Inclusion of Pattern Languages and Related Problems

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich 12
der Johann Wolfgang Goethe-Universität
in Frankfurt am Main

von
Dominik D. Freydenberger
aus Regensburg

Frankfurt 2011
(D 30)

vom Fachbereich Informatik und Mathematik (12)
der Johann Wolfgang Goethe-Universität als Dissertation angenommen.

Dekan:	Prof. Dr. Tobias Weth
Gutachter:	Prof. Dr. Nicole Schweikardt Prof. Dr. Detlef Wotschke Dr. Daniel Reidenbach
Datum der Disputation:	27. Juni 2011

Zusammenfassung

Pattern und ihre Sprachen

Pattern, d. h. endliche Wörter aus Variablen und Terminalsymbolen, stellen eine kompakte, elegante und natürliche Methode dar, gewisse kontextsensitive Sprachen zu repräsentieren. Ein Pattern erzeugt ein Wort durch eine Substitution, die alle Variablen im Pattern durch beliebige endliche Wörter über einem festen Terminalalphabet ersetzt. Die *Patternsprache* eines Pattern ist somit die Menge aller Wörter, die durch die Substitution des Pattern gebildet werden können; etwas formaler ist eine Patternsprache also die Menge aller Bilder des Pattern unter beliebigen terminalerhaltenden Homomorphismen. Wenn wir beispielsweise das Pattern $\alpha{:=}x_1\,\mathtt{a}\,x_2\,\mathtt{b}\,x_1$ (mit Variablen x_1, x_2 und Terminalsymbolen $\mathtt{a}, \mathtt{b}$) betrachten, liegen folglich (unter anderem) die Wörter $w_1{:=}\mathtt{aabbba}$, $w_2{:=}\mathtt{abababab}$ und $w_3{:=}\mathtt{aaabaa}$ in der von α erzeugten Patternsprache $L(\alpha)$, wogegen die Beispielwörter $w_4{:=}\mathtt{ba}$, $w_5{:=}\mathtt{babbba}$ und $w_6{:=}\mathtt{abba}$ nicht von α erzeugt werden können.

Die Untersuchung von Pattern in Wörtern lässt sich bis zu Thue [103] zurückverfolgen und zählt mittlerweile zu den etablierten Teilgebieten der Wortkombinatorik (s. Cassaigne [17]), während die explizite Verwendung von Pattern als Sprachgeneratoren von Angluin [4] eingeführt wurden. In Angluins Definition ist es nicht erlaubt, Variablen löschend zu ersetzen, daher wird diese Klasse von Patternsprachen in der Literatur im Allgemeinen als *NE-Patternsprachen* bezeichnet („NE“ steht in diesem Fall für „nonerasing“). Ebenfalls häufig betrachtet werden die von Shinohara [102] eingeführten sogenannten „E-Patternsprachen“ („E“ wie „erasing“ oder „extended“); bei dieser Sprachklasse ist die löschende Substitution gestattet. Im obigen Beispiel ist somit das Wort w_3 zwar in der E-Patternsprache, nicht jedoch in der NE-Patternsprache von α enthalten.

Durch diesen – scheinbar kleinen – definitorischen Unterschied besitzen die beiden Sprachklassen sehr unterschiedliche Eigenschaften. Beispielsweise lässt sich das Äquivalenzproblem (d. h. die Frage, ob zwei Pattern die gleiche Sprache erzeugen) für *NE*-Patternsprachen trivial in Polynomialzeit entscheiden, während die Entscheidbarkeit des Äquivalenzproblems für *E*-Patternsprachen ein schweres und seit rund 30 Jahren offenes Problem darstellt (s. Ohlebusch und Ukkonen [79], Reidenbach [87]).

Beide Sprachklassen wurden zuerst in der Induktiven Inferenz, einem Teilgebiet der (mathematischen) Lerntheorie, eingeführt. Ziel dieser Betrachtungen war es, zu gegebenen Mengen von Wörtern effektiv Pattern zu finden, die diese Wörter beschreiben. Inzwischen wurden Patternsprachen auch in der Theorie der formalen Sprachen ausgiebig untersucht. Wegen ihrer einfachen Definition kommen Pattern und ihre Sprachen in einer Vielzahl anderer Gebiete der Informatik und der diskreten Mathematik vor.

Aufgrund der einfachen Definition von Patternsprachen ließe sich vermuten, dass sich die meisten interessanten Eigenschaften einer Patternsprache direkt aus dem sie erzeugenden Wort ablesen lassen. Insbesondere sollte sich daher leicht entscheiden lassen, ob zwei Pattern die gleiche oder vergleichbare Sprachen erzeugen (d. h. ob eine Äquivalenz- oder Inklusionsbeziehung vorliegt), und ob ein Wort in der Sprache eines Patterns ist (das sogenannte „Wortproblem“). Tatsächlich ist diese Vermutung aber in den meisten Fällen ein Trugschluss – beispielsweise ist das Inklusionsproblem im Allgemeinen unentscheidbar, und das Wortproblem NP-vollständig.

Die vorliegende Dissertation beschäftigt sich mit verschiedenen Aspekten des Inklu-

sionsproblems. Den eigentlichen Hauptteil der Arbeit bilden die Kapitel 3 bis 7, die sich sich grob in zwei Teile unterteilen lassen.

Der erste Teil, der aus den Kapiteln 3 und 4 besteht, befasst sich direkt mit dem Inklusionsproblem für Patternsprachen und wendet eine in diesem Kontext entwickelte Technik auf eine in der Praxis weit verbreitete Erweiterung der regulären Ausdrücke an.

Der zweite Teil, bestehend aus den Kapiteln 5 bis 7, untersucht Fragestellungen zu sogenannten *deskriptive Pattern*, die aufgrund ihrer Definition eng mit dem Inklusionsproblem verwandt sind.

Im Folgenden wird der Inhalt der einzelnen Kapitel genauer vorgestellt und jeweils explizit erwähnt, in welcher Form die genannten Resultate bereits veröffentlicht wurden.

Kapitel 3

Die Entscheidbarkeit des Inklusionsproblems für Patternsprachen wurde bereits bei der Einführung der NE-Patternsprachen durch Angluin [4] als offenes Problem benannt. Sowohl der NE- als auch der E-Fall dieses Problems erwiesen sich als überraschend schwer zu lösen, bis schließlich Jiang et al. [51] der Beweis für die Unentscheidbarkeit beider Fälle gelang.

Allerdings basiert der von Jiang et al. angegebene Beweis auf der Annahme, dass die darin verwendeten Patternsprachen auf Terminalalphabeten von unbeschränkter Größe definiert werden können. Im Gegensatz dazu verwenden aber die meisten Anwendungen von Patternsprachen Alphabete von beschränkter Größe. Trotz des wegweisenden Resultats von Jiang et al. blieb das Inklusionsproblem für Patternsprachen über spezifischen Alphabeten offen.

Das erste Hauptresultat der vorliegenden Arbeit, Theorem 3.3, zeigt, dass das Inklusionproblem für E-Patternsprachen über endlichen Terminalalphabeten (mit mindestens zwei Buchstaben) unentscheidbar ist. Mittels einer Reduktion von Jiang et al. folgt hieraus außerdem die Unentscheidbarkeit des Inklusionsproblems für NE-Patternsprachen über endlichen Alphabeten mit mindestens vier Buchstaben.

Theorem 3.3 beantwortet die entsprechenden offenen Fragen von Reidenbach [85] und Salomaa [99] und wurde bereits von Barceló et al. [8] in einer Arbeit aus der Datenbanktheorie angewendet.

Im restlichen Teil von Kapitel 3 wird Entscheidbarkeit der Inklusion für stärker eingeschränkte Klassen von Patternsprachen untersucht. Sowohl der ursprüngliche Beweis von Jiang et al., als auch der darauf aufbauende Beweis von Theorem 3.10 setzen voraus, dass die verwendeten Pattern unbeschränkt viele verschiedene Variablen enthalten dürfen.

Daher ist es naheliegend, als weitere Einschränkung die Zahl der in den Pattern vorkommenden Variablen zu betrachten. Theorem 3.10 zeigt durch eine Verfeinerung des Beweises von Theorem 3.3, dass das Inklusionsproblem für E-Patternsprachen mit einer beschränkten (wenn auch vergleichsweise hohen) Zahl von Variablen weiterhin unentscheidbar ist.

Um für stärker eingeschränkte Variablenzahlen interessante Resultate unterhalb der Unentscheidbarkeit zu finden, wird (analog zu einer Technik aus der Untersuchung kleiner Turingmaschinen) außerdem ein weiteres Berechnungsmodell auf die Inklusion von Patternsprachen reduziert. Hierzu wird die Iteration der *Collatzfunktion* $\mathcal{C}$ (s. Lagari-

as [62, 63]) betrachtet[1]. Die *Collatzvermutung* besagt, dass wiederholte Iteration von $\mathcal{C}$ stets zum Zyklus 4, 2, 1 führt.

Hier stellt sich heraus (Theorem 3.11), dass bereits eine verhältnismäßig geringe Zahl von Variablen genügt, um die Iteration der Collatzfunktion in Patternsprachen zu codieren. Weiterhin zeigt Theorem 3.12, dass sich die Frage nach der Existenz von Zahlen, die die Iteration der Collatzfunktion in einen anderen Zyklus als 4, 2, 1 führen, konstruktiv auf das Inklusionsproblem für Patternsprachen mit einer geringfügig größeren Variablenzahl reduzieren lässt. Ein entsprechender Entscheidungsalgorithmus könnte also in endlicher Zeit entweder die Collatzvermutung widerlegen, oder aber die Nichtexistenz einer von zwei möglichen Klassen von Gegenbeispielen beweisen.

Auch wenn aus diesen Resultaten keine Unentscheidbarkeit der Inklusionsprobleme von Patternsprachen mit den entsprechenden Variablenzahlen folgt, zeigen die Theoreme 3.11 und 3.12, dass die entsprechenden Probleme zumindest schwer sind.

Die Inhalte dieses Kapitels wurden bereits in den Arbeiten [32] (gemeinsam mit Daniel Reidenbach, zuerst erschienen als [30]) und [12] (gemeinsam mit Joachim Bremer, ausgezeichnet mit dem „Best Paper Award“ der Konferenz DLT 2010) veröffentlicht.

Kapitel 4

Reguläre Ausdrücke zählen zu den am weitesten verbreiteten Beschreibungsmechanismen und werden sowohl in der theoretischen, als auch in der praktischen Informatik auf verschiedenste Arten angewendet. Allerdings haben sich im Lauf der Jahrzehnte in Theorie und Anwendung zwei unterschiedliche Interpretationen dieses Konzepts entwickelt.

Während die Theorie weitestgehend der klassischen Definition folgt und reguläre Ausdrücke betrachtet, die exakt die Klasse der regulären Sprachen beschreiben, erlauben die meisten modernen Implementierungen von regulären Ausdrücken die Spezifikation von Wiederholungen mittels *Variablen* (auch *Rückreferenzen* genannt). Die daraus resultierenden *erweiterten regulären Ausdrücke* können durch diese Wiederholungen auch nichtreguläre Sprachen beschreiben.

Beispielsweise erzeugt der erweiterte reguläre Ausdruck $((\mathtt{a} \mid \mathtt{b})^*)\%x\ x$ die nichtreguläre Sprache $\{ww \mid w \in \{\mathtt{a},\mathtt{b}\}^*\}$. Hierbei kann der Teilausdruck $((\mathtt{a} \mid \mathtt{b})^*)\%x$ ein beliebiges Wort $w \in \{\mathtt{a},\mathtt{b}\}^*$ erzeugen, während gleichzeitig dieses Wort w der Variablen x als Wert zugewiesen wird. Weitere Vorkommen von x erzeugen wiederum exakt das gleiche Wort w.

Andererseits führt die Verwendung von Variablen nicht zwangsläufig zu Nichtregularität; beispielsweise erzeugen (für $n \geq 1$) Ausdrücke der Form

$$\alpha_n := (\underbrace{(\mathtt{a} \mid \mathtt{b}) \ldots (\mathtt{a} \mid \mathtt{b})}_{n \text{ mal } (\mathtt{a} \mid \mathtt{b})})\%x\ x$$

die endliche (und daher reguläre Sprache) aller Wörter $ww \in \{\mathtt{a},\mathtt{b}\}^*$, die die Länge $2n$ haben. Hierbei fällt auf, dass *klassische reguläre Ausdrücke* (d. h. Ausdrücke, die keine Variablen enthalten) für diese Sprachen exponentiell länger sind als die Ausdrücke α_n.

Kapitel 4 befasst sich hauptsächlich mit den folgenden zwei Fragen: Erstens, können erweiterte reguläre Ausdrücke – in Hinsicht auf ihre Länge oder auf ihre Variablenzahl

[1] Die Funktion $\mathcal{C}$ ist definiert durch $\mathcal{C}(n) := 3n + 1$ für ungerade n, und $\mathcal{C}(n) := \frac{1}{2}n$ für gerade n.

– effektiv minimiert werden? Und zweitens, um wie viel kompakter ist die Beschreibung von regulären Sprachen durch erweiterte reguläre Ausdrücke im Vergleich zu klassischen regulären Ausdrücken?

Die Klasse der Patternsprachen ist eine Teilklasse der von erweiterten regulären Ausdrücken erzeugten Sprachen, da sich die erzeugenden Pattern ohne großen Aufwand direkt in die entsprechenden Ausdrücke konvertieren lassen. Allerdings erhöht die Verwendung des Alternierungsoperators | die Ausdruckskraft so sehr, dass deutlich schärfere Nichtentscheidbarkeitsresultate als die in Kapitel 3 enthaltenen Resultate zur Inklusion für Patternsprachen erzielt werden können.

Die Konstruktion zum Beweis von Theorem 3.10 lässt sich mit signifikantem Zusatzaufwand zu einer mächtigeren Konstruktion für erweiterte reguläre Ausdrücke umbauen (Theorem 4.14), mit deren Hilfe in Theorem 4.15 mehrere Nichtentscheidbarkeitsresultate gewonnen werden, mittels derer die beiden weiter oben genannten Fragen beantwortet werden. Erweiterte reguläre Ausdrücke können nicht effektiv minimiert werden, und der Tradeoff zwischen erweiterten und klassischen regulären Ausdrücken ist durch keine berechenbare Funktion beschränkt. Dies beantwortet auch eine offene Frage von Bordihn et al. [9] zu einem verwandten Sprachmodell.

Besondere Erwähnung verdient hierbei die Tatsache, dass die genannten negativen Eigenschaften bereits bei erweiterten regulären Ausdrücken mit einer einzigen Variable nachgewiesen werden.

Als praktische Konsequenz lässt sich feststellen, dass selbst die Verwendung einer einzigen Variable zwar deutlich kompaktere Ausdrücke erlauben kann (was sich natürlich auch positiv auf die Geschwindigkeit des Matchens der Ausdrücke auswirkt), dass aber andererseits dieser Vorteil aufgrund der Unmöglichkeit einer effektiven (oder gar effizienten) Minimierung nicht generell nutzbar ist.

Die Inhalte dieses Kapitels wurden bereits in der Arbeit [29] veröffentlicht.

Kapitel 5

Ein Pattern α ist *konsistent* mit einer Menge $S \subseteq \Sigma^*$, wenn jedes Wort von S in der von α erzeugten Sprache $L(\alpha)$ enthalten ist[2]; d. h. wenn $L(\alpha) \supseteq S$ gilt. So sind beispielsweise die Pattern $\alpha_0{:=}x$, $\alpha_1{:=}xyxyx$ und $\alpha_2{:=}x\,\mathtt{a}\,\mathtt{b}\,y$ konsistent mit der Menge $S_0{:=}\{\mathtt{ababa}, \mathtt{ababbababbab}, \mathtt{babab}\}$. Konsistente Pattern liefern also eine kompakte und leicht verständliche Darstellung von Gemeinsamkeiten der Wörter einer Menge.

Wie das oben stehende Beispiel zeigt, gibt es zu einer Menge von Wörtern im Allgemeinen eine Vielzahl von konsistenten Pattern, die intuitiv eine sehr unterschiedliche Güte haben können; beispielsweise ist das Pattern α_0 mit jeder Sprache konsistent und dürfte daher für nahezu alle Anwendungen im Allgemeinen als eine triviale und nicht sonderlich hilfreiche Approximation gelten.

Es ist daher vorteilhaft, konsistente Pattern hoher Qualität formal zu fassen. Ein in der Literatur häufig verwendetes Qualitätsmaß ist das Konzept der *deskriptiven* Pattern für eine Menge S. Ein Pattern δ dessen Sprache $L(\delta)$ in einer gegebenen Klasse P von Patternsprachen liegt, ist P-deskriptiv für eine Sprache S, wenn δ mit S konsistent ist, und außerdem $L(\delta) \supset L(\gamma) \supseteq S$ für kein Pattern γ mit $L(\gamma) \in P$ gilt.

[2]Hierbei ist natürlich von Fall zu Fall festzulegen, ob E- oder NE-Patternsprachen betrachtet werden.

Anschaulich formuliert können P-deskriptive Pattern als Erzeuger einer kleinsten innerhalb der Klasse P möglichen Generalisierungen der Zielsprachen L verstanden werden. Die vorliegende Arbeit betrachtet hierbei vor allem die Klassen der NE-Patternsprachen, der E-Patternsprachen, und die Klasse der *terminalfreien E-Patternsprachen* (also derjenigen E-Patternsprachen, die von Pattern erzeugt werden, die keine Terminalsymbole enthalten). In den ersten beiden Fällen sprechen wir gewöhnlich von *NE-deskriptiven* beziehungsweise *E-deskriptiven* Pattern.

Die wahrscheinlich naheliegendste Frage bei der Suche nach deskriptiven Pattern für beliebige Sprachen ist, ob zu jeder Sprache mindestens ein deskriptives Pattern existiert. Für NE-Patternsprachen wurde diese Frage bereits von Angluin [4] positiv beantwortet. Ebenso wurde von Jiang et al. [50] bewiesen, dass alle endlichen Sprachen ein E-deskriptives Pattern besitzen, während der Fall von E-deskriptiven Pattern für unendliche Sprachen offen blieb.

Das Hauptresultat dieses Kapitels, Theorem 5.18, besagt, dass zu jedem Terminalalphabet mit mindestens zwei Buchstaben eine Sprache existiert, für die kein Pattern E-deskriptiv ist. Um die durch die Nichtentscheidbarkeit des Inklusionsproblems entstehenden Schwierigkeiten zu umgehen, verwendet der Beweis terminalfreie E-Patternsprachen, da für diese Unterklasse die Inklusion durch ein einfaches syntaktisches Kriterium charakterisiert wird.

Die meisten der Inhalte dieses Kapitels wurden bereits in der Arbeit [33] (gemeinsam mit Daniel Reidenbach, zuerst erschienen als [31]) veröffentlicht.

Kapitel 6

Im Gegensatz zu Kapitel 5, in dem die Existenz deskriptiver Pattern im Vordergrund steht, befasst sich dieses Kapitel mit der Frage nach der Möglichkeit, deskriptive Pattern effektiv zu finden.

Hierzu wird das Konzept der *deskriptiven Generalisierung (anhand von positiven Daten)* eingeführt (kurz: das DG-Modell), das an Golds klassisches Modell (s. Gold [39]) der *Identifikation von Sprachen im Limes (anhand von positiven Daten)*, das LIM-TEXT-Modell angelehnt ist. Das LIM-TEXT-Modell wurde in der einschlägigen Literatur intensiv untersucht (s. Ng und Shinohara [76]).

Anschaulich (und vereinfacht) untersuchen wir die Existenz von Lernstrategien, die Positivbeispiele von Sprachen L wortweise einlesen und, wann immer ein bisher ungesehenes Wort eingelesen wird, ein Pattern als *Hypothese* ausgeben.

Ist P eine Klasse von Patternsprachen, so ist eine Klasse $\mathcal{L}$ von Sprachen *P-deskriptiv generalisierbar*, wenn eine berechenbare *Generalisierungsstrategie* S existiert, so dass für jede Sprache $L \in \mathcal{L}$ die Folge der von S ausgegeben Hypothesen gegen ein Pattern δ konvergiert, das P-deskriptiv für L ist.

Es muss also keine exakte Repräsentation der Zielsprachen erlernt werden, sondern nur eine Approximation. Hierbei stellt sich heraus, dass die Korrespondenz zwischen zu generalisierenden Sprachen und korrekten Hypothesen im Allgemeinen schwächer ist als bei den üblicherweise in der Literatur betrachteten Lernmodellen: Ein Pattern δ kann gleichzeitig deskriptives Pattern (und korrekte Hypothese) für zwei unvergleichbare Sprachen L_1, L_2 sein, und eine Sprache L kann durch zwei unterschiedliche deskriptive Pattern δ_1 und δ_2 generalisiert werden.

Zusätzlich charakterisiert Theorem 6.26 diejenigen Klassen von Sprachen, die anhand einer festen (und nach Ansicht des Autors kanonischen) Strategie in Bezug auf die Klasse der terminalfreien E-Patternsprachen deskriptiv generalisiert werden können.

Mittels dieser Charakterisierung lässt sich eine große deskriptiv generalisierbare Klasse von Sprachen definieren, wodurch tiefere Einsichten zur Stärke des DG-Modells gewonnen und ein Vergleich zum LIM-TEXT ermöglicht werden.

Die Inhalte dieses Kapitels wurden bereits in der Arbeit [34] (gemeinsam mit Daniel Reidenbach) veröffentlicht.

Kapitel 7

Dieses Kapitel befasst sich mit einer Vermutung, die der Autor beim Versuch aufstellte, den Beweis von Theorem 5.18 zu einer Charakterisierung derjenigen Sprachen zu erweitern, für die kein Pattern in Bezug auf die Klasse der terminalfreien E-Pattersprachen deskriptiv ist.

Dazu wird ein größeres technisches Instrumentarium eingeführt und anschließend zur Konstruktion von Gegenbeispielen zu dieser Vermutung verwendet. Darüber hinaus werden verschiedene Phänomene diskutiert, die die Schwierigkeit einer Charakterisierung der genannten Sprachen verdeutlichen.

Als Fazit dieses Kapitels lässt sich feststellen, dass die Nichtexistenz deskriptiver Pattern selbst in Bezug auf die Klasse der terminalfreien Patternsprachen komplex und wahrscheinlich nur schwer zu charakterisieren ist.

Die Inhalte dieses Kapitels wurden nicht zuvor veröffentlicht.

Acknowledgements

First of all, I am indebted to my parents, Irmtraud and Dieter, and also to my sister Lisa, for their continued support.

Furthermore, I want to thank my teachers, especially Ralf Emrich, whose exemplary courses in school made me aware of the existence of theoretical computer science, and Rolf Wiehagen, whose inspired teaching enabled me to truly appreciate the beauty of theory.

I am very grateful to my advisors, Nicole Schweikardt and Detlef Wotschke, who both gave me useful advise and feedback whenever I needed it (and not only when I asked for it), made me aware of fascinating areas I would have missed without their guidance, and provided me with valuable freedom.

I am also indebted to Daniel Reidenbach, especially for showing continuous interest in the topics presented in the present thesis, for innumerable helpful discussions and suggestions, for being a diligent and reliable co-author, for his feedback on this thesis, and for refusing to bet money on the existence of counterexamples.

I am grateful to Lars Hedrich and Manfred Schmidt-Schauß for acting as examiners.

I wish to express my gratitude to Markus Schmid and Joachim Bremer for numerous insightful and interesting discussions. I also want to thank them for importing pork crackles and for baking the world's best cookies, respectively. Joachim deserves additional thanks for also being a great co-author.

I want to thank my current and former colleagues for creating a pleasant working atmosphere: André Böhm, Sviatlana Danilava, André Hernich, Sabine Kappes, Christine Lenhart, Andreas Malcher, and Mariano Zelke. Special thanks go to Jutta Nadland, who also contributed to this atmosphere, and helped me in countless occasions.

My greatest thanks belong to Petra, who shared my joy over every new result, encouraged me to carry on in those dark times where problems seemed unsolvable, patiently waited whenever I was absent-minded, and gently stopped me whenever I was too absent-minded for my own good. In addition to this, she also proofread this thesis.

To Petra

Contents

Chapter 1

Introduction

1.1 On Patterns

Patterns – finite strings that consist of *variables* and *terminals* – are compact and natural devices for the definition of formal languages. A pattern generates a word by a *substitution* of the variables with arbitrary strings of terminals from a fixed alphabet Σ (where all occurrences of a variable in the pattern must be replaced with the same word), and its language is the set of all words that can be obtained under substitutions. In a more formal manner, the language of a pattern can be understood as the set of all images under *terminal-preserving* morphisms; i. e., morphisms that map variables to terminal strings, and each terminal to itself. For example, the pattern $\alpha = x_1 x_1 \mathtt{a}\,\mathtt{b}\, x_2$ (where x_1 and x_2 are variables, and $\mathtt{a}$ and $\mathtt{b}$ are terminals) generates the language $L(\alpha)$ of all words that have a prefix that consists of a square, followed by the word $\mathtt{a}\,\mathtt{b}$, and are followed by an arbitrary word.

The study of patterns in strings goes back to Thue [103] and is a central topic of combinatorics on words (cf. the survey by Choffrut and Karhumäki [18]), while the investigation of pattern languages was initiated by Angluin [4]. Angluin's definition of pattern languages permits only the use of *nonerasing* substitutions (hence, this class of pattern languages is called *NE-pattern languages*). Later, Shinohara [102] introduced *E-pattern languages* (E for 'erasing' or 'extended'), where erasing substitutions are permitted.

This small difference in the definitions leads to immense differences in the properties of these two classes. For example, while the equivalence problem for NE-pattern languages is trivially decidable, the equivalence problem for E-pattern languages is a hard open problem. Although both classes were first introduced in the context of *inductive inference* (which deals with the problem of learning patterns for given sets of strings, for a survey see Ng and Shinohara [76]), they have been widely studied in Formal Language Theory (cf. the surveys by Mateescu and Salomaa [69], Mitrana [72], Salomaa [98]). Due to their compact definition, patterns or their languages occur in numerous prominent areas of computer science and discrete mathematics, including *unavoidable patterns* (cf. Jiang et al. [50]), *extended regular expressions* (cf. Chapter 4), *word equations* and the *positive theory of concatenation* (cf. Section 3.4.1), and recently, *database theory* (cf. Barceló et al. [8]).

From the simple definition of pattern languages, one might expect that all properties of the language of a pattern are immediately obvious from the pattern itself (and the

information which terminal alphabet one substitutes the variables into and whether variables may be erased), and hence, comparing pattern languages (more formally, deciding the *equivalence* or *inclusion problem* of a pattern language) or deciding whether a given word belongs to the language of a pattern (the *membership problem*) should simply be a straightforward matter of comparing patterns, or the pattern and the word. With few exceptions, this is not the case – in general, inclusion for pattern languages is undecidable (cf. Chapter 3), and the membership is NP-complete even for many subclasses (cf. Section 2.2).

Thus, patterns (and their languages) offer a simple and elegant definition that leads to hard problems with the potential applications in various areas. On one side, proving a negative result for a problem on patterns (e. g. undecidability, or hardness for some complexity class) immediately extends to all larger classes that contain pattern languages, and to problems on models that use pattern languages in their definition. On the other side, the proof of a negative result often hints at appropriate restrictions that might simplify that problem. Here, pattern languages serve as a proving ground for restrictions and techniques, which (not always, but from time to time) can be extended to larger classes and models with more intricate definitions that encompass patterns. For some examples, see Ng and Shinohara [76], Reidenbach and Schmid [89], and Section 3.4 in the present thesis.

Surveys on pattern languages and various extensions and applications have been provided by A. Salomaa [96, 97], Mateescu and A. Salomaa [69], Kucherov and Rusinowitch [59], K. Salomaa [98], Mitrana [72], Reidenbach [85] (Chapter 3), and Ng and Shinohara [76]; further references on various aspects on pattern languages, particular applications and related areas are mentioned in the text of this thesis whenever it is appropriate.

1.2 On This Thesis

The present thesis examines various aspects of the inclusion problem for pattern languages, and related problems. The main focus is on E-pattern languages, as are most results, but NE-pattern languages are often considered for comparison. The present section provides a short overview of the structure of the thesis, of the content of each chapter, and of the relations between the chapters. Note that every chapter has also its own detailed introduction.

The formal part of the thesis begins with Chapter 2, where we introduce the technical preliminaries that are used throughout the thesis.

The main body consists of five chapters that form two largely independent parts: The first part, consisting of Chapters 3 and 4, deals with the inclusion problem for pattern languages, and applies a technique for patterns to multiple decision problems for a common extension of regular expressions.

More specifically, in Chapter 3, we examine the decidability of the inclusion problem for pattern languages. In 1995, Jiang, Salomaa, Salomaa and Yu [51] solved a longstanding open problem by proving that inclusion for pattern languages (in both the E and the NE case) is undecidable. We show in Theorem 3.3 that undecidability holds even when the terminal alphabet is restricted to two or four letters (in the E- and NE-case, respectively), thus answering an open question that was first posed by Reidenbach [85]

and reaffirmed by Salomaa [99]. Furthermore, we study the difficulty of deciding the inclusion for E-pattern languages that are generated by patterns with a bounded number of variables. We show that inclusion is still undecidable for patterns with a bounded but comparatively large number of variables (Theorem 3.10), and we combine the famous Collatz conjecture (cf. Lagarias [62, 63]) with a slight variation of the technique from the proof of Theorem 3.10 to prove that there is reason to expect that inclusion is still very difficult for significantly lower bounds (Theorem 3.11 and 3.12). Furthermore, in Sections 3.3.5 and 3.4, respectively, we discuss extensions of the aforementioned proof technique to larger terminal alphabets and to the material presented in Chapter 4. The content of this chapter was published in [12] and [32] (first presented as [30]).

In Chapter 4, we examine so-called *extended regular expressions*, which extend the regular expressions commonly studied in Theoretical Computer Science with variables that allow to specify repetitions (thus generating non-regular languages). The two main questions in this chapter are whether there exist effective procedures that can be used to simplify or compare extended regular expressions, and how much more succinctly extended regular expressions describe regular languages in contrast to the well-known classical regular expressions. We prove that the corresponding decision problems are undecidable, and that the tradeoff in length between extended regular expressions and regular expressions is not bounded by any recursive function. Furthermore, we show that these results hold even if the extended regular expression uses only a single variable. Almost all results in this chapter are based on Theorem 4.14, which was initially derived from the proof of Theorem 3.10. The content in this chapter was published as [29].

Although the techniques in Chapters 3 and 4 are related, the chapters themselves can be read and understood independently and in any order (as far as the author himself is able to judge this question).

The second part of this thesis consists of Chapters 5 to 7, in which we examine various aspects of *descriptive patterns*. A pattern δ from some class of patterns P is descriptive of a language L if its language $L(\delta)$ is a minimal generalization of L with respect to P; i. e., $L(\delta) \supseteq L$, and there is no $\gamma \in P$ with $L(\delta) \supset L(\gamma) \supseteq L$. As such, descriptive patterns can be considered the best approximation of languages within P. Note that, as descriptive patterns are defined using the inclusion relation of pattern languages, many problems for descriptive patterns are related to the inclusion problem.

In Chapter 5, we examine the question which languages have a descriptive pattern. After introducing and motivating the concept of descriptive patterns both informally and formally (Sections 5.1 and 5.2), we first consider a necessary condition for the non-existence of a descriptive pattern for a given language, namely the existence of a certain kind of strictly decreasing chain of pattern languages (Section 5.3). We then examine the existence of descriptive patterns for finite and infinite sets, considering patterns generating a whole class of E- or NE-pattern languages. The main result of this chapter is Theorem 5.18, in which we prove that there are languages of which no E-pattern language is descriptive, answering an open question by Jiang et al. [50]. Furthermore, in Section 5.5 we prove that the problem to compute descriptive patterns for finite sets cannot be solved efficiently, and connect the problem of finding descriptive patterns for unions of E-pattern languages to the equivalence problem for E-pattern languages.

Most of the content of this chapter was published in [33] (first presented as [31]), the only exception being the material in Section 5.5, which was published in [34] (Sec-

tion 5.5.1), or has not been published before (Section 5.5.2).

In Chapter 6, we introduce the concept of *descriptive generalization (from positive data)*. This is a new learning model that is based on Gold's model of language identification in the limit (cf. Gold [39]), but instead of learning *exact* representations of formal languages from positive data, it infers descriptive patterns as *approximations* of these languages. In Section 6.2, we discuss this model in its most general form (for arbitrary effective strategies and arbitrary classes of patterns) and a possible extension to a more abstract model that does not rely on patterns. In Section 6.3, we then consider a fixed class of pattern languages with a decidable inclusion problem (the class of *terminal-free E-pattern languages*), and a strategy that the author considers canonical. We then provide a characterization of the class that can be descriptively generalized in this setting (Theorem 6.26), which we call the class of *locking set languages*. Section 6.4 lists various properties of locking set languages. The content of this chapter was published in [34].

In Chapter 7, we examine a conjecture which the author developed in search of a generalization of the proof of Theorem 5.18 from Chapter 5 to a characterization of those languages that have no descriptive pattern with respect to the class of terminal-free E-pattern languages. Most of the chapter is devoted to the technical tools that are used to define the examples that disprove the author's conjecture and illustrate the various difficulties that such a characterization would need to overcome. This material has not been published before.

Note that Chapters 6 and 7 use the definitions introduced in Section 5.2.1. Apart from this, the content of Chapter 5 serves only as wider context for Chapter 6, both chapters can be read mostly indepentenly. This does not hold for Chapter 7, which attempts to generalize and extend the proof of Theorem 5.18. In order to appreciate the content of this chapter, the reader should be familiar with the content of Section 5.3 and, preferably, the proof of Theorem 5.18 presented in Section 5.4.2.

The thesis closes with Chapter 8, where we summarize the results of the previous chapters and point out some open questions and possible directions of future work.

Note that, with the exception of [29], all mentioned articles were published with co-authors, namely, Joachim Bremer ([12]) and Daniel Reidenbach ([30, 31, 32, 33, 34]). The author has contacted his co-authors to reaffirm that there are no disputes over the ownership of the material presented in this thesis. Whenever we consider material from any of these co-authored articles that is not the author's, we explicitly mention this fact.

Chapter 2

Preliminaries

We begin the formal part of this thesis with a detailed description of basic definitions. Although this thesis is largely self contained, we assume that the reader is familiar with the common mathematical concepts and notations, as well with the elementary insights of formal language theory (cf. Hopcroft and Ullman [46], Salomaa [95]), complexity theory (cf. Garey and Johnson [38]), and recursion theory (cf. Cutland [21], Odifreddi [77], Rogers [91]).

2.1 Basic Definitions

Let $\mathbb{Z}$ denote the set of all integers, and let $\mathbb{N}$ denote the set of all non-negative integers. For every $k \geq 0$, let $\mathbb{N}_k := \{n \in \mathbb{N} \mid n \geq k\}$, and let the symbol ∞ denote infinity.

The symbols div and mod denote the integer division and its remainder (respectively), and $\lfloor\rfloor$ denotes the floor function. The symbols $\subseteq$, $\subset$, $\supseteq$ and $\supset$ refer to subset, proper subset, superset and proper superset relation, respectively. The symbols $\emptyset$, $\mathcal{P}$ and $\setminus$ denote the empty set, the power set, and the set difference, respectively. For any set A and any $n \geq 1$, subsets $A_1, \ldots, A_n \subseteq A$ form a *partition* of A if all A_i are pairwise disjoint, and $A = \bigcup_{i=1}^{n} A_i$.

For any alphabet A, a *string* (over A) is a finite sequence of symbols from A, and λ denotes the *empty string*, or *empty word*. The symbol A^+ denotes the set of all nonempty strings over A, and $A^* := A^+ \cup \{\lambda\}$. For the *concatenation* of two strings w_1, w_2 we write $w_1 \cdot w_2$ or simply $w_1 w_2$. We say a string $v \in A^*$ is a *factor* of a string $w \in A^*$ if there are $u_1, u_2 \in A^*$ such that $w = u_1 v u_2$. If $u_1 = \lambda$ (or $u_2 = \lambda$), then v is a *prefix* of w (or a *suffix*, respectively).

For an arbitrary alphabet A, a *language* L (over A) is a set of strings over A, i. e. $L \subseteq A^*$. A language L is *empty* if $L = \emptyset$; otherwise, it is *nonempty*. A *class* $\mathcal{L}$ of languages (over A) is a set of languages over A, i. e., $\mathcal{L} \subseteq \mathcal{P}(A^*)$. Let FIN_A denote the class of all finite languages over A.

The notation $|K|$ denotes the size of a set K or the length of a string K; the term $|w|_a$ refers to the number of occurrences of the symbol a in the string w. For any $w \in A^*$ and any $n \in \mathbb{N}_0$, w^n denotes the *n-fold concatenation of w*, with $w^0 := \lambda$. Furthermore, we use $\cdot$ and the Kleene operations $*$ and $+$ on sets and strings in the usual way.

If A is an alphabet, a *(one-sided) infinite word* over A is an infinite sequence $w = (w_i)_{i=0}^{\infty}$ with $w_i \in A$ for every $i \geq 0$. We denote the set of all one-sided infinite words over A by A^ω and, for every $a \in A$, let a^ω denote the word $w = (w_i)_{i=0}^{\infty}$ with $w_i = a$ for

every $i \geq 0$. We shall only deal with infinite words $w \in A^\omega$ that have the form $w = u\,a^\omega$ with $u \in A^*$ and $a \in A$.

Concatenation of words and infinite words is defined canonically: For every $u \in A^+$ and every $v \in A^\omega$ with $v = (v_i)_{i=0}^\infty$, $u \cdot v := w \in A^\omega$, where $w_0 \cdot \ldots \cdot w_{|u|-1} = u$ and $w_{i+|u|} = v_i$ for every $i \geq 0$, and $\lambda \cdot v{:=}v$, while $v \cdot u$ is undefined. Especially, note that $a \cdot a^\omega = a^\omega$ for every $a \in A$.

For any alphabets A, B, a *morphism* is a function $h : A^* \to B^*$ that satisfies $h(vw) = h(v)h(w)$ for all $v, w \in A^*$. Given morphisms $g : A^* \to B^*$ and $h : B^* \to C^*$ (for alphabets A, B, C), their *composition* $h \circ g$ is defined by $(h \circ g)(w) := h(g(w))$ for all $w \in A^*$. For every morphism $h : A^* \to A^*$ and every $n \geq 0$, h^n denotes the *n-fold iteration of h*, i. e., $h^{n+1} := h \circ h^n$, where h^0 is the identity on A^*.

A morphism $h : A^* \to B^*$ is said to be *nonerasing* if $h(a) \neq \lambda$ for all $a \in A$. For any string $w \in C^*$, where $C \subseteq A$ and $|w|_a \geq 1$ for every $a \in C$, the morphism $h : A^* \to B^*$ is called a *renaming (of w)* if $h : C^* \to B^*$ is injective and $|h(a)| = 1$ for every $a \in C$.

2.2 Patterns and Their Languages

Let Σ be a (finite or infinite) alphabet of so-called *terminals* and X an infinite set of *variables* with $\Sigma \cap X = \emptyset$. We normally assume $\{\mathtt{a}, \mathtt{b}, \ldots\} \subseteq \Sigma$ and $\{x_1, x_2, x_3, \ldots\} \subseteq X$. A *pattern* is a non-empty string over $\Sigma \cup X$, a *terminal-free pattern* is a non-empty string over X, and a *terminal-string* (or *word*) is a string over Σ. For any pattern α, we refer to the set of variables in α as $\mathrm{var}(\alpha)$ and to the set of terminals in α as $\mathrm{symb}(\alpha)$. The set of all patterns over $\Sigma \cup X$ is denoted by Pat_Σ; the set of all terminal-free patterns is denoted by $\mathrm{Pat}_{\mathrm{tf}}$ or X^+. For every $n \geq 0$, let $\mathrm{Pat}_{n,\Sigma}$ denote the set of all patterns over Σ that contain at most n variables; that is, $\mathrm{Pat}_{n,\Sigma} := \{\alpha \in \mathrm{Pat}_\Sigma \mid |\mathrm{var}(\alpha)| \leq n\}$.

A morphism $\sigma : (\Sigma \cup X)^* \to (\Sigma \cup X)^*$ is called *terminal-preserving* if $\sigma(a) = a$ for every $a \in \Sigma$. A terminal-preserving morphism $\sigma : (\Sigma \cup X)^* \to \Sigma^*$ is called a *substitution*. The *E-pattern language* $L_{\mathrm{E},\Sigma}(\alpha)$ of α is given by

$$L_{\mathrm{E},\Sigma}(\alpha) := \{\sigma(\alpha) \mid \sigma : (\Sigma \cup X)^* \to \Sigma^* \text{ is a substitution}\},$$

and the *NE-pattern language* $L_{\mathrm{NE},\Sigma}(\alpha)$ of a pattern $\alpha \in \mathrm{Pat}_\Sigma$ is given by

$$L_{\mathrm{NE},\Sigma}(\alpha) := \{\sigma(\alpha) \mid \sigma : (\Sigma \cup X)^* \to \Sigma^* \text{ is a nonerasing substitution}\}.$$

If the intended meaning is clear, we write $L(\alpha)$ instead of $L_{\mathrm{E},\Sigma}(\alpha)$ or $L_{\mathrm{NE},\Sigma}(\alpha)$ for any $\alpha \in \mathrm{Pat}_\Sigma$. A pattern language is called *terminal-free* if it is generated by a terminal-free pattern. For convenience, we define $L_{\mathrm{E},\Sigma}(\lambda) := L_{\mathrm{NE},\Sigma}(\lambda) := \{\lambda\}$ for all terminal alphabets Σ.

Furthermore, let ePAT_Σ denote the class of all E-pattern languages over Σ, and nePAT_Σ the class of all NE-pattern languages over Σ. Likewise, we define $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$ as the class of all $L_{\mathrm{E},\Sigma}(\alpha)$ with $\alpha \in \mathrm{Pat}_{\mathrm{tf}}$, and, for any $n \geq 0$, $\mathrm{ePAT}_{n,\Sigma}$ as the class of all $L_{\mathrm{E},\Sigma}(\alpha)$ with $\alpha \in \mathrm{Pat}_{n,\Sigma}$. The classes $\mathrm{nePAT}_{\mathrm{tf},\Sigma}$ and $\mathrm{nePAT}_{n,\Sigma}$ are defined accordingly. In most of the cases, we shall consider only pattern languages over a terminal alphabet Σ that is finite, and non-unary (i. e., $|\Sigma| \geq 2$). The latter restriction that is due to the fact that unary pattern languages are a strict subset of the set of regular languages

(cf. Reidenbach [85][1]), and share few interesting properties with pattern languages over larger alphabets.

Furthermore, in order to obtain a unique representative of each class of those patterns that are identical up to a terminal-preserving renaming, we define the notion of the *canonical form* of a pattern. For this, we fix any well-founded total order $\preceq$ on X. A pattern $\alpha \in \mathrm{Pat}_\Sigma$ is in *canonical form (with respect to $\preceq$)* if all variables are introduced without gaps and in ascending order with respect to $\preceq$. More formally, $\alpha \in \mathrm{Pat}_\Sigma$ is in canonical form if, for all $\beta, \gamma \in (\Sigma \cup X)^*$ and all $x \in X$ with $\alpha = \beta x \gamma$, $y \in \mathrm{var}(\beta)$ holds for all $y \in X$ with $y \prec x$.

One of the canonical questions for classes of pattern languages is the *membership problem*; i. e., given a pattern α from some class of patterns, and a word $w \in \Sigma^*$, does w belong to $L(\alpha)$? Both problems are decidable, although not efficiently (withstanding P = NP):

Theorem 2.1 (Angluin [4], Jiang et al. [50])**.** *Let Σ be an alphabet with $|\Sigma| \geq 2$. The membership problem for* nePAT_Σ *and the membership problem for* ePAT_Σ *are* NP*-complete.*

The NE-case of Theorem 2.1 is due to Angluin [4], the E-case is an extension of the NE-case and due to Jiang, Kinber, Salomaa, Salomaa and Yu [50]. Independently, Ehrenfeucht and Rozenberg proved the following more general theorem:

Theorem 2.2 (Ehrenfeucht and Rozenberg [27])**.** *Let X be an infinite alphabet, and Σ be an alphabet with at least two letters. Given an $\alpha \in X^*$ and a $w \in \Sigma^*$, it is an* NP*-complete problem to decide whether there is a morphism $\phi : X^* \to \Sigma^*$ with $\phi(\alpha) = w$.*

Note that this result implies that even for terminal-free pattern languages, the membership problem is NP-complete. Furthermore, all proofs rely on the fact that the number of variables in the involved patterns is unbounded. As shown by Ibarra et al. [47], the membership problem can be solved in polynomial time for E- and NE-patterns if the number of variables in the patterns is bounded.

In contrast to this, the *equivalence problem* for NE-pattern languages is efficiently decidable, as equivalence of NE-pattern languages can be characterized in a trivial manner:

Theorem 2.3 (Angluin [4])**.** *Let Σ be an alphabet with $|\Sigma| \geq 2$. For patterns $\alpha, \beta \in \mathrm{Pat}_\Sigma$, $L_{\mathrm{NE},\Sigma}(\alpha) = L_{\mathrm{NE},\Sigma}(\beta)$ holds if and only if there is a terminal-preserving renaming $\phi : \mathrm{Pat}_\Sigma \to \mathrm{Pat}_\Sigma$ with $\phi(\alpha) = \beta$.*

Thus, for all non-unary alphabets, two patterns generate the same NE-language if and only if they are identical, modulo a renaming. Likewise, two patterns in canonical form generate the same NE-language if and only if they are identical. On the other hand, the equivalence problem for E-pattern languages is still open and is widely considered to be among the hardest open problems for pattern languages (see Ohlebusch and Ukkonen [79] and Freydenberger and Reidenbach [30]).

[1] Alternatively, one can prove this claim by observing that the class of unary pattern languages is identical to the class of linear unary languages (cf. Proposition 5.21), which is a proper subset of the class of regular unary languages (cf. Salomaa [95], Chapter II.7).

The decidability of the *inclusion problem* for various classes of pattern languages – the central topic of this thesis – is defined as follows: Let P_1, P_2 be two classes of patterns, and $\mathrm{PAT}_1, \mathrm{PAT}_2$ be the corresponding classes of pattern languages (either the class of all E-pattern languages or the class of all NE-pattern languages over some alphabet Σ that are generated by patterns from P_1 or P_2). We say that the *inclusion problem for* PAT_1 *in* PAT_2 *is decidable* if there exists a total computable function χ such that, for every pair of patterns $\alpha \in P_1$ and $\beta \in P_2$, χ decides on whether or not $L(\alpha) \subseteq L(\beta)$. If no such function exists, this inclusion problem is *undecidable*. If both classes of pattern languages are the same class $\mathrm{PAT}_{\star,\Sigma}$, we simply refer to the *inclusion problem of* $\mathrm{PAT}_{\star,\Sigma}$.

One of the most useful results on pattern languages is the following characterization of the inclusion for terminal-free E-pattern languages:

Theorem 2.4 (Filè [28], Jiang et al. [50])**.** *Let* $|\Sigma| \geq 2$*. For every* $\alpha, \beta \in X^+$*,* $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$ *holds if and only if there is a morphism* $\phi : X^* \to X^*$ *with* $\phi(\beta) = \alpha$*.*

While Theorem 2.4 implies that inclusion is decidable for terminal-free E-pattern languages, we can also conclude (together with Theorem 2.2) that this problem is NP-complete.

Chapter 3

Inclusion of Pattern Languages

3.1 On Inclusion for Pattern Languages

One of the most notable results on pattern languages is the proof of the undecidability of the inclusion problem (for E- as well as NE-pattern languages) by Jiang, Salomaa, Salomaa and Yu [51], a problem that was open for a long time and is of vital importance for the inductive inference of pattern languages (cf. Angluin [5], Ng and Shinohara [76]).

Unfortunately, this proof heavily depends on the availability of an unbounded number of terminals, which might be considered impractical, as pattern languages are mostly used in settings with fixed (or at least bounded) terminal alphabets. The first major theorem of the present chapter, Theorem 3.3, shows that undecidability holds even for all E-pattern languages over finite terminal alphabets with more than one letter, and for NE-pattern languages with at least four letters. This result was obtained through a considerable modification of the proof technique of Jiang et al.

As the proof by Jiang et al. and its modification used in the proof of Theorem 3.3 require the number of variables of the involved patterns to be unbounded, the author considers it a natural question whether the inclusion problems remain undecidable even if bounds are imposed on the number of variables in the pattern; especially as bounding the number of variables changes the complexity of the membership problem from NP-complete to polynomial time (cf. Ibarra et al. [47]). Similar restrictions have been studied in the theory of concatenation (cf. Section 3.4.1).

Apart from potential uses in inductive inference or other areas, and the search for an approach that could provide the leverage needed to solve the equivalence problem for E-pattern languages, the author's main motivation for deeper research into the inclusion problems is the question how strongly patterns and their languages are connected. All known cases of (nontrivial) decidability of the inclusion problem for various classes of patterns rely on the fact that for these classes, inclusion is characterized by the existence of a terminal-preserving morphism mapping one pattern to the other. This is a purely syntactical condition that, although NP-complete (cf. Ehrenfeucht and Rozenberg [27]), can be verified in a straightforward way. Finding cases of inclusion that are not covered by this condition, but still decidable, could uncover (or rule out) previously unknown phenomena, and be of immediate use for related areas of research.

The three further main results in the present chapter, Theorems 3.10, 3.11 and 3.12, can be summarized as follows: We show that the inclusion problem for E-patterns with a bounded (but large) number of variables is indeed undecidable. For smaller bounds,

we prove the existence of classes of patterns that have complicated inclusion relations, and an inclusion problem which we are not able to prove undecidable. Some of these inclusions can simulate iterations of the Collatz function, while others could (in principle) be used to settle an important part of the famous Collatz conjecture.

