Stockmeyer. The Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[Str] Howard Straubing. Tiling Problems. `http://www.cs.bc.edu/~straubin/cs385-07/tiling`. Accessed: 2018-09-03.

[vEB$^+$97] Peter van Emde Boas et al. The Convenience of Tilings. *Lecture Notes in Pure and Applied Mathematics*, pages 331–363, 1997.

[Wra76] Celia Wrathall. Complete Sets and the Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.

[WW89] Klaus Wagner and Gerd Wechsung. Computational Complexity. 1989.

Selbständigkeitserklärung