3.2 Definitions and a Preliminary Result

This section contains the definitions of the models which we simulate in the proofs of the major results presented in Section 3.3.

3.2.1 The Universal Turing Machine $\mathcal{U}$

Let $\mathcal{U}$ be the universal[1] Turing machine $U_{15,2}$ with 2 symbols and 15 states described by Neary and Woods [75]. This machine has the state set $Q = \{q_1, \ldots, q_{15}\}$ (where q_1 is the initial state) and operates on the tape alphabet $\Gamma = \{0, 1\}$ (where 0 is the blank symbol). Its transition function $\delta : \Gamma \times Q \to (\Gamma \times \{L, R\} \times Q) \cup \text{HALT}$ is depicted in Figure 3.1.

	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8
0	$0, R, q_2$	$1, R, q_3$	$0, L, q_7$	$0, L, q_6$	$1, R, q_1$	$1, L, q_4$	$0, L, q_8$	$1, L, q_9$
1	$1, R, q_1$	$1, R, q_1$	$0, L, q_5$	$1, L, q_5$	$1, L, q_4$	$1, L, q_4$	$1, L, q_7$	$1, L, q_7$

	q_9	q_{10}	q_{11}	q_{12}	q_{13}	q_{14}	q_{15}
0	$0, R, q_1$	$1, L, q_{11}$	$0, R, q_{12}$	$0, R, q_{13}$	$0, L, q_2$	$0, L, q_3$	$0, R, q_{14}$
1	$1, L, q_{10}$	HALT	$1, R, q_{14}$	$1, R, q_{12}$	$1, R, q_{12}$	$0, R, q_{15}$	$1, R, q_{14}$

Figure 3.1: The transition table of the universal Turing machine $\mathcal{U}$ which is defined in Section 3.2.1. This machine is due to Neary and Woods [75] and is, to the author's knowledge, the smallest currently known universal Turing machine over a two letter tape alphabet.

In order to discuss configurations of $\mathcal{U}$, we use the following terminology: A *configuration* of $\mathcal{U}$ is a triple $(q_i, t_{aL}, t_R) \in Q \times \Gamma^* 0^\omega \times \Gamma^* 0^\omega$, where q_i denotes the state, and t_{aL} describes the content of what we shall call the *left side of the tape*, the infinite word that starts at the position of the machine's head and extends to the left[2]. Likewise, t_R describes the *right side of the tape*, the infinite word that starts immediately to the right of the head and extends to the right (cf. Figure 3.2).

Encoding Computations of $\mathcal{U}$

We define the function $\mathrm{e} : \Gamma \to \mathbb{N}_0$ as $\mathrm{e}(0) := 0$ and $\mathrm{e}(1) := 1$, and extend this to an encoding of infinite sequences $t = (t_i)_{i=0}^{\infty}$ over Γ by $\mathrm{e}(t) := \sum_{i=0}^{\infty} 2^i \, \mathrm{e}(t_i)$. As we consider

[1] Note that $\mathcal{U}$ is indeed Turing universal: There is a computable function $c_{\mathcal{U}}$ that maps every Turing machine $\mathcal{M}$ and every configuration I of $\mathcal{M}$ into a configuration $c_{\mathcal{U}}(\mathcal{M}, I)$ of $\mathcal{U}$ such that $\mathcal{U}$ halts in finite time after being started in the configuration $c_{\mathcal{U}}(\mathcal{M}, I)$ if and only if $\mathcal{M}$ halts in finite time after being started in the configuration I.

[2] Hence t_{aL}, as the word contains both the current letter a and the part to its left. This naming was chosen intentionally, to distinguish this concept of tape sides from the one we shall use in Chapter 4.

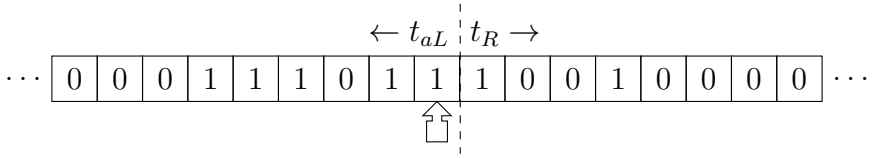

Figure 3.2: An illustration of tape words of some configuration of the universal Turing machine $\mathcal{U}$ (as defined in Section 3.2.1). The arrow below the tape symbolizes the position of the head, while the dashed lines show the borders between the left tape side and the right tape side. Assuming that all tape cells that are not shown contain 0, we observe the left tape word $t_L = 1101110^\omega$ and the right tape word $t_R = 10010^\omega$.

only configurations where all but finitely many cells of the tape consist of the blank symbol 0 (which is encoded as 0), $\mathrm{e}(t)$ is always finite and well-defined. Note that for every side t of the tape, $\mathrm{e}(t) \bmod 2$ returns the encoding of the symbol that is closest to the head (the symbol under the head for t_{aL}, and the symbol to the right of the head for t_R). Furthermore, each side can be lengthened or shortened by multiplying or dividing (respectively) its encoding $\mathrm{e}(t)$ by 2. The encoding $\mathrm{enc}_{\mathcal{U}}$ of configurations of $\mathcal{U}$ is defined by

$$\mathrm{enc}_{\mathcal{U}}(q_i, t_{aL}, t_R) := 0\, 0^{\mathrm{e}(t_R)} \#0\, 0^{\mathrm{e}(t_{aL})} \#0^i,$$

for every configuration (q_i, t_{aL}, t_R).

Encoding the tape content of any configuration of $\mathcal{U}$ such that each tape side is identified with a natural number by use of e allows us to describe the correct successor configuration using only basic arithmetical operations:

Observation 3.1. *Assume that $\mathcal{U}$ is in some configuration $C = (t_{aL}, t_R, q_i)$, and $\delta(a, q_i) = (d, M, q_j)$ for some $d \in \{0, 1\}$, some $M \in \{L, R\}$ and some $q_j \in Q$. For the (uniquely defined) successor configuration $C' = (t'_{aL}, t'_R, q_j)$, the following holds:*

1. *if $M = L$, then $\mathrm{e}(t'_{aL}) = \mathrm{e}(t_{aL}) \operatorname{div} 2$ and $\mathrm{e}(t'_R) = 2(\mathrm{e}(t_R)) + \mathrm{e}(d)$,*
2. *if $M = R$, then $\mathrm{e}(t'_{aL}) = 4(\mathrm{e}(t_{aL}) \operatorname{div} 2) + 2\,\mathrm{e}(d) + (\mathrm{e}(t_R) \bmod 2)$ and $\mathrm{e}(t'_R) = \mathrm{e}(t_R) \operatorname{div} 2$.*

Proof. We begin with the case that $M = L$. Thus, if $\mathcal{U}$ reads a in state q_i, it has to write d and to execute a left movement. It is easier to understand the respective equations if we view this not as movement of the head, but as movement of the tape. In order to derive t'_{aL} from t_{aL}, we simply move all tape cells one step to the right (instead of moving the head ones step to the left), while the cell that is currently under at the head position is moved to the right tape side. If we view t_{aL} as a binary number, it should become obvious that $\mathrm{e}(t'_{aL}) = \mathrm{e}(t_{aL}) \operatorname{div} 2$.

Likewise, all cells of the right tape side are moved one step to the right, which multiplies $\mathrm{e}(t_R)$ by two. In addition to this, the right side gains the cell that was the former head position as its leftmost tape cell. As $\mathcal{U}$ just wrote d into this cell, $\mathrm{e}(t'_R) = 2\,\mathrm{e}(t_R) + \mathrm{e}(d)$ holds.

Now consider the case of $M = R$. Although the equation for $\mathrm{e}(t'_R)$ should be clear (as it behaves analogously to the equation for $\mathrm{e}(t'_{aL})$ in the first case), the equation for $\mathrm{e}(t'_{aL})$ is more involved. First, let t''_{aL} denote the content of the left tape side immediately after writing d, but before moving the head. As the head position is the least significant bit of t_{aL} (viewed as a binary number), we observe $\mathrm{e}(t''_{aL}) = 2(\mathrm{e}(t_{aL}) \operatorname{div} 2) + \mathrm{e}(d)$. This tape side is now shifted one step to the left (which is equivalent to multiplication by two) and gains the leftmost letter of t_R as new head symbol. Thus,

$$\begin{aligned}\mathrm{e}(t'_{aL}) &= 2\,\mathrm{e}(t''_{aL}) + (\mathrm{e}(t_R) \bmod 2)\\ &= 4(\mathrm{e}(t_{aL}) \operatorname{div} 2) + 2\,\mathrm{e}(d) + (\mathrm{e}(t_R) \bmod 2)\end{aligned}$$

The intermediate result $2(\mathrm{e}(t_{aL}) \operatorname{div} 2) + \mathrm{e}(d)$ sets the tape cell under the head to the letter d, multiplying this number by 2 shifts the whole left side of the tape one cell to the left and appends a new cell containing the blank symbol 0. This symbol is then overwritten with the first letter of the right side of the tape by adding $(\mathrm{e}(t_R) \bmod 2)$. Thus, $\mathrm{e}(t'_{aL})$ can indeed be computed as claimed. □

We extend this encoding to an encoding of finite sequences of configurations $C = (C_i)_{i=1}^n$ by

$$\mathrm{enc}_{\mathcal{U}}(C) := \#\#\,\mathrm{enc}_{\mathcal{U}}(C_1)\#\#\ldots\#\#\,\mathrm{enc}_{\mathcal{U}}(C_n)\#\#.$$

Let I be any configuration of $\mathcal{U}$. A *valid computation from* I is a finite sequence $C = (C_i)_{i=1}^n$ (with $n \geq 2$) of configurations of $\mathcal{U}$ such that $C_1 = I$, C_n is a halting configuration, and C_{i+1} is the valid successor configuration of C_i for every i with $1 \leq i < n$, if C_i is not a halting configuration. In contrast to the definitions of valid computations that is commonly used in literature, this definition allows $\mathcal{U}$ to continue its run after reaching a halting configuration. Regarding tape sides, we define that any possible configuration where both tape sides have a finite value under e is a valid successor configuration of a halting configuration.

This extended definition of succession does not change the acceptance behavior of $\mathcal{U}$, and serves only to simplify the construction given in Section 3.3.2. Finally, for every configuration I of $\mathcal{U}$, let

$$\mathrm{ACCEPT}_{\mathcal{U}}(I) := \{\mathrm{enc}_{\mathcal{U}}(C) \mid C \text{ is a valid computation from } I\}.$$

This set is nonempty if and only if $\mathcal{U}$ halts (and, thus, accepts) in finite time after being started in the configuration I. Thus, this set can be used to decide the halting problem of $\mathcal{U}$. As $\mathcal{U}$ is universal, there can be no recursive function that, on input I, decides whether $\mathrm{ACCEPT}_{\mathcal{U}}(I)$ is empty or not.

3.2.2 The Collatz Function

In this section, we discuss a different (and probably less powerful) model of computation, which we shall also use in our proofs. The *Collatz function* $\mathcal{C} : \mathbb{N}_1 \to \mathbb{N}_1$ is defined by $\mathcal{C}(n) := \frac{1}{2}n$ if n is even, and $\mathcal{C}(n) := 3n + 1$ if n is odd. For any $i \geq 0$ and any $n \geq 1$, let $\mathcal{C}^0(n) := n$ and $\mathcal{C}^{i+1}(n) := \mathcal{C}(\mathcal{C}^i(n))$. A number n *leads* $\mathcal{C}$ *into a cycle* if there are i, j with $1 \leq i < j$ and $\mathcal{C}^i(n) = \mathcal{C}^j(n)$. The cycle is *nontrivial* if $\mathcal{C}^k(n) \neq 1$ for every $k \geq 0$; otherwise, it is the *trivial cycle*.

We consider the iterated Collatz function mainly in context of the *Collatz conjecture*, which has also been called the $3n+1$ conjecture, the Ulam conjecture, Kakutani's problem, Thwaites conjecture, Hasse's algorithm, or the Syracuse problem:

Collatz conjecture. *For every $N \geq 1$, there is an $n \geq 1$ such that $\mathcal{C}^n(N) = 1$.*

Consequently, the Collatz conjecture states that every natural number leads $\mathcal{C}$ into the trivial cycle $4, 2, 1$. Regardless of the considerable effort spent on this problem (see the bibliographies by Lagarias [62, 63], as well as the recent book, [64]), the conjecture remains unsolved, as the iterated function often behaves rather unpredictably. Ongoing independent computer verifications by Roosendal [92] and Oliveira e Silva [80] have verified the conjecture for all $N \leq 888 \cdot 2^{50} \approx 10^{18}$ and all $N \leq 20 \cdot 2^{58} \approx 5.764 \cdot 10^{18}$, respectively.

As their unpredictable behavior is in stark contrast to their simple definition, iterations of the Collatz function have been studied in the research of small Turing machines. Margenstern [68] conjectures that every class of Turing machines (as characterized by the number of states and symbols) that contains a machine that is able to simulate the iteration of the Collatz function, also contains a machine that has an undecidable halting problem.

Note that there are two classes of potential counterexamples that would disprove the Collatz conjecture: First, there could be an $N \geq 1$ such that $\mathcal{C}^i(N) \neq \mathcal{C}^j(N)$ for all $i \neq j$, and second, there could be an $N \geq 1$ that leads $\mathcal{C}$ into a cycle that is not the trivial cycle.

Encoding Collatz Iterations

Similar to the definition of $\mathrm{ACCEPT}_{\mathcal{U}}(I)$, we encode those iterations of the Collatz function that lead to the number 1 (and thus, to the trivial cycle) in languages over the alphabet $\{0, \#\}$. For every $N \in \mathbb{N}_1$, let

$$\mathrm{TRIV}(N) := \{\#0^{\mathcal{C}^0(N)}\#0^{\mathcal{C}^1(N)}\# \dots \#0^{\mathcal{C}^n(N)}\# \mid n \geq 1, \mathcal{C}^n(N) = 1\},$$

By definition, $\mathrm{TRIV}(N)$ is empty if and only if N does not lead $\mathcal{C}$ into the trivial cycle. As we shall see, our constructions are able to express an even stronger problem, the question whether there are any numbers that lead $\mathcal{C}$ to a nontrivial cycle. We define NTCC as the set of all strings $\#0^{\mathcal{C}^0(N)}\#0^{\mathcal{C}^1(N)}\# \dots \#0^{\mathcal{C}^n(N)}\#$, where $n, N \geq 1$, $\mathcal{C}^i(N) \neq 1$ for all $i \in \{0, \dots, n\}$, and $\mathcal{C}^j(N) = \mathcal{C}^n(N)$ for some $j < n$. Obviously, this set is nonempty if and only if there exist nontrivial cycles in the iteration of $\mathcal{C}$. As mentioned above, this is one of the two possible cases that would disprove the Collatz conjecture.

3.3 The Difficulty of Inclusion

In this section, we study the inclusion problems of various classes of pattern languages generated by patterns with a bounded number of variables.

As shown by Jiang et al. [51], the *general inclusion problem for pattern languages* is undecidable, both in the case of E- and NE-patterns:

Theorem 3.2 (Jiang et al. [51]). *Let $Z \in \{\mathrm{E}, \mathrm{NE}\}$. There is no total computable function χ_Z which, for every alphabet Σ and for every pair of patterns $\alpha, \beta \in \mathrm{Pat}_\Sigma$, decides on whether or not $L_{Z,\Sigma}(\alpha) \subseteq L_{Z,\Sigma}(\beta)$.*

Technically, Jiang et al. show that, given a so-called nondeterministic 2-counter automaton without input $\mathcal{A}$ (for details, see [51] or Freydenberger and Reidenbach [32]), one can effectively construct an alphabet Σ and patterns $\alpha_\mathcal{A}, \beta_\mathcal{A} \in \mathrm{Pat}_\Sigma$ such that $L_{\mathrm{E},\Sigma}(\alpha_\mathcal{A}) \subseteq L_{\mathrm{E},\Sigma}(\beta_\mathcal{A})$ if and only if $\mathcal{A}$ has an accepting computation. As this problem is known to be undecidable, the general inclusion problem for E-pattern languages must also be undecidable. The undecidability of the general inclusion problem for NE-pattern languages follows using an involved reduction of the E-case to the NE-case. This construction requires two additional terminal letters in Σ.

In the proof for the E-case, Σ contains one letter for every state of $\mathcal{A}$, and six further symbols that are used for technical reasons. Quite obviously, we cannot use such an approach to prove undecidability of the inclusion problem for ePAT_Σ with some fixed alphabet Σ, since we would need to limit the number of states of the automata under consideration. This step, in turn, would lead to a finite class of possible automata, and, hence, would result in a trivially decidable emptiness problem for that class. Consequently, as mentioned by Reidenbach [85] and Salomaa [99], there seems to be no straightforward way from the undecidability result by Jiang et al. [51] to the undecidability of the inclusion problem for ePAT_Σ, especially when Σ is comparatively small. Nevertheless, the inclusion problem remains undecidable for most cases of a fixed terminal alphabet:

Theorem 3.3. *Let Σ be a finite alphabet. If $|\Sigma| \geq 2$, the inclusion problem of ePAT_Σ is undecidable. If $|\Sigma| \geq 4$, the inclusion problem of nePAT_Σ is undecidable.*

Although Theorem 3.3 is part of the present thesis, the author decided to omit its proof, as the E-case is a corollary of Theorem 3.10 further down, and the proof technique used in [32] is only a special case of the more general construction we shall present here.

The proof of the E-case of Theorem 3.3 that is given in [30] consists of a major modification of the construction for the general inclusion problem for E-pattern languages, and requires that the construction is able to use an unbounded number of variables in one of the two patterns. The NE-case of the result follows from the same reduction as in the proof of Theorem 3.2 (thus, $|\Sigma| \geq 4$, as the reduction uses two additional letters), and also requires an unbounded number of variables.

As patterns with an arbitrarily large number of variables might seem somewhat artificial for many applications, the author considers it natural to bound this number in order to gain decidability of (or at least further insights on) the inclusion of pattern languages. We begin our considerations with an observation from two classical papers on pattern languages:

Theorem 3.4 (Angluin [4], Jiang et al. [50]). *The inclusion problem for nePAT_Σ in $\mathrm{nePAT}_{1,\Sigma}$ and the inclusion problem for ePAT_Σ in $\mathrm{ePAT}_{1,\Sigma}$ are decidable.*

The proofs for both cases of this theorem rely on the following sufficient condition for inclusion of pattern languages:

Theorem 3.5 (Jiang et al. [50], Angluin [4]). *Let Σ be an alphabet and $\alpha, \beta \in \mathrm{Pat}_\Sigma$. If there is a terminal-preserving morphism $\phi : (\Sigma \cup X)^* \to (\Sigma \cup X)^*$ with $\phi(\beta) = \alpha$, then $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$. If ϕ is also nonerasing, then $L_{\mathrm{NE},\Sigma}(\alpha) \subseteq L_{\mathrm{NE},\Sigma}(\beta)$.*

In fact, the proofs of both parts of Theorem 3.4 show that, for every alphabet Σ and all patterns $\alpha \in \mathrm{Pat}_\Sigma$, $\beta \in \mathrm{Pat}_{1,\Sigma}$, $L(\alpha) \subseteq L(\beta)$ holds *if and only if* there is a terminal-preserving (and, in the NE-case, nonerasing) morphism ϕ with $\phi(\beta) = \alpha$. As the existence of such a morphism is a decidable property (although in general NP-complete, cf. Theorem 2.2 and our remarks thereon), the respective inclusion problems for these classes are decidable.

There are numerous other classes of pattern languages where this condition is not only sufficient, but characteristic; e. g. the terminal-free E-pattern languages (cf. Theorem 2.4), some of their generalizations (cf. Ohlebusch and Ukkonen [79]), and pattern languages over infinite alphabets (cf. Reidenbach [85], [32]). To the author's knowledge, all nontrivial decidability results for pattern languages over non-unary alphabets rely on this property[3]. Contrariwise, the existence of patterns where inclusion is not characterized by the existence of an appropriate morphism between them is a necessary condition for an undecidable inclusion problem for this class.

The same phenomenon as in Theorem 3.4 does not occur if we swap the bounds of the classes. For the nonerasing case, this is illustrated by the following example:

Example 3.6 (Reidenbach [85], Example 3.2). *Let $\Sigma = \{\mathtt{a}_1, \ldots, \mathtt{a}_n\}$ with $n \geq 2$, and consider the pattern $\alpha_n{:=}x\,\mathtt{a}_1\,x\,\mathtt{a}_2\,x \ldots x\,\mathtt{a}_n\,x$, $\beta{:=}xyyz$. Then there is no nonerasing terminal-preserving morphism ϕ with $\phi(\beta) = \alpha_n$, but every word from $L_{\mathrm{NE},\Sigma}(\alpha_n)$ contains an inner square. Thus, $L_{\mathrm{NE},\Sigma}(\alpha_n) \subseteq L_{\mathrm{NE},\Sigma}(\beta)$.* ◇

Using a less straightforward approach, one obtains an even tighter bound:

Proposition 3.7 (Angluin [4]). *For every finite alphabet Σ, there exist patterns $\alpha \in \mathrm{Pat}_{1,\Sigma}$ and $\beta \in \mathrm{Pat}_{2,\Sigma}$ such that $L_{\mathrm{NE},\Sigma}(\alpha) \subseteq L_{\mathrm{NE},\Sigma}(\beta)$, but there is no nonerasing terminal-preserving morphism $\phi : (\Sigma \cup X)^+ \to (\Sigma \cup X)^+$ with $\phi(\beta) = \alpha$.*

Proof. The proof for the case of binary terminal alphabets is due to Angluin [4] (Example 3.8). As Angluin only sketches the extension to ternary terminal alphabets and mentions that the construction can be extended to larger alphabets in a straightforward way, we give the whole proof.

First, we define the infinite terminal alphabet $\Sigma_\infty{:=}\{\mathtt{a}_1, \mathtt{a}_2, \ldots\}$, where all $\mathtt{a}_i$ are pairwise different. Next, we define an infinite sequence of patterns $(\hat{\alpha}_i)_{i=1}^\infty$ by

$$\hat{\alpha}_1{:=}\,\mathtt{a}_1\,x,$$
$$\hat{\alpha}_{i+1}{:=}\hat{\alpha}_i\,\mathtt{a}_{i+1}\,\hat{\alpha}_i x$$

for every $i \geq 1$. In addition to this, we define a second sequence $(\alpha_i)_{i=1}^\infty$ by $\alpha_i{:=}\hat{\alpha}_i\,\mathtt{a}_i$ for every $i \geq 1$. Thus, the first three patterns in the two sequences are

$$\hat{\alpha}_1{:=}\,\mathtt{a}_1\,x, \qquad \alpha_1{:=}\,\mathtt{a}_1\,x\,\mathtt{a}_1,$$
$$\hat{\alpha}_2{:=}\,\mathtt{a}_1\,x\,\mathtt{a}_2\,\mathtt{a}_1\,xx, \qquad \alpha_2{:=}\,\mathtt{a}_1\,x\,\mathtt{a}_2\,\mathtt{a}_1\,xx\,\mathtt{a}_2,$$
$$\hat{\alpha}_3{:=}\,\mathtt{a}_1\,x\,\mathtt{a}_2\,\mathtt{a}_1\,xx\,\mathtt{a}_3\,\mathtt{a}_1\,x\,\mathtt{a}_2\,\mathtt{a}_1\,xxx, \qquad \alpha_3{:=}\,\mathtt{a}_1\,x\,\mathtt{a}_2\,\mathtt{a}_1\,xx\,\mathtt{a}_3\,\mathtt{a}_1\,x\,\mathtt{a}_2\,\mathtt{a}_1\,xxx\,\mathtt{a}_3\,.$$

[3] Nontrivial meaning that the involved classes are neither finite, nor restricted in some artificial way that leads to trivial decidability.

We shall now show that, for every alphabet $\Sigma = \{\mathtt{a}_1, \ldots, \mathtt{a}_n\} \subset \Sigma_\infty$ (with $n \geq 1$), the patterns α_n and $\beta{:=}xxy$ prove the claim – i. e., $L_{\mathrm{NE},\Sigma}(\alpha_n) \subseteq L_{\mathrm{NE},\Sigma}(\beta)$, but there is no nonerasing terminal-preserving morphism ϕ with $\phi(\beta) = \alpha$. The proof relies on the following two claims:

Claim 1. For every $n \geq 1$, no nonempty prefix of $\hat{\alpha}_n$ is a square.

Proof of Claim 1. We prove this claim by induction. For $n = 1$, $\hat{\alpha}_n = \mathtt{a}_1\, x$. The only nonempty prefixes of $\hat{\alpha}_n$ are $\hat{\alpha}_n$ itself and $\mathtt{a}_1$, neither of which is a square.

Now assume the claim holds for some $n \geq 1$ (i. e., no nonempty prefix of $\hat{\alpha}_n$ is a square). By definition, $\hat{\alpha}_{n+1} = \hat{\alpha}_n\, \mathtt{a}_{n+1}\, \hat{\alpha}_n x$. Due to the definition of $\hat{\alpha}_n$, we know that the letter $\mathtt{a}_{n+1}$ does not occur therein, and by the induction assumption, no nonempty prefix of $\hat{\alpha}_n$ is a square. Thus, the claim holds or $\hat{\alpha}_{n+1}$ as well. □ (Claim 1)

In order to state the next claim, for every $i \geq 1$, we define S_i to be the set of all nonerasing substitutions $\sigma : (\Sigma_\infty \cup X)^+ \to (\Sigma_\infty)^+$ for which the leftmost letter of $\sigma(x)$ is $\mathtt{a}_i$.

Claim 2. For every $n \geq 1$, every i with $1 \leq i \leq n$ and every $\sigma \in S_i$, $\sigma(\hat{\alpha}_n)$ has a nonempty prefix that is a square.

Proof of Claim 2. Again, we show the claim by induction. First, let $n = 1$. In this case, we only need to consider the case of $\sigma \in S_1$. For every such σ, there is a $w \in (\Sigma_\infty)^*$ such that $\sigma(x) = \mathtt{a}_1\, w$. Accordingly, as $\sigma(\hat{\alpha}_1) = \mathtt{a}_1\, \mathtt{a}_1\, w$, the claim holds.

Now assume that, for some $n \geq 1$ and all i with $1 \leq i \leq n$, $\sigma(\hat{\alpha}_n)$ has a nonempty prefix that is a square. As $\hat{\alpha}_n$ is a prefix of $\hat{\alpha}_{n+1}$, this implies that, for every $\sigma \in S_i$ with $1 \leq i \leq n$, $\sigma(\hat{\alpha}_{n+1})$ has a nonempty square as a prefix. Therefore, we only need to consider the substitutions $\sigma \in S_{n+1}$. For every such σ, there is a $w \in (\Sigma_\infty)^*$ such that $\sigma(x) = \mathtt{a}_{n+1}\, w$, and

$$\begin{aligned}\sigma(\hat{\alpha}_{n+1}) &= \sigma(\hat{\alpha}_n\, \mathtt{a}_{n+1}\, \hat{\alpha}_n x)\\ &= \sigma(\hat{\alpha}_n)\, \mathtt{a}_{n+1}\, \sigma(\hat{\alpha}_n)\, \mathtt{a}_{n+1}\, w.\end{aligned}$$

Thus, $(\sigma(\hat{\alpha}_n)\, \mathtt{a}_{n+1})^2$ is a (nonempty) prefix of $\sigma(\hat{\alpha}_{n+1})$. □ (Claim 2)

Now, for every $n \geq 1$, consider the terminal alphabet $\Sigma{:=}\{\mathtt{a}_1, \ldots, \mathtt{a}_n\}$ and the patterns $\alpha{:=}\alpha_n$ and $\beta{:=}xxy$ (where x and y are distinct variables).

For every word $w \in L_{\mathrm{NE},\Sigma}(\alpha)$, there is a nonerasing substitution $\sigma : (\Sigma \cup X)^+ \to \Sigma^+$ with $\sigma(\alpha) = w$. Therefore, $\sigma \in S_i$ for some i with $1 \leq i \leq n$, depending on the leftmost letter of $\sigma(x)$. By Claim 2, $\sigma(\hat{\alpha}_n)$ has a nonempty prefix that is a square; i. e., there are a $u \in \Sigma^+$ and a $v \in \Sigma^*$ such that $w = uuv$. We now define the substitution $\tau : (\Sigma \cup X)^+ \to \Sigma^+$ by $\tau(x){:=}u$ and $\tau(y){:=}v\, \mathtt{a}_n$. Thus, $\tau(\beta) = uuv\, \mathtt{a}_n = \sigma(\hat{\alpha}_n)\, \mathtt{a}_n = \sigma(\alpha_n) = w$, and $L_{\mathrm{NE},\Sigma}(\alpha) \subseteq L_{\mathrm{NE},\Sigma}(\beta)$.

On the other hand, assume that there is a nonerasing morphism $\phi : X^+ \to (\Sigma \cup X)^+$ with $\phi(\beta) = \alpha = \hat{\alpha}_n\, \mathtt{a}_n$. As ϕ is nonerasing, the rightmost letter of $\phi(y)$ must be $\mathtt{a}_n$. More formally, there is some $\gamma \in (\Sigma \cup X)^*$ with $\phi(y) = \gamma\, \mathtt{a}_n$. Thus, $\hat{\alpha}_n = (\phi(x))^2\gamma$; by definition of ϕ, this means that $\hat{\alpha}_n$ has a nonempty square as a prefix, which contradicts Claim 1. Therefore, no such ϕ exists. □

Thus, regardless of the size of $|\Sigma|$, even the inclusion problem of $\mathrm{nePAT}_{1,\Sigma}$ in $\mathrm{nePAT}_{2,\Sigma}$ is too complex to be characterized by the existence of a nonerasing terminal-preserving morphism between the patterns. A similar phenomenon can be observed for E-pattern languages:

Proposition 3.8. *For every finite alphabet Σ with $|\Sigma| \geq 2$, there are patterns $\alpha \in \mathrm{Pat}_{1,\Sigma}$ and $\beta \in \mathrm{Pat}_{2|\Sigma|+2,\Sigma}$ such that $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$, but there is no terminal-preserving morphism $\phi : (\Sigma \cup X)^* \to (\Sigma \cup X)^*$ with $\phi(\beta) = \alpha$.*

Proof. The patterns α and β can be directly obtained from the patterns in Reidenbach's proof of Theorem 6 in [32], by replacing each variable in α with a single variable x, and removing a common prefix.

Let $\Sigma = \{\mathtt{a}_1, \ldots, \mathtt{a}_n\}$ (where all $\mathtt{a}_i$ are distinct, i. e., $|\Sigma| = n$). Let $m{:=}n$ if n is odd, and $m{:=}n+1$ if n is even. If n is even, we also define $\mathtt{a}_m := \mathtt{a}_n$.

Next, we define

$$\alpha{:=}\, \mathtt{a}_1\, x\, \mathtt{a}_1\, x \cdot \mathtt{a}_2\, x\, \mathtt{a}_2\, x \cdot \ldots \cdot \mathtt{a}_m\, x\, \mathtt{a}_m\, x,$$
$$\beta{:=}\, \mathtt{a}_1\, \beta_1\, \mathtt{a}_1\, z_1 \cdot \mathtt{a}_2\, \beta_2\, \mathtt{a}_2\, z_2 \cdot \ldots \cdot \mathtt{a}_m\, \beta_m\, \mathtt{a}_m\, z_m,$$

with, for $1 \leq i \leq m$,

$$\beta_i{:=} \begin{cases} y_i y_{i+1} & \text{if } 1 \leq i < m, \\ y_n y_1 & \text{if } i = m, \end{cases}$$

where $y_1, z_1, \ldots, y_m, z_m$ are pairwise distinct variables.

In order to show $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$, we prove that, for every substitution σ, there is a substitution τ with $\tau(\beta) = \sigma(\alpha)$. If $\sigma(x) = \lambda$, it is easy to see that $\sigma(\alpha)$ can be created from β by erasing all variables. Therefore, we can safely assume $\sigma(x) = \mathtt{a}_j\, u$ with $1 \leq j \leq n$ and $u \in \Sigma^*$.

We define the substitution τ by

$$\tau(z_i){:=} \begin{cases} \sigma(x) & \text{if } i \neq j, \\ u\, \mathtt{a}_j\, \sigma(x) & \text{if } i = j, \end{cases}$$

for every $z_i \in \mathrm{var}(\beta)$, and by

$$\tau(y_i){:=} \begin{cases} \lambda & \text{if } i \in \mathrm{ERASE}_j, \\ a_j u & \text{if } i \notin \mathrm{ERASE}_j, \end{cases}$$

for every $y_i \in \mathrm{var}(\beta)$, where the set $\mathrm{ERASE}_j \subset \mathrm{var}(\beta)$ is defined as

$$\mathrm{ERASE}_j := \{y_s \in \mathrm{var}(\beta) \mid s = j - 2i \text{ or } s = j + 1 + 2i \text{ for some } i \geq 0\}.$$

Note that, due to our definition of ERASE_j and τ, $\tau(\beta_j) = \lambda$ and $\tau(\beta_i) = \sigma(x)$ for every $i \neq j$ hold, as ERASE_j contains exactly those x_s with either $s \leq j$, and s has the same parity as j, or $s > j$, where s and j have different parities.

In order to prove $\phi(\beta) = \sigma(\alpha)$, it suffices to show that $\phi(\mathtt{a}_i\, \beta_i\, \mathtt{a}_i\, z_i) = \sigma(\mathtt{a}_i\, x\, \mathtt{a}_i\, x)$ for every i with $1 \leq i \leq m$ – then the claim follows by definition of α and β.

For every i with $1 \leq i \leq m$ and $i \neq j$, we use $\tau(\beta_i) = \sigma(x)$ to conclude

$$\begin{aligned}\tau(\mathtt{a}_i\,\beta_i\,\mathtt{a}_i\,z_i) &= \mathtt{a}_i\,\sigma(x)\,\mathtt{a}_i\,\sigma(x)\\ &= \sigma(\mathtt{a}_i\,x\,\mathtt{a}_i\,x).\end{aligned}$$

Likewise, for the special case of $i = j$, $\tau(\beta_j) = \lambda$ leads to

$$\begin{aligned}\tau(\mathtt{a}_j\,\beta_j\,\mathtt{a}_j\,z_j) &= \mathtt{a}_j \cdot\lambda \cdot \mathtt{a}_j\,u \cdot \mathtt{a}_j\,\sigma(x)\\ &= \mathtt{a}_j\,\sigma(x)\,\mathtt{a}_j\,\sigma(x)\\ &= \sigma(\mathtt{a}_j\,x\,\mathtt{a}_j\,x).\end{aligned}$$

Thus, $\phi(\beta) = \sigma(\alpha)$, and – as σ was chosen freely – $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$.

We proceed to show that there is no terminal-preserving morphism $\phi : (\Sigma \cup X)^* \to (\Sigma \cup X)^*$ with $\phi(\beta) = \alpha$. Assume to the contrary that there is a terminal-preserving morphism ϕ with $\phi(\beta) = \alpha$. As α and β contain exactly the same occurrences of terminals, $\phi(\beta_i) = x$ and $\phi(z_i) = x$ must hold for every $i \in \{1, \ldots, m\}$. We define $\beta' := \beta_1 \cdot \ldots \cdot \beta_m$, and observe $\phi(\beta') = x^m$. By definition of β_i, $|\beta'|_{z_i} = 2$ for $1 \leq i \leq m$, and thus, $|\beta'|$ is even. This contradicts the fact that m (and, thus, $|x^m|$) is odd by definition. □

The proof also shows that, if Σ has an odd number of letters, the bound on the number of variables in the second class of patterns can be lowered to $2|\Sigma|$. We do not know whether this lower bound is strict, or if there are patterns $\alpha \in \mathrm{Pat}_{1,\Sigma}$, $\beta \in \mathrm{Pat}_{n,\Sigma}$ with $n < 2|\Sigma|$ such that $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$, but there is no terminal-preserving morphism mapping β to α.

For $|\Sigma| = 2$, according to Proposition 3.8, the inclusion of $\mathrm{ePAT}_{1,\Sigma}$ in $\mathrm{ePAT}_{6,\Sigma}$ is not characterized by the existence of such a morphism. As this bound (and the bound on NE-patterns from Proposition 3.7) are the lowest known bounds for 'morphism-free' inclusion, we want to emphasize the following problem:

Open Problem 3.9. *Let* $|\Sigma| = 2$. *Is the inclusion problem of* $\mathrm{ePAT}_{1,\Sigma}$ *in* $\mathrm{ePAT}_{6,\Sigma}$ *decidable? Is the inclusion problem of* $\mathrm{nePAT}_{1,\Sigma}$ *in* $\mathrm{nePAT}_{2,\Sigma}$ *decidable?*

In principle, both inclusion problems might be undecidable, although this conjecture is somewhat counterintuitive to the low bounds on the numbers of variables. Furthermore, even if these problems should be undecidable, the numbers of variables in the results further down in this chapter suggest that a radically different proof would be necessary.

On the other hand, the author considers these classes promising candidates for classes of pattern languages where the inclusion is decidable, but not characterized by the existence of an appropriate morphism.

As evidenced by our next main theorem, bounding the number of variables preserves the undecidability of the inclusion problem for E-pattern languages[4]:

Theorem 3.10. *Let* $|\Sigma| = 2$. *The following problems are undecidable:*

[4]Bremer has adapted Theorem 3.10 to NE-pattern languages over binary and larger alphabets, cf. Bremer [11] and Bremer and Freydenberger [12]. The resulting bounds are higher for each second class of patterns ("the βs"), but similar.

1. *The inclusion problem of* $\mathrm{ePAT}_{3,\Sigma}$ *in* $\mathrm{ePAT}_{2854,\Sigma}$,

2. *the inclusion problem of* $\mathrm{ePAT}_{2,\Sigma}$ *in* $\mathrm{ePAT}_{2860,\Sigma}$.

The proof of this theorem starts in Section 3.3.1 and continues in Section 3.3.2.

Both cases of the proof use the same basic approach as the proofs of the E-case in Theorems 3.2 and 3.3. We show that, for a given starting configuration I of the universal Turing machine $\mathcal{U}$, one can effectively construct patterns α, β in the appropriate classes of patterns such that $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$ if and only if $\mathcal{U}$ halts after starting in I. As this would decide the halting problem of the universal Turing machine $\mathcal{U}$, the inclusion problems must be undecidable.

Note that extensions to larger (finite) alphabets are discussed in Section 3.3.5. The bounds presented in this theorems are not optimal – through additional effort and some encoding tricks, it is possible to reduce each bound on the number of variables in the second pattern by a few hundred variables (cf. Section 3.3.2 for some remarks). As the resulting number would still be far away from the bounds presented in the theorems further down in this section, the author feels that these optimizations would only add additional complexity to the proofs, without providing deeper insight, and decided to give only the bounds present above, which are not as strict.

Although encoding the correct operation of a Turing machine (or any similar device) in patterns requires a considerable amount of variables, the simple structure of iterating the Collatz function $\mathcal{C}$ can be expressed in a more compact form. With far smaller bounds, we are able to obtain the following result using the same construction as for the proof of Theorem 3.10:

Theorem 3.11. *Let* Σ *be a binary alphabet. Every algorithm that decides the inclusion problem of* $\mathrm{ePAT}_{2,\Sigma}$ *in* $\mathrm{ePAT}_{74,\Sigma}$ *can be converted into an algorithm that, for every* $N \in \mathbb{N}_1$*, decides whether* N *leads* $\mathcal{C}$ *into the trivial cycle.*

The proof starts in Section 3.3.1 and continues in Section 3.3.3.

As mentioned in Section 3.2.2, this demonstrates that, even for these far tighter bounds, the inclusion relation of pattern languages can be used to express quite complicated sets. Moreover, a slight modification of the proof allows us to state the following far stronger result:

Theorem 3.12. *Let* Σ *be a binary alphabet. Every algorithm that decides the inclusion problem for* $\mathrm{ePAT}_{4,\Sigma}$ *in* $\mathrm{ePAT}_{80,\Sigma}$ *can be used to decide whether any number* $N \geq 1$ *leads* $\mathcal{C}$ *into a nontrivial cycle.*

The proof of this theorem starts in Section 3.3.1 and continues in Section 3.3.4.

This result needs to be interpreted very carefully. Of course, the existence of nontrivial cycles is trivially decidable (by a constant predicate); but these results are stronger than mere decidability, as the patterns are constructed effectively. Thus, deciding the inclusion of any of the two pairs of patterns defined in the proofs would allow us to prove the existence of a counterexample to the Collatz conjecture, or to rule out the existence of one important class of counterexamples, and thus solve 'one half' of the Collatz conjecture. More pragmatically, it is the author's opinion that these results give reason to suspect that the inclusion problems of these classes of pattern languages are probably not decidable (even if effectively, then not efficiently), and definitely very complicated.

The following table summarizes our findings on the difficulty of the inclusion problem for $\mathrm{ePAT}_{m,\Sigma}$ in $\mathrm{ePAT}_{n,\Sigma}$ for binary Σ:

m	n		
unb.	1	characterized by existence of a morphism	Theorem 3.5 ([50])
1	6	not characterized by existence of a morphism	Proposition 3.8
2	74	simulate Collatz iterations	Theorem 3.11
4	80	decide existence of non-trivial Collatz cycles	Theorem 3.12
2	2860	undecidable	Theorem 3.10
3	2854	undecidable	Theorem 3.10

Note that, as mentioned above, Theorems 3.10, 3.11 and 3.12 can also be extended to larger (finite) terminal alphabets Σ. Section 3.3.5 contains an extension that is comparatively straightforward (but leads to an increase in the number of variables in one of the patterns that depends on the size of Σ), and mentions a less straightforward extension by Bremer [11] (also contained in Bremer and Freydenberger [12]) that increases the number of variables only by 22 (independently from the size of Σ).

In addition to this, Bremer extended Theorems 3.10, 3.11 and 3.12 to NE-pattern languages (again, see [11] and [12]).

3.3.1 The Basic Construction

In this section, we describe the construction that is common to the proofs of Theorems 3.10, 3.11 and 3.12, and describe how the number of necessary variables can be derived from each actual instantiation of the construction. The proofs for Theorems 3.10, 3.11 and 3.12 use the material in the present section and continue in Section 3.3.2, 3.3.3 and 3.3.4, respectively.

As the construction is rather involved, we first give a basic sketch, before we consider the technical details of the basic construction common to all three proofs.

Sketch of the proof. In each of the proofs, our goal is to decide the emptiness of a set V, which is one of $\mathrm{ACCEPT}_{\mathcal{U}}(I)$ (for some configuration I), $\mathrm{TRIV}(N)$ (for some $N \geq 1$), or NTCC. For this, we construct two patterns α and β such that $L_{\mathrm{E},\Sigma}(\alpha) \setminus L_{\mathrm{E},\Sigma}(\beta) \neq \emptyset$ if and only if $V \neq \emptyset$. The pattern α contains two subpatterns α_1 and α_2, where α_2 is a terminal-free pattern with $\mathrm{var}(\alpha_2) \subseteq \mathrm{var}(\alpha_1) \cup \{y\}$, and y is a variable that occurs exactly once in α_2, but does not occur in α_1.

Glossing over details (and ignoring the technical role of α_2), the main goal is to define β in such a way that, for every substitution σ, $\sigma(\alpha) \in L_{\mathrm{E},\Sigma}(\beta)$ if and only if $\sigma(\alpha_1) \in \mathrm{V}$. More explicitly, the subpattern α_1 generates a set of possible strings, and β encodes a disjunction of predicates on strings that describe the complement of V through all possible errors. If one of these errors occurs in $\sigma(\alpha_1)$, we can construct a substitution τ with $\tau(\beta) = \sigma(\alpha)$. If $\mathrm{V} = \emptyset$, every $\sigma(\alpha)$ belongs to $L_{\mathrm{E},\Sigma}(\beta)$. Otherwise, any element of V can be used to construct a word $\sigma(\alpha) \notin L_{\mathrm{E},\Sigma}(\beta)$.

The actual proof. Let $\Sigma = \{0, \#\}$. As mentioned above, our goal is to construct two patterns α and β such that $L_{\mathrm{E},\Sigma}(\alpha) \setminus L_{\mathrm{E},\Sigma}(\beta) = \emptyset$ if and only if $V \neq \emptyset$; where V, is one of $\mathrm{TRIV}(N)$ (for some $N \geq 1$), NTCC, or $\mathrm{ACCEPT}_{\mathcal{U}}(I)$ (for some configuration I).

Basically, α generates a list of possible strings and provides some technical infrastructure, while β encodes a list of predicates π_1 to π_μ that describe all possible errors in the

strings generated by α by describing the complement of V. Due to the right choice of α and β, $L_{\mathrm{E},\Sigma}(\alpha) \subset L_{\mathrm{E},\Sigma}(\beta)$ holds if some word in $L_{\mathrm{E},\Sigma}(\alpha)$ satisfies none of the predicates.

Depending on which of the three proofs we want to actualize, we choose a structural parameter $\kappa \in \{2, 3\}$ and a $\mu \geq 4$. The parameter κ has two purposes: First, it determines the maximal number of parameters in each predicate, and second, if none of the predicates is satisfied, the encoded word must not contain a factor $\#^\kappa$.

In addition to this, also depending on the actual proof, we select patterns α_1 and α_2, where α_1 is a pattern that does not contain $\#^\kappa$ as a factor, and α_2 is a terminal-free pattern with $\mathrm{var}(\alpha_2) \subseteq \mathrm{var}(\alpha_1) \cup \{y\}$, where y is a variable that occurs exactly once in α_2, but does not occur in α_1.

We define

$$\alpha{:=}v\ v\ \#^4\ v\ \alpha_1\ v\ \alpha_2\ v\ \#^4\ v\ u\ v,$$

where $v{:=}0\#^3 0$ and $u{:=}0\#\#0$. The pattern α_1 will be used to generate the set of possible members of V, while α_2 serves more technical purposes.

Note that the construction in [32] can be seen as a special case of the present construction, by selecting $\alpha_1{:=}x$, $\alpha_2{:=}y$ and $\kappa{:=}3$. Our more general approach allows us to describe the intended starting and ending values of the encoded computation in α_1 without the use of additional predicates. Furthermore, as we shall see soon, the variables in $\mathrm{var}(\alpha_1) \cap \mathrm{var}(\alpha_2)$ provide us with greater control on the shape of the images of α_1.

Furthermore, let

$$\beta{:=}(x_1)^2 \dots (x_\mu)^2 \#^4 \hat{\beta}_1 \dots \hat{\beta}_\mu \#^4 \ddot{\beta}_1 \dots \ddot{\beta}_\mu,$$

with, for all $i \in \{1, \dots, \mu\}$, $\hat{\beta}_i{:=}x_i\,\gamma_i\,x_i\,\delta_i\,x_i$ and $\ddot{\beta}_i{:=}x_i\,\eta_i\,x_i$, where $x_1, \dots, x_\mu$ are pairwise distinct variables and all $\gamma_i, \delta_i, \eta_i \in X^*$ are terminal-free patterns. The patterns γ_i and δ_i shall be defined later; for now, we only mention:

1. $\eta_i{:=}z_i(\hat{z}_i)^2 z_i$ and $z_i \neq \hat{z}_i$ for all $i \in \{1, \dots, \mu\}$,
2. $\mathrm{var}(\gamma_i\delta_i\eta_i) \cap \mathrm{var}(\gamma_j\delta_j\eta_j) = \emptyset$ for all $i, j \in \{1, \dots, \mu\}$ with $i \neq j$,
3. $x_k \notin \mathrm{var}(\gamma_i\delta_i\eta_i)$ for all $i, k \in \{1, \dots, \mu\}$.

Thus, for every i, the elements of $\mathrm{var}(\gamma_i\delta_i\eta_i)$ appear nowhere but in these three factors. Let H be the set of all substitutions $\sigma : (\Sigma \cup \mathrm{var}(\alpha_1\alpha_2))^* \to \Sigma^*$. We interpret each triple $(\gamma_i, \delta_i, \eta_i)$ as a predicate $\pi_i : H \to \{0, 1\}$ in such a way that $\sigma \in H$ *satisfies* π_i if there exists a morphism $\tau : \mathrm{var}(\gamma_i\delta_i\eta_i)^* \to \Sigma^*$ with $\tau(\gamma_i) = \sigma(\alpha_1)$, $\tau(\delta_i) = \sigma(\alpha_2)$ and $\tau(\eta_i) = u$. As we shall see, $L_{\mathrm{E},\Sigma}(\alpha) \setminus L_{\mathrm{E},\Sigma}(\beta)$ exactly contains those $\sigma(\alpha)$ for which σ does not satisfy any of π_1 to π_μ. Our goal is a selection of predicates that describe the complement of V, where the predicates π_4 to π_μ provide an exhaustive list of sufficient criteria for 'non-membership' in V. We continue with further technical preparations.

A substitution σ is of *κ-bad form* if $\sigma(\alpha_1)$ contains $\#^\kappa$ as a factor, or if $\sigma(\alpha_2)$ contains $\#$. Otherwise, σ is of *κ-good form*. For $\kappa = 3$, this notion is equivalent to the concept of bad form and good form in [32].

The predicates π_1 and π_2 describe the cases where σ is of κ-bad form and are defined by

$$\gamma_1{:=}y_{1,1}(\hat{z}_1)^\kappa y_{1,2}, \qquad \gamma_2{:=}y_2,$$
$$\delta_1{:=}\hat{y}_1, \qquad \delta_2{:=}\hat{y}_{2,1}\ \hat{z}_2\ \hat{y}_{2,2},$$

where $y_{1,1}$, $y_{1,2}$, y_2, $\hat{y}_1$, $\hat{y}_{2,1}$, $\hat{y}_{2,2}$, $\hat{z}_1$ and $\hat{z}_2$ are pairwise distinct variables.

Recall that $\eta_i = z_i(\hat{z}_i)^2 z_i$ for all i. It is not difficult to see that π_1 and π_2 characterize the morphisms that are of κ-bad form:

Lemma 3.13. *A substitution $\sigma \in H$ is of κ-bad form if and only if σ satisfies π_1 or π_2.*

Proof. We begin with the *only if* direction. If $\sigma(\alpha_1) = w_1\#^\kappa w_2$ for some $w_1, w_2 \in \Sigma^*$, choose $\tau(y_{1,1}){:=}w_1$, $\tau(y_{1,2}){:=}w_2$, $\tau(\hat{z}_1){:=}\#$, $\tau(\hat{y}_1){:=}\sigma(\alpha_2)$ and $\tau(z_1){:=}0$. Then $\tau(\gamma_1) = \sigma(\alpha_1)$, $\tau(\delta_1) = \sigma(\alpha_2)$ and $\tau(\eta_1) = u$; thus, σ satisfies π_1.

If $\sigma(\alpha_2) = w_1\#w_2$ for some $w_1, w_2 \in \Sigma^*$, let $\tau(y_2){:=}\sigma(\alpha_1)$, $\tau(\hat{y}_{2,1}){:=}w_1$, $\tau(\hat{y}_{2,2}){:=}w_2$ and $\tau(\hat{z}_2){:=}\#$, and $\tau(z_2){:=}0$. It is easy to see that σ satisfies π_2.

For the *if* direction, if σ satisfies π_1, then there exists a morphism τ with $\tau(\gamma_1) = \sigma(\alpha_1)$ and $\tau(\eta_1) = 0\#^2 0$. Thus, $\tau(\hat{z}_1) = \#$ and $\tau(z_1) = 0$ must hold. Then, by definition of γ_1, $\sigma(\alpha_1) = \tau(y_{1,1})\#^\kappa\tau(y_{1,2})$, which means that σ is of κ-bad form.

Analogously, if σ satisfies π_2, then $\sigma(\alpha_2)$ contains the letter $\#$, and σ is of κ-bad form. □

Note that, if σ is of good form, $\sigma(x) \in 0^*$ for all variables $x \in \mathrm{var}(\alpha_1) \cap \mathrm{var}(\alpha_2)$. Thus, these variables provide us with greater control on the shape of $\sigma(\alpha_1)$ for the remaining predicates.

Now, Lemma 3.13 leads us to the central part of the construction:

Lemma 3.14. *For every substitution $\sigma \in H$, $\sigma(\alpha) \in L_{\mathrm{E},\Sigma}(\beta)$ if and only if σ satisfies one of the predicates π_1 to π_μ.*

Proof. We begin with the *if* direction. Assume $\sigma \in H$ satisfies some predicate π_i. Then there exists a morphism $\tau : (\mathrm{var}(\gamma_i\delta_i\eta_i))^* \to \Sigma^*$ such that $\tau(\gamma_i) = \sigma(\alpha_1)$, $\tau(\delta_i) = \sigma(\alpha_2)$ and $\tau(\eta_i) = u$. We extend τ to a substitution τ' defined by $\tau'(0){:=}0$, $\tau'(\#){:=}\#$, and, for all $x \in \mathrm{var}(\beta)$,

$$\tau'(x){:=}\begin{cases} \tau(x) & \text{if } x \in \mathrm{var}(\gamma_i\delta_i\eta_i), \\ v & \text{if } x = x_i, \\ \lambda & \text{in all other cases.} \end{cases}$$

By definition, none of the variables in $\mathrm{var}(\gamma_i\delta_i\eta_i)$ appears outside of these factors. Thus, τ' can always be defined this way. We obtain

$$\begin{aligned} \tau'(\hat{\beta}_i) &= \tau'(x_i\,\gamma_i\,x_i\,\delta_i\,x_i) \\ &= v\,\tau(\gamma_i)\,v\,\tau(\delta_i)\,v \\ &= v\,\sigma(\alpha_1)\,v\,\sigma(\alpha_2)\,v, \\ \tau'(\ddot{\beta}_i) &= \tau'(x_i\,\eta_i\,x_i) \\ &= v\,\tau(\eta_i)\,v \\ &= v\,u\,v. \end{aligned}$$

As $\tau'(\gamma_j) = \tau'(\delta_j) = \tau'(\eta_j) = \tau'(\hat{\beta}_j) = \tau'(\ddot{\beta}_j) = \lambda$ for all $j \neq i$, this leads to

$$\begin{aligned} \tau'(\beta) &= \tau'\left((x_1)^2 \dots (x_\mu)^2\#^4\hat{\beta}_1 \dots \hat{\beta}_\mu\#^4\ddot{\beta}_1 \dots \ddot{\beta}_\mu\right) \\ &= \tau'\left((x_i)^2\right)\#^4\tau'(\hat{\beta}_i)\#^4\tau'(\ddot{\beta}_i) \\ &= v\,v\,\#^4\,v\,\sigma(\alpha_1)\,v\,\sigma(\alpha_2)\,v\,\#^4\,v\,u\,v \\ &= \sigma(\alpha). \end{aligned}$$

This proves $\sigma(\alpha) \in L_{\mathrm{E},\Sigma}(\beta)$.

For the other direction, assume $\sigma(\alpha) \in L_{\mathrm{E},\Sigma}(\beta)$. If σ is of κ-bad form, then by Lemma 3.13, σ satisfies π_1 or π_2. Thus, assume $\sigma(\alpha_1)$ does not contain $\#^\kappa$ as a factor, and $\sigma(\alpha_2) \in 0^*$. Let τ be a substitution with $\tau(\beta) = \sigma(\alpha)$.

Now, as σ is of κ-good form, $\sigma(\alpha)$ contains exactly two occurrences of $\#^4$, and these are non-overlapping. As $\sigma(\alpha) = \tau(\beta)$, the same holds for $\tau(\beta)$. Thus, the equation $\sigma(\alpha) = \tau(\beta)$ can be decomposed into the system consisting of the following three equations:

$$0\#^3 0\ 0\#^3 0 = \tau\left((x_1)^2 \dots (x_\mu)^2\right), \tag{3.1}$$

$$0\#^3 0\ \sigma(\alpha_1)\ 0\#^3 0\ \sigma(\alpha_2)\ 0\#^3 0 = \tau(\hat{\beta}_1 \dots \hat{\beta}_\mu), \tag{3.2}$$

$$0\#^3 0\ u\ 0\#^3 0 = \tau(\ddot{\beta}_1 \dots \ddot{\beta}_\mu). \tag{3.3}$$

First, consider equation (3.1) and choose the smallest i for which $\tau(x_i) \neq \lambda$. Then $\tau(x_i)$ has to start with 0, and as

$$\tau\left((x_i)^2 \dots (x_\mu)^2\right) = 0\#^3 0\ 0\#^3 0,$$

it is easy to see that $\tau(x_i) = 0\#^3 0 = v$ and $\tau(x_j) = \lambda$ for all $j \neq i$ must hold.

Note that u does not contain $0\#^3 0$ as a factor, and does neither begin with $\#^3 0$, nor end on $0\#^3$. But as $\tau(\ddot{\beta}_i)$ begins with and ends on $0\#^3 0$, we can use equation (3.3) to obtain $0\#^3 0\ u\ 0\#^3 0 = \tau(\ddot{\beta}_i)$ and $\tau(\ddot{\beta}_j) = \lambda$ for all $j \neq i$. As $\ddot{\beta}_i = x_i \eta_i x_i$ and $\tau(x_i) = 0\#^3 0$, $\tau(\eta_i) = u$ must hold.

As σ is of κ-good form, $\sigma(0\#^3 0 \alpha_1 0\#^3 0 \alpha_2 0\#^3 0)$ contains exactly three occurrences of $\#^3$. But there are already three occurrences of $\#^3$ in $\tau(\hat{\beta}_i) = 0\#^3 0\ \tau(\gamma_i)\ 0\#^3 0\ \tau(\delta_i)\ 0\#^3 0$. This, and equation (3.2), lead to $\tau(\hat{\beta}_j) = \lambda$ for all $j \neq i$ and, more importantly, $\tau(\gamma_i) = \sigma(\alpha_1)$ and $\tau(\delta_i) = \sigma(\alpha_2)$. Therefore, σ satisfies the predicate π_i. □

Thus, we can select predicates π_1 to π_μ in such a way that $L_{\mathrm{E},\Sigma}(\alpha) \setminus L_{\mathrm{E},\Sigma}(\beta) = \emptyset$ if and only if $\mathrm{V} = \emptyset$ by describing $\overline{\mathrm{V}}$ through a disjunction of predicates on H. The proof of Lemma 3.14 shows that if $\sigma(\alpha) = \tau(\beta)$ for substitutions σ and τ; where σ is of κ-good form, there exists exactly one i $(3 \leq i \leq \mu)$ such that $\tau(x_i) = 0\#^3 0$.

Due to technical reasons, we need a predicate π_3 that, if unsatisfied, sets a lower bound to the length of $\sigma(\alpha_2)$, defined by

$$\gamma_3 := y_{3,1}\ \hat{y}_{3,1}\ y_{3,2}\ \hat{y}_{3,2}\ y_{3,3},$$
$$\delta_3 := \hat{y}_{3,1}\ \hat{y}_{3,2},$$

if $\kappa = 2$, or by

$$\gamma_3 := y_{3,1}\ \hat{y}_{3,1}\ y_{3,2}\ \hat{y}_{3,2}\ y_{3,3}\ \hat{y}_{3,3}\ y_{3,4},$$
$$\delta_3 := \hat{y}_{3,1}\ \hat{y}_{3,2}\ \hat{y}_{3,3},$$

if $\kappa = 3$; where in either case all of $y_{3,1}$ to $y_{3,4}$ and $\hat{y}_{3,1}$ to $\hat{y}_{3,3}$ are pairwise distinct variables.

Clearly, if some $\sigma \in H$ satisfies π_3, $\sigma(\alpha_2)$ is a concatenation of κ (possibly empty) factors of $\sigma(\alpha_1)$. Thus, if σ satisfies none of π_1 to π_3, $\sigma(\alpha_2)$ has to be longer than the κ longest non-overlapping sequences of 0s in $\sigma(\alpha_1)$. This allows us to identify a class of predicates definable by a rather simple kind of expression, which we use to define π_4 to π_μ in a less technical way. Note that any meaningful use of this construction requires α_2

to contain at least one variable that does not occur in α_1, as otherwise, π_3 would always be satisfied.

Let $X_\kappa := \{\hat{x}_1, \ldots, \hat{x}_\kappa\} \subset X$, let G_κ denote the set of those substitutions in H that are of κ-good form and let R be the set of all substitutions $\rho : (\Sigma \cup X_\kappa)^* \to \Sigma^*$ for which $\rho(\hat{x}_i) \in 0^*$ for all i with $1 \leq i \leq \kappa$. For patterns $\zeta \in (\Sigma \cup X_\kappa)^*$, we define $R(\zeta) := \{\rho(\zeta) \mid \rho \in R\}$.

Definition 3.15. *A predicate $\pi : G_\kappa \to \{0,1\}$ is called a* κ-simple predicate for α_1 *if there exist a pattern $\zeta \in (\Sigma \cup X_\kappa)^*$ and languages $L_1, L_2 \in \{\Sigma^*, \{\lambda\}\}$ such that a substitution σ satisfies π if and only if $\sigma(\alpha_1) \in L_1\, R(\zeta)\, L_2$. If $L_1 = L_2 = \Sigma^*$, we call π an* factor-predicate. *If only $L_1 = \Sigma^*$ and $L_2 = \{\lambda\}$, π is called a* suffix-predicate, *and if $L_1 = \{\lambda\}$ and $L_2 = \Sigma^*$, a* prefix-predicate.

From a slightly different point of view, the elements of X_κ can be understood as numerical parameters describing (concatenational) powers of 0, with substitutions $\rho \in R$ acting as assignments. For example, if $\sigma \in G_\kappa$ satisfies a κ-simple predicate π if and only if $\sigma(\alpha_1) \in \Sigma^* R(\#\hat{x}_1\#\hat{x}_2 0\#\hat{x}_1)$, we can also write that σ satisfies π if and only if $\sigma(\alpha_1)$ has a suffix of the form $\#0^m\#0^n0\#0^m$ (with $m, n \in \mathbb{N}_0$), which could also be written as $\#0^m\#0^*0\#0^m$, as n occurs only once in this expression. Although these predicates do not explicitly allow arithmetical operations on the numerical parameters, we use expressions like 0^{m+2n+1} as a shorthand for $0^m0^n0^n0$.

As in the original construction, the predicate π_3 allows us to express all κ-simple predicates:

Lemma 3.16. *For every κ-simple predicate π_S having n numerical parameters with $n \leq \kappa$, there exists a predicate π defined by terminal-free patterns γ, δ, η such that for all substitutions $\sigma \in G_\kappa$:*

1. *if σ satisfies π_S, then σ also satisfies π or π_3,*

2. *if σ satisfies π, then σ also satisfies π_S.*

Proof. We first consider the case of $L_1 = L_2 = \Sigma^*$. Assume π_S is a κ-simple predicate, and $\zeta \in (\Sigma \cup X_\kappa)^*$ is a pattern such that $\sigma \in G_\kappa$ satisfies π_S if and only if $\sigma(\alpha_1) \in L_1 R(\zeta) L_2$. Then define $\gamma := y_1\, \zeta'\, y_2$, where ζ' is obtained from ζ by replacing all occurrences of 0 with a new variable z and all occurrences of # with a different variable $\hat{z}$, while leaving all present elements of X_κ unchanged. Furthermore, $\delta := \hat{x}_1 \ldots \hat{x}_\kappa \hat{y}$. Finally, in order to stay consistent with the η_i appearing in β, let $\eta := z(\hat{z})^2 z$. Note that $\hat{x}_1$, $\hat{x}_2$, $\hat{x}_3$, y_1, y_2, z and $\hat{z}$ are pairwise distinct variables.

Now, assume $\sigma \in G_\kappa$ satisfies π_S. Then there exist words $w_1, w_2 \in \Sigma^*$ and a substitution $\rho \in R$ such that $\sigma(\alpha_1) = w_1 \rho(\zeta) w_2$. If $\sigma(\alpha_2)$ is not longer than any κ non-overlapping factors of the form 0^* of $\sigma(\alpha_1)$ combined, π_3 is satisfied. Otherwise, we can define τ by setting $\tau(y_1) := w_1$, $\tau(y_2) := w_2$, $\tau(z) := 0$, $\tau(\hat{z}) := \#$, $\tau(\hat{x}_i) := \rho(\hat{x}_i)$ for all $i \in \{1, \ldots, k\}$ where $\hat{x}_i$ appears in ζ and $\tau(\hat{x}_i) := \lambda$ where $\hat{x}_i$ does not appear in ζ. Finally, let $\tau(\hat{y}) := 0^m$, where

$$m := |\sigma(\alpha_2)| - \sum_{\hat{x} \in \mathrm{var}(\zeta)} |\tau(\hat{x})|$$

($m > 0$ holds, as σ does not satisfy π_3). Then $\tau(\zeta') = \rho(\zeta)$, and

$$\begin{aligned}\tau(\gamma) &= \tau(y_1)\,\tau(\zeta')\,\tau(y_2)\\ &= w_1\,\rho(\zeta)\,w_2 = \sigma(\alpha_1),\\ \tau(\delta) &= 0^{|\sigma(\alpha_2)|} = \sigma(\alpha_2),\\ \tau(\eta) &= \tau(z\,(\hat{z})^2\,z)\\ &= 0\#\#0 = u.\end{aligned}$$

Therefore, σ satisfies π, which concludes this direction.

For the other direction, assume $\sigma \in G_\kappa$ satisfies π. Then there is a morphism τ such that $\sigma(\alpha_1) = \tau(\gamma)$, $\sigma(\alpha_2) = \tau(\delta)$ and $\tau(\eta) = u$. As $\eta = z\,(\hat{z})^2\,z$ and $u = 0\#\#0$, $\tau(z) = 0$ and $\tau(\hat{z}) = \#$ must hold. By definition $\tau(y_1), \tau(y_2) \in \Sigma^*$. If we define $\rho(\hat{x}_i) := \tau(\hat{x}_i)$ for all $\hat{x}_i \in \mathrm{var}(\delta)$, we see that $\sigma(\alpha_1) \in L_1 R(\zeta) L_2$ holds. Thus, σ satisfies π_S as well.

The other three cases for choices of L_1 and L_2 can be handled analogously by omitting y_1 or y_2 as needed. Note that this proof also works in the case $\zeta = \lambda$. □

Intuitively, if σ does not satisfy π_3, then $\sigma(\alpha_2)$ (which is in 0^*, due to $\sigma \in G_\kappa$) is long enough to provide building blocks for κ-simple predicates using variables from X_κ.

All that remains for each of the proofs is to choose an appropriate set of predicates.

Then it is easy to see how many variables each predicate in β requires. First, every predicate π_i has a corresponding variable x_i, for μ variables in total. The predicates π_1 and π_2 each use five further variables, π_3 uses $2\kappa + 3$ additional variables. In total, β contains $\mu + 2\kappa + 13$ variables for the predicates π_1 to π_3 and the variables x_i, and the additional variables that are required to encode the remaining predicates π_4 to π_μ.

Each of these predicates requires:

1. three variables for y_i, z_i and $\hat{z}_i$,
2. one variable for each numerical parameter (or occurrence of 0^*),
3. one additional variable if it is a prefix or a suffix predicate,
4. two additional variables if it is a factor predicate.

Thus, each predicate requires at least 3 and at most 8 variables.

3.3.2 Undecidability (Proof of Theorem 3.10)

For both claims of the proof, we show that, given any configuration I of $\mathcal{U}$, we can construct patterns α and β from the appropriate classes such that $L_{\mathrm{E},\Sigma}(\alpha) \setminus L_{\mathrm{E},\Sigma}(\beta) = \emptyset$ if and only if $\mathrm{ACCEPT}_{\mathcal{U}}(I) = \emptyset$. The predicates for the proofs of the two claims of this theorem are very similar, they differ only at the choice of α_1 and α_2, and an additional predicate that is required for the second case. For the first claim, we choose $\mu := 333$, for the second, $\mu := 334$. In either case, we choose $\kappa = 3$.

For the first claim of the theorem, we define

$$\alpha_1 := \#\#\,\mathrm{enc}_{\mathcal{U}}(I)\#\#x_1\#00x_2x_2\#0^{10}\#\#, \qquad \alpha_2 := x_2y,$$

where x_1, x_2 and y are pairwise distinct variables; for the second,

$$\alpha_1 := \#\# \operatorname{enc}_{\mathcal{U}}(I) \#\# x \# 0^{10} \#\#, \qquad \alpha_2 := y,$$

where x and y are distinct variables. Ultimately, if $\sigma(\alpha) \notin L_{\mathrm{E},\Sigma}(\beta)$, $\sigma(\alpha_1)$ is supposed to contain an encoding of a valid computation that starts in the configuration I, and leads to an accepting configuration. The variable x_2 in the subpattern α_1 of the first claim will have an image from 0^*, which means that the left tape of the final configuration has an odd encoding, and thus contains 1, while the machine is in state q_{10}. For the second claim, this condition will be checked by an additional predicate, which requires 6 additional variables in β.

Our first intermediate goal is a set of predicates that (if unsatisfied) forces $\sigma(\alpha_1)$ into a basic shape common to all elements of $\mathrm{ACCEPT}_{\mathcal{U}}(I)$. In other words, we want to remove all cases where

$$\sigma(\alpha_1) \notin (\#\#0^+\#0^+\#0^+)^+\#\#,$$

or $\sigma(\alpha_1)$ contains a factor $0^{16}\#\#$ and thus, an encoding of a state q_n with $n > 15$ (such a state does not exist in $\mathcal{U}$).

To achieve this goal, we define predicates π_4 to π_7 by κ-simple predicates as follows:

$$\begin{aligned}
&\pi_4 : \sigma(\alpha_1) \text{ contains a factor } \#\#0^+\#\#,\\
&\pi_5 : \sigma(\alpha_1) \text{ contains a factor } \#\#0^+\#0^+\#\#,\\
&\pi_6 : \sigma(\alpha_1) \text{ contains a factor } \#\#0^+\#0^+\#0^+\#0,\\
&\pi_7 : \sigma(\alpha_1) \text{ contains a factor } 0^{16}\#\#.
\end{aligned}$$

Due to Lemma 3.16, the predicates π_1 to π_7 do not strictly give rise to a characterization of substitutions with images that are not an encoding of a sequence of configurations of $\mathcal{U}$, as there are $\sigma \in G_\kappa$ where $\sigma(\alpha_1)$ is of the right shape, but π_3 is satisfied due to $\sigma(\alpha_2)$ being too short. But this problem can be avoided by choosing $\sigma(\alpha_2)$ long enough to leave π_3 unsatisfied.

Thus, if σ satisfies none of the predicates π_1 to π_7, $\sigma(\alpha_1)$ is an encoding of a sequence of configurations of $\mathcal{U}$ that starts with I, and ends in a halting configuration (for the first claim we prove), or a configuration in state q_{10} (for the second claim).

The remaining predicates will describe all errors where one of the encoded configurations is not a valid successor of its preceding configuration[5]. We will first consider all errors in state transitions, and then all errors in the tape contents.

In principle, we could now define predicates that, for every state $q_i \in Q$, every input letter $a \in \Gamma$, list all states that are not the successor state of q_i on input a. In order to save predicates (and thereby variables), our approach is a little bit more involved. Every state has at most two legal successor states, and the states q_6, q_{10} and q_{15} have only one successor. Thus, we can first exclude forbidden successor states regardless of the input letter, and then handle the few remaining cases. Furthermore, we are able to express the fact that a successor state has a larger number than possible.

[5]Note that, at this point, the construction uses 5 factor predicates (in addition to π_1 to π_3); one for each possible number of numerical parameters from 0 to 3. Even this small number of predicates requires 52 variables in β, and is only able to express the basic shape of encoded configurations.

$$S = \begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}$$

Figure 3.3: The matrix S, listing the possible (and impossible) immediate successors and predecessors of the states. Lines denote the sets of impossible state pairs that are described by the predicates π_8 to π_{34}. The remaining occurrences of 0 are handled by the predicates π_{35} to π_{66}.

In order to determine a good choice of predicates, it helps to visualize the relations of possible predecessor and successor states in a matrix. We define the 15×15 matrix $S = (s_{i,j})_{i,j=1}^{15}$ by

$$s_{i,j} := \begin{cases} 1 & \text{if there is an } a \in \Gamma \text{ with } \delta(q_i, a) = q_j, \\ 0 & \text{otherwise.} \end{cases}$$

For a graphical representation of S and the predicates that are derived from it, see Figure 3.3. Intuitively, $s_{i,j}$ equals 0 if and only if q_j can never be a valid immediate successor of q_j, regardless of the input letter.

First, we construct a predicate

$$\pi_8 : \sigma(\alpha_1) \text{ contains a factor } \#0^1\#\#0^+\#0^+\#0^3.$$

This predicates handles all cases where the encoding contains a configuration with state q_1, where the next state is some q_j with $j \geq 3$. In the same spirit, we can define a predicate that handles all configurations where q_1 is preceded by a state q_j with $j \geq 10$, which is also impossible in a valid computation:

$$\pi_9 : \sigma(\alpha_1) \text{ contains a factor } 0^{10}\#\#0^+\#0^+\#0^1\#.$$

Intuitively, π_8 describes all occurrences of 0 in the first row of S, while π_9 describes the bottom block of 6 occurrences of 0 in the first column.

We define similar predicates π_{10} to π_{33} for all states q_2 to q_{13}; each predicate handles the longest continuous block of 0s when reading a row from the right, or a column from the bottom.

Using the matrix S it is easy to see that this is not possible for q_{14}, as this state has q_{15} as successor and as predecessor. Similarly, the state q_{15} is handled by a single predicate

$$\pi_{34} : \sigma(\alpha_1) \text{ contains a factor } \#0^{15}\#\#0^{+}\#0^{+}\#0^{15}\#$$

that describes the lone 0 in the bottom right corner of S. Each of the 27 predicates π_8 to π_{34} is a factor predicate with 2 numerical parameters.

It seems like reordering the states could transform the matrix and reduce the number of predicates for single occurrences of 0. But after some experimentation, the author and Joachim Bremer decided that the expected small savings would not warrant the considerable effort. Further savings might be achieved by the use of an optimized machine with a different matrix; although (as we shall see) the general overhead that is necessary to encode computations makes the author doubt that these savings would be high enough to motivate the search for a machine with an undecidable halting problem that is minimal in this respect.

There are still 32 occurrences of 0 that have at least one 1 between them and the right side or the bottom of S. Thus, for each $s_{i,j}$ with this property, we define a predicate

$$\pi_k : \sigma(\alpha_1) \text{ contains } \#0^{i}\#\#0^{+}\#0^{+}\#0^{j}\#$$

for an appropriate k. This leads to the 32 predicates π_{35} to π_{66}, also factor predicates with 2 numerical parameters.

Now, only 24 possible errors need to be considered. For every state $q_i \in Q \setminus \{q_6, q_{10}, q_{15}\}$, and every input letter $a \in \Gamma$, we need to describe the error that the succeeding state is the one possible successor state that would have been reached from q_i by reading the complement of a. This leads to the predicates π_{67} to π_{90}; as an example, we define the two predicates that handle the invalid successor states of q_1:

$$\begin{aligned} &\pi_{67} : \sigma(\alpha_1) \text{ contains } \#00^{2m}\#0^{1}\#\#0^{+}\#0^{+}\#0^{1}\#; && m \in \mathbb{N}_0, \\ &\pi_{68} : \sigma(\alpha_1) \text{ contains } \#00^{2m+1}\#0^{1}\#\#0^{+}\#0^{+}\#0^{2}\#; && m \in \mathbb{N}_0. \end{aligned}$$

The first of these two predicates describes all cases where the machine is in the state q_1, reads 0 (as $\mathrm{enc}_{\mathrm{X}}(t_{aL}) \bmod 2 = 0 = \mathrm{e}(0)$) and stays in the state q_1, while π_{68} describes all cases where the machine transitions to q_2 upon reading 1 in state q_1.

No such predicates are required for the states q_6 and q_{15}, as these have only one possible successor state. As we permitted the machine to continue working after reaching a halting computation, the same applies to q_{10}. The 24 predicates π_{67} to π_{90} are factor predicates with three numerical parameters (as the starts count as numerical parameters that occur only once).

Thus, if σ satisfies none of the predicates π_1 to π_{90}, $\sigma(\alpha_1)$ encodes a sequence of configurations that starts with the initial configuration I and ends on the state q_{10} (as mentioned before, we also know that in the proof of the first claim, the final configuration is an accepting configuration, but this fact will be discussed later). Furthermore, we know that all transitions of the states are correct. Therefore, all that remains is to define a set of predicates that handle errors in the manipulation of the tape.

For this, we need to distinguish between left movements and right movements. Before we proceed to the definition of the predicates for tape errors in each of these cases, we

take a closer look at the intended behavior of valid computations, and their encodings in $\mathrm{ACCEPT}_{\mathcal{U}}(I)$. Assume $\mathcal{U}$ is in some state q_i, while the tape contains t_{aL} on the left and t_R on the right side. Let a denote the input letter, i. e., $\mathrm{e}(a) = (\mathrm{e}(t_{aL}) \bmod 2)$. Let t'_{aL} and t'_R denote the left and the right tape side of the succeeding valid configuration, respectively.

First, consider the case that $\delta(q_i, a) = (d, L, q_j)$ for some state $q_j \in Q$ and an output letter $d \in \Gamma$. In this case, according to Observation 3.1,

$$\begin{aligned} \mathrm{e}(t'_{aL}) &= \mathrm{e}(t_{aL}) \operatorname{div} 2, \\ \mathrm{e}(t'_R) &= 2(\mathrm{e}(t_R)) + \mathrm{e}(d). \end{aligned}$$

Thus, every tape error can be understood as a difference between the supposed e-value of the encoded side, and the actual e-value. As we shall see, all these differences can be described by a finite number of simple predicates, simulating arithmetic operations with the numerical parameters.

We begin with predicates for values that are too large, which can be defined with less effort than for too small values. For some appropriate $k > 90$, define the predicates

$$\begin{aligned} \pi_k &: \sigma(\alpha_1) \text{ contains } \#00^{2m+\mathrm{e}(a)}\#0^i\#\#0^+\#00^{m+1}; && m \in \mathbb{N}_0, \\ \pi_{k+1} &: \sigma(\alpha_1) \text{ contains } \#00^m\#00^{2n+\mathrm{e}(a)}\#0^i\#\#00^{2m+\mathrm{e}(d)+1}; && m, n \in \mathbb{N}_0. \end{aligned}$$

These capture all cases where, upon reading a in state q_i, the left or the right side of the tape (respectively) in the succeeding configuration contains more than it is supposed to (more meaning that its image under e is larger).

The following predicate describes all cases where the encoding of the left side of the tape is too small:

$$\pi_{k+2} : \sigma(\alpha_1) \text{ contains } \#00^{2(m+n+1)+\mathrm{e}(a)}\#0^i\#\#0^+\#00^m\#; \qquad m, n \in \mathbb{N}_0.$$

We capture the same case for the right side of the tape by the following two cases:

$$\begin{aligned} \pi_{k+3} &: \sigma(\alpha_1) \text{ contains } \#00^{2m+\mathrm{e}(a)}\#0^i\#\#00^{2n+(1-\mathrm{e}(d))}\#; && m, n \in \mathbb{N}_0, \\ \pi_{k+4} &: \sigma(\alpha_1) \text{ contains } \#00^{l+m+1}\#00^{2n+\mathrm{e}(a)}\#0^i\#\#00^{2m+\mathrm{e}(d)}\#; && l, m, n \in \mathbb{N}_0. \end{aligned}$$

As $\mathrm{e}(t'_R) = 2(\mathrm{e}(t_R)) + \mathrm{e}(d)$ holds, we know that every case with $\mathrm{e}(t'_R) \bmod 2 \neq \mathrm{e}(d)$ contains an error, which is described by π_{k+3}. Assuming that this predicate is not satisfied, we can use π_{k+4} to capture all cases where $\mathrm{e}(t'_R) \bmod 2$ equals $\mathrm{e}(d) \bmod 2$, but is too small.

This concludes the definitions of tape errors for L movements. Every combination of q_i and a that results in an L-movement requires 5 factor predicates π_k to π_{k+4}; the first two use 2 parameters, the other three use 3 parameters. In total, $\mathcal{U}$ has 15 combinations (q_i, a) that lead to an L-movement. Therefore, we need 75 predicates for tape errors of L-movements, which brings us to an intermediate total of 165 predicates.

Next, assume $\delta(q_i, a) = (d, R, q_j)$ for some state $q_j \in Q$ and an output letter $d \in \Gamma$. Recall that, according to Observation 3.1,

$$\begin{aligned} \mathrm{e}(t'_{aL}) &= 2(2(\mathrm{e}(t_{aL}) \operatorname{div} 2) + \mathrm{e}(d)) + (\mathrm{e}(t_R) \bmod 2) \\ &= 4(\mathrm{e}(t_{aL}) \operatorname{div} 2) + 2\,\mathrm{e}(d) + (\mathrm{e}(t_R) \bmod 2), \\ \mathrm{e}(t'_R) &= \mathrm{e}(t_R) \operatorname{div} 2. \end{aligned}$$

Again, we define predicates for all cases where $\mathrm{e}(t'_{aL})$ or $\mathrm{e}(t'_R)$ is higher or lower than prescribed by these equations.

For fixed q_i and a, encoding R-steps is more involved than encoding L-steps, as we need to distinguish the two possible cases for $t_R \bmod 2$. This is the reason we chose to count the head of $\mathcal{U}$ to the left side of the tape, as we have only 14 R-movements, but 15 L-movements. Larger savings could be achieved by using a different machine with a larger difference in the number of L- and R-movements; but as mentioned before, we do not think that these slight improvements warrant the effort.

For an appropriate $k > 165$, we define the following four predicates for cases where one of the sides of the tapes contains too much:

$$\begin{aligned}
\pi_k &: \sigma(\alpha_1) \text{ contains } \#00^{2m}\#00^{2n+\mathrm{e}(a)}\#0^i\#\#0^+\#00^{2(2n+\mathrm{e}(d))+1}; && m,n \in \mathbb{N}_0,\\
\pi_{k+1} &: \sigma(\alpha_1) \text{ contains } \#00^{2m+1}\#00^{2n+\mathrm{e}(a)}\#0^i\#\#0^+\#00^{2(2n+\mathrm{e}(d))+2}; && m,n \in \mathbb{N}_0,\\
\pi_{k+2} &: \sigma(\alpha_1) \text{ contains } \#00^{2m}\#00^{2n+\mathrm{e}(a)}\#0^i\#\#00^{m+1}; && m,n \in \mathbb{N}_0,\\
\pi_{k+3} &: \sigma(\alpha_1) \text{ contains } \#00^{2m+1}\#00^{2n+\mathrm{e}(a)}\#0^i\#\#00^{m+1}; && m,n \in \mathbb{N}_0.
\end{aligned}$$

The first two describe the cases where t'_{aL} is too large (with $\mathrm{e}(t_R)$ being even or odd, respectively), the second two the cases where $\mathrm{e}(t'_R)$ is too large.

Next, we define two predicates that are satisfied if t'_R is too small:

$$\begin{aligned}
\pi_{k+4} &: \sigma(\alpha_1) \text{ contains } \#00^{2(l+m+1)}\#00^{2n+\mathrm{e}(a)}\#0^i\#\#00^l\#; && l,m,n \in \mathbb{N}_0,\\
\pi_{k+5} &: \sigma(\alpha_1) \text{ contains } \#00^{2(l+m+1)+1}\#00^{2n+\mathrm{e}(a)}\#0^i\#\#00^l\#; && l,m,n \in \mathbb{N}_0.
\end{aligned}$$

Again, we need to distinguish whether $\mathrm{e}(t_R)$ is even (π_{k+4}) or odd (π_{k+5}). This concludes the definition of predicates for t'_R.

As $t'_{aL} = 4(\mathrm{e}(t_{aL}) \operatorname{div} 2)+2\,\mathrm{e}(d)+(\mathrm{e}(t_R) \bmod 2)$, we know that for every R-movement in a valid computation, the congruence class of $\mathrm{e}(t'_{aL})$ modulo 4 is either $2\,\mathrm{e}(d)$ or $2\,\mathrm{e}(d)+1$, depending on $t_{R,0}$ (recall that $t_{R,0}$ is the first cell to the right of the head). Thus, regardless of that tape cell, the congruence classes of $2-\mathrm{e}(d)$ and $3-\mathrm{e}(d)$ modulo 4 can be excluded with the following two predicates:

$$\begin{aligned}
\pi_{k+6} &: \sigma(\alpha_1) \text{ contains } \#00^{2m+\mathrm{e}(a)}\#0^i\#\#0^+\#00^{4n+(2-\mathrm{e}(d))}\#; && m,n \in \mathbb{N}_0,\\
\pi_{k+7} &: \sigma(\alpha_1) \text{ contains } \#00^{2m+\mathrm{e}(a)}\#0^i\#\#0^+\#00^{4n+(3-\mathrm{e}(d))}\#; && m,n \in \mathbb{N}_0.
\end{aligned}$$

Furthermore, depending on $t_{R,0}$, we can also exclude the class $2\,\mathrm{e}(d) + (1 - \mathrm{e}(t_{R,0}))$ modulo 4. For this, we need to distinguish the two possible cases for $\mathrm{e}(t_{R,0})$ and define the predicates

$$\begin{aligned}
\pi_{k+8} &: \sigma(\alpha_1) \text{ contains } \#00^{2l}\#00^{2m+\mathrm{e}(a)}\#0^i\#\#00^l\#00^{4n+2\,\mathrm{e}(d)+1}\#; && l,m,n \in \mathbb{N}_0,\\
\pi_{k+9} &: \sigma(\alpha_1) \text{ contains } \#00^{2l+1}\#00^{2m+\mathrm{e}(a)}\#0^i\#\#00^l\#00^{4n+2\,\mathrm{e}(d)}\#; && l,m,n \in \mathbb{N}_0.
\end{aligned}$$

Finally, the last two predicates handle the case where $\mathrm{e}(t'_{aL})$ is of the right congruence class modulo 4, but too small. Again, we need to distinguish the two possible values of $\mathrm{e}(t_{R,0})$:

$$\begin{aligned}
\pi_{k+10} &: \sigma(\alpha_1) \text{ contains } \#00^{2l}\#00^{2(m+n+1)\,\mathrm{e}(a)}\#0^i\#\#00^l\#00^{4m+2\,\mathrm{e}(d)}\#; && l,m,n \in \mathbb{N}_0,\\
\pi_{k+11} &: \sigma(\alpha_1) \text{ contains } \#00^{2l+1}\#00^{2(m+n+1)\,\mathrm{e}(a)}\#0^i\#\#00^l\#00^{4m+2\,\mathrm{e}(d)+1}\#; && l,m,n \in \mathbb{N}_0.
\end{aligned}$$

Note that the last four predicates already assume t'_R has transitioned correctly. This is acceptable, as errors on this side of the tape are handled by the previous predicates.

We see that every one of the 14 R-movements of $\mathcal{U}$ requires 12 factor predicates π_k to π_{k+11}. Of these, π_{k+2} and π_{k+3} use 2 parameters, all others use 3 parameters. Adding these 168 predicates allows us to conclude that $\mu = 333$ was indeed a correct choice for the first claim.

For the second claim, we also add the suffix predicate

$$\pi_{334} : \sigma(\alpha_1) \text{ ends on } \#00^{2n}\#0^{10}\#\#; \qquad\qquad n \in \mathbb{N}_0.$$

This predicate eliminates all computations where the last configuration is not accepting.

Now, if there is a $\sigma(\alpha) \notin L_{\mathrm{E},\Sigma}(\beta)$, $\sigma(\alpha_1)$ encodes a computation of $\mathcal{U}$ that starts in I and reaches the state q_{10}, while $\mathrm{e}(t_{aL})$ is odd. That means that the machine reads 1 in q_{10} and halts. On the other hand, if there is a valid computation $(C_i)_{i=0}^n$ with $C_0 = I$, we can define σ by $\sigma(\alpha_1) := \mathrm{enc}_{\mathrm{X}}(C)$ and (for example) $\sigma(\alpha_2) := 0^{|\sigma(\alpha_1)|}$. Then none of the predicates is satisfied, and $\sigma(\alpha) \notin L_{\mathrm{E},\Sigma}(\beta)$.

Thus, for both claims, $L_{\mathrm{E},\Sigma}(\alpha) \setminus L_{\mathrm{E},\Sigma}(\beta) = \emptyset$ if and only if $\mathrm{ACCEPT}_{\mathcal{U}}(I) = \emptyset$. As I was chosen freely, this question must be undecidable.

All that remains is to count the number of variables in β. For the first claim, the types of predicates are distributed as follows:

1. 1 factor predicate with no parameter (π_7),
2. 1 factor predicate with one parameter (π_4),
3. 133 factor predicates with two parameters (π_5, π_8 to π_{66}, 3 per L-instruction, 2 per R-instruction),
4. 195 factor predicates with three parameters (π_6, π_{67} to π_{90}, 2 per L-instruction, 10 per R-instruction).

Therefore, in the first case, we have

$$\begin{aligned} |\operatorname{var}(\beta)| &= \mu + 2\kappa + 13 + 5 + 6 + 133 \cdot 7 + 195 \cdot 8 \\ &= 333 + 6 + 13 + 5 + 6 + 931 + 1560 = 2854. \end{aligned}$$

Thus, our construction proves that the inclusion problem for $\mathrm{ePAT}_{3,\Sigma}$ in $\mathrm{ePAT}_{2854,\Sigma}$ is undecidable.

The suffix predicate π_{334} uses one parameter and requires 6 additional variables (as μ needs to be increased by one), bringing the total amount of variables in β to 2860. This demonstrates undecidability of the inclusion problem for $\mathrm{ePAT}_{2,\Sigma}$ in $\mathrm{ePAT}_{2860,\Sigma}$.

3.3.3 Simulating Any Collatz Iteration (Proof of Theorem 3.11)

Here, for any given $N \geq 1$, we use the construction to decide the emptiness of $\mathrm{TRIV}(N)$.

Let $\kappa := 2$, $\mu := 10$, $\alpha_1 := \#0^N\# \, x \, \#0\#$ and $\alpha_2 := y$, where x and y are distinct variables. Due to the results in Section 3.3.1, we know that if there is a substitution σ with $\sigma(\alpha) \notin L_{\mathrm{E},\Sigma}(\beta)$, then

$$\sigma(\alpha_1) \subseteq \#0^N\# \, (0^+\#)^+ \, 0\#.$$

Therefore, every word from this set difference is already an encoding of a finite sequence over $\mathbb{N}_1$, with N as the first, and 1 as the last number. All that remains is to choose predicates π_4 to π_μ that describe every pair of successive numbers n_i and n_{i+1} where $n_{i+1} \neq \mathcal{C}(n_i)$.

We begin with the cases where $n_{i+1} > \mathcal{C}(n_i)$, which are handled by the following two predicates:

$$\pi_4 : \sigma(\alpha_1) \text{ contains a factor } \#0^{2m}\#0^{m+1} \text{ for some } m \in \mathbb{N}_0,$$
$$\pi_5 : \sigma(\alpha_1) \text{ contains a factor } \#0^{2m+1}\#0^{6m+3+2} \text{ for some } m \in \mathbb{N}_0.$$

It is easy to see that π_4 is satisfied if and only if the encoded sequence contains successive numbers n_i and n_{i+1} where n_i is even, and $n_{i+1} > \frac{1}{2}n_i = \mathcal{C}(n_i)$. Likewise, π_5 does the same for odd n_i: If n_i is odd, there is an $m \in \mathbb{N}_0$ with $n_i = 2m+1$, and $\mathcal{C}(n_i) = 3n_i+1 = 6m+3+1$.

Next, we define a predicate that describes all cases where n_i is even, and $n_{i+1} < \mathcal{C}(n_i)$:

$$\pi_6 : \sigma(\alpha_1) \text{ contains a factor } \#0^{2m+2n+2}\#0^{m}\# \text{ for some } m, n \in \mathbb{N}_0.$$

Obviously, if this predicate is satisfied, n_i is even, and $n_{i+1} < \mathcal{C}(n_i)$. For the other direction, let n_i be even, $n_{i+1} < \mathcal{C}(n_i)$, and define $m{:=}n_i$, $n{:=}\frac{1}{2}n_i - n_{i+1} - 1$. Then $2m+2n+2 = n_i$, which means that the corresponding substitution satisfies this predicate.

Capturing all cases where n_i is odd and $n_{i+1} < \mathcal{C}(n_i)$ is a little bit more involved. We define the following four predicates:

$$\pi_7 : \sigma(\alpha_1) \text{ contains a factor } \#0^{2m+1}\#0^{2n+1}\# \text{ for some } m, n \in \mathbb{N}_0,$$
$$\pi_8 : \sigma(\alpha_1) \text{ contains a factor } \#0^{2m+1}\#0^{6n}\# \text{ for some } m, n \in \mathbb{N}_0,$$
$$\pi_9 : \sigma(\alpha_1) \text{ contains a factor } \#0^{2m+1}\#0^{6n+2}\# \text{ for some } m, n \in \mathbb{N}_0,$$
$$\pi_{10} : \sigma(\alpha_1) \text{ contains a factor } \#0^{2m+2n+3}\#0^{6n+4}\# \text{ for some } m, n \in \mathbb{N}_0.$$

By definition of the Collatz function, if n_i is odd, then $\mathcal{C}(n_i)$ must be congruent to 4 modulo 6. The first three of these predicates handle all the cases where n_i is odd, but n_{i+1} is in the wrong congruence class modulo 6; i. e., either n_{i+1} is odd (π_7) or division by 6 leads to a remainder of 0 or 2 (π_8 and π_9, respectively). The remaining predicate π_{10} is satisfied if and only if n_i is odd, n_{i+1} is congruent to 4 modulo 6, and $n_{i+1} < \mathcal{C}(n_i)$.

Thus, if there is a $\sigma(\alpha) \notin L_{\mathrm{E},\Sigma}(\beta)$, $\sigma(\alpha_1)$ contains an encoding of a sequence $n_0, \ldots, n_l$ for some $l \geq 2$ with $n_i = \mathcal{C}^i(N)$ for every i, and $n_l = 1$. This means that N leads the Collatz function to the trivial cycle, and thus, $\mathrm{TRIV}(N) \neq \emptyset$.

On the other hand, assume $\mathrm{TRIV}(N) \neq \emptyset$. Then there is an $l \geq 2$ with $\mathcal{C}^l(N) = 1$. Let $\sigma(x){:=}0^{\mathcal{C}^1(N)}\#0^{\mathcal{C}^2(N)}\#\ldots\#0^{\mathcal{C}^{l-1}(N)}$ and $\sigma(y){:=}0^m$, where $m{:=}|\sigma(\alpha_1)|$. As we have seen, σ satisfies none of the predicates π_1 to π_{10}, and thus, $\sigma(\alpha) \notin L_{\mathrm{E},\Sigma}(\beta)$.

The total number of variables in β can be calculated as follows: First, we require $\mu + 2\kappa + 13$ variables from the basic construction and π_1 to π_3. As π_4 and π_5 are factor predicates with one numerical parameter, they each require 6 additional variables. Likewise, the predicates π_6 to π_{10} require 7 variables each. Thus, β contains $\mu + 2\kappa + 13 + 12 + 35 = 74$ different variables.

3.3.4 Simulating All Collatz Iterations (Proof of Theorem 3.12)

In order to decide the emptiness of NTCC, we choose $\kappa{:=}2$, $\mu{:=}11$ $\alpha_1{:=}\#x_1\#x_2\#x_3\#x_2\#$ and $\alpha_2{:=}x_2y$, where x_1, x_2, x_3 and y are pairwise distinct variables.

We use the same predicates π_4 to π_{10} as in the previous section for the encoding of TRIV(N), and the additional predicate

$$\pi_{11} : \sigma(\alpha_1) \text{ contains the factor } \#0\#.$$

Considering the previous section, it is easy to see that $L_{\mathrm{E},\Sigma}(\alpha) \setminus L_{\mathrm{E},\Sigma}(\beta) \neq \emptyset$ if and only if there is a number leading to a nontrivial cycle: Assume there is a substitution σ with $\sigma(\alpha) \notin L_{\mathrm{E},\Sigma}(\beta)$. This substitution satisfies none of the predicates π_1 to π_{10}, and must be of 2-E-good form. Therefore, $\sigma(x_2) \in 0^+$, which means that the sequence encoded in $\sigma(\alpha_1)$ contains the number $|\sigma(x_2)|$ at least twice. Due to π_{11}, this sequence does not contain the number 1, which means that the encoded sequence contains a nontrivial cycle of the Collatz function. Thus, NTCC is empty if and only if $L_{\mathrm{E},\Sigma}(\alpha) \setminus L_{\mathrm{E},\Sigma}(\beta)$ is empty.

As π_{11} is a 2-simple factor predicate with no numerical parameters, its subpatterns require five new variables in β (in addition to x_{11}), bringing the total number of variables in β to 80.

Therefore, any algorithm that decides the inclusion problem of $\mathrm{ePAT}_{4,\Sigma}$ in $\mathrm{ePAT}_{80,\Sigma}$ can be used to determine in finite time whether there exists any nontrivial cycle of the Collatz function by deciding whether $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$.

3.3.5 Extensions to Larger Terminal Alphabets

The construction from Section 3.3.1 can be extended to larger (finite) terminal alphabets. Assume that $\Sigma = \{0, \#, \mathtt{a}_1, \ldots, \mathtt{a}_n\}$ for some $n \geq 1$.

We first discuss a comparatively straightforward extension that leads to a huge increase in variables in the following section. In the section after that, we compare these results to the ones that follow from a more efficient modification by Bremer [11] (which is also contained in the upcoming full version of Bremer and Freydenberger [12]).

Straightforward Extension

We extend H to the set of all substitutions $\sigma : (\Sigma \cup \{x, y\})^* \to \Sigma^*$, but do not extend the definition of substitutions of κ-good form to our new and larger alphabet. Thus, $\sigma \in H$ is of κ-good form if $\sigma(\alpha_1) \in \{0, \#\}^*$, $\sigma(\alpha_2) \in 0^*$ and $\sigma(\alpha_1)$ does not contain $\#^\kappa$ as a factor.

In addition to the predicates π_1 to π_μ, for each new letter $\mathtt{a}_i$, we define a predicate $\pi_{\mu+2i-1}$ which describes the cases that $\sigma(\alpha_1)$ contains an occurrence of $\mathtt{a}_i$, and a predicate $\pi_{\mu+2i}$ which describes the cases that $\sigma(\alpha_2)$ contains an occurrence of $\mathtt{a}_i$. Instead of defining

$$\alpha{:=}v\, v\, \#^4\, v\, \alpha_1\, v\, \alpha_2\, v\, \#^4\, v\, u\, v,$$

with $v{:=}0\#^30$ and $u{:=}0\#\#0$, we use

$$\alpha{:=}v\, v\, \#^4\, v\, \alpha_1\, v\, \alpha_2\, v\, \#^4\, v\, \hat{u}\, v,$$

with $\hat{u}{:=}0\#\#\,\mathtt{a}_1\,\mathtt{a}_1 \ldots \mathtt{a}_n\,\mathtt{a}_n\,0$. In addition to this, we add the new predicates $\pi_{\mu+1}$ to $\pi_{\mu+2n}$ (which we still leave unspecified for a moment) to β and use

$$\eta_i{:=}z_i(\hat{z}_i)^2(\ddot{z}_{i,1})^2 \ldots (\ddot{z}_{i,n})^2 z_i$$

instead of $\eta_i = z_i(\hat{z}_i)^2 z_i$, where all z_i, $\hat{z}_i$, $\ddot{z}_{i,j}$ are pairwise distinct variables. Referring to the new shape of u, we can make the following observation:

Lemma 3.17. *Let $n \geq 1$, let $x_1 \ldots, x_n \in X$ be pairwise distinct, let $\{\mathtt{a}_1, \ldots, \mathtt{a}_n\} \subseteq \Sigma$ and let*

$$\gamma = x_1\,(x_2)^2 \ldots (x_n)^2\,x_1.$$

If there is a morphism $\sigma : X^* \to \Sigma^*$ with $\sigma(\alpha) = \mathtt{a}_1\,(\mathtt{a}_2)^2 \ldots (\mathtt{a}_n)^2\,\mathtt{a}_1$, then $\sigma(x_i) = \mathtt{a}_i$ for each $i \in \{1, \ldots, n\}$.

Proof. Assume $\sigma\,(x_1\,(x_2)^2 \ldots (x_n)^2\,x_1) = \mathtt{a}_1\,(\mathtt{a}_2)^2 \ldots (\mathtt{a}_n)^2\,\mathtt{a}_1$. If $\sigma(x_1) = \lambda$, then

$$\sigma\left((x_2)^2 \ldots (x_n)^2\right) = \mathtt{a}_1\,(\mathtt{a}_2)^2 \ldots (\mathtt{a}_n)^2\,\mathtt{a}_1$$

leads to an immediate contradiction. But $\sigma(x_1) \neq \lambda$ implies $\sigma(x_1) = \mathtt{a}_1$. Therefore,

$$\sigma\left((x_2)^2 \ldots (x_n)^2\right) = (\mathtt{a}_2)^2 \ldots (\mathtt{a}_n)^2$$

must hold. Now, for every $i \in \{2, \ldots, n\}$ with $\sigma(x_i) \neq \lambda$, $|\sigma(x_i)| = 1$ must hold, as $\sigma(\gamma)$ does not contain squares that are longer than two letters. Thus, every $(x_i)^2$ generates at most one factor $(\mathtt{a}_j)^2$, and every factor $(\mathtt{a}_j)^2$ has to be generated by some $(x_i)^2$. We conclude that for every x_i there is a j with $\sigma(x_i) = \mathtt{a}_j$. Of course, this is only possible if $i = j$ in all cases; therefore, the claim holds. □

Lemma 3.17 allows $\pi_{\mu+1}$ to $\pi_{\mu+2n}$ to be analogously constructed to π_2. To this end, we define

$$\gamma_{\mu+2i-1}{:=}y_{\mu+2i-1,1}\;\ddot{z}_{\mu+2i-1,i}\;y_{\mu+2i-1,2}, \qquad \gamma_{\mu+2i}{:=}y_{\mu+2i},$$
$$\delta_{\mu+2i-1}{:=}\hat{y}_{\mu+2i-1}, \qquad \delta_{\mu+2i}{:=}\hat{y}_{\mu+2i,1}\;\ddot{z}_{\mu+2i,i}\;\hat{y}_{\mu+2i,2}.$$

for each $i \in \{1, \ldots, n\}$. Again, all $y_{j,k}$, $\hat{y}_{j,k}$, z_j, $\hat{z}_j$ and $\ddot{z}_{j,k}$ are pairwise different variables. Now Lemma 3.13 applies (mutatis mutandis) as for binary alphabets, and since all substitutions of good form behave for Σ as for the binary alphabet, we can use the very same predicates and the same reasoning as before to prove undecidability of the inclusion problem for ePAT_Σ.

Of course, if $|\Sigma| = 2 + n$, this construction adds

- $2n$ predicates with $5 + n$ variables each (z_i, $\hat{z}_i$, $\ddot{z}_{i,1}$ to $\ddot{z}_{i,n}$, and the three variables in γ_i and δ_i for each i with $\mu + 1 \leq i \leq 2\mu + 2n$), and also
- n variables ($\ddot{z}_{i,1}$ to $\ddot{z}_{i,n}$) for each of the μ predicates that were used in the original construction,

for a total of $n(10 + n + \mu)$ additional variables in comparison to the construction for the binary case.

Using this technique, we arrive at the following results:

Corollary 3.18. *Let $|\Sigma| = 2 + n$ with $n \geq 1$. The following holds:*

1. *The inclusion problem of* $\mathrm{ePAT}_{3,\Sigma}$ *in* $\mathrm{ePAT}_{n^2+343n+2854,\Sigma}$ *is undecidable.*

2. *The inclusion problem of* $\mathrm{ePAT}_{2,\Sigma}$ *in* $\mathrm{ePAT}_{n^2+344n+2860,\Sigma}$ *is undecidable.*

3. *Every algorithm that decides the inclusion problem of* $\mathrm{ePAT}_{2,\Sigma}$ *in* $\mathrm{ePAT}_{n^2+20n+74,\Sigma}$ *can be converted into an algorithm that, for every $N \in \mathbb{N}_1$, decides whether N leads $\mathcal{C}$ into the trivial cycle.*

4. *Every algorithm that decides the inclusion problem for* $\mathrm{ePAT}_{4,\Sigma}$ *in* $\mathrm{ePAT}_{n^2+21n+80,\Sigma}$ *can be used to decide whether any number $N \geq 1$ leads $\mathcal{C}$ into a nontrivial cycle.*

This follows from the proofs of Theorem 3.10 (with $\mu = 333$ and $\mu = 334$), Theorem 3.11 (with $\mu = 10$) and Theorem 3.12 (with $\mu = 11$).

Bremer's Extensions

Using some clever coding tricks he discovered on the corresponding construction for NE-patterns, Bremer (cf. Bremer [11] and the upcoming full version of Bremer and Freydenberger [12]) developed an extension that modifies the basic construction to larger alphabets Σ with $|\Sigma| = 2 + n$, using only 22 additional variables.

This leads to the following bounds that beat the bounds from Corollary 3.18 in almost all cases:

Corollary 3.19. *Let $|\Sigma| = 2 + n$ with $n \geq 1$. The following holds:*

1. *The inclusion problem of* $\mathrm{ePAT}_{3,\Sigma}$ *in* $\mathrm{ePAT}_{2876,\Sigma}$ *is undecidable.*

2. *The inclusion problem of* $\mathrm{ePAT}_{2,\Sigma}$ *in* $\mathrm{ePAT}_{2882,\Sigma}$ *is undecidable.*

3. *Every algorithm that decides the inclusion problem of* $\mathrm{ePAT}_{2,\Sigma}$ *in* $\mathrm{ePAT}_{96,\Sigma}$ *can be converted into an algorithm that, for every $N \in \mathbb{N}_1$, decides whether N leads $\mathcal{C}$ into the trivial cycle.*

4. *Every algorithm that decides the inclusion problem for* $\mathrm{ePAT}_{4,\Sigma}$ *in* $\mathrm{ePAT}_{102,\Sigma}$ *can be used to decide whether any number $N \geq 1$ leads $\mathcal{C}$ into a nontrivial cycle.*

The only case where the bounds from Corollary 3.19 are not lower than the corresponding bounds from Corollary 3.18 occur for $n = 1$. Then, the bound for the number of variables in the β-pattern in item 3 (the reachability of the trivial Collatz cycle) is 95 due to Corollary 3.18, but 96 due to Corollary 3.19. Furthermore, both corollaries give a bound of 102 for item 4.

In addition to this, Bremer extended the construction to use terminal-free β-patterns, both in the E- and in the NE-case.

3.4 From Pattern Inclusion to Regular Expressions

This section describes how a comparison of Theorem 3.10 and its proof to similar results on the theory of concatenation led the author to the results on extended regular expression that we shall consider in Chapter 4. We begin in Section 3.4.1 with an introduction to word equations and continue in Section 3.4.2 with a discussion of the similarities to the proof of Theorem 3.10.

3.4.1 Word Equations and the Theory of Concatenation

This section provides a short introduction on word equations, one of the central topics in combinatorics on words that is related to pattern languages. Far more information can be found in Choffrut and Karhumäki [18], and Karhumäki et al. [52]. This material serves mostly as additional context for the explanations in Section 3.4.2.

Given a finite terminal alphabet Σ and a disjoint infinite variable alphabet X, a *word equation* (over the alphabet Σ) is a pair $(\alpha, \beta) \in (\Sigma \cup X)^* \times (\Sigma \cup X)^*$, usually denoted by $\alpha = \beta$. Furthermore, a *solution* of a word equation $\alpha = \beta$ is a terminal-preserving morphism $\sigma : (\Sigma \cup X)^* \to \Sigma^*$ with

$$\sigma(\alpha) = \sigma(\beta).$$

Given an equation $\eta = (\alpha, \beta)$, we define $\mathrm{var}(\eta) = \mathrm{var}(\alpha \cup \beta)$. It is easy to see that, when speaking from a pattern point of view, a word equation might be considered an equation on patterns (note that some variables might occur on both sides of the equation).

Moreover, given a language $L \subset \Sigma^*$, we say that L is *expressible* by an equation η if there is an $x \in \mathrm{var}(\eta)$ such that

$$L = \{\sigma(x) \mid \sigma \text{ is a solution of } \eta\}.$$

This definition can be directly extended to k-ary relations over Σ^* (with $k \geq 0$): A relation $R \subseteq (\Sigma^*)^k$ is expressible by an equation η if there are variables $x_1, \ldots, x_k \in \mathrm{var}(\eta)$ such that

$$R = \{(\sigma(x_1), \ldots, \sigma(x_k)) \mid \sigma \text{ is a solution of } \eta\}.$$

Obviously, for every alphabet Σ, every E-pattern language $L_{\mathrm{E},\Sigma}(\alpha)$ over Σ is expressible by the equation $x = \alpha$, where x is any variable that does not occur in α. Later in this section, we shall see that this also holds for $L_{\mathrm{NE},\Sigma}(\alpha)$.

Let $\Sigma := \{\mathtt{a}, \mathtt{b}\}$ and consider the word equation $\eta = (x\,\mathtt{a}\,\mathtt{b}, \mathtt{a}\,\mathtt{b}\,x)$. Using some elementary combinatorics on words (cf. Perrin [81]), one can show that η expresses the language $(\mathtt{a}\,\mathtt{b})^*$. Similarly, the equation $xz = zy$ expresses the *conjugacy relation* $\{(x, y) \mid \text{there exist } u, v \in \Sigma^* \text{ with } x = uv, y = vu\}$ (depending on the choice of the terminal alphabet Σ^*).

One surprising result in this area is that even very simple equations can express comparatively complicated relations, for details, see Czeizler's alternative proof [22] of the observation of Hmelevskiĭ[6] [43] on the set of solutions of the equation $x\,\mathtt{a}\,\mathtt{b}\,y = y\,\mathtt{b}\,\mathtt{a}\,x$.

[6] Actually, Хмелевский. Some authors use the alternative transcription *Khmelevski*.

Although word equations are a quite powerful mechanism, there exist comparatively simple languages and relations that cannot be expressed by word equations. Moreover, like for pattern languages, the question whether a language is expressible depends heavily on the choice of the terminal alphabet Σ. For example, given alphabets Σ and $\Sigma' \supset \Sigma$, one can show that Σ^* is not expressible by any word equation over Σ' (cf. Karhumaäki et al. [52], Example 23). Furthermore, for every non-unary alphabet Σ, the relation $\{(u, v) \mid u, v \in \Sigma^*, |u| = |v|\}$ cannot be expressed by any word equation.

In his seminal paper, Makanin [67] proves that the satisfiability problem for word equations is decidable; i. e., there is an algorithm that, given a word equation η over some constant alphabet Σ, decides whether η has a solution. By most measures, Makanin's proof can be considered to be among the most complicated results in theoretical computer science. Both the technique and the revision underwent multiple simplifications by various authors, and culminated by the (to the author's knowledge) most recent variant by Diekert [24]. In addition to this, there is an algorithm by Plandowski [82] that decides the satisfiability in PSPACE, and another algorithm by Plandowski and Rytter [83] that is conjectured to decide satisfiability in NP (for an explanation, cf. [82]). Note that this research has already found an application outside of combinatorics on words: The main result of [83] has been used by Schaefer et al. [101] to show that so-called string graphs can be recognized in NP (instead of the previously known upper bound of NEXPTIME).

As additional remark, NP-hardness of satisfiability follows immediately from the NP-hardness of the membership problem for pattern languages.

As explained by Choffrut and Karhumäki [18] as well as Karhumäki et al. [52], the notion of a word equation can be extended to that of a *Boolean formula of equations*. We say that Φ is a Boolean formula of equations with range Σ and variable alphabet X if it is obtained from equations over alphabets Σ and X by the operations of disjunction, conjunction and negation. A terminal-preserving morphism $\sigma : (\Sigma \cup X)^* \to \Sigma^*$ is a *solution* of Φ if and only if Φ gets the value true whenever the values of the equations of Φ get the values true or false depending on whether or not σ is a solution of the corresponding equation. A relation $R \subset \Sigma^k$ (with $k \geq 0$) is *expressible* by a Boolean formula of equations Φ if there are Variables $x_1, \ldots, x_k \in X$ such that

$$R = \{(\sigma(x_1), \ldots, \sigma(x_k)) \mid \sigma \text{ is a solution of } \Phi\}.$$

Surprisingly, any language or relation of words that is expressible by a Boolean formula is expressible by a single equation (Theorem 7 in Karhumäki et al. [52]). Note that the elimination of a conjunction requires no additional variables, while the elimination of disjunction and negation as presented in [52] introduce two additional variables, or a finite number of variables that depends on the size of Σ, respectively.

This observation connects word equations to the existential theory of concatenation (cf. Choffrut and Karhumäki [18]), as every word equation $\alpha = \beta$ using variables $x_1, \ldots, x_k$ for some $n \geq 1$ can be understood as the existential formula

$$\exists x_1 \ldots \exists x_n : \alpha = \beta.$$

Consequently, every pattern language can be easily expressed in the existential theory of concatenation: For every pattern $\alpha \in \mathrm{Pat}_\Sigma$ with $\mathrm{var}(\alpha) = \{x_1, \ldots, x_n\}$ for some $n \geq 0$,

its pattern languages $L_{\mathrm{E},\Sigma}(\alpha)$ and $L_{\mathrm{NE},\Sigma}(\alpha)$ can be expressed as

$$L_{\mathrm{E},\Sigma}(\alpha) = \{w \mid \exists x_1 \ldots \exists x_n : w = \alpha\},$$
$$L_{\mathrm{NE},\Sigma}(\alpha) = \{w \mid \exists x_1 \ldots \exists x_n : w = \alpha \wedge x_1 \neq \lambda \wedge \ldots \wedge x_n \neq \lambda\}.$$

Considering this, Makanin's result shows that *intersection emptiness*[7] for pattern languages is decidable for E- and NE-pattern languages. For E-pattern languages, one ensures that $\mathrm{var}(\alpha) \cap \mathrm{var}(\beta) = \emptyset$ using a renaming of the variables if necessary, and uses Makanin's (or Plandowski's) algorithm to check whether the equation $\alpha = \beta$ has a solution. For NE-pattern languages, one additionally specifies that $x_i \neq \lambda$ for all involved variables x_i and converts the corresponding Boolean formula into an equation.

As already shown by Quine [84] in 1946, satisfiability is undecidable in the whole theory of concatenation (i. e, where every combination of quantifiers is permitted). In the following years, the borderline of proven undecidability has been moved closer to Makanin's decidability result (cf. Choffrut and Karhumäki [18]). To the author's knowledge, the smallest fragment of the theory of concatenation that is known to be undecidable is the $\forall\exists^3$-positive theory (cf. Durnev [26]). This fragment consists of all formulae of the form $\forall x_1 \exists x_2 \exists x_3 \exists x_4 \Phi$, where Φ is a Boolean formula of equations over the variables $x_1, \ldots, x_4$ that uses only conjunctions and disjunctions, but no negations.

We return to this material in Section 3.4.2. There, we discuss how a comparison of the proof of Durnev's result to a similar result on pattern languages inspired the author to obtain the results presented in Chapter 4.

3.4.2 Word Equations and Theorem 3.10

As mentioned in Section 3.4.1, pattern languages can be viewed as a special case of languages that are expressible by word equations. For any two patterns $\alpha, \beta \in \mathrm{Pat}_\Sigma$ with $\mathrm{var}(\alpha) = \{x_1, \ldots, x_m\}$ and $\mathrm{var}(\beta) = \{y_1, \ldots, y_n\}$ for some $m, n \geq 0$ (where all x_i and all y_j are pairwise distinct), the inclusion $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$ holds if and only if the formula

$$\forall x_1 \ldots \forall x_m \exists y_1 \ldots \exists y_n : \alpha = \beta$$

is satisfiable. As such, the inclusion problem for pattern languages belongs to a comparatively restricted fragment of the positive $\forall\exists$-theory of concatenation, and the undecidability of the inclusion problem of $\mathrm{ePAT}_{m,\Sigma}$ in $\mathrm{ePAT}_{n,\Sigma}$ immediately implies that satisfiability of the positive $\forall^m\exists^n$-positive theory of concatenation is undecidable. On the other hand, the lower bounds of $\forall^3\exists^{2854}$-positive and $\forall^2\exists^{2860}$-positive that follow from Theorem 3.10 are not very close to the lowest currently known bound of $\forall^1\exists^3$-positive that is due to Durnev [26].

In addition to this, Durnev's proof is significantly shorter than the proof of Theorem 3.10 presented in this thesis. In principle, both proofs use similar approaches: A large disjunction describes all errors in the encodings of a machine with an undecidable halting problem, and if anything remains, there is at least one accepting run of that machine.

One major advantage of Durnev's construction is disjunction, which allows the 'reuse' of variables in different predicates, while every predicate in the basic construction as described in Section 3.3.1 uses its own set of variables.

[7] Given $\alpha, \beta \in \mathrm{Pat}_\Sigma$, is $L(\alpha) \cap L(\beta) = \emptyset$?

This observation served as the author's main inspirations for the results on extended regular expressions presented in Chapter 4: Extended regular expressions have a built-in alternation operator that works similarly to conjunction in the theory of concatenation. Furthermore, extended regular expressions allow more structural control than pattern languages[8] – while a pattern has to allow every possible value from Σ^* (or Σ^+, in the NE-case), an extended regular expression can be restricted to regular subsets of Σ^*.

One of the early precursors of Theorem 4.14 stated that the inclusion problem $L(\alpha) \subseteq L(\beta)$ is undecidable, where α is a proper regular expression, and β is an extended regular expression with at most 6 variables (from the set RegEx(6) in the terminology we introduce in Chapter 4).

The next improvement followed from understanding that the greater control in extended regular expressions can be used to remove the auxiliary structure that is used in the basic construction for patterns to ensure κ-good form. All this could be done using only regular features. While the construction for pattern languages reduces various problems to the problem whether $L_{\mathrm{E},\Sigma}(\alpha) \setminus L_{\mathrm{E},\Sigma}(\beta) = \emptyset$ for two constructed patterns α and β, the same problem could be reduced to the question whether $L(\alpha) = \Sigma^*$ for some constructed extended regular expression α.

Furthermore, using the right choice of reduced problems and encodings (the problems on the domain of extended Turing machines from Chapter 4), we can also reduce more complicated problems to appropriate problems for extended regular expressions (as we shall see, Π^0_2-complete and Σ^0_2-complete in the arithmetical hierarchy (cf. Odifreddi [77], Rogers [91]), in contrast to the Σ^0_1-completeness of the inclusion problem for pattern languages that follows from the proof of Theorem 3.10).

This realization and further optimizations which resulted in extended regular expressions with only one variable finally led to the results that are presented in Chapter 4. The main conceptual idea of the proof of Theorem 3.10 (in contrast to the proof of Theorem 3.3 as presented in [32]), the use of a single universal machine instead of infinitely many general machines, survives in the extension described in Section 4.3.1.

Possibly, it might have been easier to develop the results in Chapter 4 directly, without the potentially distracting detour over the inclusion of pattern languages. But, at least in the author's case, tackling the problems for pattern languages (which are narrower in their results, but harder to prove) provided the experience and education that made the actual proof of Theorem 4.14 comparatively simple[9].

[8] There has also been work on extending pattern languages by allowing restrictions on the possible replacements, i. e., giving the variables a certain type. For examples, see Koshiba [58], and Dumitrescu et al. [25].

[9] As a closing footnote that is not directly related to the material in the present section, the author would like to relate a long and perhaps not too interesting anecdote on the coincidences that led to the proofs of Theorem 3.3 and Theorem 3.10. Readers who do not care about these things are invited (and probably well advised) to skip this footnote.

Around July 2007, the author accidentally met Steffen Lange in the ICE from Kaiserslautern to Frankfurt. Not only had they chosen the same train, but also seats that where almost next to each other. In their short discussion, Lange remarked to the author (who was reading [51]) that someone should definitely answer the then open question on the decidability of the alphabet-specific inclusion problem. Although the author agreed, he was not too optimistic that he could be that someone, as he had tried to understand the proofs in [51] for several times. A few weeks later, in late August, the author had to spend a few hours in his new and at that time quite empty apartment, waiting for the meter-reader. Due to several circumstances, he had nothing to pass the time but a copy of [51] and

a few pens. For unknown reasons, the meter-reader missed this appointment, but while waiting, the author finally understood the proof of Theorem 3.2, and how it could be adapted to prove Theorem 3.3.

The idea for Theorem 3.10 also occurred serendipitously. In June 2009, the author had to undergo an ambulant surgery that required general anesthesia. While sitting in the waiting room, inspiration struck, and he started to sketch the proof; but when he was hushed into the operating room, he had only finished half of the proof. Apparently, anesthesia can cause memory loss, but not in this case. Almost immediately after waking up, the author reoriented himself and finished the sketch – correctly, as he later verified.

Chapter 4

Real Regular Expressions: Decidability and Succinctness

4.1 On Extended Regular Expressions

Since being introduced by Kleene [55] in 1956, regular expressions have developed into a central device of theoretical and applied computer science. On one side, research into the theoretical properties of regular expressions, especially various aspects of their complexity, is still a very active area of investigation (see Holzer and Kutrib [45] for a survey with numerous recent references). On the other side, almost all modern programming language offer regular expression matching in their standard libraries or application frameworks, and most text editors allow the use of regular expressions for search and replacement functionality.

But, due to practical considerations (cf. Friedl [36]), most modern matching engines have evolved to use an extension to regular expressions that allows the user to specify non-regular languages. In addition to the features of regular expressions as they are mostly studied in theory (which we, from now on, call *proper regular expressions*), and apart from the (regularity preserving) "syntactic sugar" that most implementations use, these *extended regular expressions*[1] contain *back references*, also called *variables*, which specify repetitions that increase the expressive power beyond the class of regular languages. For example, the (non-regular) language $L = \{ww \mid w \in \{\mathtt{a},\mathtt{b}\}^*\}$ is generated by the extended regular expression $\alpha := ((\mathtt{a} \mid \mathtt{b})^*)\,\%x\,x$.

This expression can be understood as follows (for a more formal treatment, see Definition 4.3): For any expression β, $(\beta)\%x$ matches the same expression as β, and binds the match to the variable x. In the case of this example, the subexpression $((\mathtt{a} \mid \mathtt{b})^*)\,\%x$ can be matched to any word $w \in \{\mathtt{a},\mathtt{b}\}^*$, and when it is matched to w, the variable x is assigned the value w. Any further occurrence of x repeats w, leading to the language of all words of the form ww with $w \in \{\mathtt{a},\mathtt{b}\}^*$. Analogously, the expression $((\mathtt{a} \mid \mathtt{b})^*)\,\%x\,xx$ generates the language of all www with $w \in \{\mathtt{a},\mathtt{b}\}^*$.

Although this ability to specify repetitions is used in almost every modern match-

[1] The author is aware that the term "extended regular expression" is also used for a different model (used in conjunction with star-free languages). Personally, he would have preferred to use the term "real expressions", especially considering the fact that (proper) regular expressions are also called "rational expressions". As there are already far too many names for the same concept (e. g., rewbr, regex, practical regular expression), he decided to follow the recent literature.

ing engine (e. g., the programming languages PERL and Python), the implementations differ in various details, even between two versions of the same implementation of a programming language (for some examples, see Câmpeanu and Santean [14]). Nonetheless, there is a common core to these variants, which was first formalized by Aho [1]. Later, Câmpeanu et al. [13] introduce a different formalization that is closer to the real world syntax, and addresses some questions of semantics that were left open in [1]. In addition to this, the *pattern expressions* by Câmpeanu and Yu [15] and the *H-expressions* by Bordihn et al. [9] use comparable repetition mechanisms and possess similar expressive power.

Still, theoretical investigation of extended regular expressions has been comparatively rare (especially when compared to their more prominent subclass); see e. g. Larsen [65], Della Penna et al. [23], Câmpeanu and Santean [14], Carle and Narendran [16], and Reidenbach and Schmid [89].

In contrast to their widespread use in various applications, extended regular expressions have some undesirable properties. Most importantly, their *membership problem* (the question whether an expression matches a word) is NP-complete (cf. Aho [1]); the exponential part in the best known upper bounds depends on the number of different variables in the expression. Of course, this compares unfavorably to the efficiently decidable membership problem of proper regular expressions (cf. Aho [1]). On the other hand, there are cases where extended regular expressions express regular languages far more succinctly than proper regular expressions. Consider the following example:

Example 4.1. *For $n \geq 1$, let $L_n := \{www \mid w \in \{\mathtt{a},\mathtt{b}\}^+, |w| = n\}$. This language is generated by the expression*

$$\alpha_n := (\underbrace{(\mathtt{a} \mid \mathtt{b}) \ldots (\mathtt{a} \mid \mathtt{b})}_{n \text{ times } (\mathtt{a} \mid \mathtt{b})})\%x\, xx.$$

Moreover, every language L_n is finite and, hence, regular. With some effort, one can show that (for sufficiently large n), every proper regular expression for L_n is exponentially longer than α_n. ◇

Due to the repetitive nature of the words of languages L_n in Example 4.1, it is not surprising that the use of variables provides a shorter description of L_n. The following example might be considered less straightforward:

Example 4.2. *Consider the expression $\alpha := (\mathtt{a} \mid \mathtt{b})^* \left((\mathtt{a} \mid \mathtt{b})^+\right) \%x\, x\, (\mathtt{a} \mid \mathtt{b})^*$. It is a well-known fact (and easily verified by an exhaustive list of all possibilities) that every word $w \in \{\mathtt{a},\mathtt{b}\}^*$ with $|w| \geq 4$ can be expressed in the form $w = uxxv$, with $u, v \in \{\mathtt{a},\mathtt{b}\}^*$ and $x \in \{\mathtt{a},\mathtt{b}\}^+$. Thus, the expression α matches all but finitely many words; hence, its language $L(\alpha)$ is regular.* ◇

The phenomenon used in Example 4.2 is strongly related to the notion of *avoidable patterns* (cf. Cassaigne [17]), and involves some very hard combinatorial questions. We observe that extended regular expressions can be used to express regular languages more succinctly than proper regular expressions do, and that it might be hard to convert an extended regular expression into a proper regular expression for the same language.

The two central questions studied in the present chapter are as follows: First, how hard is it to minimize extended regular expressions (both with respect to their length,

and with respect to the number of variables they contain), and second, how succinctly can extended regular expressions describe regular languages? These natural questions are also motivated by practical concerns: If a given application reuses an expression many times, it might pay off to invest resources in the search for an expression that is shorter, or uses fewer variables, and thus can be matched more efficiently.

We approach this question through related decidability problems (e. g. the universality problem) and by studying lower bounds on the tradeoff between the size of extended regular expressions and proper regular expressions.

The main technical contribution of the present chapter is the proof that all these decision problems are undecidable (some are not even semi-decidable), even for extended regular expressions that use only a single variable. Thus, while bounding the number of variables in extended regular expressions (or, more precisely, the number of variable bindings) reduces the complexity of the membership problem from NP-complete to polynomial (cf. Aho [1]), we show that extending proper regular expressions with only a single variable already results in undecidability of various problems.

As a consequence, extended regular expressions cannot be minimized effectively, and the tradeoff between extended and proper regular expressions is not bounded by any recursive function (a so-called *non-recursive tradeoff*, cf. Kutrib [61]). Thus, although the use of the "right" extended regular expression for a regular expression might offer arbitrary advantages in size (and, hence, parsing speed), these optimal expressions cannot be found effectively. These results highlight the power of the variable mechanism, and demonstrate that different restrictions than the number of variables ought to be considered.

The structure of the further parts of this chapter is as follows: In Section 4.2, we introduce most of the technical details that serve as the fundament of this chapter, the most important one being Theorem 4.14. In Section 4.3, we use Theorem 4.14 to derive Theorem 4.15 – the main undecidability result – and its consequences, thus answering our questions on minimizability and relative succinctness of extended regular expressions.

The technical preparations in Section 4.2 and the proof of Theorem 4.14 are of considerable length; readers who are primarily interested in the answer to these questions might prefer to read Section 4.3 before reading Section 4.2.

As explained in Section 3.4, the proof of Theorem 4.14, the main technical theorem in the present chapter, was inspired by the proof of Theorem 3.10 in Chapter 3.

4.2 Definitions and Preliminary Results

This section contains the definition of extended regular expressions, and various tools which we shall use later on in the present chapter.

4.2.1 Extended Regular Expressions

We now introduce syntax and semantics of extended regular expressions. Apart from some changes in terminology, this formalization is due to Aho [1]:

Definition 4.3. *Let* Σ *be an infinite set of* terminals*,* X *an infinite set of* variables*, and the set of* metacharacters *consist of* λ*,* $($*,* $)$*,* $|$*,* **, and* $\%$*, where all three sets are*

pairwise disjoint. Syntax and semantics of real regular expressions*, or* extended regular expressions*, are defined inductively as follows:*

1. *Each terminal $a \in \Sigma$ is an extended regular expression that matches the word a.*
2. *Each variable $x \in X$ is an extended regular expression that matches the word defined by x.*
3. *If α_1 and α_2 are extended regular expressions, then $(\alpha_1 \mid \alpha_2)$ is an extended regular expression that matches any word matched by α_1 or by α_2.*
4. *If α_1 and α_2 are extended regular expressions, then $(\alpha_1\alpha_2)$ is an extended regular expression that matches any word of the form vw, where v matches α_1 and w matches α_2.*
5. *If α is an extended regular expression, then $(\alpha)^*$ is an extended regular expression that matches any word of the form $w_1 \dots w_n$ with $n \geq 0$, where α matches each w_i with $1 \leq i \leq n$.*
6. *If α is an extended regular expression that matches a word w, and $x \in X$, then $(\alpha)\%x$ is an extended regular expression that matches the word w, and x is bound to the value w.*
7. *If α is an extended regular expression, then (α) is an extended regular expression that matches the same words as α.*

We denote the set of all extended regular expressions by RegEx*. For every extended regular expression α, we use $L(\alpha)$ to denote the set of all words that are matched by α, and call $L(\alpha)$* the language generated by α.

A proper regular expression *is an extended regular expression that contains neither %, nor any variable from X.*

Note that, as in [1], some peculiarities of the semantics of extended regular expressions are not addressed in this definition (some examples are mentioned further down). Câmpeanu et al. [13] offer an alternative definition that explicitly deals with some technical peculiarities that are omitted in Aho's definition, and is closer to the syntax of the programming language PERL. The proofs presented in this chapter are not affected by these differences and can be easily adapted to the definition of Câmpeanu et al., or any similar mechanism (e. g., those given by Câmpeanu and Yu [15] or Bordihn et al. [9]).

We shall use the notation $(\alpha)^+$ as a shorthand for $\alpha(\alpha)^*$, and freely omit parentheses whenever the meaning remains unambiguous. When doing this, we assume that there is a precedence on the order of the applications of operations, with * and $^+$ ranking over concatenation ranking over the alternation operator $|$.

We illustrate the intended semantics of extended regular expressions using the following examples in addition to the examples in Section 4.1:

Example 4.4. *Consider the following extended regular expressions:*

$$\alpha_1 := ((\mathtt{a} \mid \mathtt{b})^*)\,\%x\; xx\;\; ((\mathtt{a} \mid \mathtt{b})^*)\,\%x\; x, \qquad \alpha_2 := (((\mathtt{a} \mid \mathtt{b})^*)\,\%x\; x)^+ .$$

These expressions generate the following languages:

$$L(\alpha_1) = \{vvvww \mid v, w \in \{\mathtt{a}, \mathtt{b}\}^*\}\,, \qquad L(\alpha_2) = \{w_1 w_1 \dots w_n w_n \mid n \geq 1, w_i \in \{\mathtt{a}, \mathtt{b}\}^*\}\,.$$

Note that both expressions rely on the fact that variables can be bound multiple times, and implicitly assume that we parse from left to right. ◇

From a formal point of view, the fact that variables have a scope and the possibility to rebind variables (as in Example 4.4) can cause unexpected side effects and would normally require a more formal definition of semantics, instead of our "definition by example". Furthermore, Aho's definition does not deal with pathological cases in which some variables might be unbound, e. g. like $((\mathtt{a})\%x \mid \mathtt{b})\, x$. Although the definition by Câmpeanu et al. [13] addresses these problems, we still use Aho's notation, because it is more convenient for the proof of Theorem 4.14, the main technical tool of the present chapter.

As we use variables (and variable bindings) in a comparatively restricted way that does not require rebinding of variables or assumptions on variable scopes, and as we do not rely on any special features or conventions that are peculiar to Aho's definition, all proofs can be easily adapted to the notation of Câmpeanu et al.

In general, the membership problem for RegEx is NP-complete, as shown in Theorem 6.2 in Aho [1]. As explained in that proof, this problem is solvable in polynomial-time if the number of different variables is bounded. It is not clear how (or if) Aho's reasoning applies to expressions like α_2 in our Example 4.4; therefore, we formalize a slightly stronger restriction than Aho, and consider the following subclasses of RegEx:

Definition 4.5. *For $k \geq 0$, let* $\mathrm{RegEx}(k)$ *denote the class of all extended regular expressions α that satisfy the following properties:*

1. *α contains at most k occurrences of the metacharacter %,*
2. *if α contains a subexpression $(\beta)^*$, then the metacharacter % does not occur in β,*
3. *for every $x \in X$ that occurs in α, α contains exactly one occurrence of $\%x$.*

Intuitively, these restrictions on extended regular expressions in $\mathrm{RegEx}(k)$ limit not only the number of different variables, but also the total number of possible variable bindings, to at most k.

Note that $\mathrm{RegEx}(0)$ is equivalent to the class of proper regular expressions; furthermore, observe that $\mathrm{RegEx}(k) \subset \mathrm{RegEx}(k+1)$ for every $k \geq 0$.

Referring to the extended regular expressions given in Example 4.4, we observe that, as $\%x$ occurs twice in α_1, α_1 is not element of any $\mathrm{RegEx}(k)$ with $k \geq 0$, but the extended regular expression $\alpha_1' := ((\mathtt{a} \mid \mathtt{b})^*)\,\%x\; x\; ((\mathtt{a} \mid \mathtt{b})^*)\,\%y\; y$ generates the same language as α_1, and $\alpha_1' \in (\mathrm{RegEx}(2) \setminus \mathrm{RegEx}(1))$. In contrast to this, $\alpha_2 \notin \mathrm{RegEx}(k)$ for all $k \geq 0$, as % occurs inside a $()^*$ subexpression (as we defined $^+$ through *).

For any $k \geq 0$, we say that a language L is a $\mathrm{RegEx}(k)$-*language* if there is some $\alpha \in \mathrm{RegEx}(k)$ with $L(\alpha) = L$.

We also consider the class FRegEx of all extended regular expressions that do not use the operator * (or $^+$), and its subclasses $\mathrm{FRegEx}(k) := \mathrm{FRegEx} \cap \mathrm{RegEx}(k)$ for $k \geq 0$. Thus, FRegEx contains exactly those expressions that generate finite (and, hence,

regular) languages. Analogously, for every $k \geq 0$, we define a class CoFRegEx(k) as the class of all $\alpha \in \text{RegEx}(k)$ such that $L(\alpha)$ is cofinite. Unlike the classes FRegEx(k), these classes have no straightforward syntactic definition – as we shall prove in Theorem 4.15, cofiniteness is not semi-decidable for RegEx(k) (if $k \geq 1$).

4.2.2 Decision Problems and Descriptional Complexity

Most of the technical reasoning in the present chapter is centered around the following decision problems:

Definition 4.6. *Let Σ denote a fixed terminal alphabet. For all $k, l \geq 0$, we define the following decision problems for* RegEx(k)*:*

Universality: *Given $\alpha \in \text{RegEx}(k)$, is $L(\alpha) = \Sigma^*$?*

Cofiniteness: *Given $\alpha \in \text{RegEx}(k)$, is $\Sigma^* \setminus L(\alpha)$ finite?*

RegEx(l)-ity:** *Given $\alpha \in \text{RegEx}(k)$, is there a $\beta \in \text{RegEx}(l)$ with $L(\alpha) = L(\beta)$?*

In addition to this, we also define inclusion and equivalence for RegEx(k) canonically. Although we do not deal with the more general case, all these problems can be extended to decision problems for the whole class RegEx.

As we shall see, Theorem 4.15 – one of this chapter's main technical results – states that these problems are undecidable (to various degrees). We use the undecidability of the universality problem to show that there is no effective procedure that minimizes extended regular expressions with respect to their length, and the undecidability of RegEx(l)-ity to conclude the same for minimization with respect to the number of variables. Furthermore, cofiniteness and RegEx(l)-ity help us to obtain various results on the relative succinctness of proper and extended regular expressions.

By definition, RegEx(l)-ity holds trivially for all RegEx(k) with $k \leq l$. If $l = 0$, we mostly use the more convenient term *regularity* (for RegEx(k)), instead of RegEx(0)-ity. Note that, even for RegEx(0), universality is already PSPACE-complete (see Aho et al. [2]).

In order to examine the relative succinctness of RegEx(1) in comparison with RegEx(0), we use the following notion of complexity measures:

Definition 4.7. *Let $\mathcal{R}$ be a class of extended regular expressions. A* complexity measure *for $\mathcal{R}$ is a total recursive function $c : \mathcal{R} \to \mathbb{N}$ such that, for every alphabet Σ, the set of all $\alpha \in \mathcal{R}$ with $L(\alpha) \subseteq \Sigma^*$*

1. *can be effectively enumerated in order of increasing $c(\alpha)$, and*
2. *does not contain infinitely many extended regular expressions with the same value $c(\alpha)$.*

This definition includes the canonical concept of the length, as well as most of its natural extensions (for example, in our context, one could define a complexity measure that gives additional weight to the number or distance of occurrences of variables, or their nesting level). Kutrib [61] provides more details on (and an extensive motivation of) complexity measures. Using this definition, we are able to define the notion of tradeoffs between classes of extended regular expressions:

Definition 4.8. *Let $k > l \geq 0$ and let c be a complexity measure for* RegEx(k) *(and thereby also for* RegEx(l)*). A recursive function $f_c : \mathbb{N} \to \mathbb{N}$ is said to be a* recursive upper bound for the tradeoff between RegEx(k) and RegEx(l) *if, for all those $\alpha \in$* RegEx(k) *for which $L(\alpha)$ is a* RegEx(l)*-language, there is a $\beta \in$* RegEx(l) *with $L(\beta) = L(\alpha)$ and $c(\beta) \leq f_c(c(\alpha))$.*

If no recursive upper bound for the tradeoff between RegEx(k) *and* RegEx(l) *exists, we say that* the tradeoff between RegEx(k) and RegEx(l) is non-recursive*.*

There is a considerable amount of literature on a wide range of non-recursive tradeoffs between various description mechanisms; for a survey, see Kutrib [61].

4.2.3 Generalized Sequential Machines

In Section 4.2.4, we shall use a so-called *generalized sequential machine (GSM)* and its associated mapping. A full formal definition of GSMs and the mappings they define is provided in Chapter 11.2 of Hopcroft and Ullman [46], we only sketch the concept on a superficial level that is sufficient to understand how we use it in the proof of Lemma 4.13.

Generalized sequential machines generalize nondeterministic finite automata by supplementing every transition with an output; i. e., whenever a GSM reads a symbol, it also emits a string (as specified by its transition relation). Thus, every path through a GSM also yields the concatenation of the emitted strings as an output. Applying a GSM M to a word w yields the language $M(w)$ that consists of every string emitted by M along an accepting path for w. Likewise, for every language L, $M(L) := \cup_{w \in L} M(w)$. As M maps L to $M(L)$, this process is called a *GSM mapping*.

As we shall see, our use of a GSM in the proof of Lemma 4.13 is comparatively simple: First, the machine is deterministic, and second, it is only used on words where it is guaranteed that the machine reaches the accepting state. This allows us to ignore all special cases that might require a more formal definition of GSMs. In either case, our use is compatible with the extensive formal definition in [46].

One important fact is that the class of regular languages is closed under GSM mappings: Every GSM mapping can be expressed as the result of the application of an inverse morphism, the intersection with a regular language, and the application of a morphism, and as the class of regular languages is closed under each of these three operations, closure under GSM mappings follows (cf. [46]).

Therefore, GSM mappings can be used to simplify proofs of non-regularity, as can be seen in the following example:

Example 4.9. *Consider the language $L := \{\mathtt{a}^i\,\mathtt{b}\,\mathtt{a}^n\,\mathtt{b}\,\mathtt{b}\,\mathtt{a}^{2n}\,\mathtt{b}^m \mid i \geq 0, n, m \geq 1\}$ and assume that we want to prove that L is not regular. Furthermore, assume that we are familiar enough with the use of the Pumping Lemma to show that $L' := \{\mathtt{a}^n\,\mathtt{b}^n \mid n \geq 1\}$ is not regular, but that we do not see how we apply the Pumping Lemma to show that L is not regular.*

Our goal is to define a GSM M such that $M(L) = L'$. If L is regular, then $M(L)$ must be regular as well, which leads to the desired contradiction. To this end, we define the M as depicted in Figure 4.1. Now assume that M is fed a word $w \in L$. By definition, $w = \mathtt{a}^i\,\mathtt{b}\,\mathtt{a}^n\,\mathtt{b}\,\mathtt{b}\,\mathtt{a}^{2n}\,\mathtt{b}^m$ for some $i \geq 0$ and some $n, m \geq 1$. First, M loops on in q_1 reading all letters of the prefix $\mathtt{a}^i$, and emits λ. When all these $\mathtt{a}$ have been consumed, M reads

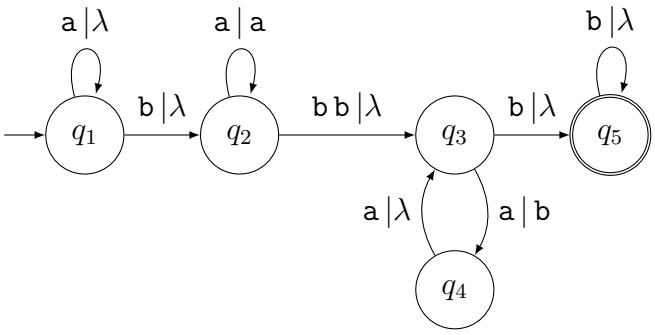

Figure 4.1: The GSM M that is used in Example 4.9. As explained therein, M is used to map each word $\mathtt{a}^i\,\mathtt{b}\,\mathtt{a}^n\,\mathtt{b}\,\mathtt{b}\,\mathtt{a}^{2n}\,\mathtt{b}^m$ with $i \geq 0$ and $n, m \geq 1$ to the word $\mathtt{a}^n\,\mathtt{b}^n$.

the letter $\mathtt{b}$ *and consumes it, emits* λ*, and enters state* q_2*. At this point, the unread part of* w *is* $\mathtt{a}^n\,\mathtt{b}\,\mathtt{b}\,\mathtt{a}^{2n}\,\mathtt{b}^m$*, and* M *has emitted* λ*.*

Next, M *loops in state* q_2*, emitting* $\mathtt{a}$ *for every* $\mathtt{a}$ *from* $\mathtt{a}^n$*. After all these* $\mathtt{a}$ *have been read,* M *reads* $\mathtt{b}\,\mathtt{b}$*, emits* λ*, and enters* q_3*. Now, the unread part of* w *is* $\mathtt{a}^{2n}\,\mathtt{b}^m$*, and* $\mathtt{a}^n$ *has been emitted. Note that, in a strict interpretation of the definition,* M *would only be allowed to read one letter in every transition. Purists can easily handle this problem by introducing an additional state between* q_2 *and* q_3*.*

Then M *cycles between* q_3 *and* q_4*, processing the part* $\mathtt{a}^{2n}$ *of* w*. For every two* $\mathtt{a}$ *the machine reads, it emits a single* $\mathtt{b}$*. As* $w \in L$*, we can rely on the fact that there is an even number of* $\mathtt{a}$ *present. At some point, all these* $\mathtt{a}$ *have been processed, and* M *is in* q_3 *and has emitted* $\mathtt{a}^n\,\mathtt{b}^n$ *when it encounters the first* $\mathtt{b}$ *of* $\mathtt{b}^m$*. The machine enters state* q_5 *and consumes the remaining* $\mathtt{b}$*, without emitting any further letters. Thus,* $M(w) = \mathtt{a}^n\,\mathtt{b}^n$*, and therefore,* $M(L) = L'$*.*

If we assume L *is regular, then* $L' = M(L)$ *must be regular as well, which contradicts our initial assumption that* L' *is not regular.* ◇

As Example 4.9 shows, GSM mappings are a powerful tool that can be used to reduce the question whether a language is regular to languages where non-regularity is easier to establish.

4.2.4 Extended Turing Machines

On a superficial level, we prove Theorem 4.15 by using Theorem 4.14 (which we introduce further down in the present section) to reduce various undecidable decision problems for Turing machines to appropriate problems for extended regular expressions (the problems from Definition 4.6). This is done by giving an effective procedure that, given a Turing machine $\mathcal{M}$, returns an extended regular expression that generates the complement of a language that encodes all accepting runs of $\mathcal{M}$.

On a less superficial level, this approach needs to deal with certain technical peculiarities that make it preferable to study a variation of the Turing machine model. An *extended Turing machine* is a 3-tuple $\mathcal{X} = (Q, q_1, \delta)$, where Q and q_1 denote the state set and the initial state. All extended Turing machines operate on the tape alphabet $\Gamma{:=}\{0, 1\}$ and use 0 as the blank letter. The transition function δ is a function $\delta : \Gamma \times Q \to (\Gamma \times \{L, R\} \times Q) \cup \{\text{HALT}\} \cup (\{\text{CHECK}_\text{R}\} \times Q)$. The movement instructions

L and R and the HALT-instruction are interpreted canonically – if $\delta(a, q) = (b, M, p)$ for some $M \in \{L, R\}$ (and $a, b \in \Gamma$, $p, q \in Q$), the machine replaces the symbol under the head (a) with b, moves the head to the left if $M = L$ (or to the right if $M = R$), and enters state p. If $\delta(a, q) = \text{HALT}$, the machine halts and accepts.

The command CHECK_R works as follows: If $\delta(a, q) = (\text{CHECK}_\text{R}, p)$ for some $p \in Q$, $\mathcal{X}$ immediately checks (without moving the head) whether the right side of the tape (i. e., the part of the tape that starts immediately to the right of the head) contains only the blank symbol 0. If this is the case, $\mathcal{X}$ enters state p; but if the right side of the tape contains any occurrence of 1, $\mathcal{X}$ stays in q. As the tape is never changed during a CHECK_R-instruction, this leads $\mathcal{X}$ into an infinite loop, as it will always read a in q, and will neither halt, nor change its state, head symbol, or head position.

Although it might seem counterintuitive to include an instruction that allows our machines to search the whole infinite side of a tape in a single step and without moving the head, this command is expressible in the construction we use in the proof of Theorem 4.14, and it is needed for the intended behavior.

We partition the tape of an extended Turing machine $\mathcal{X}$ into three disjoint areas: The *head symbol*, which is (naturally) the tape symbol at the position of the head, the *right tape side*, which contains the tape word that starts immediately to the right of the head symbol and extends rightward into infinity, and the *left tape side*, which starts immediately left to the head symbol and extends infinitely to the left. When speaking of a configuration, we denote the head symbol by a and refer to the contents of the left or right tape side as the *left tape word* t_L or the *right tape word* t_R, respectively. For an illustration and further explanations, see Figure 4.2.

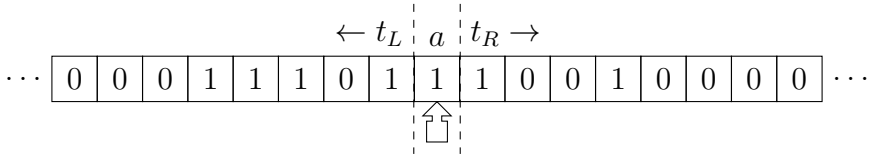

Figure 4.2: An illustration of tape words of an extended Turing machine (as defined in Section 4.2.4). The arrow below the tape symbolizes the position of the head, while the dashed lines show the borders between the left tape side, the head position and the right tape side. Assuming that all tape cells that are not shown contain 0, we observe the left tape word $t_L = 10111\,0^\omega$, the right tape word $t_R = 1001\,0^\omega$, and the head letter $a = 1$.

A *configuration* of an extended Turing machine $\mathcal{X} = (Q, q_1, \delta)$ is a tuple (t_L, t_R, a, q), where $t_L, t_R \in \Gamma^* 0^\omega$ are the left and right tape word, $a \in \Gamma$ is the head symbol, and $q \in Q$ denotes the current state. The symbol $\vdash_\mathcal{X}$ denotes the successor relation on configurations of $\mathcal{X}$, i. e., $C \vdash_\mathcal{X} C'$ if $\mathcal{X}$ enters C' immediately after C.

We define $\text{dom}_\text{X}(\mathcal{X})$, the *domain* of an extended Turing machine $\mathcal{X} = (Q, q_1, \delta)$, to be the set of all tape words $t_R \in \Gamma^* 0^\omega$ such that $\mathcal{X}$, if started in the configuration $(0^\omega, t_R, 0, q_1)$, halts after finitely many steps.

The definition of dom_X is motivated by the properties of the encoding that we shall use. Usually, definitions of the domain of a Turing machine rely on the fact that the end of the input is marked by a special letter \$ or an encoding thereof (cf. Minsky [70]). As we shall see, our use of extended regular expressions does not allow us to express

the fact that every input is ended by exactly one \$ symbol. Without the $\mathrm{CHECK}_\mathrm{R}$-instruction in an extended Turing machine $\mathcal{X}$, we then would have to deal with the unfortunate side effect that a nonempty $\mathrm{dom}_\mathrm{X}(\mathcal{X})$ could never be finite: Assume $w \in \Gamma^*$ such that $w\,0^\omega \in \mathrm{dom}_\mathrm{X}(\mathcal{X})$. The machine can only see a finite part of the right side of the tape before accepting. Thus, there is a $v \in \Gamma^*$ such that both $wv1\,0^\omega \in \mathrm{dom}_\mathrm{X}(\mathcal{X})$ and $wv0\,0^\omega \in \mathrm{dom}_\mathrm{X}(\mathcal{X})$, as $\mathcal{X}$ will not reach the part where $wv1$ and $wv0$ differ. This observation leads to $wvx\,0^\omega \in \mathrm{dom}_\mathrm{X}(\mathcal{X})$ for every $x \in \Gamma^*$, and applies to various other extensions of the Turing machine model. As Lemma 4.13 – and thereby most of the main results in Section 4.2.5 – crucially depends on the fact that there are extended Turing machines with a finite domain, we use $\mathrm{CHECK}_\mathrm{R}$ to allow our machines to perform additional sanity checks on the input and to overcome the limitations that arise from the lack of the input markers ¢ and \$.

Using a classical coding technique for two-symbol Turing machines (see Minsky [70]) and the corresponding undecidability results, we establish the following negative results on decision problems for extended Turing machines:

Lemma 4.10. *Consider the following decision problems for extended Turing machines:*

Emptiness: *Given an extended Turing machine $\mathcal{X}$, is $\mathrm{dom}_\mathrm{X}(\mathcal{X})$ empty?*

Finiteness: *Given an extended Turing machine $\mathcal{X}$, is $\mathrm{dom}_\mathrm{X}(\mathcal{X})$ finite?*

Then emptiness is not decidable, and neither finiteness nor its complement are semi-decidable.

Proof. We show these results on extended Turing machines by reducing each of these problems for "non-extended" Turing machines (or, as we call them, *general Turing machines*, to its counterpart for extended Turing machines. A general Turing machine is a 7-tuple $\mathcal{M} = (Q, q_1, \hat{\Gamma}, 0, ¢, \$, \delta)$, where Q is a finite set of states, q_1 is the initial state, $0 \in \hat{\Gamma}$ is the blank tape symbol, $¢, \$ \in \hat{\Gamma}$ are distinct special symbols (with $¢, \$ \neq 0$) that are used to mark the beginning and end (respectively) of an input of $\mathcal{M}$, and

$$\delta : \hat{\Gamma} \times Q \to (\hat{\Gamma} \times \{L, R\} \times Q) \cup \{\mathrm{HALT}\}$$

is the transition function. We interpret δ as for extended Turing machines, and use the same notion of tape words and configurations as for extended Turing machines.

The *domain* $\mathrm{dom}_\mathrm{T}(\mathcal{M})$ *of a general Turing machine* $\mathcal{M} = (Q, q_1, \hat{\Gamma}, 0, ¢, \$, \delta)$, is defined to be the set of all $w \in (\hat{\Gamma} \setminus \{¢, \$\})^*$ such that $\mathcal{M}$, if started in the configuration $(0^\omega, t_R, 0, q_1)$ with $t_R = ¢w\$\,0^\omega$, halts after finitely many steps.

The definition of $\mathrm{dom}_\mathrm{T}(\mathcal{M})$ corresponds to the definition of the language of a Turing machine as given by Hopcroft and Ullman [46] and Minsky [70]. As for extended Turing machines, we consider the following decision problems for general Turing machines:

Emptiness: Given a general Turing machine $\mathcal{M}$, is $\mathrm{dom}_\mathrm{T}(\mathcal{M})$ empty?

Finiteness: Given a general Turing machine $\mathcal{M}$, is $\mathrm{dom}_\mathrm{T}(\mathcal{M})$ finite?

Emptiness of dom_T is undecidable due to Rice's Theorem; and due to the Rice-Shapiro Theorem, both finiteness of dom_T and its complement are not semi-decidable (cf. Cutland [21], Hopcroft and Ullman [46][2]).

[2] In [46], the Rice-Shapiro Theorem is called "Rice's Theorem for recursively enumerable index sets" (Chapter 8.4).

In order to prove the present lemma's claims on extended Turing machines, we now define an effective procedure that, given a general Turing machine $\mathcal{M}$, returns an extended Turing machine $\mathcal{X}$ such that $\mathrm{dom}_{\mathrm{T}}(\mathcal{M})$ is empty (or finite) if and only if $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ is empty (or finite).

First, assume that $\mathcal{M}$ is defined over some tape alphabet $\hat{\Gamma} \supseteq \{\$, ¢, 0\}$. Using the common technique to simulate Turing machines with larger tape alphabets on Turing machines with a binary tape alphabet (cf. Chapter 6.3.1 in Minsky [70]), we choose a $k \geq 1$ with $2^k \geq |\hat{\Gamma}|$ and fix any injective function $b_k : \hat{\Gamma} \to \Gamma^k$ (every letter from $\hat{\Gamma}$ is encoded by a block of k letters from Γ), with $b_k(0) = 0^k$ (the blank symbol of $\mathcal{M}$ is mapped to k successive blank symbols of $\mathcal{X}$). We extend this function b_k canonically to an injective morphism $b_k : \hat{\Gamma}^* 0^\omega \to \Gamma^* 0^\omega$. Moreover, we partition the tape of $\mathcal{X}$ into non-overlapping blocks of k tape cells, each representing a single tape cell of $\mathcal{M}$ as encoded by b_k.

The main idea of the construction is that $\mathcal{X}$ works in two phases. First, it checks that its right tape word is $b_k(¢\hat{w}\$\, 0^\omega) = b_k(¢\hat{w}\$)\, 0^\omega$ for some $\hat{w} \in (\hat{\Gamma} \setminus \{¢, \$\})^*$. If this is the case, $\mathcal{X}$ simulates $\mathcal{M}$, always reading blocks of k letters at a time and interpreting every block $b_k(a)$ as input a for $\mathcal{M}$.

More explicitly, the first phase works as follows: If started on an input $w \in \Gamma^* 0^\omega$, $\mathcal{X}$ scans w and checks whether the first block of k letters is $b_k(¢)$ (using its finite control to store the $k-1$ letters of the block, and evaluating the whole block after reading its k-th letter). If this is not the case, $\mathcal{X}$ enters an infinite loop (and thus, rejects implicitly). Otherwise, $\mathcal{X}$ continues scanning to the right, evaluating every block of k letters until a block with $b_k(\$)$ is encountered. On its way to the right, $\mathcal{X}$ performs the following checks: If a block contains some $b_k(a)$ with $a \in (\hat{\Gamma} \setminus \{¢, \$\})$, $\mathcal{X}$ examines the next block. If a block contains $b_k(¢)$ or some sequence of k letters that is not an image of any letter from $\hat{\Gamma}$, $\mathcal{X}$ enters an infinite loop. If a k-letter block containing $b_k(\$)$ is found, $\mathcal{X}$ moves the head to the last letter of this block and executes the $\mathrm{CHECK}_{\mathrm{R}}$-command. This leads the machine to enter an infinite loop if any occurrence of the non-blank symbol $1 \in \Gamma$ follows.

Thus, if there is no $\hat{w} \in (\hat{\Gamma} \setminus \{¢, \$\})^*$ such that $w = b_k(¢\hat{w}\$)\, 0^\omega$, $\mathcal{X}$ will never find a block $b_k(\$)$, and will never halt. Intuitively, $\mathcal{X}$ (implicitly) rejects any input that does not satisfy its sanity criteria by refusing to halt.

But if $w = b_k(¢\hat{w}\$)\, 0^\omega$ for some $\hat{w} \in (\hat{\Gamma} \setminus \{¢, \$\})^*$, no tape cell containing 1 is found. Then $\mathcal{X}$ enters its second phase: The machine returns to the left side of w (which it recognizes by the unique block containing $b_k(¢)$), and simulates $\mathcal{M}$ on the corresponding input $¢\hat{w}\$$ with $b_k(¢\hat{w}\$)\, 0^\omega = w$, always using the finite control to read blocks $b_k(a)$ of length k which represent a tape letter $a \in \hat{\Gamma}$ as input for $\mathcal{M}$, and halting if and only if $\mathcal{M}$ halts. By definition, the left tape side is initially empty; hence, due to $b_k(0) = 0^k$, and due to the sanity check using $\mathrm{CHECK}_{\mathrm{R}}$, we do not even need to keep track which part of the tape $\mathcal{X}$ has already seen.

Thus, if $w \in \mathrm{dom}_{\mathrm{X}}(\mathcal{X})$, there is exactly one $\hat{w} \in (\Gamma \setminus \{¢, \$\})^*$ with $\hat{w} \in \mathrm{dom}_{\mathrm{T}}(\mathcal{M})$ and $b_k(¢\hat{w}\$) = w$. Likewise, for every $\hat{w} \in \mathrm{dom}_{\mathrm{T}}(\mathcal{M})$ (which, by definition, implies that $\hat{w}$ does not contain any ¢ or \$), $b_k(¢\hat{w}\$)\, 0^\omega \in \mathrm{dom}_{\mathrm{X}}(\mathcal{X})$. Thus, $\mathrm{dom}_{\mathrm{T}}(\mathcal{M}) = \emptyset$ if and only if $\mathrm{dom}_{\mathrm{X}}(\mathcal{X}) = \emptyset$, and likewise, $\mathrm{dom}_{\mathrm{T}}(\mathcal{M})$ is finite if and only if $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ is finite.

As the whole construction process can be realized effectively, any algorithm that (semi-)decides any of these two problems for extended Turing machines could be con-

verted into an algorithm that (semi-)decides the corresponding problem for general Turing machines.

Due to the fact that emptiness of the domain for general Turing machines is not decidable, and as neither finiteness nor its complement is semi-decidable, the claim follows. □

Those who are interested in these problems' exact position in the arithmetical hierarchy (cf. Odifreddi [77]) can use Propositions X.9.5 and X.9.6 from Odifreddi [78] and the canonical reasoning on the order of quantifiers for the respective levels to observe that – for general and for extended Turing machines – emptiness of the domain is Π^0_1-complete, while finiteness of the domain is Σ^0_2-complete (hence, its complement is Π^0_2-complete).

In order to simplify some technical aspects of our further proofs below, we adopt the following convention on extended Turing machines:

Convention 4.11. *Every extended Turing machine*

1. *has the state set* $Q = \{q_1, \ldots, q_\nu\}$ *for some* $\nu \geq 1$, *where* q_1 *is the initial state,*
2. *has* $\delta(0, q_1) = (0, L, q_2)$,
3. *has* $\delta(a, q) = \text{HALT}$ *for at least one 2-tuple* $(a, q) \in \Gamma \times Q$.

Obviously, every extended Turing machine can be directly (and effectively) adapted to satisfy these criteria.

As every tape word contains only finitely many occurrences of 1, we can interpret tape sides as natural numbers in the following (canonical) way which we already used in the proof of Theorem 3.10 in Chapter 3: For sequences $t = (t_i)_{i=0}^\infty$ over Γ, define

$$\mathrm{e}(t) := \sum_{i=0}^{\infty} 2^i \,\mathrm{e}(t_i),$$

where $\mathrm{e}(0) := 0$ and $\mathrm{e}(1) := 1$. Most of the time, we will not distinguish between single letters and their values under e, and simply write a instead of $\mathrm{e}(a)$ for all $a \in \Gamma$. It is easily seen that e is a bijection between $\mathbb{N}$ and $\Gamma^* 0^\omega$, the set of all tape words over Γ. Intuitively, every tape word is read as a binary number, starting with the cell closest to the head as the least significant bit, extending toward infinity.

Expressing the three parts of the tape (left and right tape word and head symbol) as natural numbers allows us to compute the tape parts of successor configurations using elementary integer operations. The following observation is straightforward, but a very important tool for out proof of Theorem 4.14 further down:

Observation 4.12. *Assume that an extended Turing machine* $\mathcal{X} = (Q, q_1, \delta)$ *is in some configuration* $C = (t_L, t_R, a, q_i)$, *and* $\delta(a, q_i) = (b, M, q_j)$ *for some* $b \in \Gamma$, *some* $M \in \{L, R\}$ *and some* $q_j \in Q$. *For the (uniquely defined) successor configuration* $C' = (t'_L, t'_R, a', q_j)$ *with* $C \vdash_{\mathcal{X}} C'$, *the following holds:*

$$\begin{array}{llll} \textit{If } M = L\text{:} & \mathrm{e}(t'_L) = \mathrm{e}(t_L) \operatorname{div} 2, & \mathrm{e}(t'_R) = 2\,\mathrm{e}(t_R) + b, & a' = \mathrm{e}(t_L) \bmod 2, \\ \textit{if } M = R\text{:} & \mathrm{e}(t'_L) = 2\,\mathrm{e}(t_L) + b, & \mathrm{e}(t'_R) = \mathrm{e}(t_R) \operatorname{div} 2, & a' = \mathrm{e}(t_R) \bmod 2. \end{array}$$

These equations are fairly obvious, especially considering Figure 4.2 and our previous reasoning on the Observation 3.1 (the analogous observation on the configurations of the universal machine $\mathcal{U}$). When moving the head in direction M, $\mathcal{X}$ turns the tape cell that contained the least significant bit of $\mathrm{e}(t_M)$ into the new head symbol, while the other tape side gains the tape cell containing the new letter b that was written over the head symbol as new least significant bit.

Using the encoding e, we define an encoding $\mathrm{enc}_{\mathrm{X}}$ of configurations of $\mathcal{X}$ by

$$\mathrm{enc}_{\mathrm{X}}\left(t_L, t_R, a, q_i\right) := 00^{\mathrm{e}(t_L)}\#00^{\mathrm{e}(t_R)}\#00^{\mathrm{e}(a)}\#0^i$$

for every configuration (t_L, t_R, a, q_i) of $\mathcal{X}$. We extend $\mathrm{enc}_{\mathrm{X}}$ to an encoding of finite sequences $C = (C_i)_{i=1}^n$ (where every C_i is a configuration of $\mathcal{X}$) by

$$\mathrm{enc}_{\mathrm{X}}(C) := \#\#\,\mathrm{enc}_{\mathrm{X}}(C_1)\#\#\,\mathrm{enc}_{\mathrm{X}}(C_2)\#\#\ldots\#\#\,\mathrm{enc}_{\mathrm{X}}(C_n)\#\#.$$

A *valid computation of* $\mathcal{X}$ is a sequence $C = (C_i)_{i=1}^n$ of configurations of $\mathcal{X}$ where C_1 is an initial configuration (i. e. some configuration $(0^\omega, w, 0, q_1)$ with $w \in \Gamma^* 0^\omega$), C_n is a halting configuration, and for every $i < n$, $C_i \vdash_{\mathcal{X}} C_{i+1}$. Thus, let

$$\begin{aligned}\mathrm{VALC}(\mathcal{X}) &= \{\mathrm{enc}_{\mathrm{X}}(C) \mid C \text{ is a valid computation of } \mathcal{X}\},\\ \mathrm{INVALC}(\mathcal{X}) &= \{0, \#\}^* \setminus \mathrm{VALC}(\mathcal{X}).\end{aligned}$$

The main tool in the proof of Theorem 4.15 is Theorem 4.14 (still further down), which states that, given an extended Turing machine $\mathcal{X}$, one can effectively construct an expression from RegEx(1) that generates $\mathrm{INVALC}(\mathcal{X})$. Note that in $\mathrm{enc}_{\mathrm{X}}(C)$, ## serves as a boundary between the encodings of individual configurations, which will be of use in the proof of Theorem 4.14. Building on Convention 4.11, we observe the following fact on the regularity of $\mathrm{VALC}(\mathcal{X})$ for a given extended Turing machine $\mathcal{X}$:

Lemma 4.13. *For every extended Turing machine* $\mathcal{X}$, $\mathrm{VALC}(\mathcal{X})$ *is regular if and only if* $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ *is finite.*

Proof. The *if* direction follows immediately: As $\mathcal{X}$ is deterministic, every word in $\mathrm{VALC}(\mathcal{X})$ corresponds to exactly one word from $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ (and the computation of $\mathcal{X}$ on that word). Thus, if $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ is finite, $\mathrm{VALC}(\mathcal{X})$ is also finite, and thus, regular.

For the *only if* direction, let $\mathcal{X} = (Q, q_1, \delta)$, and assume that $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ is infinite, while $\mathrm{VALC}(\mathcal{X})$ is regular. The main idea of the proof is to show that this assumption implies the regularity of the language

$$L_{\mathcal{X}} := \{0^{\mathrm{e}(t_R)}\#0^{\mathrm{e}(t_R)} \mid t_R \in \mathrm{dom}_{\mathrm{X}}(\mathcal{X})\}.$$

Due to $L_{\mathcal{X}}$ being an infinite subset of $\{0^n\#0^n \mid n \geq 0\}$, we can then obtain a contradiction using the Pumping Lemma. In order to achieve this result, we use our convention that $\mathcal{M}$ does not halt on the very first configuration (cf. Convention 4.11).

As $\mathcal{X}$ is deterministic, every word $w \in \mathrm{VALC}(\mathcal{X})$ corresponds to exactly one tape word $t_R \in \mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ and its accepting computation. This means that w has a prefix that encodes the initial configuration (t_L, t_R, a, q_1) with $t_L = 0^\omega$ and $a = 0$, and its successor configuration (t'_L, t'_R, a', q_j). Recall that, by Convention 4.11, $\delta(0, q_1) = (0, L, q_2)$. Using

the first equation in Observation 4.12, we conclude $\mathrm{e}(t'_L) = 0$, $\mathrm{e}(t'_R) = 2\,\mathrm{e}(t_R)$, and $a' = 0$. This means that there are $w_p, w' \in \{0, \#\}^*$ such that $w = w_p w'$, and

$$w_p = \#\#0\underbrace{}_{\mathrm{e}(t_L)}\#0\underbrace{0^{\mathrm{e}(t_R)}}_{\mathrm{e}(t_R)}\#0\underbrace{}_{a}\#\underbrace{0}_{q_1}\#\#0\underbrace{}_{\mathrm{e}(t'_L)}\#0\underbrace{0^{\mathrm{e}(t'_R)}}_{\mathrm{e}(t'_R)}\#0\underbrace{}_{a'}\#\underbrace{0^2}_{q_2}\#\#. \tag{4.1}$$

We now define a GSM M (see Section 4.2.3 for the definition of generalized sequential machines) to transform $\mathrm{VALC}(\mathcal{X})$ into the language $L_{\mathcal{X}}$. The GSM M is defined by the transition diagram in Figure 4.3.

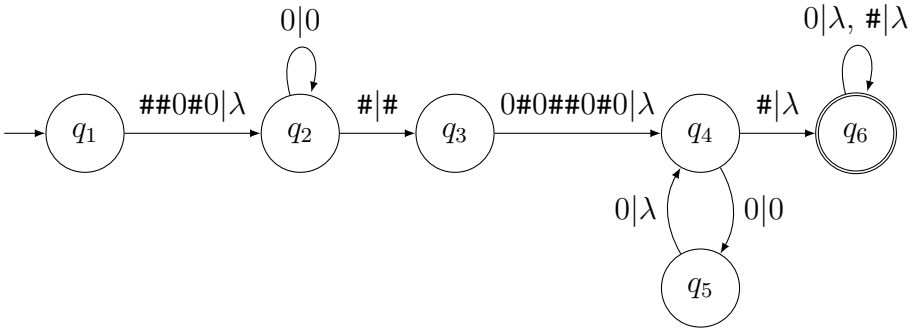

Figure 4.3: The GSM M that is used in the proof of Lemma 4.13. Every transition shows the string that is read to the left of the | symbol, and the emitted string to the right. First, M erases ##0#0 and keeps the following continuous block of 0s and the # after it. It then erases 0#0##0#0 and halves the number of 0s in the next continuous block of 0s (using the loop between q_4 and q_5). After that block (as recognizable by #), all following letters are erased. Note that M relies on the fact that it is only used on words $w = w_p w'$, where w_p of the form that is described in 4.1. For all such words, w' is completely erased in the loop in q_6.

Compared to the definition of generalized sequential machines from Section 4.2.3, this definition of M uses a slightly streamlined notation by allowing M to read multiple letters in the transition between q_1 and q_2 and between q_3 and q_4. By introducing additional states, one can easily convert M into a GSM that reads one letter after the other. E. g., instead of directly transitioning from q_1 to q_2 in a single step reading ##0#0 and emitting λ, M could be redefined by adding four additional states $q_{1,1}$ to $q_{1,4}$, such that M goes from q_1 to $q_{1,1}$ reading #, then $q_{1,2}$ reading #, then $q_{1,3}$ reading 0, then to $q_{1,4}$ reading #, and finally to q_2 reading 0, while emitting λ in every step.

Thus, for every $w \in \mathrm{VALC}(\mathcal{X})$, there are words $w_p, w' \in \{0, \#\}^*$ such that $w = w_p w'$, where w_p satisfies Equation 4.1 and encodes some values $\mathrm{e}(t_R)$ and $\mathrm{e}(t'_R)$. Now assume that M is applied to w. First note that M is deterministic, thus, $M(w)$ can contain at most one word (in principle, $M(w)$ could even be empty if M does not reach the accepting state q_6, but considering the structure of all words in $\mathrm{VALC}(\mathcal{X})$, it is fairly easy to see that this is impossible).

When transitioning from q_1 to q_2, M removes the prefix ##0#0 from w. This erases the encoding of $\mathrm{e}(t_L)$, the surrounding #, and the 0 that precedes the encoding of $\mathrm{e}(t_R)$. While M loops in q_2, every 0 from the encoding of $\mathrm{e}(t_R)$ is passed through unchanged, as is the # that marks the end of that encoding when transitioning to q_2. Therefore, when M has reached q_3, it has emitted $0^{\mathrm{e}(t_R)}\#$ and read the part $\#\#0\#00^{\mathrm{e}(t_R)}\#$ of the input.

In the transition from q_3 to q_4, M reads 0#0##0#0 and emits only λ – thus, the GSM passes over those parts of w_p that encode a, q_1 (of $\mathcal{X}$) and $\mathrm{e}(t'_L)$, and over the 0 that precedes the encoding of $\mathrm{e}(t'_R)$, and emits only λ. In the loop between q_4 and q_5, M emits half of the 0s that encode $e(t'_R)$. When reaching the # that marks the end of that block, M transitions to q_6 and erases all remaining letters of w_p and all of w'. Thus, $M(w) = 0^{\mathrm{e}(t_R)}\#0^{\frac{1}{2}\,\mathrm{e}(t'_R)} = 0^{\mathrm{e}(t_R)}\#0^{\mathrm{e}(t_R)}$. Hence, if we consider not a single $w \in \mathrm{VALC}(\mathcal{X})$, but the whole language, we have

$$\begin{aligned} M(\mathrm{VALC}(\mathcal{X})) &= \bigcup_{w \in \mathrm{VALC}(\mathcal{X})} M(w) \\ &= \{0^{\mathrm{e}(t_R)}\#0^{\mathrm{e}(t_R)} \mid t_R \in \mathrm{dom}_{\mathrm{X}}(\mathcal{X})\} = L_{\mathcal{X}}. \end{aligned}$$

By our initial assumption, $\mathrm{VALC}(\mathcal{X})$ is regular, and as the class of regular languages is closed under GSM mappings, $L_{\mathcal{X}}$ must be regular as well. Also by our initial assumption, $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ is infinite, which means that $L_{\mathcal{X}}$ is an infinite subset of $\{0^n\#0^n \mid n \geq 0\}$. Using the Pumping Lemma (cf. Hopcroft and Ullman [46]), we can obtain the intended contradiction, as pumping any sufficiently large word from $L_{\mathcal{X}}$ would lead to a word that is not a subset of $0^*\#0^*$, or to a word $0^m\#0^n$ with $m \neq n$. □

4.2.5 The Main Construction

Using the technical preparations from Sections 4.2.3 and 4.2.4, we are now able to state Theorem 4.14, the main technical result that forms the fundament for the undecidability result Theorem 4.15, on which in turn all further results in the present chapter are based. In order to ease navigation of the text, this section contains only the statement and its proof; the consequences of Theorem 4.14 (most importantly, Theorem 4.15) are explained in Section 4.3.

Theorem 4.14. *For every extended Turing machine $\mathcal{X}$, one can effectively construct an extended regular expression $\alpha_{\mathcal{X}} \in \mathrm{RegEx}(1)$ such that $L(\alpha_{\mathcal{X}}) = \mathrm{INVALC}(\mathcal{X})$.*

Proof. Let $\mathcal{X} = (Q, q_1, \delta)$ be an extended Turing machine. Let $\nu \geq 2$ denote the number of states of $\mathcal{X}$; by Convention 4.11, $Q = \{q_1, \ldots, q_\nu\}$ for some $\nu \geq 2$. Intuitively, each element w of $\mathrm{INVALC}(\mathcal{X})$ contains at least one error that prevents w from being an encoding of a valid computation of $\mathcal{X}$. We distinguish two kinds of errors:

1. *structural errors*, where a word is not an encoding of any sequence $(C_i)_{i=1}^n$ over configurations of $\mathcal{X}$ for some n, or the word is such an encoding, but C_1 is not an initial, or C_n is not a halting configuration, and

2. *behavioral errors*, where a word is an encoding of some sequence of configurations $(C_i)_{i=1}^n$ of $\mathcal{X}$, but there is an $i < n$ such that $C_i \vdash_{\mathcal{X}} C_{i+1}$ does not hold.

The extended regular expression $\alpha_{\mathcal{X}}$ is defined by

$$\alpha_{\mathcal{X}} := \alpha_{struc} \mid \alpha_{beha},$$

where the subexpressions α_{struc} and α_{beha} describe all structural and all behavioral errors, respectively. Both expressions shall be defined later. Note that the variable reference

mechanism shall be used only for some extended regular expressions in α_{beha}; most of the encoding of INVALC($\mathcal{X}$) can be achieved with proper regular expressions. In order to define α_{struc}, we take a short detour and consider the language

$$\begin{aligned} S_{\mathcal{X}} := & \left(\#\#0^+\#0^+\#0\{\lambda,0\}\#\left\{0^i \mid 1 \leq i \leq \nu\right\}\right)^+\#\# \\ & \cap \#\#0\#0^+\#0\#0\#\#\{0,\#\}^* \\ & \cap \{0,\#\}^*\#\left\{00^a\#0^i\#\# \mid a \in \Gamma, \delta(a,q_i) = \text{HALT}\right\}. \end{aligned}$$

Note that $S_{\mathcal{X}}$ is exactly the set of all $\text{enc}_{\text{X}}(C)$, where $C = (C_i)_{i=1}^n$ (with $n \geq 2$) is a sequence of configurations of $\mathcal{X}$; with C_1 being an initial configuration (where neither the left tape side nor the head cell contain any 1), and C_n being a halting configuration ($n \geq 2$ follows from our Convention 4.11 that $\mathcal{X}$ cannot halt in the first step). In other words, all that distinguishes $S_{\mathcal{X}}$ from VALC($\mathcal{X}$) is that for $S_{\mathcal{X}}$, we do not require that $C_i \vdash_{\mathcal{X}} C_{i+1}$ holds for all $i < \nu$. Thus, VALC($\mathcal{X}$) $\subseteq S_{\mathcal{X}}$.

Furthermore, $S_{\mathcal{X}}$ is a regular language, as it is obtained by an intersection of three regular languages. Thus, $\{0,\#\}^* \setminus S_{\mathcal{X}}$ is also a regular language, and we define α_{struc} to be any proper regular expression with $L(\alpha_{struc}) = \{0,\#\}^* \setminus S_{\mathcal{X}}$. It is easy to see that such an α_{struc} can be constructed effectively solely from $\mathcal{X}$, for example by constructing a deterministic finite automaton A for $S_{\mathcal{X}}$, complementing A (by turning accepting into non-accepting states, and vice versa), and converting the resulting nondeterministic automaton into a proper regular expression. The DFA A depends only on ν and the halting instructions occurring in δ and can be constructed effectively, as can all the conversions that lead to α_{struc} (again, cf. Hopcroft and Ullman [46]). The exact shape of α_{struc} is of no significance to this proof, as we require only that the expression is a proper regular expression, and can be obtained effectively.

As mentioned above, VALC($\mathcal{X}$) $\subseteq S_{\mathcal{X}}$, and thus, INVALC($\mathcal{X}$) $\supseteq L(\alpha_{struc})$. Furthermore, all elements of INVALC($\mathcal{X}$) $\setminus L(\alpha_{struc})$ are elements of $S_{\mathcal{X}}$ and encode a sequence $(C_i)_{i=1}^n$ ($n \geq 2$) of configurations of $\mathcal{X}$ such that $C_i \vdash_{\mathcal{X}} C_{i+1}$ does not hold for at least one i, $1 \leq i < n$.

Thus, INVALC($\mathcal{X}$) $\setminus L(\alpha_{struc})$ contains exactly those words from $S_{\mathcal{X}}$ that encode a sequence of configurations with at least one behavioral error. Therefore, when defining α_{beha} to describe all these remaining errors, we can safely assume that the word in question is an element of $S_{\mathcal{X}}$, as otherwise, it is already contained in $L(\alpha_{struc})$. This allows us to reason about the yet to be defined elements of INVALC($\mathcal{X}$) purely in terms of the execution of $\mathcal{X}$, as the encoding is already provided by the structure of $S_{\mathcal{X}}$, and to understand all errors that are yet to be defined as incorrect transitions between configurations.

We distinguish three kinds of behavioral errors in the transition between a configuration $C = (t_L, t_R, a, q_i)$ and a configuration $C' = (t'_L, t'_R, a', q_j)$, where $C \vdash_{\mathcal{X}} C'$ does not hold:

1. *state errors*, where q_j has a wrong value,
2. *head errors*, where a' is wrong, and
3. *tape side errors*, where t'_L or t'_R contains an error (characterized by $\text{e}(t'_L)$ or $\text{e}(t'_R)$ being different from the value that is expected according to Observation 4.12).

Each of these types of errors shall be handled by an expression α_{state}, α_{head} or α_{tape} (respectively, of course), and we define

$$\alpha_{beha} := (\alpha_{state} \mid \alpha_{head} \mid \alpha_{tape}).$$

Basically, each of these expressions lists all combinations of $a \in \Gamma$ and $q_i \in Q$, and describes the corresponding errors of the respective kind. The error that $\mathcal{X}$ continues its computation after encounter a HALT-instruction is considered a state error and handled in α_{state} (thus, we do not need to consider HALT-instructions in α_{head} and α_{tape}). We can already note that α_{head} and α_{state} are proper regular expressions, as variables and the % metacharacter occur only in α_{tape} (recall that, as $\alpha_{\mathcal{X}} \in \mathrm{RegEx}(1)$, we are only allowed to use % once in the whole expression).

State errors: We begin with the definition of α_{state}. For every $a \in \Gamma$ and every i with $q_i \in Q$, we define a proper regular expression $\alpha_{a,i}^{state}$, and let

$$\alpha_{state} := \left(\alpha_{0,1}^{state} \mid \alpha_{1,1}^{state} \mid \alpha_{0,2}^{state} \mid \alpha_{1,2}^{state} \mid \ldots \mid \alpha_{0,\nu}^{state} \mid \alpha_{1,\nu}^{state}\right),$$

where each $\alpha_{a,i}^{state}$ lists all 'forbidden' follower states for q_i on a. More formally, if $\delta(a, q_i) = \mathrm{HALT}$, let

$$\alpha_{a,i}^{state} := (0 \mid \#)^* \#00^a \#0^i \#\#0(0 \mid \#)^*.$$

For all words in $S_{\mathcal{X}}$, this expression describes all cases where $\mathcal{X}$ reads a in state q_i, and continues instead of halting. First, as mentioned above, we only need to consider words from $S_{\mathcal{X}}$, as all other words are already matched by α_{struc}. Due to the definition of $\mathrm{enc}_{\mathrm{X}}$, every ## in words from $S_{\mathcal{X}}$ marks the boundary between two encoded configurations, and every string $\#0^i$ immediately to the left of such a ## encodes a state q_i. Likewise, when continuing to the left, $\#00^a$ encodes the head letter a. Thus, whenever a word from $S_{\mathcal{X}}$ contains a string $\#00^a\#0^i\#\#$, there is a configuration where $\mathcal{X}$ is in state q_i and reads a. As $\delta(a, q_i) = \mathrm{HALT}$, there may not be a succeeding configuration, and this definition of $\alpha_{a,i}^{state}$ describes all cases where $\mathcal{X}$ continues after reading a in q_i. Note that we do not need to deal with cases where ## is followed by yet another #, as such words are not contained in $S_{\mathcal{X}}$ and, thus, contained in $L(\alpha_{struc})$.

For those cases where $\delta(a, q_i) = (b, M, q_j)$ for some $M \in \{L, R\}$, some $b \in \Gamma$, and some $q_j \in Q$, we define

$$\alpha_{a,i}^{state} := (0 \mid \#)^* \#00^a \#0^i \#\#0^+ \#0^+ \#0^+ \#\alpha_j^{not} \#\#(0 \mid \#)^*,$$

where α_j^{not} is any proper regular expression with

$$L(\alpha_j^{not}) = \{0^k \mid 1 \leq k \leq \nu \text{ and } k \neq j\}.$$

Again, we use $\#00^a\#0^i\#\#$ to identify an encoding of a configuration with head letter a in state q_i. To the right of ##, the subexpression $0^+\#0^+\#0^+\#$ is used to skip over the encodings of t'_L, t'_R and a', as we only deal with state errors (for now). By definition, the invalid successor states are exactly all states from $Q \setminus \{q_j\}$, and these are described by α_j^{not}. Thus, if a word from $S_{\mathcal{X}}$ contains any state error when reading a in q_i, the whole word belongs to $\alpha_{a,i}^{state}$, and $\alpha_{a,i}^{state}$ only matches such words.

Finally, if $\delta(a, q_i) = (\text{CHECK}_\text{R}, q_j)$ for some $q_j \in Q$, we define

$$\begin{aligned}\alpha_{a,i}^{state} := &\big((0 \mid \#)^*\#0\#00^a\#0^i\#\#0^+\#0^+\#0^+\#\alpha_j^{not}\#\#(0 \mid \#)^*\big)\\ &\mid \big((0 \mid \#)^*\#00^+\#00^a\#0^i\#\#0^+\#0^+\#0^+\#\alpha_i^{not}\#\#(0 \mid \#)^*\big)\,,\end{aligned}$$

where α_j^{not} is defined as in the preceding paragraph. This expression is slightly more complicated, as it needs to distinguish two cases. Recall that the CHECK_R-instruction is to be interpreted as follows: If $t_R = 0^\omega$, $\mathcal{X}$ is to change into state q_j; and if $t_R \neq 0^\omega$, $\mathcal{X}$ is to stay in q_i, which will lead to an infinite loop. The first line of the definition handles all cases where $t_R = 0^\omega$, while the second handles those where $t_R \neq 0^\omega$. Again, both cases use $\#00^a\#0^i\#\#$ to identify configurations where $\mathcal{X}$ is in state q_i reading a.

In the first case, the string $\#0\#00^a\#0^i\#\#$ contains the additional information that $\text{e}(t_R) = 0$, and thus, $t_R = 0^\omega$. The correct successor state would be q_j, and the expression skips over the encodings of t'_L, t'_R and a' (using $0^+\#0^+\#0^+\#\alpha_j^{not}\#$) and then matches all states but q_j.

Likewise, in the second case, $\#00^+\#00^a\#0^i\#\#$ matches all cases where (when reading a in q_i) $\text{e}(t_R) > 0$, which is equivalent to $t_R \neq 0^\omega$. Again, the expression skips over the encodings of t'_L, t'_R and a' and uses α_i^{not} to identify all states that are not the correct successor state q_i.

Head errors: As α_{state} handles all cases where a halting configuration is followed by any other configuration, we can restrict our definition of the various head errors to cases where a non-halting instruction should be executed. We define

$$\alpha_{head} := \big(\alpha_{0,1}^{head} \mid \alpha_{1,1}^{head} \mid \alpha_{0,2}^{head} \mid \alpha_{1,2}^{head} \mid \ldots \mid \alpha_{0,\nu}^{head} \mid \alpha_{1,\nu}^{head}\big)\,,$$

omitting those $\alpha_{a,i}^{head}$ with $\delta(a, q_i) = \text{HALT}$. For all $a \in \Gamma$, $q_i \in Q$ with $\delta(a, q_i) \neq \text{HALT}$, we define $\alpha_{a,i}^{head}$ as follows.

If $\delta(a, q_i) = (b, L, q_j)$ (for some $q_j \in Q$), let

$$\begin{aligned}\alpha_{a,i}^{head} := &\big((0 \mid \#)^*\#0(00)^*\#0^+\#00^a\#0^i\#\#0^+\#0^+\#00\#(0 \mid \#)^*\big)\\ &\mid \big((0 \mid \#)^*\#00(00)^*\#0^+\#00^a\#0^i\#\#0^+\#0^+\#0\#(0 \mid \#)^*\big)\,.\end{aligned}$$

According to the first equation in Observation 4.12, after a left movement of the head, $a' = \text{e}(t_L) \bmod 2$ must hold. The two lines in the $\alpha_{a,i}^{head}$ distinguish the two possible cases for $\text{e}(t_L) \bmod 2$. In both cases, we once again identify a and q_i in the encoding using $\#00^a\#0^i\#$. In the first line, the expression ignores $\text{e}(t_R)$ (using the 0^+ to the left of $\#00^a\#$), and describes all cases where $\text{e}(t_L)$ is even (by the $(00)^*$ part of $\#0(00)^*$). To the right of $\#\#$, the expression skips t'_L and t'_R and finds $a' = 1$, thus exactly those cases where $\text{e}(t_L)$ is even, but $a' = 1$. Likewise, the second line handles the cases where $\text{e}(t_L)$ is odd, but $a' = 0$. As $a' \in \{0, 1\}$ is ensured by $S_\mathcal{X}$, these expressions describe exactly the head errors after L-movements.

Likewise, if $\delta(a, q_i) = (b, R, q_j)$ (for some $q_j \in Q$), let

$$\begin{aligned}\alpha_{a,i}^{head} := &\big((0 \mid \#)^*\#0(00)^*\#00^a\#0^i\#\#0^+\#0^+\#00\#(0 \mid \#)^*\big)\\ &\mid \big((0 \mid \#)^*\#00(00)^*\#00^a\#0^i\#\#0^+\#0^+\#0\#(0 \mid \#)^*\big)\,.\end{aligned}$$

This expression uses the second equation in Observation 4.12, $a' = \mathrm{e}(t_R) \bmod 2$, and works like the expression for L-moves, the only difference being that it does not skip over the encoding of t_R.

Finally, if $\delta(a, q_i) = \mathrm{CHECK_R}(q_j)$ for some $q_j \in Q$, we define

$$\alpha_{a,i}^{head} := (0 \mid \#)^* \#00^a \#0^i \#\#0^+ \#0^+ \#00^{1-a} \#(0 \mid \#)^*.$$

As $\mathrm{CHECK_R}$-instructions do not change the tape or the head symbol, we just need to describe the case where $a' \neq a$. The expression identifies an encoding of a configuration with head symbol a in state q_i (again using ## as a navigation tool), skips over t'_L and t'_R, and finds a head symbol $a' = 1$ if $a = 0$, or $a' = 0$ if $a = 1$. As a is fixed within every $\alpha_{a,i}^{head}$, we can use the shorthand notation 0^{1-a} without any formal problems (as it is just another notation for 0 or λ, depending on a).

Tape side errors: As mentioned above, α_{tape} is the only expression in this proof that uses variables and variable bindings. In fact, as we operate in RegEx(1), we are only allowed to use a single variable (which shall be called x), and bind it only once in all of α_{tape}.

In order to increase the readability, we shall define α_{tape} using numerous subexpressions. As most of these expressions contain the binding operator %, simply connecting them with | (as we did with the proper regular expressions in the previous cases) would force us out of RegEx(1). The main idea of this part of the proof is that, in all these expressions, the binding occurs only in a prefix that they all have in common. This allows us to 'factor out' the variable binding, and to capture all tape side errors without leaving RegEx(1). Therefore, we do not need not be worried by the fact that most of the following definitions contain $\%x$; as we shall see, the resulting expression α_{struc} contains only a single $\%x$.

In this section, we do not follow our usual order, as we discuss tape side errors for L- and R-instructions after the tape side errors for $\mathrm{CHECK_R}$-instructions. If $\delta(a, q_i)$ is a $\mathrm{CHECK_R}$-instruction, we define

$$\begin{aligned}
\alpha_{L,>}^{a,i} &:= (0 \mid \#)^* \#0(0^*)\%x\#0^+\#00^a\#0^i\#\#0x0^+\#(0 \mid \#)^*,\\
\alpha_{L,<}^{a,i} &:= (0 \mid \#)^* \#0(0^*)\%x0^+\#0^+\#00^a\#0^i\#\#0x\#(0 \mid \#)^*,\\
\alpha_{R,>}^{a,i} &:= (0 \mid \#)^* \#0(0^*)\%x\#00^a\#0^i\#\#0^+\#0x0^+\#(0 \mid \#)^*,\\
\alpha_{R,<}^{a,i} &:= (0 \mid \#)^* \#0(0^*)\%x0^+\#00^a\#0^i\#\#0^+\#0x\#(0 \mid \#)^*.
\end{aligned}$$

Intuitively, for $M \in \{L, R\}$, $\alpha_{M,>}^{a,i}$ is used to describe all successor configurations (after reading a in q_i), where $\mathrm{e}(t'_M) > \mathrm{e}(t_M)$. Likewise, $\alpha_{M,<}^{a,i}$ handles all those cases where $\mathrm{e}(t'_M) < \mathrm{e}(t_M)$. We discuss the correctness of these expressions using $\alpha_{L,>}^{a,i}$ as an example, the three other expressions behave analogously. If $\alpha_{L,>}^{a,i}$ matches a word from $S_{\mathcal{X}}$, x is bound to some word 0^n with $n \geq 0$ (due to $(0^*)\%x$). As this 0^n belongs to the fourth block of 0s when counting from ## to the left, $0(0)^n$ corresponds to $00^{e(t_L)}$ for a configuration (t_L, t_R, a, q_i). Analogously, the subexpression $\#\#0x0^+\#$ matches the block that encodes $\mathrm{e}(t'_L)$. As $\#\#00^n0^+\#$ is expanded to some $\#\#00^n0^m$ (with $m \geq 1$), we know that $\mathrm{e}(t'_L) = 0^m0^n > \mathrm{e}(t_L)$. Likewise, for every $\mathrm{e}(t'_L) > \mathrm{e}(t_L)$, we can find an appropriate $m \geq 1$ and expand 0^+ to 0^m. Thus, $\alpha_{L,>}^{a,i}$ matches exactly those cases in which $\mathcal{X}$ reads

a in q_i, and the resulting $\mathrm{e}(t'_L)$ is larger than it should be (i. e., larger than $\mathrm{e}(t_L)$, as $\mathrm{CHECK_R}$-instructions do not change the tape).

The three other expressions behave analogously; but note that for $\alpha^{a,i}_{R,>}$ and $\alpha^{a,i}_{R,<}$, the subexpression $0(0^*)\%x$ matches a the coding of $\mathrm{e}(t_R)$ instead of $\mathrm{e}(t_L)$, as can be seen by the location of ##.

Handling tape side errors for configurations that lead to a left or right movement of the head follows the same basic principle, but is a little more complicated, as we have to deal with changes to the tape. First, recall that the equations given in Observation 4.12 allow us to compute $\mathrm{e}(t'_L)$ and $\mathrm{e}(t'_R)$ from $\mathrm{e}(t_L)$, $\mathrm{e}(t_R)$ and $\delta(a, q_i)$.

If $\delta(a, q_i) \in (b, L, q_j)$ for some $q_j \in Q$, we define

$$\begin{aligned}
\alpha^{a,i}_{L,>} &:= (0 \mid \#)^*\#0(0^*)\%xx(0 \mid \lambda)\#0^+\#00^a\#0^i\#\#0x0^+\#(0 \mid \#)^*,\\
\alpha^{a,i}_{L,<} &:= (0 \mid \#)^*\#0(0^*)\%xx000^*\#0^+\#00^a\#0^i\#\#0x\#(0 \mid \#)^*,\\
\alpha^{a,i}_{R,>} &:= (0 \mid \#)^*\#0(0^*)\%x\#00^a\#0^i\#\#0^+\#0xx0^b0^+\#(0 \mid \#)^*,\\
\alpha^{a,i}_{R,<} &:= (0 \mid \#)^*\#0(0^*)\%x0^+\#00^a\#0^i\#\#0^+\#0xx0^b\#(0 \mid \#)^*,\\
\alpha^{a,i}_{mod} &:= (0 \mid \#)^*\#00^a\#0^i\#\#0^+\#0(00)^*0^{1-b}\#(0 \mid \#)^*.
\end{aligned}$$

These expressions fulfill the same purpose as their equally named counterparts we defined to handle tape side errors for configurations in which a $\mathrm{CHECK_R}$-instruction is executed, i. e., they describe the cases where $\mathrm{e}(t'_L)$ or $\mathrm{e}(t'_R)$ contains too much or too little. For technical reasons that shall be explained a little later, we also use a proper regular expression $\alpha^{a,i}_{mod}$ to describe the errors where $\mathrm{e}(t_R)$ has the wrong parity.

We begin with the expressions that handle errors on the left tape side. According to the first equation in Observation 4.12, the correct t'_L is characterized by $\mathrm{e}(t'_L) = \mathrm{e}(t_L) \operatorname{div} 2$.

First, we consider $\alpha^{a,i}_{L,>}$. As before, a and q_i are matched in $\#00^a\#0^i\#\#$. To the left of that block, the expression skips the encoding of $\mathrm{e}(t_R)$, and matches $00^{\mathrm{e}(t_L)}$ with the expression $0(0^*)\%xx(0 \mid \lambda)$. Note that x is bound to $0^{\mathrm{e}(t_L) \operatorname{div} 2}$, and $0^{\mathrm{e}(t_L) \bmod 2}$ matches 0 or λ in $(0 \mid \lambda)$. To the right of ##, $0x0^+$ matches the encoding of $00^{\mathrm{e}(t'_L)}$. It is easily seen that this describes exactly those cases where $\mathrm{e}(t'_L) > (\mathrm{e}(t_L) \operatorname{div} 2)$, as the 0^+ is used to match all possible values of $\mathrm{e}(t'_L) - (\mathrm{e}(t_L) \operatorname{div} 2)$.

Next, consider $\alpha^{a,i}_{L,<}$. Note that if $\mathrm{e}(t'_L) = (\mathrm{e}(t_L) \operatorname{div} 2)$, either $\mathrm{e}(t_L) = 2\,\mathrm{e}(t'_L)$ or $\mathrm{e}(t_L) = 2\,\mathrm{e}(t'_L) + 1$ holds. Thus, $\mathrm{e}(t'_L)$ is too small if and only if $\mathrm{e}(t_L) > \mathrm{e}(t'_L) + 1$, which holds if and only if $\mathrm{e}(t_L) = \mathrm{e}(t'_L)+2+m$ for some $m \geq 0$. We can easily see that (for words from $S_{\mathcal{X}}$), $\alpha^{a,i}_{L,<}$ matches exactly those encodings of successive configurations (t_L, t_R, a, q_i) and (t'_L, t_R, a', q_j), where $\mathrm{e}(t_L) = 2n + 2 + m$ and $\mathrm{e}(t'_L) = n$ for some $m, n \geq 0$, as x binds to 0^n, and $0(0^*)\%xx000^*$ corresponds to $00^n0^n000^m = 00^{\mathrm{e}(t_L)}$. Thus, this expression handles exactly those cases where $\mathrm{e}(t'_L)$ is too small.

Hence, $\alpha^{a,i}_{L,>}$ and $\alpha^{a,i}_{L,<}$ handle all errors on the left tape side (for given a, q_i with $\delta(a, q_i) = (b, L, q_j)$). As we still need to handle errors on the right tape side, recall that (according to the first equation in Observation 4.12), the correct right tape word t'_R is characterized by $\mathrm{e}(t'_R) = 2\,\mathrm{e}(t_R) + b$.

It is easy to see that $\alpha^{a,i}_{R,>}$ handles exactly those words (from $S_{\mathcal{X}}$, with $\mathcal{X}$ reading a in q_i) where $\mathrm{e}(t'_R)$ is too large. First, x is bound to $0^{\mathrm{e}(t_R)}$. For t'_R, we have $0\,xx0^b0^+$, and thus, $\mathrm{e}(t'_R) = 2\,\mathrm{e}(t_R) + b + n$, with $n \geq 1$, where 0^+ expresses the difference between $\mathrm{e}(t'_R)$ and its intended value.

The final case, where t'_R is too small, is more complicated. We handle this case with two expressions. First, note that $\mathrm{e}(t'_R) \bmod 2 = b$ must hold. The expression $\alpha^{a,i}_{mod}$ describes those cases where this condition is not satisfied; i. e., $\mathcal{X}$ reads a in q_i, but $\mathrm{e}(t'_R) \bmod 2 \neq b$. Therefore, we can restrict our definition of $\alpha^{a,i}_{R,<}$ to those cases where $\mathrm{e}(t'_R)$ and b have the same parity. In these cases, $\mathrm{e}(t'_R) = 2n + b$ for some $n \geq 0$, but $\mathrm{e}(t_R) > n$, which holds if and only if there is an m $geq0$ with $\mathrm{e}(t_R) = n+m+1$. The words from $S_\mathcal{X}$ that match $\alpha^{a,i}_{R,<}$ (but not $\alpha^{a,i}_{mod}$) are exactly those that satisfy this condition: The variable x is bound to 0^n, while 0^+ corresponds to $m+1$. Thus, $\alpha^{a,i}_{mod} \mid \alpha^{a,i}_{R,<}$ handles the cases where $\mathrm{e}(t'_R)$ is too small (but not exactly those cases, even when restricted to $S_\mathcal{X}$, as $\alpha^{a,i}_{mod}$ also matches encodings of computations where $\mathrm{e}(t'_R)$ is too large and of the wrong parity).

As we have seen, these five expressions describe all tape side errors occurring during left movements of the head. Analogously, if $\delta(a, q_i) \in (b, R, q_j)$ for some $q_j \in Q$, we define

$$\begin{aligned}
\alpha^{a,i}_{R,>} &:= (0 \mid \#)^*\#0(0^*)\%xx(0 \mid \lambda)\#00^a\#0^i\#\#0^+\#0x0^+\#(0 \mid \#)^*,\\
\alpha^{a,i}_{R,<} &:= (0 \mid \#)^*\#0(0^*)\%xx000^*\#00^a\#0^i\#\#0^+\#0x\#(0 \mid \#)^*,\\
\alpha^{a,i}_{L,>} &:= (0 \mid \#)^*\#0(0^*)\%x\#0^+\#00^a\#0^i\#\#0xx0^b0^+\#(0 \mid \#)^*,\\
\alpha^{a,i}_{L,<} &:= (0 \mid \#)^*\#0(0^*)\%x0^+\#0^+\#00^a\#0^i\#\#0xx0^b\#(0 \mid \#)^*,\\
\alpha^{a,i}_{mod} &:= (0 \mid \#)^*\#00^a\#0^i\#\#0(00)^*0^{1-b}\#(0 \mid \#)^*.
\end{aligned}$$

As already suggested by the similarities (of the equations in Observation 4.12, and of the definitions of the five expressions), tape side errors for R-movements can be handled analogously to tape side errors for L-movements. Thus, it can be easily verified that these expressions describe exactly the tape side errors for all transitions where $\mathcal{X}$ reads a in state q_i (again assuming that we only consider words from $S_\mathcal{X}$).

We can now combine all these expressions to define α_{tape}. First, note that whenever it is defined, $\alpha^{a,i}_{mod}$ is a proper regular expression. We define

$$\alpha_{mod} := \alpha^{0,1}_{mod} \mid \alpha^{1,1}_{mod} \mid \alpha^{0,2}_{mod} \mid \alpha^{1,2}_{mod} \mid \ldots \mid \alpha^{0,\nu}_{mod} \mid \alpha^{1,\nu}_{mod},$$

omitting those $\alpha^{a,i}_{mod}$ that are undefined (i. e., cases where $\delta(a, q_i)$ is a HALT- or a $\mathrm{CHECK_R}$-instruction).

Next, note that all other expressions for tape side errors use exactly one variable binding, and start with the common prefix $(0 \mid \#)^*\#0(0^*)\%x$. For every $\alpha^{a,i}_{M,c}$ (with $M \in \{L, R\}$ and $c \in \{>, <\}$), let $\hat{\alpha}^{a,i}_{M,c}$ be the (uniquely defined) extended regular expression that satisfies

$$\alpha^{a,i}_{M,c} = (0 \mid \#)^*\#0(0^*)\%x\ \hat{\alpha}^{a,i}_{M,c}.$$

In other words, $\hat{\alpha}^{a,i}_{M,c}$ is obtained from $\alpha^{a,i}_{M,c}$ by factoring out the prefix that contains the variable binding. We then combine all this expressions into a single expression

$\alpha_{var} \in \mathrm{RegEx}(1)$ by

$$\alpha_{var} = (0 \mid \#)^*\#0(0^*)\%x\Big(\hat{\alpha}^{0,1}_{L,>} \mid \hat{\alpha}^{0,1}_{L,<} \mid \hat{\alpha}^{0,1}_{R,>} \mid \hat{\alpha}^{0,1}_{L,R} \mid \hat{\alpha}^{1,1}_{L,>} \mid \hat{\alpha}^{1,1}_{L,<} \mid \hat{\alpha}^{1,1}_{R,>} \mid \hat{\alpha}^{1,1}_{L,R} \mid \ldots \mid \hat{\alpha}^{0,\nu}_{L,>} \mid \hat{\alpha}^{0,\nu}_{L,<} \mid \hat{\alpha}^{0,\nu}_{R,>} \mid \hat{\alpha}^{0,\nu}_{L,R} \mid \hat{\alpha}^{1,\nu}_{L,>} \mid \hat{\alpha}^{1,\nu}_{L,<} \mid \hat{\alpha}^{1,\nu}_{R,>} \mid \hat{\alpha}^{1,\nu}_{L,R}\Big),$$

omitting all subexpressions that refer to (a, q_i) where $\delta(a, q_i) = \mathrm{HALT}$. Finally, we set

$$\alpha_{tape} := \alpha_{mod} \mid \alpha_{var}.$$

As discussed before, it is easy to see that $L(\alpha_{tape})$ is the union of all the languages that are generated by the various regular expressions we defined to handle tape side errors, and thus, $L(\alpha_{tape})$ contains exactly those words from $S_{\mathcal{X}}$ that encode a tape side error at some point of the encoded computation. Therefore, for every word $w \in \{0, \#\}^*$, $w \in L(\alpha_{\mathcal{X}})$ if and only if

$$w \in L(\alpha_{struc}) \cup L(\alpha_{state}) \cup L(\alpha_{head}) \cup L(\alpha_{tape}),$$

and this holds if and only if w contains a structural or behavioral error. Hence, we observe that $w \in L(\alpha_{\mathcal{X}})$ if and only $w \in \mathrm{INVALC}(\mathcal{X})$, and thereby, $L(\alpha_{\mathcal{X}}) = \mathrm{INVALC}(\mathcal{X})$.

Finally, as this proof defines $\alpha_{\mathcal{X}}$ constructively, using only ν and δ, it also describes an effective procedure to compute $\alpha_{\mathcal{X}}$ from $\mathcal{X}$. This concludes the proof of Theorem 4.14. □

Note that in the proof of Theorem 4.14, the single variable x is bound only to words that match the expression 0^*. This shows that the "negative" properties we list in Section 4.3 hold even if we restrict RegEx(1) by requiring that the variable can only be bound to some proper regular expressions. In addition to this, it can be easily adapted to pattern expressions (cf. Câmpeanu and Yu [15]) and H-expressions (cf. Bordihn et al. [9]). These H-expressions are based on *H-systems*, which were introduced by Albert and Wegner [3].

Theorem 4.14 can even be adapted to these simpler H-systems. An H-system is a 4-tuple $H = (X, \Sigma, L_1, \phi)$, where X and Σ are finite alphabets (the *meta alphabet* and the *terminal alphabet*, respectively), $L_1 \subseteq X^*$ is called the *meta language*, and $\phi : X \to \mathcal{P}$ is a function that assigns to each $x \in X$ a language $\phi(x) = L_x \subseteq \Sigma^*$. The language of H is defined as

$$L(H) := \{h(w) \mid w \in L_1, h \text{ is a morphism with } h(x) \in \phi(x) \text{ for all } x \in X\}.$$

Less formally, every letter x from the meta alphabet is replaced uniformly with a word from $\phi(x)$.

Furthermore, if $\mathcal{L}_1$ and $\mathcal{L}_2$ are classes of languages, $\mathcal{H}(\mathcal{L}_1, \mathcal{L}_2)$ denotes the class of H-system languages of $\mathcal{L}_1$ and $\mathcal{L}_2$, i. e., the class of all languages that are generated by H-systems that use a language $L_1 \in \mathcal{L}_1$ as metalanguage, and have $\phi(x) \in \mathcal{L}_2$ for every x in their meta alphabet X.

As explained by Bordihn et al. [9], one can easily show that all classes of pattern languages are subclasses of $\mathcal{H}(\mathrm{ONE}, \mathrm{REG})$ (where ONE denotes the class of languages that contain only one word, and REG denotes the class of regular languages). It is easy to see that, for every extended Turing machine $\mathcal{X}$, the expressions $\alpha_{\mathcal{X}}$ that is derived from the proof of Theorem 4.14 can be easily converted into an H-system $H = (X_H, \Sigma_H, L_1, \phi)$ with $L(H) = \mathrm{INVALC}(\mathcal{X})$ and $L(H) \in \mathcal{H}(\mathrm{REG}, \mathrm{REG})$ by proceeding as follows.

First, we select $X_h{:=}\{0, \#, x\}$ and $\Sigma_H{:=}\{0, \#\}$. We then replace the single occurrence of $(0^*)\%x$ in $\alpha_{\mathcal{X}}$ with x, and obtain a proper regular expression $\hat{\alpha}_{\mathcal{X}}$ over the alphabet Σ_H. This allows us to choose $L_1{:=}L(\hat{\alpha}_{\mathcal{X}})$, while $L_1 \in \mathrm{REG}$ holds. Next, we define $\phi(0) = \{0\}$, $\phi(\#) = \{\#\}$, and $\phi(x){:=}\{0\}^*$. Hence, $L(H) = L(\alpha_{\mathcal{X}}) = \mathrm{INVALC}(\mathcal{X})$ follows immediately.

4.3 Undecidability and Its Consequences

As mentioned in Section 4.1, the central questions of the present chapter are whether we can minimize extended regular expressions (under any complexity measure as defined in Definition 4.7, or with respect to the number variables), and whether there is a recursive upper bound on the tradeoff between extended and proper regular expressions. We approach these questions by proving various degrees of undecidability for the decision problems given in Definition 4.6. Building on Theorem 4.14 and Lemma 4.10, we can almost immediately state this section's main theorem:

Theorem 4.15. *For* RegEx(1)*, universality is not decidable; and regularity, cofiniteness, and their complements are not semi-decidable.*

Proof. We prove each of the claims by reduction from one of three problems for extended Turing machines that are listed in Lemma 4.10 (these being emptiness, finiteness, and the complement of finiteness). Each reduction uses the same construction: Given an extended Turing machine $\mathcal{X}$, we construct an extended regular expression $\alpha_{\mathcal{X}} \in \mathrm{RegEx}(1)$ with $\mathrm{INVALC}(\mathcal{X}) = L(\alpha_{\mathcal{X}})$ (this is possible according to Theorem 4.14).

Then $\mathrm{dom}_{\mathrm{X}}(\mathcal{X}) = \emptyset$ if and only if $\mathrm{VALC}(\mathcal{X}) = \emptyset$, which holds if and only if $\mathrm{INVALC}(\mathcal{X}) = \{0, \#\}^*$, which holds if and only if $L(\alpha_{\mathcal{X}}) = \{0, \#\}^*$. Thus, any algorithm that decides universality for RegEx(1) could be used to decide the emptiness of the domain for extended Turing machines, which is not decidable according to Lemma 4.10.

Furthermore, $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ is finite if and only if $\mathrm{VALC}(\mathcal{X})$ is finite, which holds if and only if $\mathrm{VALC}(\mathcal{X})$ is regular (according to Lemma 4.13), which holds if and only if $\mathrm{INVALC}(\mathcal{X})$ is regular (as the class of regular languages is closed under complementation), which holds if and only if $L(\alpha_{\mathcal{X}})$ is regular. Hence, semi-decidability of regularity for RegEx(1) would lead to semi-decidability of finiteness of $\mathrm{dom}_{\mathrm{X}}$, a problem that is not semi-decidable (according to Lemma 4.10)

Likewise, as $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ is finite if and only if $L(\alpha_{\mathcal{X}})$ is regular, $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ is infinite if and only if $L(\alpha_{\mathcal{X}})$ is not regular. Therefore, semi-decidability of non-regularity for RegEx(1) contradicts the fact that the complement of finiteness of $\mathrm{dom}_{\mathrm{X}}$ is not semi-decidable (see Lemma 4.10).

As $\mathrm{INVALC}(\mathcal{X})$ is cofinite if and only if $\mathrm{INVALC}(\mathcal{X})$ is regular, the results for regularity and non-regularity also show that neither cofiniteness nor non-cofiniteness is semi-decidable for RegEx(1). □

Of course, all these undecidability results also hold for every RegEx(k) with $k \geq 2$, and for the whole class RegEx of extended regular expressions (as RegEx(1) is contained in all these classes)[3]. Theorem 4.15 also demonstrates that inclusion and equivalence are undecidable for RegEx(1) (and, hence, all of RegEx). We also see, as an immediate consequence to Theorem 4.15, that there is no algorithm that minimizes the number of variables in an extended regular expression, as such an algorithm could be used to decide regularity[4].

Note that in the proof of Theorem 4.15, the single variable x is bound only to words that match the expression 0^*. This shows that the "negative" properties of extended regular expressions we derive from Theorem 4.15 hold even if we restrict RegEx(1) by requiring that the variable can only be bound to a very restricted proper regular expression. Furthermore, the proof also applies to the extension of proper regular expressions through numerical parameters that is proposed in Della Penna et al. [23]. In addition to this, the construction from Theorem 4.14 (which we shall use to prove Theorem 4.15, and consequently, all other results in the present chapter) can be refined to also include bounds on the number of occurrences of the single variable – see Section 4.3.1. Furthermore, as explained at the end of Section 4.2.5, we can adapt these proofs to the class $\mathcal{H}(\mathrm{REG}, \mathrm{REG})$ of H-system languages, and conclude the same levels of undecidability of the respective decision problems for $\mathcal{H}(\mathrm{REG}, \mathrm{REG})$. As $\mathcal{H}(\mathrm{REG}, \mathrm{REG})$ is a subclass of the class of H-expression languages (cf. Bordihn et al. [9]), this also proves that equivalence for H-expression languages is undecidable, a problem that was explicitly left open in [9].

From the undecidability of universality, we can immediately conclude that RegEx(1) cannot be minimized effectively:

Corollary 4.16. *Let c be a complexity measure for* RegEx(1)*. Then there is no recursive function m_c that, given an expression $\alpha \in$ RegEx(1), returns an expression $m_c(\alpha) \in$ RegEx(1) with*

1. *$L(m_c(\alpha)) = L(\alpha)$, and*
2. *$c(\beta) \geq c(m_c(\alpha))$ for every $\beta \in$ RegEx(1) with $L(\beta) = L(\alpha)$.*

[3]Note that cofiniteness for extended regular expressions is a more general case of the question whether a pattern is avoidable over a fixed terminal alphabet, an important open problem in pattern avoidance (cf. Currie [20]). Example 4.2 illustrates this relation for the pattern xx over a binary alphabet. See Reidenbach [85] (Proposition 3.4) for a more detailed explanation.

[4]Those who are interested in the exact position of these problems in the arithmetical hierarchy (cf. Odifreddi [77]) can conclude that universality is Π_1^0-complete, while regularity and co-finiteness are Σ_2^0-complete. For each of these problems, hardness for the respective class follows from the respective completeness of each of the problems on extended Turing machines used in the proof of Theorem 4.15 (see the remark at the end of the proof of Lemma 4.10). Membership in the respective level of the hierarchy is easily proved using the appropriate representation for that class; e. g., universality of some $L(\alpha)$ can be expressed as $\forall w \in \Sigma^* : w \in L(\alpha)$. As the membership problem for extended regular expressions is decidable, this proves that universality is in Π_1^0. Likewise, non-regularity can be expressed as

$$\forall \beta \in \mathrm{RegEx}(0) : \exists w \in \Sigma^* : [w \in L(\alpha) \Leftrightarrow w \notin L(\beta)],$$

which shows that non-regularity is in Π_2^0. Actually, under a strict interpretation of the definition given by Odifreddi [77], we would need to quantify over natural numbers; but as there are computable bijections between $\mathbb{N}$ and Σ^*, as well as between $\mathbb{N}$ and RegEx(0), we can omit this technical detail.

Proof. Let c be a complexity measure for RegEx(1) and assume there is such a function m_c. Let Σ be any finite alphabet with $|\Sigma| \geq 2$, and let

$$U_c := \{m_c(\alpha) \mid L(\alpha) = \Sigma^*\}.$$

By definition of c, U_c is a finite set, and therefore recursive. As $L(\alpha) = \Sigma^*$ if and only if $m_c(\alpha) \in U_c$, m_c and the characteristic function of U_c can be used to decide universality for RegEx(1), a problem that is not decidable (cf. Theorem 4.15). This is a contradiction. □

Following the classic proof method of Hartmanis [41] (cf. Kutrib [61]), we can use the fact that non-regularity is not semi-decidable to obtain a result on the relative succinctness of extended and proper regular expressions:

Corollary 4.17. *There are non-recursive tradeoffs between* RegEx(1) *and* RegEx(0). *This holds even if we consider only the tradeoffs between* CoFRegEx(1) *and* CoFRegEx(0), *using a complexity measure for* RegEx(1).

Proof. The result for RegEx(1) and RegEx(0) follows immediately from Theorem 4.15 and Theorem 4 in Hartmanis [41][5]: As non-RegEx(0)-ity is not semi-decidable for RegEx(1), the tradeoff between RegEx(1) and RegEx(0) is non-recursive.

Thus, in order to prove the existence of non-recursive tradeoffs between RegEx(1) and RegEx(0), it would suffice to invoke Theorem 4 from Hartmanis [41]. This does not apply for the claimed non-recursive tradeoff between CoFRegEx(1) and CoFRegEx(0), as it is not possible to define a complexity measure for CoFRegEx(1), as our definition of complexity measures requires that the expressions can be enumerated effectively in order of increasing size. According to Theorem 4.15, non-cofiniteness for RegEx(1) is not semi-decidable, hence, this enumeration criterion cannot be satisfied.

Nonetheless, this problem can be solved using a slight adaption of Hartmanis' proof scheme. Therefore, we first consider the canonical proof for the existence of non-recursive tradeoffs between RegEx(1) and RegEx(0), and then state the necessary modifications.

Applied to RegEx(1) and RegEx(0), Hartmanis' proof scheme gives the following proof: As non-regularity is not semi-decidable for RegEx(1), the set

$$\Delta := \{\alpha \in \text{RegEx}(1) \mid L(\alpha) \text{ is not regular}\}$$

is not partially recursive. Now assume that, for a given complexity measure c, the tradeoff from RegEx(1) to RegEx(0) is recursively bounded by some recursive function f_c. We can then use f_c to construct a semi-decision procedure for Δ as follows: Given some $\alpha \in \text{RegEx}(1)$, we compute $n := f_c(c(\alpha))$, and let

$$F_n := \{\beta \in \text{RegEx}(0) \mid c(\beta) \leq n\}.$$

[5] As it would require considerable additional definitional effort, the author decided against including an exact formulation of Hartmanis' theorem in the present thesis (e. g., we defined complexity measures and tradeoffs only for classes of regular expressions, not for arbitrary description systems). Basically, Hartmanis' theorem states that, for two description systems D_1, D_2 and respective complexity measures c_1, c_2, non-recursive tradeoffs between D_2 and D_1 exist if non-D_1-ity is not semi-decidable for descriptions from D_2. See also Kutrib [61] for more detailed explanations than in [41].

As c is a complexity measure, F_n is finite, and we can effectively list all its elements (as we can effectively list all $\beta \in \mathrm{RegEx}(1)$ with $c(\beta) \leq n$, and we can decide whether $\beta \in \mathrm{RegEx}(0)$ by searching β for occurrences of variables or the metacharacter %). For each $\beta \in F_n$, we then semi-decide $L(\beta) \neq L(\alpha)$ by checking $w \in L(\alpha)$ and $w \in L(\beta)$ for all $w \in \Sigma^*$ successively.

If $L(\alpha)$ is not regular, then for every $\beta \in F_n$, we find some w_β in finite time that proves $L(\beta) \neq L(\alpha)$ (as w_β is not contained in one of the two languages, but in the other), and we can proceed to the next expression in F_n. If $L(\alpha)$ is regular, and f_c is a bound on the tradeoff from RegEx(1) to RegEx(0), there is a $\beta \in F_n$ with $L(\beta) = L(\alpha)$, and the procedure will never terminate. If no such β can be found, we know that $\alpha \in \Delta$, and the procedure can return 1. Thus, we can construct a semi-decision procedure for Δ, which contradicts the established fact that Δ is not partially recursive.

Likewise, we can still prove the existence of non-recursive tradeoffs if we restrict the claim to expressions from CoFRegEx(1) and CoFRegEx(0). According to Theorem 4.15, non-cofiniteness for RegEx(1) is not semi-decidable. Thus, given a complexity measure c for RegEx(1), a bound f_c on the tradeoff from CoFRegEx(1) to CoFRegEx(0) could be used to give a semi-decision algorithm for the set

$$\Delta_C := \{\alpha \in \mathrm{RegEx}(1) \mid L(\alpha) \text{ is not cofinite}\}.$$

First, note that (as mentioned above) the use of a complexity measure for RegEx(1) instead of CoFRegEx(1) is intentional and serves to avoid complications with the fact that cofiniteness is not decidable for RegEx(1) (recall Theorem 4.15). We can construct a semi-decision procedure for Δ_C from f_c using almost the same reasoning as above: Given an $\alpha \in \mathrm{RegEx}(1)$, we define $n := f(c(\alpha))$, and let

$$F_{C,n} := \{\beta \in \mathrm{RegEx}(0) \mid c(\beta) \leq n, L(\beta) \text{ is cofinite}\}.$$

Enumerating all elements of $F_{C,n}$ is slightly more difficult than enumerating all elements of F_n (see above), as we also need to decide whether $L(\beta)$ is cofinite for every $\beta \in F_n$. Luckily, this is not a problem, as cofiniteness is decidable for RegEx(0) (e. g., given a $\beta \in \mathrm{RegEx}(0)$, one could convert β into an equivalent DFA, compute its complement, and check the resulting DFA for loops that contain a final state and are reachable from the initial state). From here on, the proof continues as above, *mutatis mutandis*. □

Thus, no matter which complexity measure and which computable upper bound we assume for the tradeoff, there is always a regular language L that can be described by an *extended* regular expression from RegEx(1) so much more succinctly that every *proper* regular expression for L has to break that bound. Obviously, this has also implications for the complexity of matching regular expressions: Although membership is "easier" for proper regular expressions than for extended regular expressions, there are regular languages that can be expressed far more efficiently through extended regular expressions than through proper regular expressions.

Recall Example 4.1, where we gave extended regular expressions that describe finite languages. In this restricted case, there exists an effective conversion procedure – hence, the tradeoffs are recursive:

Lemma 4.18. *For every $k \geq 1$, the tradeoff between* FRegEx(k) *and* FRegEx(0) *is recursive (even when considering complexity measures for* RegEx(k) *instead of* FRegEx(k)*).*

Proof. Let $k \geq 1$, and let c be a complexity measure for RegEx(k). By definition, no $\alpha \in \text{FRegEx}(k)$ contains a Kleene star (or Kleene plus). Thus, given an $\alpha \in \text{FRegEx}(k)$, we can effectively compute the finitely many words in $L(\alpha)$ by exhausting all possible combinations of choices for each alternation symbol in α, and computing the corresponding word for each combination, handling all bindings accordingly.

For example, the expression $\alpha{:=}(\texttt{a} \mid \texttt{b})(\texttt{a} \mid \texttt{b})\%x(\texttt{a} \mid \texttt{b})\%y\ x\ y$ contains three alternation symbols, totaling $2^3 = 8$ possible combinations of choices, each corresponding to one of the words in $L(\alpha)$. More generally, if $\alpha \in \text{FRegEx}(k)$ contains n alternation symbols, $L(\alpha)$ contains at most 2^n words $w_1, \ldots, w_i$ ($i \leq 2^n$), and we can compute an $\hat{\alpha} \in \text{FRegEx}(0)$ with $L(\hat{\alpha}) = L(\alpha)$ simply by computing these words $w_1, \ldots, w_i$ and defining $\hat{\alpha}{:=}w_1 \mid \ldots \mid w_i$.

Fixing an effective procedure that computes $\hat{\alpha}$, we can directly define the recursive bound $f_c : \mathbb{N} \to \mathbb{N}$ as follows: For $n \geq 0$, we define

$$F_n{:=}\{\hat{\alpha} \in \text{FRegEx}(0) \mid \alpha \in \text{FRegEx}(k), c(\alpha) = n\}.$$

By definition of complexity measures, every F_n is finite, and given any n, we can effectively list all patterns from F_n (again, as c is a complexity measure, and membership in FRegEx(k) is decidable in a straightforward manner for RegEx(k)). For any $n \geq 0$, we define

$$f_c(n){:=}\max\{c(\hat{\alpha}) \mid \hat{\alpha} \in F_n\}.$$

As every F_n can be listed effectively, every F_n is finite, and as $c(\hat{\alpha})$ can be computed effectively, f_n is a recursive function. Furthermore, f_c is a bound on the tradeoff from FRegEx(1) to FRegEx(0) by definition. □

Although the class of RegEx-languages is not closed under complementation (Lemma 2 in Câmpeanu et al. [13]), there are languages L such that both L and its complement $\Sigma^* \setminus L$ are RegEx-languages (e. g., all regular languages). Combining Lemma 4.18 and Corollary 4.17, we can immediately conclude that there are cases where it is far more efficient to describe the complement of a RegEx(1)-language, as opposed to the language itself:

Corollary 4.19. *Let Σ be a finite alphabet. Let c be a complexity measure for* RegEx(1). *For any recursive function $f_c : \mathbb{N} \to \mathbb{N}$, there exists an $\alpha \in$ RegEx(1) such that $\Sigma^* \setminus L(\alpha)$ is a* RegEx(1)*-language, and for every $\beta \in$ RegEx(1) with $L(\beta) = \Sigma^* \setminus L(\alpha)$, $c(\beta) \geq f_c(c(\alpha))$.*

Proof. Assume to the contrary that, for some complexity measure c for RegEx(1), there is a recursive function $f_1 : \mathbb{N} \to \mathbb{N}$ such that for every $\alpha \in \text{RegEx}(1)$, if $\Sigma^* \setminus L(\alpha)$ is a RegEx(1)-language, there is a $\beta \in \text{RegEx}(1)$ with $L(\alpha) = L(\beta)$ and $c(\beta) \leq f_1(c(\alpha))$ (in other words, f_1 is a recursive bound on the blowup of complementation for RegEx(1)).

We can now use f_1 and Lemma 4.18 to obtain a recursive bound on the tradeoff between CoFRegEx(1) and CoFRegEx(0), which contradicts Corollary 4.17.

First, note that according to Lemma 4.18, there is a recursive bound $f_2 : \mathbb{N} \to \mathbb{N}$ on the tradeoff between FRegEx(1) and FRegEx(0).

Furthermore, we can easily prove that there is a recursive bound $f_3 : \mathbb{N} \to \mathbb{N}$ on the blowup that occurs in the complementation for FRegEx(1), as f_3 might be computed as follows: For every input n, there are finitely many $\alpha \in \text{RegEx}(0)$ with $c(\alpha) = n$.

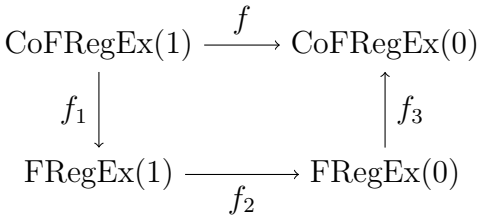

Figure 4.4: An illustration of the functions that are used in the proof of Corollary 4.19.

Finiteness for RegEx(0) is obviously decidable; thus, we can effectively construct the finite set

$$F_n := \{\alpha \in \mathrm{FRegEx}(0) \mid c(\alpha) = n\}.$$

For every $\alpha \in F_n$, we (effectively) construct an $\overline{\alpha} \in \mathrm{RegEx}(0)$ with $L(\overline{\alpha}) = \Sigma^* \setminus L(\alpha)$, for example, by converting α into a DFA, complementing it, and converting the resulting DFA into a proper regular expression, all using the standard techniques as described in Hopcroft and Ullman [46]. We then check all $\beta \in \mathrm{RegEx}(0)$, ordered by growing size of $c(\beta)$, until we find the smallest β (w.r.t. c) with $L(\beta) = \Sigma^* \setminus L(\alpha)$, and refer to this β as $\tilde{\alpha}$ (again, this is possible due to the decidability of equivalence for RegEx(0), cf. [46]), and define

$$C_n := \{\tilde{\alpha}_n \mid \alpha_n \in F_n\}.$$

Finally, we define $f_3(n)$ to be the maximum of all $c(\tilde{\beta})$ with $\beta \in C_n$. By definition, f_3 is recursive, and serves as an upper bound for the blowup that occurs when complementing expressions from FRegEx(0).

Now, consider the function $f : \mathbb{N} \to \mathbb{N}$ that is defined by $f(n) := f_3(f_2(f_1(n)))$ for every $n \in \mathbb{N}$. We shall see that our assumption implies that f is a recursive bound on the tradeoff between CoFRegEx(1) and CoFRegEx(0), which contradicts Corollary 4.17. An illustration of this argument can be found in Figure 4.4.

First, observe that f is a recursive function, as f_1, f_2, and f_3 are recursive functions. Due to Corollary 4.17, there is an $\alpha_{fc} \in \mathrm{CoFRegEx}(1)$ such that $L(\alpha_{fc})$ is regular, but for every $\beta \in \mathrm{RegEx}(0)$ (and hence, $\beta \in \mathrm{CoFRegEx}(0)$) with $L(\alpha_{fc}) = L(\beta)$, $c(\beta) > f(c(\alpha_{fc}))$.

By our assumption, there is an $\overline{\alpha}_{fc} \in \mathrm{FRegEx}(1)$ with $L(\overline{\alpha}_{fc}) = \Sigma^* \setminus L(\alpha_{fc})$ and $c(\overline{\alpha}_{fc}) \leq f_1(c(\alpha_{fc}))$.

According to Lemma 4.18, there is a $\overline{\beta}_{fc} \in \mathrm{FRegEx}(0)$ with $L(\overline{\beta}_{fc}) = L(\overline{\alpha}_{fc}) = \Sigma^* \setminus L(\alpha_{fc})$, and $c(\overline{\beta}_{fc}) \leq f_2(c(\overline{\alpha}_{fc}))$.

Finally, as explained above, there is a $\beta_{fc} \in \mathrm{CoFRegEx}(0)$ with $L(\beta_{fc}) = \Sigma^* \setminus (\overline{\beta}_{fc}) = L(\alpha_{fc})$ and

$$\begin{aligned} c(\beta_{fc}) &\leq f_3(c(\overline{\beta}_{fc})) \\ &\leq f_3(f_2(c(\overline{\alpha}_{fc}))) \\ &\leq f_3(f_2(f_1(c(\alpha_{fc})))) \\ &= f(c(\alpha_{fc})). \end{aligned}$$

This contradicts our choice of α_{fc} and concludes the proof. □

With some additional technical effort, we can extend the previous results on undecidability of RegEx(l)-ity and on tradeoffs between RegEx(k) and RegEx(0) to arbitrary levels of the hierarchy of RegEx(k)-languages:

Lemma 4.20. *Let $k \geq 1$. For* RegEx($k+1$)*, both* RegEx(k)*-ity and its complement are not semi-decidable.*

Proof. We adapt the construction from the proof of Theorem 4.14 to the larger alphabet $\Sigma_k{:=}\{0, \#, \$, \mathtt{a}_1, \mathtt{b}_1, \ldots, \mathtt{a}_k, \mathtt{b}_k\}$, where 0, #, \$, all $\mathtt{a}_i$, and all $\mathtt{b}_i$ are pairwise distinct letters. For every i with $1 \leq i \leq k$, let $\Sigma_i{:=}\{\mathtt{a}_i, \mathtt{b}_i\}$. Given an extended Turing machine $\mathcal{X}$, we construct the extended regular expression $\alpha_\mathcal{X} \in$ RegEx(1) as in the proof of Theorem 4.14. For every i with $1 \leq i \leq k$, we define an expression $\alpha_i \in$ RegEx(1) by

$$\alpha_i{:=}((\mathtt{a}_i \mid \mathtt{b}_i)^*)\%x_i x_i,$$

where every x_i is distinct from the variable x in $\alpha_\mathcal{X}$, and from all x_j with $j \neq i$. Finally, we define $\alpha_{\mathcal{X},k} \in$ RegEx($k+1$) by

$$\begin{aligned}\alpha_{\mathcal{X},k}{:=}&\alpha_\mathcal{X}\$\alpha_1\$\ldots\$\alpha_k\\ &= \alpha_\mathcal{X}\$((\mathtt{a}_1 \mid \mathtt{b}_1)^+)\%x_1x_1\$\ldots\$((\mathtt{a}_k \mid \mathtt{b}_k)^+)\%x_kx_k.\end{aligned}$$

Thus, $\alpha_{k,\mathcal{X}} \in$ RegEx($k+1$). It suffices to show that $L(\alpha_{k,\mathcal{X}})$ is a RegEx(k)-language if and only if $L(\alpha_\mathcal{X})$ is regular, as neither regularity nor non-regularity are semi-decidable for RegEx(1), cf. Theorem 4.15.

The *if* direction is obvious: If $L(\alpha_\mathcal{X})$ is regular, we can (non-effectively) replace that part of $\alpha_{\mathcal{X},k}$ with an appropriate proper regular expression, and obtain an expression from RegEx(k) for $L(\alpha_{\mathcal{X},k})$.

For the *only if* direction, first note that, for every i, $L(\alpha_i) = \{ww \mid w \in \{a_i, b_i\}^+\}$, a language that is well known to be not regular (as can be easily verified with the Pumping Lemma, cf. Hopcroft and Ullman [46]). Likewise, note that $L_k{:=}L(\alpha_1\$\ldots\$\alpha_k)$ is not a RegEx($k-1$) language, as can be seen by the following line of reasoning: Assume there is an $\alpha \in$ RegEx($k-1$) with $L(\alpha) = L_k$. By definition, α contains at most $k-1$ different variables. Note that, whenever α is matched to a $w \in L_k$, every variable x in α that is bound and also referenced when matching w contains only terminals from a single set $\Sigma_i \cup \{\$\}$, as $L_k \subset \{\mathtt{a}_1, \mathtt{b}_1\}^+\$\ldots\$\{\mathtt{a}_k, \mathtt{b}_k\}^+$, and repeating any string that contains some $\mathtt{a}_j$ or $\mathtt{b}_j$ with $j \neq i$ would break this structure. As every $L(\alpha_i)$ needs to use at least one variable, and no variable that is matched to a letter other than \$ can cross the boundaries between the different $L(\alpha_i)$, there can be no $\alpha \in$ RegEx($k-1$) with $L(\alpha) = L_k$.

If $L(\alpha_\mathcal{X})$ is not regular, this reasoning extends to $L(\alpha_{k,\mathcal{X}})$ and RegEx(k), as we need at least one variable to generate $L(\alpha_\mathcal{X})$, and k variables for L_k, for a total of $k+1$ variables.

Thus, $L(\alpha_{\mathcal{X},k})$ is a RegEx(k)-language if and only if $\alpha_\mathcal{X}$ is regular. As seen in the proof of Theorem 4.15, neither RegEx(k)-ity nor its complement is semi-decidable for RegEx(k). □

Non-recursive tradeoffs between RegEx($k+1$) and RegEx(k) for every $k \geq 1$ follow immediately, using Hartmanis' proof technique as in the proof of Corollary 4.17.

4.3.1 A Technical Note on Bounded Occurrences of Variables

Although the extended regular expressions $\alpha_{\mathcal{X}}$ that follow from the construction in the proof of Theorem 4.14 use only one variable x, there is no bound on the number of occurrences of x in $\alpha_{\mathcal{X}}$. In fact, the number of occurrences grows with the number of fields in the transition table δ of $\mathcal{X}$, and limiting that number would not allow to simulate infinitely many extended Turing machines, which is required to obtain the results in the present chapter.

Nonetheless, similar to the proof of Theorem 3.10, these limitations can be overcome by using a single universal Turing machine: First, let $\psi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a universal partially recursive function, i. e, for every partially recursive function $\phi : \mathbb{N} \to \mathbb{N}$, there is a $m \geq 0$ such that $\psi(m, n) = \phi(n)$ for every $n \geq 0$. It is an elementary fact of recursion theory that such a function exists (cf. Cutland [21], Odifreddi [77]), and moreover, there is a Turing machine $\mathcal{U}$ over some tape alphabet $\hat{\Gamma} \subseteq \{0, ¢, \$, \mathtt{a}, \mathtt{b}\}$ such that

$$\mathrm{dom}_{\mathrm{T}}(\mathcal{U}) = \{\mathtt{a}^m\, 0\, \mathtt{b}^n \mid \psi(m, n) \text{ is defined}\}.$$

Using the same construction as in the proof of Lemma 4.10, we can build an extended Turing machine $\hat{\mathcal{U}} = (Q, q_1, \delta)$ that simulates $\mathcal{U}$, using an appropriate injective function $b_k : \hat{\Gamma} \to \Gamma^k$ with $b_k = 0^k$, for some appropriate $k \geq 3$. Instead of constructing a single extended regular expression $\alpha_{\hat{\mathcal{U}}}$, we construct an expression $\alpha_{\hat{\mathcal{U}},m}$ for every function number m, using a slight modification of the proof of Theorem 4.14. Instead of allowing arbitrary contents for the right tape word in the initial configuration (as can be seen in the second line of the definition of the language $S_{\mathcal{X}}$), we define

$$\begin{aligned} S_{\hat{\mathcal{U}},m} := {} & \left(\#\#0^+\#0^+\#0\{\lambda, 0\}\#\left\{0^i \mid 1 \leq i \leq \nu\right\}\right)^+\#\# \\ & \cap\, \#\#0\#0^{\mathrm{e}(b_k(¢\,\mathtt{a}^m\,0))}\left(0^{2^{(2+m)k}}\right)^+\#0\#0\#\#\{0, \#\}^* \\ & \cap\, \{0, \#\}^*\#\left\{00^a\#0^i\#\# \mid a \in \hat{\Gamma}, \delta(a, q_i) = \mathrm{HALT}\right\}. \end{aligned}$$

The only difference to the definition of $S_{\mathcal{X}}$ is the second line. Evidently, the new definition allows exactly those initial right tape sides t_R for which $\mathrm{e}(t_R) = \mathrm{e}(b_k(¢\,\mathtt{a}^m\,0)) + i(2^{(2+m)k})$ for some $i \geq 0$. As $|b_k(¢\,\mathtt{a}^m\,0)| = (2+m)k$, these are exactly that t_R have $b_k(¢\,\mathtt{a}\,0^m)$ as a prefix.

The construction of $\alpha_{\hat{\mathcal{U}},m}$ then uses $S_{\hat{\mathcal{U}},m}$ to construct α_{struc}, and proceeds as in the original proof. Thus, $\{0, \#\}^* \setminus L(\alpha_{\hat{\mathcal{U}},m})$ is exactly that set that corresponds to the valid computations of $\mathcal{U}$ on some input that starts with $\mathtt{a}^m\,0$. Furthermore, the number of occurrences of x in every $\alpha_{\hat{\mathcal{U}},m}$ depends only on δ, not on m, which means that we can bound that number.

For any partially recursive function $\phi : \mathbb{N} \to \mathbb{N}$, let $\mathrm{dom}(\phi) := \{n \geq 0 \mid \phi(n) \text{ is defined}\}$, and, for every $m \geq 0$, let ψ_m denote the function that is defined by $\psi_m(n) := \psi(m, n)$ for every $n \geq 0$. Then $L(\alpha_{\hat{\mathcal{U}},m})$ is

- regular if and only if $\mathrm{dom}(\psi_m)$ is finite,
- cofinite if and only if $\mathrm{dom}(\psi_m)$ is finite,
- universal if and only if $\mathrm{dom}(\psi_m)$ is empty,

following the same reasoning as for $L(\alpha_{\mathcal{X}})$. As finiteness of dom is not semi-decidable, and emptiness not decidable (cf. Odifreddi [78] or the analogous results for Turing machines in Lemma 4.10), we arrive at the same conclusions as in Theorem 4.15 and Corollaries 4.17 and 4.19 for RegEx(1) with a limited number of occurrences of the single variable.

Chapter 5

Existence of Descriptive Patterns

5.1 On Patterns Descriptive of a Set of Strings

The study of pattern languages was initiated by Angluin [4] in the context of Gold's intensively studied learning paradigm of language identification in the limit from positive data (Gold [39], cf. Section 6.2.1). In this model, it is a requirement for the computational learner to infer, for any positive presentation of any language in some class, an *exact* description of that language.

While this maximum accuracy of the output of the inference procedure is clearly a natural goal, it has a number of downsides, the most obvious one being the fact that it can lead to significant limitations to the learning power of the model. From a more applied point of view, there is another important reason why one might wish to relax it and settle for receiving an approximation of the language from the learner: depending on the class of languages to be inferred, the corresponding grammars or acceptors might have undesirable properties, i. e., they might have computationally hard decision problems or be incomprehensible to a (human) user. Thus, in various settings it might be perfectly acceptable for an inference procedure to output a compact and reasonably precise approximation of the language instead of producing a precise yet arbitrarily complex grammar.

The problem of finding a *consistent* pattern α for an arbitrary set S of strings (i. e., a pattern α with $L(\alpha) \supseteq S$) is often referred to as *(string) pattern discovery*, and many of its applications are derived from tasks in bioinformatics (cf. Brazma et al. [10]). In contrast to the inductive inference approach to pattern languages, where a pattern shall be inferred that exactly describes the given language, string pattern discovery faces the problem that S can typically have many consistent patterns showing very different characteristics. For instance, both

$$\alpha_1 := xyxyx \text{ and}$$
$$\alpha_2 := x\,\mathtt{a\,b}\,y$$

are consistent with the language

$$S_0 := \{\mathtt{a\,b\,a\,b\,a},\ \mathtt{a\,b\,a\,b\,b\,a\,b\,a\,b\,b\,a\,b},\ \mathtt{b\,a\,b\,a\,b}\},$$

and the pattern $\alpha_0{:=}x$ is consistent with every set of strings, anyway. Hence, the algorithms of string pattern discovery require an underlying notion of the quality of a pattern in order to determine what patterns to strive for. With regard to the above example set and patterns, it seems quite likely that one might not be interested in a procedure outputting α_0 when reading S_0. Concerning α_1 and α_2, however, it is, a priori, by no means evident which of them to prefer. Thus, the definition of the quality of a pattern might often depend on the field of application where string pattern discovery is conducted. In addition to this, it is a worthwhile goal to develop *generic* notions of quality of consistent patterns that can inform the design of pattern discovery algorithms (we expand on the topic of finding patterns common to sets of strings in Chapter 6).

In this regard, the *descriptiveness* of patterns is a well-known and plausible concept, that is also used within the scope of inductive inference (cf. Ng and Shinohara [76]). A pattern δ is said to be descriptive of a given set S of strings if there is no pattern α satisfying $L(\delta) \supset L(\alpha) \supseteq S$. Intuitively, this means that if δ is descriptive of S, then no consistent pattern for S provides a strictly closer match than δ. Thus, although δ does not need to be unique (as to be further discussed below), it is guaranteed that it is one of the most accurate approximations of S that can be provided by patterns. While descriptiveness is unquestionably an appropriate notion of quality of consistent patterns, the fact that inclusion is undecidable in general (the main topic of Chapter 3, as the reader is probably aware) and still combinatorially involved for certain classes where it is decidable (cf. Theorem 2.4) leads to major technical challenges. This aspect is crucial to the subsequent formal parts of this chapter.

Since the definition of a descriptive pattern is based on the concept of pattern languages, the question of whether NE- or E-pattern languages are chosen can have a significant impact on the descriptiveness of a pattern. This is reflected by the terminology we use: we call a pattern δ an NE-descriptive pattern if it is descriptive in terms of its NE-pattern language and the NE-pattern languages of all patterns in $(\Sigma \cup X)^+$; accordingly, we call δ E-descriptive if its descriptiveness is based on interpreting all patterns as generators of E-pattern languages. In order to illustrate these terms, we now briefly discuss the descriptiveness of the example patterns introduced above (though the full verification of our corresponding claims is not always straightforward and might require certain tools to be introduced later). If we deal with S_0 and the patterns in the context of NE-pattern languages, then it can be stated that both α_1 and α_2 are NE-descriptive of S_0, since no NE-pattern languages can comprise S_0 and, at the same time, be a proper sublanguage of the NE-pattern languages of α_1 or α_2. If we study S_0 in terms of E-pattern languages, it turns out that α_1 is also E-descriptive of S_0, i. e. there is no pattern generating an E-pattern language that is consistent with S_0 and strictly included in the E-pattern language of α_1. However, the second NE-descriptive example pattern α_2 is not E-descriptive of S_0, since the E-pattern language generated by

$$\alpha_3{:=}x\,\mathtt{a\,b\,a\,b}\,y$$

is a proper sublanguage of the E-pattern language of α_2 and comprises S_0. The pattern α_3, in turn, is even E-descriptive of S_0, but not NE-descriptive, since it is not consistent with S_0 if we disallow empty substitutions. Exactly the same holds for $\alpha_4{:=}x\,\mathtt{b\,a\,b\,a}\,y$, which also is consistent with S_0 if we allow the empty substitution of variables, generates an E-pattern language that is strictly included in the E-pattern language of α_2 and is

E-descriptive, but not NE-descriptive of S_0.

The present chapter examines the basic underlying problem of descriptive pattern discovery, namely the *existence* of such patterns; this means that we study the question of whether or not, for a given language S, there is a pattern that is descriptive of S (we examine aspects of the actual problem of finding descriptive patterns in Chapter 6). Regarding the existence of a descriptive pattern, four different cases can be considered: NE-descriptive patterns of finite languages, NE-descriptive patterns of infinite languages, E-descriptive patterns of finite languages and E-descriptive patterns of infinite languages. The problem of the existence of the former three types of descriptive patterns is either trivial or has already been solved in previous publications. We therefore largely study the latter case, and the corresponding main result answers a question posed by Jiang, Kinber, Salomaa, Salomaa and Yu [50]. Our technical considerations do not only provide insights into the actual topic of this chapter, but – due to the definition of descriptive patterns – also reveal vital phenomena related to the inclusion of E-pattern languages and, hence, the topology of the class of terminal-free E-pattern languages; both inside the class, and in relation to all languages. Due to the way the inclusion of terminal-free E-pattern languages is characterized (cf. Theorem 2.4), this implies that we have to deal with combinatorial properties of morphisms in free monoids. Furthermore, crucial parts of our reasoning are based on infinite *unions* of pattern languages, which means that the work in this chapter shows additional connections to so-called multi-pattern languages (cf. Dumitrescu et al. [25]). While [25] features unions of pattern languages where the generating patterns form a context-free language, the material presented in this chapter is essentially based on multi-pattern languages where the underlying set of patterns – apart from an infinite variable alphabet we have to use – is defined similarly to an HD0L language over an infinite alphabet (see Section 5.2.2). We extend this connection in Chapter 7.

Finally, in Section 5.5, we study the question whether descriptive patterns can be derived effectively, or even efficiently. There, we answer another question posed by Jiang, Kinber, Salomaa, Salomaa and Yu [50] and show how a related question is connected to the open and infamous question on the decidability of the equivalence problem for E-pattern languages.

5.2 Preliminaries

We now can introduce our terminology on the main topic of the second part of this thesis, namely the descriptiveness of a pattern. For any alphabet Σ and any language $S \subseteq \Sigma^*$, a pattern $\delta \in \mathrm{Pat}_\Sigma$ is said to be *NE-descriptive (of S)* provided that $L_{\mathrm{NE},\Sigma}(\delta) \supseteq S$ and, for every $\alpha \in \mathrm{Pat}_\Sigma$ with $L_{\mathrm{NE},\Sigma}(\alpha) \supseteq S$, $L_{\mathrm{NE},\Sigma}(\alpha) \not\subset L_{\mathrm{NE},\Sigma}(\delta)$. Analogously, δ is called *E-descriptive (of S)* if $L_{\mathrm{E},\Sigma}(\delta) \supseteq S$ and, for every $\alpha \in \mathrm{Pat}_\Sigma$ with $L_{\mathrm{E},\Sigma}(\alpha) \supseteq S$, $L_{\mathrm{E},\Sigma}(\alpha) \not\subset L_{\mathrm{E},\Sigma}(\delta)$.

More generally, let $\mathrm{PAT}_{\star,\Sigma}$ be a class of NE-pattern languages or a class of E-pattern languages over Σ, and let $\mathrm{Pat}_{\star,\Sigma}$ be the corresponding class of generating patterns. We say that a pattern $\delta \in (\Sigma \cup X)^+$ is $\mathrm{PAT}_{\star,\Sigma}$-*descriptive* of a language $L \subseteq \Sigma^*$ if and only if $L(\delta) \in \mathrm{PAT}_{\star,\Sigma}$, $L(\delta) \supseteq L$, and there is no pattern α with $L(\alpha) \in \mathrm{PAT}_{\star,\Sigma}$ satisfying $L \subseteq L(\alpha) \subset L(\delta)$. Furthermore, $D_{\mathrm{PAT}_{\star,\Sigma}}(L)$ denotes the set of all patterns that are $\mathrm{PAT}_{\star,\Sigma}$-descriptive of L.

5.2.1 Properties of Terminal-Free E-Pattern Languages

Obviously, the definition of a descriptive pattern is based on the inclusion of pattern languages, which is an undecidable problem for both the full class of NE-pattern languages and the full class of E-pattern languages (cf. Chapter 3). A significant part of our subsequent technical considerations, however, can be restricted to terminal-free E-pattern languages, and here the inclusion problem is known to be characterized by the existence of a morphism between the patterns and, hence, decidable. This directly results from the characterization given in Theorem 2.4. While Theorem 2.4 is a powerful tool when dealing with the inclusion of terminal-free E-pattern languages, the examination of the descriptiveness of a pattern requires insights into *proper* inclusion relations, and therefore we use some further combinatorial results on morphisms in free monoids to give a more convenient criterion that can replace the use of Theorem 2.4.

In accordance with Reidenbach and Schneider [90], we designate a terminal-free pattern $\alpha \in X^+$ as *morphically imprimitive* if there is a pattern $\beta \in X^*$ satisfying the following conditions: $|\beta| < |\alpha|$ and there are morphisms $g, h : X^* \to X^*$ such that $g(\alpha) = \beta$ and $h(\beta) = \alpha$. Otherwise, α is *morphically primitive*. Let $\alpha \in X^+$ be morphically primitive. For any $\alpha \in X^+$, a *morphic root* of α is any morphically primitive β for which there is a morphism $\phi : X^* \to X^*$ with $\phi(\alpha) = \beta$.

A morphism $h : X^* \to X^*$ is said to be an *imprimitivity morphism (for α)* provided that $|h(\alpha)| > |\alpha|$ and there is a morphism $g : X^* \to X^*$ satisfying $(g \circ h)(\alpha) = \alpha$. Referring to these concepts, we now can give a characterization of certain proper inclusion relations between terminal-free E-pattern languages:

Lemma 5.1. *Let Σ be an alphabet, $|\Sigma| \geq 2$, $\alpha \in X^+$ a morphically primitive pattern and $h : X^* \to X^*$ a morphism. Then $L_{\mathrm{E},\Sigma}(h(\alpha)) \subset L_{\mathrm{E},\Sigma}(\alpha)$ if and only if h is neither an imprimitivity morphism for α nor a renaming of α.*

Proof. We first consider the *if* direction: If h is neither an imprimitivity morphism for α nor a renaming of α, then $|h(\alpha)| \leq |\alpha|$ or there is no morphism g mapping $h(\alpha)$ to α. In the latter case, due to Theorem 2.4, $L_{\mathrm{E},\Sigma}(h(\alpha)) \not\supseteq L_{\mathrm{E},\Sigma}(\alpha)$. In the former case, if there is a morphism g mapping $h(\alpha)$ to α, then α is not morphically primitive, which contradicts the condition of the lemma. Hence, there is no such morphism, and this again implies $L_{\mathrm{E},\Sigma}(h(\alpha)) \not\supseteq L_{\mathrm{E},\Sigma}(\alpha)$. Since Theorem 2.4 shows that $L_{\mathrm{E},\Sigma}(h(\alpha)) \subseteq L_{\mathrm{E},\Sigma}(\alpha)$, we have $L_{\mathrm{E},\Sigma}(h(\alpha)) \subset L_{\mathrm{E},\Sigma}(\alpha)$.

We proceed with the *only if* direction: If $L_{\mathrm{E},\Sigma}(h(\alpha)) \subset L_{\mathrm{E},\Sigma}(\alpha)$, then there is no morphism mapping $h(\alpha)$ to α. However, the definition of an imprimitivity morphism mapping α to some pattern β implies the existence of a morphism mapping β to α again. The same trivially holds for any renaming of α. Thus, h is neither an imprimitivity morphism for α nor a renaming of α. □

The question of whether a given morphism is an imprimitivity morphism for a pattern can be easily answered using the following insight:

Theorem 5.2 (Reidenbach and Schneider [90])**.** *Let $\alpha \in X^+$ be a morphically primitive pattern. Then a morphism $h : X^* \to X^*$ is an imprimitivity morphism for α if and only if*

1. *for every $x \in \mathrm{var}(\alpha)$, there exists an $x_h \in \mathrm{var}(h(x))$ such that $|h(x)|_{x_h} = 1$ and $|h(y)|_{x_h} = 0$ for every $y \in \mathrm{var}(\alpha) \setminus \{x\}$, and*

2. *there exists an* $x \in \mathrm{var}(\alpha)$ *with* $|h(x)| \geq 2$.

Evidently, Lemma 5.1 can only be applied if there is a tool for checking whether a terminal-free pattern is morphically primitive. This is provided by the following characterization:

Theorem 5.3 (Reidenbach and Schneider [90])**.** *A pattern* $\alpha \in X^+$ *is morphically primitive if and only if there is no factorization*

$$\alpha = \beta_0\, \gamma_1\, \beta_1\, \gamma_2\, \beta_2\, \ldots\, \beta_{n-1}\, \gamma_n\, \beta_n$$

with $n \geq 1$, $\beta_k \in X^*$ *and* $\gamma_k \in X^+$, $k \leq n$, *such that*

1. $|\gamma_k| \geq 2$ *for every* k, $1 \leq k \leq n$,
2. $\mathrm{var}(\beta_0 \ldots \beta_n) \cap \mathrm{var}(\gamma_1 \ldots \gamma_n) = \emptyset$,
3. *for every* k, $1 \leq k \leq n$, *there exists an* $x_k \in \mathrm{var}(\gamma_k)$ *such that* $|\gamma_k|_{x_k} = 1$ *and, for every* k', $1 \leq k' \leq n$, *if* $x_k \in \mathrm{var}(\gamma_{k'})$ *then* $\gamma_k = \gamma_{k'}$.

Reidenbach and Schneider call such a factorization for a pattern α an *imprimitivity factorization (of* α*)*. Thus, with Lemma 5.1, Theorem 5.2 and Theorem 5.3 we now have an appropriate tool for deciding on particular proper inclusion relations between terminal-free E-pattern languages.

Note that there are further characterizations of morphically primitive (or imprimitive) patterns. For example, as shown by Head [42], a pattern α has an imprimitivity factorization if and only if it is fixed point of a nontrivial morphism (i. e., a morphism that is not the identity morphism). Another characterization via the existence of so-called unambiguous morphisms is due to Freydenberger et al. [35].

Furthermore, we say that a pattern $\alpha \in X^+$ is *succinct* if it is the shortest generator of its E-pattern language $L_{\mathrm{E},\Sigma}(\alpha)$ (in other words, $|\beta| \geq |\alpha|$ for every $\beta \in X^+$ with $L_{\mathrm{E},\Sigma}(\beta) = L_{\mathrm{E},\Sigma}(\alpha)$). Then the following holds:

Theorem 5.4 (Reidenbach [88])**.** *A pattern* $\alpha \in X^+$ *is succinct if and only if it is morphically primitive.*

An immediate consequence of Theorem 2.4 is not only that inclusion for terminal-free E-pattern languages is decidable, but also that it cannot be decided effectively (if P $\neq$ NP):

Corollary 5.5. *Let* Σ *be an alphabet with* $|\Sigma| \geq 2$. *Then the inclusion problem for* $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$ *is* NP*-complete.*

Proof. As shown by Ehrenfeucht and Rozenberg (cf. Theorem 2.2), it is NP-complete to decide, given patterns $\alpha, \beta \in X^+$, whether there is a morphism mapping β to α. □

Note that Corollary 5.5 crucially depends on the fact that, when deciding $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$, $|\,\mathrm{var}(\beta)|$ is unbounded.

Although membership and inclusion problem for terminal-free E-pattern languages are NP-complete (cf. Theorem 2.2 and Corollary 5.5, respectively), the situation is different for the equivalence problem.

In conjunction with Theorem 5.4, a recent result by Holub [44] demonstrates that the equivalence problem can be decided in polynomial time:

Theorem 5.6. *There is a polynomial-time algorithm deciding, for any pair of terminal-free patterns α, β and for any alphabet Σ with $|\Sigma| \geq 2$, on whether $L_{\mathrm{E},\Sigma}(\alpha) = L_{\mathrm{E},\Sigma}(\beta)$.*

Proof. As mentioned by Reidenbach [88], two succinct terminal-free patterns generate the same E-pattern language if and only if they are identical up to a renaming or, in other words, if they have the same canonical form.

According to Holub [44], given a pattern $\alpha \in X^+$, one can compute a morphic root in polynomial time. Thus, to decide whether $L_{\mathrm{E},\Sigma}(\alpha) = L_{\mathrm{E},\Sigma}(\beta)$ holds for patterns $\alpha, \beta \in X^+$, one first computes morphic roots α_ρ and β_ρ of α and β, respectively. As mentioned in Theorem 5.4, α_ρ and β_ρ are succinct patterns. One then converts α_ρ and β_ρ into canonical form. Taking into account that $|\operatorname{var}(\alpha_\rho)|$ and $|\operatorname{var}(\beta_\rho)|$ are bounded by the lengths of the patterns, this can be done in log-linear time. Finally, $L_{\mathrm{E},\Sigma}(\alpha) = L_{\mathrm{E},\Sigma}(\beta)$ holds if and only if the canonical forms of α_ρ and β_ρ are identical, this can be checked in linear time. Thus, the whole procedure works in polynomial time. □

5.2.2 L Systems

From time to time, we shall mention relations or similarities between the content of Chapters 5 to 7 to so-called *L systems*. As a service to the reader, the present section provides a very short overview over the types of L systems we shall refer to. L systems were a very active area of Formal Language Theory from the late 1960s deep into the 1990s, and were topic of thousands of articles. This section only visits the aspects that are relevant to the present thesis; a far more extensive introduction with pointers to various surveys and bibliographies is Kari et al. [53].

In principle, most L systems can be considered a parallel variant of Chomsky grammars. In contrast to this more general view, we consider only *deterministic* L systems, which can be understood as iterating one or multiple morphisms, instead of using a set of rules (i. e., in deterministic L systems, in every step, every letter is replaced in the same way).

The most basic type of L system considered in this thesis is the *D0L system*. A D0L system over some alphabet Σ consists of a morphism $\phi : \Sigma^* \to \Sigma^*$ and a word $w \in \Sigma^*$ (the *axiom*). A D0L system $G = (\phi, w)$ generates the language $L(G) := \{\phi^i(w) \mid i \geq 0\}$ – in other words, $L(G)$ consists of all the words that can be obtained from w by iteration of ϕ. We illustrate this definition using the following examples from Rozenberg and Salomaa [93]:

Example 5.7. *Let $\Sigma := \{\mathtt{a}\}$ and define the morphism $\phi : \Sigma^* \to \Sigma^*$ through $\phi(\mathtt{a}) := \mathtt{a}^2$. Then D0L system $G_1 := (\phi, \mathtt{a})$ generates the language $L(G_1) = \{a^{2^i} \mid i \geq 0\}$.*

Let $\Sigma := \{\mathtt{a}, \mathtt{b}\}$ and define $\phi : \Sigma^* \to \Sigma^*$ by $\phi(\mathtt{a}) := \mathtt{a}\,\mathtt{b}^2\,\mathtt{a}$ and $\phi(\mathtt{b}) := \lambda$. The D0L system $G_2 := (\phi, \mathtt{a}\,\mathtt{b}^2\,\mathtt{a})$ generates the language $L(G_2) = \{(\mathtt{a}\,\mathtt{b}^2\,\mathtt{a})^{2^n} \mid n \geq 0\}$. ◇

Section 1.2 of [53] shows a D0L system that models the development of a red alga and explains how L systems are often used as models of plant growth. Each application of the morphism (or, for nondeterministic L systems, a rewriting rule) can be understood as the passage of one time step, and each of the images $\phi^i(w)$ corresponds to one stage of the growth of a plant. In Chapter 7, we pick up this growth metaphor.

The theory of L systems uses a mostly uniform terminology; each class of L systems is identified by the combination of letters appended in front of the L. For example,

D0L systems are L systems that are deterministic (hence 'D') and context-free in the application of rules (hence the '0', see below). Of the commonly used letters listed at the end of Section 2.3 in [53], the present thesis uses the following subset:

- D. deterministic, only one choice, only one choice in each table
- F. finite set of axioms, rather than only one axoim
- H. homomorphism, morphism, image under morphism
- L. Lindenmayer, appears in the name of all developmental systems
- O. actually number 0 but often read as the letter, information 0-sided, no interaction, rewriting context-free
- T. tables, sets of rules, diversification of developmental instructions[1]

Of the possible combinations of these letters, we use D0L (as introduced above), HD0L, DT0L, DF0L, and DTF0L. An *HD0L system* over an alphabet Σ consists of two morphisms $\phi, \psi : \Sigma^* \to \Sigma^*$ and an axiom $w \in \Sigma^*$. The HD0L system $G = (\phi, \psi, w)$ generates the language $L(G) := \{\psi(\phi^i(w)) \mid i \geq 0\}$. In other words, G applies ψ to every word of the language that is generated by the D0L system (ψ, w). For further illustration, consider the following example:

Example 5.8. *Let* $\Sigma := \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ *and consider the morphisms* $\phi, \psi : \Sigma^* \to \Sigma^*$ *that are defined by* $\phi(\mathtt{a}) := \mathtt{a\,b\,b\,c}$, $\phi(\mathtt{b}) := \mathtt{b\,c}$, $\phi(\mathtt{c}) := \mathtt{c}$, *and* $\psi(\mathtt{a}) := \psi(\mathtt{b}) := \psi(\mathtt{c}) := \mathtt{a}$. *Then the HD0L system* $G := (\phi, \psi, \mathtt{a})$ *generates the language* $L(G) = \{\mathtt{a}^{i^2} \mid i \geq 0\}$, *as can be easily verified by considering that, for every* $i \geq 0$, *both* $|\phi^{i+1}(\mathtt{a})|_{\mathtt{a}} + |\phi^{i+1}(\mathtt{a})|_{\mathtt{b}} = 2i + 1$ *and* $|\phi^{i+1}(\mathtt{a})|_{\mathtt{c}} = i^2$ *hold.* ◇

Furthermore, a *DF0L system* is defined by a single morphism ϕ and a finite set of axioms, and generates the language that consists of all words that can be obtained from any of the axioms through iteration of ϕ.

Analogously, a *DT0L system* uses only a single axiom, but a finite set of morphisms, and its language contains all words that can be obtained by applying any combination of these morphisms to the axiom (where each morphism may be used an unbounded number of times).

Finally, a *DTF0L system* uses a finite set of axioms and a finite set of morphisms. As is to be expected, its language consists of all words that can be obtained from any of the axioms through any combination of the morphisms.

As a side note on the wider context, see Harju and Karhumäki [40] for a short explanation how questions on D0L systems and HD0L systems have been shown to be related to the satisfiability problem for word equations (cf. Section 3.4.1).

[1] The author would like to use this as an occasion to repeat an anecdote on the origin of L systems, and a different explanation of the letter 'L', quoted directly from Rozenberg and Salomaa [94]:

> Aristid Lindenmayer was passing by a lecture room where a lecture about formal languages was being held. The lecturer was speaking of $L(G)$. "Algae!" was Aristid's immediate reaction. (The pronounciations of "L(G)" and "algae" are very similar, at least for most of us.) After that Aristid started to investigate applications of language theory to developmental biology. [..] The following (true) story is an example of Aristid's modesty. At one of the American conferences somebody asked Aristid what the L in "L systems" stands for. This happened at a time when L systems where already well-known. Aristid's answer was "languages"!

5.3 Descriptive Patterns and Infinite Strictly Decreasing Chains of Pattern Languages

Before we state and prove the main results of the present chapter, we discuss some simple yet enlightening observations that establish a connection between descriptiveness of patterns and infinite strictly decreasing chains of pattern languages over some fixed alphabet, i. e. sequences $(L_i)_{i=0}^{\infty}$ of pattern languages satisfying, for every $j \geq 0$, $L_j \supset L_{j+1}$. This aspect is already briefly mentioned by Jiang et al. [50].

Since, by definition, a descriptive pattern generates a smallest pattern language comprising a language S, S does not have a descriptive pattern if and only if no pattern language L satisfying $L \supseteq S$ is smallest. Hence, the existence of a descriptive pattern essentially depends on the existence of a pattern language that is not contained in an infinite strictly decreasing chain:

Observation 5.9. *Let Σ be an alphabet with $|\Sigma| \geq 2$ and let $S \subseteq \Sigma^*$ be a language. Then there is no pattern that is NE-descriptive (or E-descriptive) of S if and only if, for every pattern α with $L_{\mathrm{NE},\Sigma}(\alpha) \supseteq S$ (or $L_{\mathrm{E},\Sigma}(\alpha) \supseteq S$, respectively) there is*

- *a sequence of patterns $\alpha_i \in \mathrm{Pat}_\Sigma$, $i \geq 0$, satisfying, for every $j \geq 0$,*
  - *$L_{\mathrm{NE},\Sigma}(\alpha_j) \supset L_{\mathrm{NE},\Sigma}(\alpha_{j+1})$ (or $L_{\mathrm{E},\Sigma}(\alpha_j) \supset L_{\mathrm{E},\Sigma}(\alpha_{j+1})$, respectively) and*
  - *$L_{\mathrm{NE},\Sigma}(\alpha_j) \supseteq S$ (or $L_{\mathrm{E},\Sigma}(\alpha_j) \supseteq S$, respectively)*

  and
- *an $n \geq 0$ with $L_{\mathrm{NE},\Sigma}(\alpha_n) = L_{\mathrm{NE},\Sigma}(\alpha)$ (or $L_{\mathrm{E},\Sigma}(\alpha_n) = L_{\mathrm{E},\Sigma}(\alpha)$, respectively).*

Proof. Directly from the definition of an NE-descriptive (or E-descriptive) pattern. □

Consequently, the question of whether there is a descriptive pattern for a language S requires insights into the inclusion problem for pattern languages. As discussed in Chapter 3, this problem is undecidable in the general case, but it is decidable for the class of terminal-free E-pattern languages (though NP-complete, cf. Section 5.2.1).

In order to illustrate and substantiate Observation 5.9 and as a reference for further considerations in Section 5.4, we now give some examples of strictly decreasing chains of pattern languages. We begin with a sequence of patterns that has almost identical properties for both NE- and E-pattern languages:

Example 5.10. *Let Σ be any alphabet. For every $i \geq 0$, we define $\alpha_i := x_1^{2^i}$, i. e. $\alpha_0 = x_1$, $\alpha_1 = x_1^2$, $\alpha_2 = x_1^4$, $\alpha_3 = x_1^8$ and so on. It can be easily seen that, for every $j \geq 0$, there is a morphism $h : \{x_1\}^+ \to \{x_1\}^+$, defined by $h(x_1) := x_1^2$, satisfying $h(\alpha_j) = \alpha_{j+1}$. Since, for both NE- and E-pattern languages, the existence of such a morphism is a sufficient condition for an inclusion relation (cf. Lemma 3.1 by Angluin [4] and Theorem 2.3 by Jiang et al. [50], respectively), $L_{\mathrm{NE},\Sigma}(\alpha_j) \supseteq L_{\mathrm{NE},\Sigma}(\alpha_{j+1})$ and $L_{\mathrm{E},\Sigma}(\alpha_j) \supseteq L_{\mathrm{E},\Sigma}(\alpha_{j+1})$ are satisfied. In the given example, it is evident that all inclusions of NE-pattern languages are strict. The same holds for the inclusion of E-pattern languages; alternatively, for all but unary alphabets Σ, it is directly proven by Lemma 5.1 (using Theorem 5.2 and Theorem 5.3) given in Section 5.2.1. Hence, the sequence of α_i leads to an infinite*

strictly decreasing chain for NE-pattern languages as well as for E-pattern languages. Nevertheless, the sequence of patterns is irrelevant in the context of Observation 5.9, as the sets $S_{\mathrm{NE}} := \bigcap_{i=0}^{\infty} L_{\mathrm{NE},\Sigma}(\alpha_i)$ *and* $S_{\mathrm{E}} := \bigcap_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(\alpha_i)$*, i. e. those languages all patterns are consistent with, satisfy* $S_{\mathrm{NE}} = \emptyset$ *and* $S_{\mathrm{E}} = \{\lambda\}$*.* ◇

Our next example looks quite similar to Example 5.10, but here a difference between NE- and E-pattern languages can be noted:

Example 5.11. *Let* Σ *be an alphabet with* $|\Sigma| \geq 2$*. For every* $i \geq 0$*, we define* $\alpha_i := x_1^{2^i} y^2$*, i. e.* $\alpha_0 = x_1 y^2$*,* $\alpha_1 = x_1^2 y^2$*,* $\alpha_2 = x_1^4 y^2$*,* $\alpha_3 = x_1^8 y^2$ *and so on. Referring to the same facts as mentioned in Example 5.10, it can be shown that the patterns again define one infinite strictly decreasing chain of NE-pattern languages and another one of E-pattern languages. However, while the set* $S_{\mathrm{NE}} := \bigcap_{i=0}^{\infty} L_{\mathrm{NE},\Sigma}(\alpha_i)$ *again is empty,* $S_{\mathrm{E}} := \bigcap_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(\alpha_i)$ *now equals* $L_{\mathrm{E},\Sigma}(y^2)$*. Hence, we have a chain of E-pattern languages that are all consistent with a nontrivial language. Nevertheless,* $L_{\mathrm{E},\Sigma}(y^2)$ *obviously has a descriptive pattern, namely* $\delta := y^2$*, and this of course holds for all infinite sequences of patterns where* S_{E} *equals an E-pattern language. Consequently, the existence of a single infinite strictly decreasing chain of E-pattern languages* L_i *satisfying, for every* $i \geq 0$*,* $L_i \supseteq S$*, does not mean that there is no E-descriptive pattern for* S*. Furthermore, it is worth mentioning that we can replace* S_{E} *with a finite language and still preserve the above described properties of the* α_i *and* δ*. For* $\Sigma \supseteq \{\mathtt{a}, \mathtt{b}\}$*, this is demonstrated, e. g., by the language* $S := \{\mathtt{a}\,\mathtt{a}, \mathtt{b}\,\mathtt{b}\}$*, which satisfies, for every* $i \geq 0$*,* $S \subseteq L_{\mathrm{E},\Sigma}(\alpha_i)$ *and has the E-descriptive pattern* δ*.* ◇

Our final example presents a special phenomenon of E-pattern languages, namely the existence of bi-infinite strictly decreasing/increasing chains of such languages:

Example 5.12. *Let* Σ *be an alphabet with* $|\Sigma| \geq 2$*. For every* $i \in \mathbb{Z}$*, we define*

$$\alpha_i := \begin{cases} x_1^{2^{-i}} & \text{if } i \text{ is negative,} \\ x_1^2 x_2^2 \ \ldots \ x_{i+2}^2 & \text{else.} \end{cases}$$

Hence, for example, from $i = -3$ *to* $i = 2$ *the patterns read* $\alpha_{-3} = x_1^8$*,* $\alpha_{-2} = x_1^4$*,* $\alpha_{-1} = x_1^2$*,* $\alpha_0 = x_1^2 x_2^2$*,* $\alpha_1 = x_1^2 x_2^2 x_3^2$*, and* $\alpha_2 = x_1^2 x_2^2 x_3^2 x_4^2$*. Using Theorem 5.3, it is easy to show that all patterns are morphically primitive. Theorem 5.2 demonstrates that all morphisms mapping an* α_k *to an* α_j*,* $j < k$*, are not imprimitivity morphisms. Therefore we can conclude from Lemma 5.1 that* $L_{\mathrm{E},\Sigma}(\alpha_j) \subset L_{\mathrm{E},\Sigma}(\alpha_k)$ *if and only if* $j < k$*. For the given patterns,* $S_{\mathrm{E}} := \bigcap_{i=-\infty}^{\infty} L_{\mathrm{E},\Sigma}(\alpha_i)$ *equals* $\{\lambda\}$*, but if we define, for every* $i \in \mathbb{Z}$*,* $\alpha'_i := y^2 \alpha_i$*, then these* α'_i *generate a bi-infinite strictly decreasing/increasing chain of E-pattern languages where* $S_{\mathrm{E}} := \bigcap_{i=-\infty}^{\infty} L_{\mathrm{E},\Sigma}(\alpha'_i) = L_{\mathrm{E},\Sigma}(y^2)$ *is an E-pattern language.* ◇

Note that the example patterns given above are terminal-free merely for the sake of convenience. They can be effortlessly turned into certain patterns containing terminal symbols and still showing equivalent properties.

5.4 Existence of Descriptive Patterns

In the present chapter we study the existence of patterns that are descriptive of sets S of strings. According to the remarks in Section 5.1, four main cases can be considered,

depending on whether S is finite or infinite and whether NE- or E-descriptiveness is examined. We focus on the existence of E-descriptive patterns for infinite languages since, for the other three cases, answers are absolutely straightforward or directly or indirectly provided by Angluin [4] and Jiang et al. [50]. In order to give a comprehensive description and further explain some of the formal concepts and statements we nevertheless also briefly describe the known or trivial cases.

Using Observation 5.9, the question of the existence of *NE-descriptive* patterns can be easily answered for all types of languages S. We begin with the case of a *finite* S. Here, it is primarily necessary to observe that a word w can only be covered by a pattern α through nonerasing substitutions if α is not longer than w. Hence, for any finite alphabet Σ and any word over Σ, there are only finitely many NE-pattern languages over Σ covering this word; this property of a class of languages is commonly referred to as *finite thickness* (cf. Wright [105]). Quite obviously, the same holds for infinite alphabets Σ, since the number of different terminal symbols that can occur in patterns covering w is limited by the number of different terminal symbols in w. With regard to infinite sequences of patterns (generating languages that all differ from each other) over a fixed alphabet, this means that none of them can contain infinitely many patterns that cover, e. g., the shortest word in a given finite set of strings. This immediately shows that, for every finite S, there exists an NE-descriptive pattern:

Corollary 5.13 (Angluin [4])**.** *Let Σ be an alphabet and $S \subseteq \Sigma^+$ a finite language. Then there is a pattern $\delta \in \mathrm{Pat}_\Sigma$ that is NE-descriptive of S.*

Note that Angluin [4] does not explicitly state Corollary 5.13, but directly studies more challenging questions by introducing a procedure computing an NE-descriptive pattern for any finite language S and examining the computational complexity of the problem of finding such patterns for finite languages (cf. Theorems 5.29 and 5.30 in Section 5.5.1).

With regard to NE-descriptive patterns for *infinite* languages S, the same reasoning as for finite languages S leads to the analogous result:

Proposition 5.14. *Let Σ be an alphabet and $S \subseteq \Sigma^+$ an infinite language. Then there is a pattern $\delta \in \mathrm{Pat}_\Sigma$ that is NE-descriptive of S.*

Proof. Directly from Observation 5.9 and the finite thickness of the class of NE-pattern languages. □

A closer look at the underlying reasoning proving Corollary 5.13 and Proposition 5.14 reveals that it does not need to consider whether any infinite sequence of patterns leads to an infinite strictly decreasing chain of NE-pattern languages (as featured by Observation 5.9), but can be completely based on the concept of finite thickness. If we nevertheless wish to examine the properties of such chains, then we can easily observe that, for every sequence of patterns α_i, $i \geq 0$, with $L_{\mathrm{NE},\Sigma}(\alpha_i) \supset L_{\mathrm{NE},\Sigma}(\alpha_{i+1})$, the set $S_{\mathrm{NE}} := \bigcap_{i=0}^{\infty} L_{\mathrm{NE},\Sigma}(\alpha_i)$ necessarily is empty. Hence, Examples 5.10 and 5.11 illustrate the only option possible.

With regard to *E-descriptiveness*, the situation is more complex. As shown by Examples 5.11 and 5.12, the class of E-pattern languages does not have finite thickness and there are even finite and infinite languages that are contained in all E-pattern languages

of an infinite strictly decreasing chain. Nevertheless, it is known that every nontrivial *finite* language has an E-descriptive pattern:

Theorem 5.15 (Jiang et al. [50])**.** *Let Σ be an alphabet and $S \subseteq \Sigma^*$ a finite language, $S \neq \{\lambda\}$. Then there is a pattern $\delta \in \mathrm{Pat}_\Sigma$ that is E-descriptive of S.*

The proof for Theorem 5.15 given by Jiang et al. [50] demonstrates that for every finite language S an upper bound n can be given such that, for every pattern α consistent with S, there exists a pattern β satisfying $|\beta| \leq n$ and $S \subseteq L_{\mathrm{E},\Sigma}(\beta) \subseteq L_{\mathrm{E},\Sigma}(\alpha)$. So if, for any finite S, there is a sequence of patterns α_i, $i \geq 0$, leading to an infinite strictly decreasing chain of E-pattern languages comprising S – which implies that there is no upper bound for the length of the α_i – then all but finitely many of these patterns need to have variables that are not required for generating the words in S. This phenomenon is illustrated by Example 5.11, where only the subpattern y^2 of all patterns is necessary in order to map the patterns to the words in S_{E}.

In the proof for Theorem 5.15, the upper bound n for the length of the α_i equals the sum of the lengths of the words in S. Thus, this method cannot be adopted when investigating the existence of E-descriptive patterns for *infinite* sets of words. In fact, as to be demonstrated below, we here need to consider two subcases depending on the number of different letters occurring in the words of S. If the underlying alphabet is unary, then the descriptiveness of a pattern is related to the inclusion relation of E-pattern languages over this unary alphabet. The structure of such E-pattern languages, however, is significantly simpler than that of E-pattern languages over larger alphabets; in particular, the full class of these languages is a specific subclass of the regular languages (namely the linear unary languages). Therefore it can be shown that, for every sequence of patterns $(\alpha_i)_{i=0}^{\infty}$ leading to an infinite strictly decreasing chain of E-pattern languages over a unary alphabet, the language $S_{\mathrm{E}} := \bigcap_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(\alpha_i)$ is finite. Referring to Observation 5.9, this directly leads to the following result:

Theorem 5.16. *Let Σ be an alphabet, $|\Sigma| = 1$, and $S \subseteq \Sigma^*$ an infinite language. Then there is a pattern $\delta \in \mathrm{Pat}_\Sigma$ that is E-descriptive of S.*

The proof for Theorem 5.16 is given in Section 5.4.1.

In contrast to this, Example 5.11 demonstrates that, for alphabets with at least two letters, there is an infinite strictly decreasing chain of E-pattern languages such that the intersection of all these languages is infinite. Since this intersection is an E-pattern language, Example 5.11 can nevertheless not be used to establish a result that differs from those given for the other cases. In order to answer the question of whether this holds true for all such chains, we now consider a more sophisticated infinite sequence of patterns, that is defined as follows:

Definition 5.17. *We define the pattern*

$$\alpha_0 := y^2 z^2$$

and the morphism $\phi : X^* \to X^*$ (note that we assume $X \supseteq \{y, z, x_0, x_1, x_2 \ldots\}$) through, for every $i \geq 0$,

$$\begin{aligned} \phi(x_i) &:= x_{i+1}, \\ \phi(y) &:= y^2 x_1, \\ \phi(z) &:= x_1 z^2. \end{aligned}$$

Then, for every $i \geq 0$, the pattern α_{i+1} is given by

$$\alpha_{i+1} := \phi(\alpha_i) = \phi^i(\alpha_0).$$

This means that, for example,

$$\begin{aligned}
\alpha_1 &= y^2x_1\, y^2x_1\, x_1z^2\, x_1z^2,\\
\alpha_2 &= (y^2x_1y^2x_1x_2)\, (y^2x_1y^2x_1x_2)\, (x_2x_1\, z^2x_1z^2)\, (x_2x_1z^2x_1z^2),\\
\alpha_3 &= (y^2x_1y^2x_1x_2\, y^2x_1y^2x_1x_2\, x_3)\, (y^2x_1y^2x_1x_2\, y^2x_1y^2x_1x_2\, x_3)\\
&\quad \cdot (x_3\, x_2x_1z^2x_1z^2\, x_2x_1z^2x_1z^2)\, (x_3\, x_2x_1z^2x_1z^2\, x_2x_1z^2x_1z^2),
\end{aligned}$$

and, for sufficiently large i,

$$\begin{aligned}
\alpha_i &= (((((\ldots((((y^2x_1)^2x_2)^2x_3)^2x_4)^2\ldots x_{i-4})^2x_{i-3})^2x_{i-2})^2x_{i-1})^2x_i)^2\\
&\quad \cdot (x_i(x_{i-1}(x_{i-2}(x_{i-3}(x_{i-4}\ldots(x_4(x_3(x_2(x_1z^2)^2)^2)^2)^2\ldots)^2)^2)^2)^2)^2.
\end{aligned}$$

It can be shown that this sequence $(\alpha_i)_{i=0}^{\infty}$ defines an infinite strictly decreasing chain of E-pattern languages. Furthermore, if we define the morphism $\psi : X^* \to X^*$ through $\psi(x_i) := x_i$ and $\psi(y) := \psi(z) := x_0$, then, for every alphabet Σ with $|\Sigma| \geq 2$,

$$L_\Sigma := \bigcup_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(\psi(\alpha_i))$$

satisfies $L_\Sigma \subseteq \bigcap_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(\alpha_i)$. As a side note, it is worth mentioning that L_Σ is a multi-pattern language (cf. Dumitrescu et al. [25]) where the set $\{\psi(\alpha_i) \mid i \geq 0\}$ of generating patterns is defined similarly to an HD0L language (albeit over an infinite alphabet of variables); such a concept has not been considered by previous literature[2]. Finally, it can be demonstrated that the sequence $(\alpha_i)_{i=0}^{\infty}$ has a very particular property, since for every pattern γ with $L_{\mathrm{E},\Sigma}(\gamma) \supseteq L_\Sigma$ there exists an α_i satisfying $L_{\mathrm{E},\Sigma}(\gamma) \supseteq L_{\mathrm{E},\Sigma}(\alpha_i)$. Referring to Observation 5.9, this implies the main result of the present chapter:

Theorem 5.18. *For every alphabet Σ with $|\Sigma| \geq 2$ there is an infinite language $L_\Sigma \subset \Sigma^*$ that has no E-descriptive pattern.*

The proof for Theorem 5.18 is given below in Section 5.4.2. Consequently, when searching for descriptive patterns, the case of E-descriptive patterns of infinite languages over alphabets of at least two letters is the only one where the existence of such patterns is not always guaranteed. This directly answers a question posed by Jiang et al. [50].

Consequently, when searching for descriptive patterns, the case of E-descriptive patterns of infinite languages over alphabets of at least two letters is the only one where the existence of such patterns is not always guaranteed. This directly answers a question posed by Jiang et al. [50].

Finally, it can be shown that, while the proof of Theorem 5.18 is based on the particular shape of the infinite union L_Σ of E-pattern languages described above, L_Σ can be replaced by a language L_Σ^t which, for every pattern $\psi(\alpha_i)$, $i \geq 0$, contains just a single word. In order to describe this insight more precisely, we have to introduce the following concept:

[2]We revisit this observation in Chapter 7, especially in Section 7.2.

Definition 5.19. *A language L is called* properly thin *if, for every $n \geq 0$, L contains at most one word of length n.*

Referring to this definition, we can strengthen Theorem 5.18 as follows:

Corollary 5.20. *For every alphabet Σ with $|\Sigma| \geq 2$, there is an infinite properly thin language $L^t_\Sigma \subset \Sigma^*$ that has no E-descriptive pattern.*

The proof of Corollary 5.20 is given in Section 5.4.3. Note that Section 7.3.1 in Chapter 7 contains the sketch of a modification to the proof of Theorem 5.18 that uses a variation of the chain.

5.4.1 Proof of Theorem 5.16

Before we give the actual proof of Theorem 5.16, we introduce some concepts that are only relevant to this section.

To begin with, we extend the operations addition, subtraction, multiplication and division from the natural numbers to operations on natural numbers with sets of natural numbers in the canonical way; i. e., for $\star \in \{+, -, \cdot, /\}$ and $M \subseteq \mathbb{N}$, $b \in \mathbb{N}$ let $M \star b := \{m \star b \mid m \in M\}$. Note that in all cases where we use division or subtraction, the results will always be natural numbers; furthermore, we make free use of the commutativity of multiplication and addition and write $b + M$ or $b \cdot M$ instead of $M + b$ or $M \cdot b$, respectively. For any (possibly infinite) $M \subseteq \mathbb{N}$, let $\gcd(M)$ denote the *greatest common divisor* of all elements of M.

Let $n \geq 1$ and $M = \{m_1, \ldots, m_n\} \subset \mathbb{N}_1$. We define the *linear hull of M* as $\mathrm{lin}(M) := \{m \mid m = k_1 m_1 + \ldots + k_n m_n \text{ for some } k_1, \ldots, k_n \in \mathbb{N}\}$, and $\mathrm{lin}(\emptyset) := \{0\}$.

It is obvious that every unary language L is isomorphic to its Parikh set $P(L) := \{|w| \mid w \in L\} \subseteq \mathbb{N}$. We say that a unary language L is *linear* if there is a $b \geq 0$ and a finite set $G \subset \mathbb{N}$ such that $P(L) = b + \mathrm{lin}(G)$. This allows us to state the following observation on unary pattern languages:

Proposition 5.21. *A unary language is linear if and only if it is a pattern language.*

Proof. Let $\Sigma = \{\mathtt{a}\}$. We begin with the *if* direction. Let $\alpha \in \mathrm{Pat}_\Sigma$ with $\mathrm{var}(\alpha) = \{x_1, \ldots, x_n\}$ for some $n \geq 0$. Let $b := |\alpha|_\mathtt{a}$ and, for $1 \leq i \leq n$, $g_i := |\alpha|_{x_i}$; furthermore, we define $\beta := \mathtt{a}^b\, x_1^{g_1} \ldots x_n^{g_n}$. As Σ is unary, $L_{\mathrm{E},\Sigma}(\alpha) = L_{\mathrm{E},\Sigma}(\beta)$ holds, and it is easy to see that $P(L_{\mathrm{E},\Sigma}(\beta)) = b + \mathrm{lin}(\{g_1, \ldots, g_n\})$.

Conversely, if some language $L \subseteq \Sigma^*$ is linear, then there exist a $b \geq 0$ and a finite set $G = \{g_1, \ldots, g_n\} \supset \mathbb{N}$ (with $n \geq 0$) satisfying $P(L) = b + \mathrm{lin}(G)$. If we define β as above, $P(L_{\mathrm{E},\Sigma}(\beta)) = b + \mathrm{lin}(G) = P(L)$ leads to $L_{\mathrm{E},\Sigma}(\beta) = L$. □

Also, note this important fact on linear hulls:

Lemma 5.22. *For every finite $M \subset \mathbb{N}$, there exists an $n \geq 1$ with $\mathrm{lin}(M) \supseteq \gcd(M) \cdot \mathbb{N}_n$.*

Proof. The case of $\gcd(M) = 1$ is well known, a proof can be found in Chapter 3.15 of Wilf [104]. If $\gcd(M) > 1$, let $M' := M / \gcd(M)$. Then, as $\gcd(M') = 1$, there is an $n \geq 1$ such that $\mathrm{lin}(M') \supseteq \mathbb{N}_n$, and therefore, $\mathrm{lin}(M) = \gcd(M) \cdot \mathrm{lin}(M') \supseteq \gcd(M) \cdot \mathbb{N}_n$. □

Now that all necessary tools have been introduced, we are ready for the proof of Theorem 5.16:

Proof. Let $\Sigma{:=}\{\mathtt{a}\}$. Furthermore, let

$$\begin{aligned} b &:= \min(P(S)), \\ P'_S &:= P(S) - b, \\ g &:= \gcd(P'_S), \\ P''_S &:= P'_S/g \end{aligned}$$

and $\alpha{:=}\,\mathtt{a}^b\, x_1^g$. It is easy to verify that $L_{\mathrm{E},\Sigma}(\alpha) \supseteq S$, $P(L_{\mathrm{E},\Sigma}(\alpha)) = b + g \cdot \mathbb{N}$ and $P(S) = b + g \cdot P''_S$. Although α is not necessarily E-descriptive of S, we shall see that there is always only a finite number of pattern languages between $L_{\mathrm{E},\Sigma}(\alpha)$ and S.

Since Σ is unary, we have, for every pattern $\beta \in \mathrm{Pat}_\Sigma$ with $L_{\mathrm{E},\Sigma}(\alpha) \supset L_{\mathrm{E},\Sigma}(\beta) \supseteq S$,

$$P(L_{\mathrm{E},\Sigma}(\alpha)) \supset P(L_{\mathrm{E},\Sigma}(\beta)) \supseteq P(S).$$

This, in turn, is equivalent to

$$b + g \cdot \mathbb{N} \supset P(L_{\mathrm{E},\Sigma}(\beta)) \supseteq b + g \cdot P''_S.$$

Due to this relation and Proposition 5.21, we can conclude with some effort that there is a finite $G_\beta \supset \mathbb{N}$ with $P(L_{\mathrm{E},\Sigma}(\beta)) = b + g \cdot \mathrm{lin}(G_\beta)$. Therefore,

$$b + g \cdot \mathbb{N} \supset b + g \cdot \mathrm{lin}(G_\beta) \supseteq b + g \cdot P''_S,$$

which is equivalent to

$$\mathbb{N} \supset \mathrm{lin}(G_\beta) \supseteq P''_S.$$

As $\gcd(P''_S) = 1$, there is a finite $C_S \subset P''_S$ with $\gcd(C_S) = 1$. We observe that

$$\mathrm{lin}(G_\beta) \supseteq P''_S \supset C_S,$$

and, as C_S is a finite subset of $\mathrm{lin}(G_\beta)$,

$$\mathrm{lin}(G_\beta) \supseteq \mathrm{lin}(C_S).$$

Due to Lemma 5.22, there is an $n \geq 0$ such that $\mathrm{lin}(C_S) \supseteq \mathbb{N}_n$, and thus, $\mathrm{lin}(G_\beta) \supseteq \mathbb{N}_n$, which leads to $P(L_{\mathrm{E},\Sigma}(\beta)) \supseteq b + g \cdot \mathbb{N}_n$.

Now, assume that there is an infinite sequence $(\beta_i)_{i=0}^\infty$ over Pat_Σ such that $L_{\mathrm{E},\Sigma}(\alpha) \supset L_{\mathrm{E},\Sigma}(\beta_i) \supset L_{\mathrm{E},\Sigma}(\beta_{i+1}) \supset S$ for every $i \geq 0$. Then there is an infinite sequence $(G_{\beta_i})_{i=0}^\infty$ of finite subsets of $\mathbb{N}$ with, for every $i \geq 0$, $P(L_{\mathrm{E},\Sigma}(\beta_i)) = b + g \cdot \mathrm{lin}(G_{\beta_i})$ and $\mathrm{lin}(G_{\beta_i}) \supset \mathrm{lin}(G_{\beta_{i+1}}) \supset \mathbb{N}_n$. As $\mathbb{N}_n$ is cofinite, such an infinite sequence cannot exist – therefore, due to Observation 5.9, there must be some pattern that is E-descriptive of S. □

This concludes the proof of Theorem 5.16.

5.4.2 Proof of Theorem 5.18

In order to prove Theorem 5.18, we define L_Σ through the infinite sequence of patterns α_i, $i \geq 0$, given by Definition 5.17 in such a way that the words of L_Σ are structurally so close to the patterns α_i that, for every pattern $\delta \in \mathrm{Pat}_\Sigma$ with $L_{\mathrm{E},\Sigma}(\delta) \supseteq L_\Sigma$, there is a $j \geq 0$ with $L_{\mathrm{E},\Sigma}(\delta) \supset L_{\mathrm{E},\Sigma}(\alpha_j) \supset L_\Sigma$. Thus, regardless of how closely $L_{\mathrm{E},\Sigma}(\delta)$ approximates L_Σ, there is always an α_j that provides a better description of L_Σ.

As briefly mentioned above, L_Σ is an infinite union of E-pattern languages. The corresponding patterns are derived from the patterns α_i by a morphism $\psi : X^* \to X^*$, defined through

$$\begin{aligned} \psi(x_i) &:= x_i, \\ \psi(y) := \psi(z) &:= x_0. \end{aligned}$$

Applying ψ to the patterns α_i, we receive an infinite sequence of patterns $(\beta_i)_{i=0}^\infty$; i. e., we define, for every $i \geq 0$, $\beta_i := \psi(\alpha_i)$. As the rather simple structure of ψ suggests, any pattern β_j, $j \geq 0$, is structurally very close to the patterns α_j, since the only difference is that both y and z are replaced by the variable x_0:

$$\begin{aligned} \beta_0 &= x_0^2\, x_0^2, \\ \beta_1 &= x_0^2 x_1\, x_0^2 x_1\, x_1 x_0^2\, x_1 x_0^2, \\ \beta_2 &= (x_0^2 x_1 x_0^2 x_1 x_2)\, (x_0^2 x_1 x_0^2 x_1 x_2)\, (x_2 x_1\, x_0^2 x_1 x_0^2)\, (x_2 x_1 x_0^2 x_1 x_0^2), \\ \beta_3 &= (x_0^2 x_1 x_0^2 x_1 x_2\, x_0^2 x_1 x_0^2 x_1 x_2\, x_3)\, (x_0^2 x_1 x_0^2 x_1 x_2\, x_0^2 x_1 x_0^2 x_1 x_2\, x_3) \\ &\quad (x_3\, x_2 x_1 x_0^2 x_1 x_0^2\, x_2 x_1 x_0^2 x_1 x_0^2)\, (x_3\, x_2 x_1 x_0^2 x_1 x_0^2\, x_2 x_1 x_0^2 x_1 x_0^2), \\ &\vdots \\ \beta_i &= (((((\ldots((((x_0^2 x_1)^2 x_2)^2 x_3)^2 x_4)^2 \ldots x_{i-4})^2 x_{i-3})^2 x_{i-2})^2 x_{i-1})^2 x_i)^2 \\ &\quad (x_i(x_{i-1}(x_{i-2}(x_{i-3}(x_{i-4} \ldots (x_4(x_3(x_2(x_1 x_0^2)^2)^2)^2)^2 \ldots)^2)^2)^2)^2)^2. \end{aligned}$$

Finally, for any alphabet Σ with $|\Sigma| \geq 2$, we define $L_\Sigma := \bigcup_{i=0}^\infty L_{\mathrm{E},\Sigma}(\beta_i)$.

The relation between the patterns β_i can again be expressed by a morphism, namely $\mu : X^* \to X^*$ given by

$$\begin{aligned} \mu(x_i) &:= \begin{cases} \lambda & \text{if } i = 0, \\ x_{i-1} & \text{if } i > 0, \end{cases} \\ \mu(y) &:= y, \\ \mu(z) &:= z. \end{aligned}$$

Figuratively speaking, the morphism μ permits us to move downward in the sequence $(\beta_i)_{i=0}^\infty$ (note that μ is given for the variables y and z due to technical reasons arising later). This is illustrated by Figure 5.1 and further substantiated by the following lemma:

Lemma 5.23. *For all $i, j \geq 0$, $\mu^j(\beta_{i+j}) = \beta_i$.*

Proof. If $j = 0$, the claim is trivially true. We now consider $j = 1$. By definition,

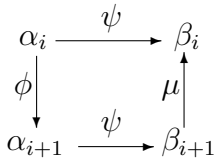

Figure 5.1: Morphic relations between the elements of the sequences $(\alpha_i)_{i=0}^{\infty}$ and $(\beta_i)_{i=0}^{\infty}$.

$\mu(\beta_{i+1}) = (\mu \circ \psi \circ \phi)(\alpha_i)$. The morphism $\mu \circ \psi \circ \phi : X^* \to X^*$ works as follows:

$$\begin{aligned}(\mu \circ \psi \circ \phi)(x) &= \begin{cases}(\mu \circ \psi)(x_{k+1}) & \text{if } x = x_k,\\ (\mu \circ \psi)((y)^2 x_1) & \text{if } x = y,\\ (\mu \circ \psi)(x_1 (z)^2) & \text{if } x = z\end{cases}\\ &= \begin{cases}\mu(x_{k+1}) & \text{if } x = x_k,\\ \mu((x_0)^2 x_1) & \text{if } x = y,\\ \mu(x_1 (x_0)^2) & \text{if } x = z\end{cases}\\ &= \begin{cases}x_k & \text{if } x = x_k,\\ x_0 & \text{if } x = y \text{ or } x = z\end{cases}\\ &= \psi(x).\end{aligned}$$

Therefore, $\mu(\beta_{i+1}) = \psi(\alpha_i) = \beta_i$. For all larger values of j, the claim holds by induction. □

Referring to Theorem 2.4, Figure 5.1 already illustrates certain inclusion relations between the languages generated by the patterns α_i and β_j, $i, j \geq 0$. The following lemma shows that these inclusions are proper, which in particular means that the patterns in $(\alpha_i)_{i=0}^{\infty}$ lead to a strictly decreasing chain of E-pattern languages (as featured by Observation 5.9). Additionally, the lemma describes the relation of the given E-pattern languages to L_Σ. A summary of selected inclusion relations is provided by Figure 5.2.

Lemma 5.24. *For every $i \geq 0$, the following statements hold:*

1. $L_{\mathrm{E},\Sigma}(\alpha_i) \supset L_{\mathrm{E},\Sigma}(\alpha_{i+1}) \supset L_\Sigma$,
2. $L_{\mathrm{E},\Sigma}(\alpha_i) \supset L_{\mathrm{E},\Sigma}(\beta_i)$,
3. $L_{\mathrm{E},\Sigma}(\beta_i) \subset L_{\mathrm{E},\Sigma}(\beta_{i+1}) \subset L_\Sigma$.

Proof. For every $i \geq 0$, the proper inclusion relations $L_{\mathrm{E},\Sigma}(\alpha_i) \supset L_{\mathrm{E},\Sigma}(\alpha_{i+1})$, $L_{\mathrm{E},\Sigma}(\beta_{i+1}) \supset L_{\mathrm{E},\Sigma}(\beta_i)$ and $L_{\mathrm{E},\Sigma}(\alpha_i) \supset L_{\mathrm{E},\Sigma}(\beta_i)$ follow from Lemma 5.1: By definition, $\alpha_{i+1} = \phi(\alpha_i)$ and $\beta_i = \psi(\alpha_i)$, and, due to Lemma 5.23, $\beta_i = \mu(\beta_{i+1})$. Furthermore, the following claim holds true:

Claim. For every $i \in \mathbb{N}$, the patterns α_i and β_i are morphically primitive.

Proof of Claim. According to Theorem 5.3, every morphically imprimitive pattern γ must – among other requirements that need to be satisfied – contain at least one variable that,

$$\begin{array}{ccc}
L_{\mathrm{E},\Sigma}(\alpha_0) & \supset & L_{\mathrm{E},\Sigma}(\beta_0) \\
\cup & & \cap \\
L_{\mathrm{E},\Sigma}(\alpha_1) & \supset & L_{\mathrm{E},\Sigma}(\beta_1) \\
\cup & & \cap \\
L_{\mathrm{E},\Sigma}(\alpha_2) & \supset & L_{\mathrm{E},\Sigma}(\beta_2) \\
\cup & & \cap \\
L_{\mathrm{E},\Sigma}(\alpha_3) & \supset & L_{\mathrm{E},\Sigma}(\beta_3) \\
\cup & & \cap \\
L_{\mathrm{E},\Sigma}(\alpha_4) & \supset & L_{\mathrm{E},\Sigma}(\beta_4) \\
\cup & & \cap \\
\vdots & & \vdots
\end{array}$$

Figure 5.2: Inclusion relations between the E-pattern languages of α_i and β_j, $i, j \geq 0$.

for each of its occurrences in γ, has the same left neighbors or the same right neighbors. More formally, there must be an $x \in \mathrm{var}(\gamma)$ such that there exists a factorization

$$\gamma = \widehat{\gamma}_1 \, \chi_{x,L} \, x \, \chi_{x,R} \, \widehat{\gamma}_2 \, \chi_{x,L} \, x \, \chi_{x,R} \, \widehat{\gamma}_3 \, \cdots \, \widehat{\gamma}_{n-1} \, \chi_{x,L} \, x \, \chi_{x,R} \, \widehat{\gamma}_n$$

with $n \geq 2$, $\chi_{x,L}, \chi_{x,R}, \widehat{\gamma}_1, \widehat{\gamma}_2, \ldots, \widehat{\gamma}_n \in X^* \setminus \{x\}$ and $\chi_{x,L} \neq \lambda$ or $\chi_{x,R} \neq \lambda$.

If we now consider any pattern α_i, $i \geq 0$, then neither y nor z nor x_i can have that property, because they have squared occurrences. More precisely, for $x \in \{y, z, x_i\}$, $\alpha_i = \ldots xx \ldots$, which due to $\chi_{x,L}, \chi_{x,R} \in X^* \setminus \{x\}$ implies $\chi_{x,L} = \lambda$ and $\chi_{x,R} = \lambda$. For every $x_j \in \mathrm{var}(\alpha_i) \setminus \{y, z, x_i\}$, $\alpha_i = \ldots x_j x_{j+1} \ldots$ and $\alpha_i = \ldots x_j y \ldots$, and this means that $\chi_{x_j,R} = \lambda$. Furthermore, for every such x_j, α_i satisfies $\alpha_i = \ldots x_{j+1} x_j \ldots$ and $\alpha_i = \ldots z x_j \ldots$, and this implies $\chi_{x_j,L} = \lambda$. In other words, there is no variable in α_i that, for each of its occurrences, has the same left neighbors or the same right neighbors. Consequently, α_i is morphically primitive.

If we substitute x_0 for y and z in the above reasoning, then it shows that every β_i, $i \geq 0$, is morphically primitive, too. This proves the correctness of the Claim. □*(Claim)*

Finally, according to Theorem 5.2, ϕ, ψ and μ are not imprimitivity morphisms for the patterns they are applied to; by definition, none of the morphisms in question is a renaming of any of the patterns involved. Thus, all conditions of Lemma 5.1 are satisfied, and this directly proves the correctness of our initial statement. In addition to this, these inclusion relations immediately imply $L_{\mathrm{E},\Sigma}(\alpha_i) \supset L_{\mathrm{E},\Sigma}(\beta_j)$ for all $i, j \geq 0$.

For every $i \geq 0$, the inclusion $L_\Sigma \supseteq L_{\mathrm{E},\Sigma}(\beta_i)$ follows from the definition of L_Σ, which in turn immediately leads to $L_\Sigma \neq L_{\mathrm{E},\Sigma}(\beta_i)$, as otherwise $L_{\mathrm{E},\Sigma}(\beta_{i+1}) \supset L_{\mathrm{E},\Sigma}(\beta_i)$ would not be satisfied.

By definition, for every $w \in L_\Sigma$, there is an $i \geq 0$ with $w \in L_{\mathrm{E},\Sigma}(\beta_i)$; and therefore, $w \in L_{\mathrm{E},\Sigma}(\alpha_j)$ for every $j \geq 0$, which implies $L_{\mathrm{E},\Sigma}(\alpha_j) \supseteq L_\Sigma$. Finally, $L_{\mathrm{E},\Sigma}(\alpha_j) = L_\Sigma$ would contradict $L_{\mathrm{E},\Sigma}(\alpha_j) \supset L_{\mathrm{E},\Sigma}(\alpha_{j+1}) \supseteq L_\Sigma$. □

Regarding the possible existence of a pattern δ that is E-descriptive of L_Σ, the language $L_{\mathrm{E},\Sigma}(\delta)$ must, by definition, not be a superlanguage of any of the E-pattern languages in the strictly decreasing chain established by Lemma 5.24. More precisely, for

every pattern $\delta \in \mathrm{Pat}_\Sigma$, if there is an $i \geq 0$ with $L_{\mathrm{E},\Sigma}(\delta) \supseteq L_{\mathrm{E},\Sigma}(\alpha_i)$, we have

$$L_{\mathrm{E},\Sigma}(\delta) \supseteq L_{\mathrm{E},\Sigma}(\alpha_i) \supset L_{\mathrm{E},\Sigma}(\alpha_{i+1}) \supset L_\Sigma,$$

which leads to the following lemma:

Lemma 5.25. *If $\delta \in \mathrm{Pat}_\Sigma$ and $L_{\mathrm{E},\Sigma}(\delta) \supseteq L_{\mathrm{E},\Sigma}(\alpha_i)$ for some $i \geq 0$, then δ is not E-descriptive of L_Σ.*

Therefore, although the language that is generated by a pattern that is E-descriptive of L_Σ (if any) has to contain every language $L_{\mathrm{E},\Sigma}(\beta_i)$, it may not contain any single language $L_{\mathrm{E},\Sigma}(\alpha_i)$. The main idea of the construction is that this requirement is inherently contradictory, as we shall see that whenever a pattern δ can generate every language $L_{\mathrm{E},\Sigma}(\beta_i)$, then δ can generate almost all of the languages $L_{\mathrm{E},\Sigma}(\alpha_i)$ as well.

We now assume to the contrary that there is a pattern $\delta \in \mathrm{Pat}_\Sigma$ that is E-descriptive of L_Σ. As $\lambda \in L_\Sigma \subseteq L_{\mathrm{E},\Sigma}(\delta)$, δ cannot contain any terminals. Therefore, Theorem 2.4 permits us to describe all relevant inclusion relations through morphisms.

According to Theorem 2.4, for every $i \geq 0$, there is a morphism $\theta_i : X^* \to X^*$ such that $\theta_i(\delta) = \beta_i$, since $L_{\mathrm{E},\Sigma}(\delta) \supseteq L_{\mathrm{E},\Sigma}(\beta_i)$ holds by definition. We now choose an infinite sequence of morphisms $(\theta_i)_{i=0}^\infty$ such that for every $i \geq 0$,

1. $\theta_i(\delta) = \beta_i$, and
2. θ_i erases as many variables of δ as possible; i. e., for every morphism ρ with $\rho(\delta) = \theta_i(\delta) = \beta_i$,

$$|\{x \in \mathrm{var}(\delta) \mid \rho(x) = \lambda\}| \leq |\{x \in \mathrm{var}(\delta) \mid \theta_i(x) = \lambda\}|.$$

Such a sequence must exist, as $\mathrm{var}(\delta)$ is finite. Furthermore, we choose integers m, n such that θ_m and θ_{m+n} erase exactly the same variables of δ; i. e., for all $x \in \mathrm{var}(\delta)$, $\theta_m(x) = \lambda$ if and only if $\theta_{m+n}(x) = \lambda$. Again, this is possible due to $\mathrm{var}(\delta)$ being finite. Due to technical reasons and without loss of generality, we assume $m, n \geq 2$.

As we shall see, this choice allows us to modify θ_{m+n} in such a way that the resulting morphism maps δ to α_{m+1}, which (according to Lemma 5.25) leads to the desired contradiction. Our modification mostly targets those variables in $\mathrm{var}(\delta)$ that contain occurrences of x_{n-1} in their images under θ_{m+n}. To this end, we define

$$\begin{aligned}
\widehat{X} &:= \{x \in \mathrm{var}(\delta) \mid x_{n-1} \in \mathrm{var}(\theta_{m+n}(x))\},\\
\widehat{X}_L &:= \{x \in \widehat{X} \mid \theta_{m+n}(x) \text{ contains } x_{n-2}x_{n-1},\ x_{n-1}x_n \text{ or } x_{n-1}x_0 \text{ as a factor}\},\\
\widehat{X}_R &:= \{x \in \widehat{X} \mid \theta_{m+n}(x) \text{ contains } x_{n-1}x_{n-2},\ x_nx_{n-1} \text{ or } x_0x_{n-1} \text{ as a factor}\}.
\end{aligned}$$

In order to construct a well-defined morphism, we need to show that $\widehat{X}_R$ and $\widehat{X}_L$ form a partition of X_κ; as we shall see, $\widehat{X}_L$ contains exactly those variables that are mapped to occurrences of x_{n-1} in the left side of β_{m+n}, while $\widehat{X}_R$ contains those variables that are mapped to occurrences on the right side. Then we can use these variables as "anchors" for a modification of θ_{m+n} that permits us to obtain α_{n+1} from δ.

Our corresponding reasoning is based on the following insight:

Lemma 5.26. *For every $x \in \mathrm{var}(\delta)$, if $\theta_{m+n}(x)$ contains a variable x_i with $i < n$, then $\theta_{m+n}(x)$ also contains a variable x_j with $j \geq n$.*

Proof. First, recall that $\theta_{m+n}(\delta) = \beta_{m+n}$ and $(\mu^n \circ \theta_{m+n})(\delta) = \beta_m$ (cf. Lemma 5.23). Assume to the contrary that there is an $x \in \mathrm{var}(\delta)$ such that $\mathrm{var}(\theta_{m+n}(x)) \neq \emptyset$ and $\mathrm{var}(\theta_{m+n}(x)) \subseteq \{x_0, \ldots, x_{n-1}\}$. Note that for all $n \geq 0$,

$$\mu^n(x_i) = \begin{cases} \lambda & i < n, \\ x_{i-n} & i \geq n. \end{cases}$$

Therefore, $\mu^n(x_i) = \lambda$ if and only if $i < n$; and thus $(\mu^n \circ \theta_{m+n})(x) = \lambda$.

Moreover, for every $y \in \mathrm{var}(\delta)$, if $\theta_{m+n}(y) = \lambda$, then $(\mu^n \circ \theta_{m+n})(y) = \lambda$. But θ_m and θ_{m+n} erase exactly the same variables of δ. Thus, although $\mu^n \circ \theta_{m+n}$ erases more variables than θ_m, $(\mu^n \circ \theta_{m+n})(\delta) = \beta_m = \theta_m(\delta)$ holds, which is a contradiction to the second criterion in our choice of $(\theta_i)_{i=0}^{\infty}$. □

Note that this implies that, for all $x \in X_\kappa$, $|\theta_{m+n}(x)| \geq 2$. Now we can prove that $\widehat{X}_L$ and $\widehat{X}_R$ form a partition of X_κ:

Lemma 5.27. *$\widehat{X}_L \cup \widehat{X}_R = X_\kappa$ and $\widehat{X}_L \cap \widehat{X}_R = \emptyset$.*

Proof. To see that $\widehat{X}_L \cup \widehat{X}_R = X_\kappa$ must hold, recall the shape of β_{m+n}:

$$\begin{aligned} \beta_{m+n} =& ((\ldots(((\ldots((x_0)^2 x_1)^2 \ldots x_{n-2})^2 x_{n-1})^2 x_n)^2 \ldots)^2 x_{m+n})^2 \\ & (x_{m+n}(\ldots(x_n(x_{n-1}(x_{n-2}\ldots(x_1(x_0)^2)^2\ldots)^2)^2)^2\ldots)^2)^2. \end{aligned}$$

Due to Lemma 5.26, $|\theta_{m+n}(x)| \geq 2$ for each $x \in X_\kappa$. Thus, every $\theta_{m+n}(x)$ contains not only an occurrence of x_{n-1}, but at least one left or right neighbor. If some occurrence of x_{n-1} lies in the left half of β_{m+n}, its left neighbor is always an occurrence of x_{n-2} (recall that $n \geq 2$), and its right neighbor is either x_n or x_0. On the other hand, if it lies in the right half of β_{m+n}, its right neighbor is always x_{n-2}, and its left neighbor is either x_0 or x_n. Thus, if some $x \in X_\kappa$ is mapped to an occurrence of x_{n-1} in the left half of β_{m+n}, $\theta_{m+n}(x)$ contains a factor $x_{n-2}x_{n-1}$, $x_{n-1}x_n$ or $x_{n-1}x_0$, and $x \in \widehat{X}_L$. Likewise, if it is mapped to an occurrence in the right half, $\theta_{m+n}(x)$ contains $x_{n-1}x_{n-2}$, $x_n x_{n-1}$ or $x_0 x_{n-1}$, and $x \in \widehat{X}_R$. Therefore, $\widehat{X}_L \cup \widehat{X}_R = X_\kappa$.

In order to prove disjointness, we make another structural observation: We can safely assume that every variable in δ occurs at least twice – otherwise $L_{\mathrm{E},\Sigma}(\delta) = \Sigma^* \supset L_{\mathrm{E},\Sigma}(\alpha_0)$ would hold, and δ would not be E-descriptive of L_Σ according to Lemma 5.25. Thus, there is no variable x such that $x_{m+n}x_{m+n}$ is a factor of $\theta_{m+n}(x)$, as $x_{m+n}x_{m+n}$ occurs only once in β_{m+n}. This means that $x_{m+n}x_{m+n}$ forms an insurmountable barrier: For every occurrence of a variable from $\mathrm{var}(\delta)$, its image under θ_{m+n} lies either in the left or the right half of β_{m+n}. But if this image is longer than a single letter, the images of all occurrences of this variable must be mapped to the same side of β_{m+n}. According to Lemma 5.26, this is true for all variables of X_κ. Therefore, for every $x \in X_\kappa$, either $x \in \widehat{X}_L$ or $x \in \widehat{X}_R$ holds, which implies $\widehat{X}_L \cap \widehat{X}_R = \emptyset$. □

This permits us to define a morphism $\rho : X^* \to X^*$ through

$$\rho(x) := \begin{cases} (\widehat{\rho}_L \circ \theta_{m+n})(x) & \text{if } x \in \widehat{X}_L, \\ (\widehat{\rho}_R \circ \theta_{m+n})(x) & \text{if } x \in \widehat{X}_R, \\ \theta_{m+n}(x) & \text{otherwise,} \end{cases}$$

where the morphisms $\widehat{\rho}_L, \widehat{\rho}_R : X^* \to X^*$ are given by

$$\widehat{\rho}_L(x) := \begin{cases} y & \text{if } x = x_{n-1}, \\ x & \text{otherwise,} \end{cases} \qquad \widehat{\rho}_R(x) := \begin{cases} z & \text{if } x = x_{n-1}, \\ x & \text{otherwise.} \end{cases}$$

According to Lemma 5.27, the morphism ρ is well-defined, and, as to be proven next, $(\mu^{n-1} \circ \rho)(\delta) = \alpha_{m+1}$. Applying ρ to δ leads to

$$\begin{aligned} \rho(\delta) =& ((\dots(((\dots((x_0)^2 x_1)^2 \dots x_{n-2})^2 y)^2 x_n)^2 \dots)^2 x_{m+n})^2 \\ & (x_{m+n}(\dots(x_n(z(x_{n-2}\dots(x_1(x_0)^2)^2\dots)^2)^2)^2\dots)^2)^2, \end{aligned}$$

and, as $(m+n)-(n-1) = m+1$ and $\mu(x_i) = \lambda$ for every $i \leq n-2$, we obtain

$$\begin{aligned} (\mu^{n-1} \circ \rho)(\delta) &= ((\dots((y)^2 x_1)^2 \dots)^2 x_{m+1})^2 (x_{m+1}(\dots(x_1(z)^2)^2\dots)^2)^2 \\ &= \alpha_{m+1}. \end{aligned}$$

The morphism $\mu^{n-1} \circ \rho$ maps δ to α_{m+1}, and, thus, Theorem 2.4 immediately leads to $L_{\mathrm{E},\Sigma}(\delta) \supseteq L_{\mathrm{E},\Sigma}(\alpha_{m+1})$. Therefore, due to Lemma 5.25, the pattern δ cannot be E-descriptive of L_Σ. This contradiction concludes the proof of Theorem 5.18.

5.4.3 Proof of Corollary 5.20

The proof of Corollary 5.20 is based on the following technical lemma, that is given by Jiang et al. [51] in the context of their proof of Theorem 2.4:

Lemma 5.28 (Jiang et al. [51])**.** *Let Σ be an alphabet, $\Sigma \supseteq \{\mathtt{a}, \mathtt{b}\}$, and let $\alpha, \beta \in X^+$ be terminal-free patterns, $k{:=}|\beta|$. Let the morphism $\tau_k : X^* \to X^*$ be given by, for every $i \geq 0$,*

$$\tau_k(x_i) := \mathtt{a}\,\mathtt{b}^{ki+1}\,\mathtt{a}\;\mathtt{a}\,\mathtt{b}^{ki+2}\,\mathtt{a}\;\dots\;\mathtt{a}\,\mathtt{b}^{ki+k-1}\,\mathtt{a}\;\mathtt{a}\,\mathtt{b}^{ki+k}\,\mathtt{a}\,.$$

Then $\tau_k(\alpha) \in L_{\mathrm{E},\Sigma}(\beta)$ if and only if there exists a morphism $h : X^* \to X^*$ satisfying $h(\beta) = \alpha$.

Furthermore, we wish to point out that the patterns α_i and β_i, $i \geq 0$, referred to in the present section are defined in Section 5.4.2.

We prove Corollary 5.20 by giving a thin language $L^t_\Sigma \subset L_\Sigma$ such that for every $\delta \in \mathrm{Pat}_\Sigma$ with $L_{\mathrm{E},\Sigma}(\delta) \supseteq L^t_\Sigma$ and for infinitely many $i \geq 0$, there is a morphism $\theta_i : X^* \to X^*$ with $\theta_i(\delta) = \beta_i$. Then for every such δ, there is a $j \geq 0$ with $L_{\mathrm{E},\Sigma}(\delta) \supset L_{\mathrm{E},\Sigma}(\alpha_j) \supset L^t_\Sigma$.

Proof. Let $\mathtt{a}, \mathtt{b} \in \Sigma$ with $\mathtt{a} \neq \mathtt{b}$. For every $n \geq 1$, we define a substitution $\tau_n : X^* \to \Sigma^*$ by

$$\tau_n(x_i) := \mathtt{a}\,\mathtt{b}^{ni+1}\,\mathtt{a}\;\mathtt{a}\,\mathtt{b}^{ni+2}\,\mathtt{a}\;\dots\;\mathtt{a}\,\mathtt{b}^{ni+n-1}\,\mathtt{a}\;\mathtt{a}\,\mathtt{b}^{ni+n}\,\mathtt{a},$$

and we assume that τ_0 denotes the constant λ-function. We then define

$$L^t_\Sigma := \bigcup_{n \geq 0}^{\infty} \tau_n(\beta_n).$$

It is easy to see that L^t_Σ is properly thin, as for every $n \geq 0$, $|\tau_n(\beta_n)| < |\tau_{n+1}(\beta_{n+1})|$.

We assume to the contrary that there is a pattern $\delta \in \mathrm{Pat}_\Sigma$ that is E-descriptive of L_Σ^t. First note that – since $L_{\mathrm{E},\Sigma}(\alpha_i) \supset L_{\mathrm{E},\Sigma}(\alpha_{i+1}) \supset L_\Sigma \supset L_\Sigma^t$ for every $i \geq 0$ (see Lemma 5.24) – there is no $j \geq 0$ with $L_{\mathrm{E},\Sigma}(\delta) \supseteq L_{\mathrm{E},\Sigma}(\alpha_j)$ (as described by Lemma 5.25). Furthermore, as $\tau_0(\beta_0) = \lambda$, $\lambda \in L_\Sigma^t \subseteq L_{\mathrm{E},\Sigma}(\delta)$ holds, and therefore δ must be terminal-free.

According to Lemma 5.28, for every $\delta \in X^+$ and every $n \geq |\delta|$, $\tau_n(\beta_n) \in L_{\mathrm{E},\Sigma}(\delta)$ if and only if there is a morphism $\theta_n : X^* \to X^*$ such that $\theta_n(\delta) = \beta_n$. Furthermore, for every $m < n$ and the morphism μ introduced in Section 5.4.2, $(\mu^{n-m} \circ \theta_n)(\delta) = \mu^{n-m}(\beta_n) = \beta_{n-(m-n)} = \beta_m$ holds. Thus, there is an infinite sequence $(\theta_i)_{i=0}^\infty$ with $\theta_i(\delta) = \beta_i$ for all $i \geq 0$, which allows us to construct a morphism that maps δ to some α_j just as in the proof for Theorem 5.4.2. Thus, $L_{\mathrm{E},\Sigma}(\delta) \supset L_{\mathrm{E},\Sigma}(\alpha_j) \supset L_\Sigma^t$, and this contradicts our assumption of δ being E-descriptive of L_Σ^t. □

In the same way, we can also remove an arbitrary infinite number of words from L_Σ^t, as long as infinitely many words $\tau_n(\beta_n)$ remain.

5.5 Computing Descriptive Patterns

In addition to the question whether a set has a descriptive pattern, it is also of interest to ask whether such a pattern can be derived effectively, and if so, efficiently. Naturally, this question needs to specify for which kind of sets we consider.

5.5.1 Computing Descriptive Patterns for Finite Sets

In this section, we discuss the effective and efficient solvability of computing a pattern that is descriptive of a finite set. As for many questions on pattern languages, the first results are due to Angluin:

Theorem 5.29 (Angluin [4]). *Let Σ be an alphabet. There is an effective procedure which, given a finite $S \subseteq \Sigma^*$ as input, outputs a pattern α that is NE-descriptive of S.*

On the other hand, with an additional restriction that is used to circumvent the problem of the undecidable inclusion problem, NE-descriptive patterns cannot be computed effectively (assuming P $\neq$ NP):

Theorem 5.30 (Angluin [4]). *Let Σ be an alphabet with $|\Sigma| \geq 2$. If* P $\neq$ NP*, then there is no polynomial-time algorithm that, given a finite set $S \subset \Sigma^*$, finds a pattern δ of maximum possible length that is NE-descriptive of S.*

The situation for the E-case is a little bit less clear. As mentioned in Section 5.4, Jiang et al. [50] claim after the proof of Theorem 5.15 that this proof also provides an algorithm that, given a finite language S, computes a pattern that is E-descriptive of that language. But, even as the search space is finite for every S (as the proof only considers patterns of a restricted length), a general algorithm as described in [50] would still require an algorithm that solves the inclusion problem for E-pattern languages. There is no indication in [50] how such an algorithm could be implemented without using a procedure that decides the inclusion for E-pattern languages (cf. Salomaa [100]). Therefore, the following question has to be considered open:

Open Problem 5.31. *Let Σ be a finite alphabet, $|\Sigma| \geq 2$. Is there an effective procedure which, given a finite $S \subseteq \Sigma^*$ as input, outputs a pattern α that is E-descriptive of S?*

As we shall see, such an algorithm, even if it should exist, is probably not efficient (cf. Theorem 5.33). Of course, as inclusion is decidable for terminal-free E-pattern languages, we can answer this question positively for this restricted class:

Proposition 5.32. *For every Σ with $|\Sigma| \geq 2$ and every finite nonempty $S \in \mathrm{FIN}_\Sigma$, $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(S) \neq \emptyset$, and a $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(S)$ can be effectively computed.*

Proof. This can be shown using the same reasoning as in the proof of Theorem 8.1 by Jing et al. [50]. Let $S = \{w_1, \ldots, w_n\} \subset \Sigma^*$ for some $n \geq 1$ and denote

$$c_S := \sum_{i=1}^{n} |w_i|.$$

Our claim is that for every $\alpha \in X^+$ with $L_{\mathrm{E},\Sigma}(\alpha) \supset S$, there is a $\beta \in X^+$ with $L_{\mathrm{E},\Sigma}(\alpha) \supseteq L_{\mathrm{E},\Sigma}(\beta) \supseteq S$ and $|\beta| \leq c_S$. If $L_{\mathrm{E},\Sigma}(\alpha) \supset S$, there are morphisms $\phi_1, \ldots, \phi_n : X^* \to \Sigma^*$ with $\phi_i(\alpha) = w_i$ for $i \in \{1, \ldots, n\}$. Let $R := \{x \in \mathrm{var}(\alpha) \mid \phi_i(x) = \lambda \text{ for all } i\}$, define the morphism $\rho : X^* \to X^*$ through

$$\rho(x) := \begin{cases} \lambda & \text{if } x \in R, \\ x & \text{if } x \notin R, \end{cases}$$

and let $\beta := \rho(\alpha)$. It is easily seen that $|\beta| \leq c_S$ and $\phi_i(\beta) = (\phi_i \circ \rho)(\alpha) = \phi_i(\alpha) = w_i$ for every i. Thus, $L_{\mathrm{E},\Sigma}(\alpha) \supseteq L_{\mathrm{E},\Sigma}(\beta) \supset S$ holds.

As there are only finitely many terminal-free patterns (modulo renaming) of length at most c_S, there must be a pattern $\delta \in X^+$ with $|\delta| \leq c_S$, $L_{\mathrm{E},\Sigma}(\alpha) \supseteq L_{\mathrm{E},\Sigma}(\delta)$ and $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(S)$.

This also leads to the desired effective procedure that returns a $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(S)$: As an initial value, choose $\alpha := x_1$. In every step, try to find a pattern $\beta \in X^+$ with $L_{\mathrm{E},\Sigma}(\alpha) \supset L_{\mathrm{E},\Sigma}(\beta) \supseteq S$ and $|\beta| \leq c_S$. If such a pattern is found, let $\alpha := \beta$, and search again. If no such pattern is found, $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(S)$ must hold. As the search space is finite, and all involved inclusions problems are decidable (cf. Theorem 2.4), the whole procedure is effective. □

Similar to the case of NE-pattern languages, even if the problem can be solved effectively for the whole class of E-pattern languages, any such algorithm would not be efficient (unless P = NP):

Theorem 5.33. *If* P $\neq$ NP*, then there is no polynomial-time algorithm that, for any alphabet Σ and every finite set $S \subset \Sigma^*$ of words, computes a pattern that is* ePAT_Σ*-descriptive of S.*

Proof. We prove the contraposition of the theorem. Thus, we assume that there is an algorithm χ_δ computing, for any alphabet Σ, any finite set $S \subseteq \Sigma^*$ of words, a pattern that is ePAT_Σ-descriptive of S, and we shall use χ_δ to decide the inclusion problem for terminal-free E-pattern languages in polynomial time. Since this problem is NP-complete (see Corollary 5.5), we can conclude P = NP.

Let $\alpha, \beta \in X^*$ be any patterns. W. l. o. g., we assume that $\mathrm{var}(\alpha) \cap \mathrm{var}(\beta) = \emptyset$. Let Σ be an alphabet with $|\Sigma| = |\mathrm{var}(\alpha)| + |\mathrm{var}(\beta)|$, and let the morphism $r : (\mathrm{var}(\alpha) \cup \mathrm{var}(\beta))^* \to \Sigma^*$ be a renaming. This implies that there is an inverse morphism r^{-1} with $r^{-1}(r(\alpha)) = \alpha$ and $r^{-1}(r(\beta)) = \beta$, and $\mathrm{symb}(r(\alpha)) \cap \mathrm{symb}(r(\beta)) = \emptyset$. We define $S{:=}\{r(\alpha), r(\beta)\}$, and we use χ_δ to compute a pattern δ that is ePAT_Σ-descriptive of S. Since $\mathrm{symb}(r(\alpha)) \cap \mathrm{symb}(r(\beta)) = \emptyset$, δ is terminal-free. Furthermore, we can use the reasoning given by Jiang et al. [50] on their Theorem 8.1 to verify that $|\delta| \leq |r(\alpha)|+|r(\beta)|$. We can therefore use a polynomial-time algorithm $\chi_=$ (the existence of which is ensured by Theorem 5.6) to decide on whether $L_{\mathrm{E},\Sigma'}(\beta) = L_{\mathrm{E},\Sigma'}(\delta)$ for any alphabet Σ' with $|\Sigma'| \geq 2$. Since the question of whether $L_{\mathrm{E},\Sigma'}(\beta)$ equals $L_{\mathrm{E},\Sigma'}(\delta)$ does not depend on the actual size of Σ' and since $|\Sigma| \geq 2$, we may, w. l. o. g., define $\Sigma'{:=}\Sigma$.

We now show that $L_{\mathrm{E},\Sigma}(\beta) = L_{\mathrm{E},\Sigma}(\delta)$ if and only if $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$:

If $L_{\mathrm{E},\Sigma}(\alpha) \not\subseteq L_{\mathrm{E},\Sigma}(\beta)$, then, according to Theorem 2.4, there exists no morphism $\phi : X^* \to X^*$ satisfying $\phi(\beta) = \alpha$. Thus, there does not exist a substitution σ with $\sigma(\beta) = r(\alpha)$, since otherwise the morphism $\phi{:=}r^{-1} \circ \sigma$ would satisfy $\phi(\beta) = \alpha$. On the other hand, by definition, there exists a substitution τ with $\tau(\delta) = r(\alpha)$, and therefore $r(\alpha) \in L_{\mathrm{E},\Sigma}(\delta) \setminus L_{\mathrm{E},\Sigma}(\beta)$. This immediately implies $L_{\mathrm{E},\Sigma}(\beta) \neq L_{\mathrm{E},\Sigma}(\delta)$.

If $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$, then there exist substitutions σ, σ' with $\sigma(\beta) = r(\alpha)$ and $\sigma'(\beta) = r(\beta)$. Hence, $S \subseteq L_{\mathrm{E},\Sigma}(\beta)$. Furthermore, since δ is ePAT_Σ-descriptive of S, there exists a substitution τ with $\tau(\delta) = r(\beta)$. Thus, $r^{-1} \circ \tau(\delta) = \beta$, and therefore, according to Theorem 2.4, $L_{\mathrm{E},\Sigma}(\beta) \subseteq L_{\mathrm{E},\Sigma}(\delta)$. If we now assume to the contrary that $L_{\mathrm{E},\Sigma}(\beta) \neq L_{\mathrm{E},\Sigma}(\delta)$, then this implies $L_{\mathrm{E},\Sigma}(\beta) \subset L_{\mathrm{E},\Sigma}(\delta)$. Consequently, $L_{\mathrm{E},\Sigma}(\delta) \supset L_{\mathrm{E},\Sigma}(\beta) \supseteq S$. This is a contradiction to the assumption that δ is ePAT_Σ-descriptive of S. Hence, $L_{\mathrm{E},\Sigma}(\beta) = L_{\mathrm{E},\Sigma}(\delta)$.

Consequently, since $|r(\alpha)|+|r(\beta)|$ is polynomial in $|\alpha|+|\beta|$ and the runtimes of χ_δ and $\chi_=$ are polynomial in $|r(\alpha)| + |r(\beta)|$, we have a polynomial-time algorithm deciding the inclusion problem for the class of terminal-free E-pattern languages over any alphabet Σ with $|\Sigma| \geq 2$. Due to our initial remarks, this proves the theorem. □

Theorem 5.33 addresses a problem left open by Jiang et al. [50], and it provides a result that is, apart from the fact that it depends on an unbounded terminal alphabet, stronger than Angluin's corresponding statement on NE-descriptive patterns (cf. Theorem 5.30).

Since Theorem 5.33 can be proved using terminal-free patterns only, we can strengthen the corresponding result as follows:

Corollary 5.34. *If* P ≠ NP*, then there is no polynomial-time algorithm that, for any alphabet* Σ *and every finite set* $S \subset \Sigma^*$ *of words, computes a pattern that is* ePAT_Σ*-descriptive of* S*.*

5.5.2 E-Descriptive Patterns and the Equivalence Problem

For every class of pattern languages, the question whether a pattern is descriptive of a language is related to the inclusion problem, and, as the proof of Theorem 5.33 illustrates, to the equivalence problem. We consider the following question:

Open Problem 5.35. *Let* Σ *be a finite alphabet with* $|\Sigma| \geq 2$*.*

1. *Are there patterns* $\alpha, \beta \in \mathrm{Pat}_\Sigma$ *such that no pattern is E-descriptive of* $L_{\mathrm{E},\Sigma}(\alpha) \cup L_{\mathrm{E},\Sigma}(\beta)$*?*

2. *Assuming that for all* $\alpha, \beta \in \mathrm{Pat}_\Sigma$ *there is a pattern that is E-descriptive of* $L_{\mathrm{E},\Sigma}(\alpha) \cup L_{\mathrm{E},\Sigma}(\beta)$*, can such a pattern be obtained effectively?*

Purely from intuition, it seems plausible that such a E-descriptive pattern exists for all pairs of patterns, and in principle, the idea that it can be computed is not too improbable. On the other hand, the existence of such an algorithm D and the decidability of the equivalence problem are mutually exclusive:

Theorem 5.36. *Let* Σ *be a finite alphabet with* $|\Sigma| \geq 2$*. Assume there is an algorithm* D *that, given* $\alpha, \beta \in \mathrm{Pat}_\Sigma$*, outputs a pattern* $D(\alpha, \beta)$ *that is E-descriptive of* $L_{\mathrm{E},\Sigma}(\alpha) \cup L_{\mathrm{E},\Sigma}(\beta)$*. Then the equivalence problem for* ePAT_Σ *is undecidable.*

Proof. Let Σ be a finite alphabet with $|\Sigma| \geq 2$, assume such an algorithm D exists, and assume for the sake of contradiction that the equivalence problem for ePAT_Σ is decidable.

Then inclusion for ePAT_Σ is decidable in the following straightforward way: Given any two patterns $\alpha, \beta \in \mathrm{Pat}_\Sigma$, we compute $\delta = D(\alpha, \beta)$. By definition, δ is E-descriptive of $L_{\mathrm{E},\Sigma}(\alpha) \cup L_{\mathrm{E},\Sigma}(\beta)$.

We now prove that $L_{\mathrm{E},\Sigma}(\delta) = L_{\mathrm{E},\Sigma}(\beta)$ if and only if $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$. For the *only if* direction, assume $L_{\mathrm{E},\Sigma}(\delta) = L_{\mathrm{E},\Sigma}(\beta)$. As δ is E-descriptive of $L_{\mathrm{E},\Sigma}(\alpha) \cup L_{\mathrm{E},\Sigma}(\beta)$, $L_{\mathrm{E},\Sigma}(\delta) \supseteq L_{\mathrm{E},\Sigma}(\alpha) \cup L_{\mathrm{E},\Sigma}(\beta)$, and thus, $L_{\mathrm{E},\Sigma}(\beta) \supseteq L_{\mathrm{E},\Sigma}(\alpha)$.

For the *if* direction, if $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$, then $L_{\mathrm{E},\Sigma}(\alpha) \cup L_{\mathrm{E},\Sigma}(\beta) = L_{\mathrm{E},\Sigma}(\beta)$, and if $L_{\mathrm{E},\Sigma}(\delta)$ is E-descriptive of $L_{\mathrm{E},\Sigma}(\beta)$, $L_{\mathrm{E},\Sigma}(\delta) = L_{\mathrm{E},\Sigma}(\beta)$ must hold.

Therefore, deciding $L_{\mathrm{E},\Sigma}(\delta) = L_{\mathrm{E},\Sigma}(\beta)$ allows to decide $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$. As the inclusion problem for ePAT_Σ is undecidable (cf. Theorem 3.3), the claim follows. □

As a contraposition of this result, if equivalence for ePAT_Σ is decidable, no such algorithm exists, either because there are patters $\alpha, \beta \in \mathrm{Pat}_\Sigma$ such that no pattern is E-descriptive of $L_{\mathrm{E},\Sigma}(\alpha) \cup L_{\mathrm{E},\Sigma}(\beta)$, or because, although there is always such a descriptive pattern, it can not be found effectively.

Therefore, research in this direction might offer an alternative angle to attack the equivalence problem for E-pattern languages, which is still open (and considered the central open problem for pattern languages). Although it has been conjectured that this problem is decidable (e. g. by Salomaa [96], Ohlebusch and Ukkonen [79]), the proof methods proposed by Ohlebusch and Ukkonen have been disproven (cf. Reidenbach [87], Freydenberger and Reidenbach [32]).

Chapter 6

Inferring Descriptive Generalizations

6.1 On Descriptive Generalizations

While Chapter 5 is mostly concerned with the *existence* of descriptive patterns, the present chapter expands on the topic of *finding* descriptive patterns for certain languages. As mentioned in the introduction to Chapter 5, descriptive patterns have applications (or at least potential applications) in inductive inference and pattern discovery.

This chapter revisits and expands this notion. More specifically, we already mentioned in Section 5.1 that Gold's model of inductive inference, the model of language identification in the limit from positive data (cf. Section 6.2.1), requires the learner to derive exact descriptions of the target languages, and that it can be worthwhile to relax this requirement to a learner that merely approximates the languages using easily interpretable approximations.

In the present chapter, we introduce and study such a variant of Gold's model, where the requirement of exact language identification is dropped and replaced with that of inference of patterns that are descriptive of the target languages (with respect to a class $\mathrm{PAT}_\star$ of pattern languages). By definition, descriptive patterns generate supersets of the languages of which they are descriptive; hence, this approach does not yield an arbitrary approximation of a language, but rather a *generalization*. Moreover, as no other pattern language (in $\mathrm{PAT}_\star$) provides a stricter generalization, descriptive patterns can be regarded as *least generalizations*.

Since descriptiveness captures a natural understanding of patterns providing a desirable generalization of languages and, furthermore, descriptive patterns can be used to devise Gold-style learners precisely identifying classes of pattern languages from positive data, this concept has been thoroughly investigated (see, e. g., Angluin [4]).

In contrast to Chapter 5 (and most established definitions of descriptiveness in the literature), we do not restrict our view to the full class of E- or NE-pattern languages as the class $\mathrm{PAT}_\star$ of admissible pattern languages, but allow the class $\mathrm{PAT}_\star$ to be chosen arbitrarily.

To summarize the proposed model of inference, we consider a learner that reads a positive presentation of a language and, after having seen a new input word, outputs a pattern, the so-called *hypothesis*. We then say that, for a class $\mathcal{L}$ of languages and a class $\mathrm{PAT}_\star$ of pattern languages, the learner $\mathrm{PAT}_\star$-*descriptively generalizes* $\mathcal{L}$ if and only

if, for every positive presentation of every language $L \in \mathcal{L}$, the sequence of hypotheses produced by the learner converges to a pattern δ that is descriptive of L with respect to the class $\mathrm{PAT}_\star$. A more formal definition of the model is given in Section 6.2.2.

The main difference between descriptive generalization and related approaches (see, e. g., Arimura et al. [7], Mukouchi [73], Kobayashi and Yokomori [56, 57] and, indirectly, Jain et al. [48]) is that we have a distinct split between a class $\mathcal{L}$ of languages to be inferred and an arbitrary class $\mathrm{PAT}_\star$ of pattern languages determining the set of admissible hypotheses. This leads to a compact and powerful model that yields interesting insights into the question of to which extent the generalizability of $\mathcal{L}$ depends on properties of $\mathcal{L}$ or of $\mathrm{PAT}_\star$. We briefly discuss this topic in Section 6.2.3, and we mention in Section 6.2.4 that descriptive generalization can be interpreted as a natural instance of a very general and simple inference model which, to the best of the author's knowledge, has not been considered so far.

In Section 6.3, we investigate the model for a fixed and rich class $\mathrm{PAT}_\star$, namely the class of *terminal-free E-pattern languages*. Our studies reveal that, for this choice of $\mathrm{PAT}_\star$, descriptive generalization and inductive inference from positive data are incomparable, and they show that there are major and natural classes of formal languages that can be descriptively generalized according to the proposed model, but not precisely inferred in Gold's model. Technically, the decision to focus on terminal-free E-pattern languages leads to a number of substantial combinatorial challenges for pattern languages, and we present various respective insights and tools of intrinsic interest. Note that almost all results in this section can be immediately adapted to all classes of E-pattern languages for which inclusion is characterized by the existence of a terminal-preserving morphism between the patterns (for some examples, see the list in Section 3.3, after Theorem 3.5).

6.2 Inferring Descriptive Generalizations

In the present section, we first discuss Gold's model of language identification in the limit from positive data (cf. Gold [39]), and provide a short overview of some related previous results. After that, we formally introduce the notion of inferring descriptive generalizations, establish some of its basic properties (mainly by characterizing, for any class of pattern languages determining the set of valid hypotheses, those indexed families that can be generalized in the proposed model) and, finally, present a much more general inference paradigm that captures the essence of this new approach.

6.2.1 Inductive Inference in the Limit from Positive Data

This section introduces Gold's model of language identification in the limit from positive data, as well as some notions that we shall use in our definition of descriptive generalization.

Let $\mathcal{L}$ be a class of languages over some alphabet A. Then $\mathcal{L}$ is said to be indexable provided that there exists an indexing $(L_i)_{i=0}^{\infty}$ of languages L_i such that, first, $\mathcal{L} = \{L_i \mid i \geq 0\}$ and, second, there exists a total computable function χ which uniformly decides the membership problem for $(L_i)_{i=0}^{\infty}$ – i. e., for every $w \in A^*$ and for every $i \geq 0$, $\chi(w, i) = 1$ if and only if $w \in L_i$. In this case, we call $\mathcal{L} = (L_i)_{i=0}^{\infty}$ an *indexed family (of recursive languages)*. Of course, in this notation for an indexed family (which conforms

with the use in the literature) the equality symbol "=" does not refer to an equality in the usual sense, but is merely a symbol indicating that $\mathcal{L}$ contains all languages in $(L_i)_{i=0}^{\infty}$ and vice versa.

Example 6.1. *The class of context-free languages is indexable. An appropriate indexing $(L_i)_{i=0}^{\infty}$ can be constructed in a straightforward manner by defining a computable bijection f between the set of natural numbers and the set of all context-free grammars, while the corresponding computable function χ can be derived from f and any parsing algorithm for context-free grammars.*

Likewise, it is easy to see that most natural classes of pattern languages (be it E-pattern languages, NE-pattern languages, the full class of pattern languages, or any of its subclasses that arises from a decidable set of patterns) are indexable, as are most of the classes commonly studied in formal language theory (e. g., regular and context-sensitive languages). ◇

For any alphabet Σ and any nonempty language $L \subseteq \Sigma^*$, we call a total function $t : \mathbb{N}_0 \to \Sigma^*$ a *text* of L if and only if it satisfies $\{t(i) \mid i \geq 0\} = L$. Moreover, for every text t and every $n \geq 0$, t^n encodes the first n values of t in a single string, i. e., $t^n{:=}t(1)\nabla t(2)\nabla t(3)\nabla \ldots \nabla t(n)$ with $\nabla \notin \Sigma$; additionally, we define $t[n]{:=}\{t(i) \mid i \leq n\}$. Finally, $\text{text}(L)$ denotes the set of all (computable and non-computable, repetitive and non-repetitive) texts of a language L.

Let $\mathcal{L} = (L_i)_{i=0}^{\infty}$ be an indexed family of nonempty languages over an alphabet Σ. Then $\mathcal{L}$ is *inferrable (from positive data)* (or *learnable*) if and only if there exists a computable function $S : (\Sigma \cup \{\nabla\})^* \to \mathbb{N}_0$ such that, for every $i \geq 0$ and for every $t \in \text{text}(L_i)$, $S(t^n)$ is defined for every $n \geq 0$, and there is an $m \geq 0$ with $L_{S(t^m)} = L_i$ and $S(t^n) = S(t^m)$ for every $n \geq m$.

We call S a *(learning) strategy* and, for every $n \geq 0$, $S(t^n)$ a *hypothesis* of S. The notation LIM-TEXT refers to the class of all classes of languages that are inferrable from positive data.

Note that this concept of learnability only requires the learner to converge towards a correct hypothesis in finite time. While a learning strategy has to produce a hypothesis in every step, it is not required to decide that this hypothesis is correct (or even consistent with the available data). Hence, although it is certain that, for every language in the class, there is a point at which the strategy reaches a correct hypothesis without changing that hypothesis, we cannot recognize this point.

Furthermore, similarly to the notion of computability[1], the mere existence of such an effective learning strategy does not guarantee that this strategy can be found, or that it is efficient.

An important topic in [39] are the so-called *superfinite* classes of formal languages,

[1]The author considers the following quote by Angluin [6] an apt commentary on the similarity of inductive inference and computability theory:

> Inductive Inference [..] is to computational learning theory roughly as computability theory is to complexity and analysis of algorithms. Inductive Inference and computability theory are historically prior to and part of their polynomially-obsessed younger counterparts, share a body of techniques from recursion theory, and are a source of potent ideas and analogies in their respective fields.

i. e., those classes that contain all finite languages (over some fixed alphabet) and at least one infinite language. Gold proved the following negative result:

Theorem 6.2 (Gold [39]). *Let $\mathcal{L} = (L_i)_{i=0}^{\infty}$ be an indexable class of languages that contains all finite and at least one infinite language. Then $\mathcal{L} \notin$* LIM-TEXT.

As the class of regular languages is superfinite, this proves that no level of the Chomsky hierarchy is inferrable from positive data. This was taken as an indication that no rich and natural class of formal languages is learnable in this model, until Angluin proved (more than a decade later) the following result:

Theorem 6.3 (Angluin [4]). *For every alphabet Σ,* $\mathrm{nePAT}_\Sigma \in$ LIM-TEXT.

In addition to this, the following characterization of languages that can be inferred in the limit from positive data is due to Angluin:

Theorem 6.4 (Angluin [5]). *An indexed family $\mathcal{L} = (L_i)_{i=0}^{\infty}$ of nonempty recursive languages is in* LIM-TEXT *if and only if there exists an effective procedure which, for every $j \geq 0$, enumerates a set T_j such that*

- *T_j is finite,*
- *$T_j \subseteq L_j$, and*
- *there does not exist a $j' \geq 0$ with $T_j \supseteq L_{j'} \supset L_j$.*

In contrast to the situation for NE-pattern languages, the learnability of E-pattern languages was open for two decades, apart from the following result on unary and infinite alphabets:

Proposition 6.5 (Mitchell [71]). *Let Σ be an alphabet with $|\Sigma| \in \{1, \infty\}$. Then* $\mathrm{ePAT}_\Sigma \in$ LIM-TEXT.

But as for many other problems on pattern languages, the most interesting and most difficult alphabet sizes are those in the (considerable) gap between 1 and ∞. Finally, Reidenbach proved the following negative results:

Theorem 6.6 (Reidenbach [86]). *Let Σ be an alphabet with $|\Sigma| = 2$. Then* $\mathrm{ePAT}_\Sigma \notin$ LIM-TEXT.

Theorem 6.7 (Reidenbach [88]). *Let Σ be an alphabet with $|\Sigma| \in \{3, 4\}$. Then* $\mathrm{ePAT}_\Sigma \notin$ LIM-TEXT.

For all larger finite alphabets, the learnability of E-pattern languages remains open. In contrast to this, the classes of terminal-free E-pattern languages show the following curious discontinuity:

Theorem 6.8 (Reidenbach [88]). *Let Σ be an alphabet. Then* $\mathrm{ePAT}_{\mathrm{tf},\Sigma} \in$ LIM-TEXT *if and only if $|\Sigma| \neq 2$.*

As we shall see, the phenomena behind Theorem 6.8 also affect our reasoning in the present chapter.

Further results on inductive inference, in particular from positive data, and pointers to related learning models can be found in the surveys provided by Reidenbach [85] and by Ng and Shinohara [76].

6.2.2 The Inference Paradigm

Building on (some of) the definitions we introduced in Section 6.2.1, we formalize the explanations on the model given in Section 6.1 as follows:

Let $\mathcal{L}$ be a class of nonempty languages over an alphabet Σ, and let $\mathrm{PAT}_{\star,\Sigma}$ be a class of NE-pattern languages or a class of E-pattern languages over Σ. Then $\mathcal{L}$ is $\mathrm{PAT}_{\star,\Sigma}$-*descriptively generalizable* (or, if $\mathrm{PAT}_{\star,\Sigma}$ is understood, *(descriptively) generalizable* for short) if and only if there exists a computable function $S : (\Sigma \cup \{\nabla\})^* \to (\Sigma \cup X)^+$ such that, for every $L \in \mathcal{L}$ and for every $t \in \mathrm{text}(L)$, $S(t^n)$ is defined for every $n \geq 0$, and there is a $\delta \in (\Sigma \cup X)^+$ with $\delta \in D_{\mathrm{PAT}_{\star,\Sigma}}(L)$ and there is an $m \geq 0$ with $S(t^n) = \delta$ for every $n \geq m$.

We call S a *(generalization) strategy* and, for every $n \geq 0$, $S(t^n)$ a *hypothesis* of S. The notation $\mathrm{DG}_{\mathrm{PAT}_{\star,\Sigma}}$ refers to the class of all classes of languages that are $\mathrm{PAT}_{\star,\Sigma}$-descriptively generalizable.

Consequently, and as already mentioned in Section 6.1, we have an inference model where the class to be inferred and the *hypothesis space* (we shall use this term in a rather informal manner for both the class $\mathrm{PAT}_{\star,\Sigma}$ and any set $\mathrm{Pat}_\star$ of patterns satisfying $\mathrm{PAT}_{\star,\Sigma} = \{L_\Sigma(\alpha) \mid \alpha \in \mathrm{Pat}_\star\}$) are entirely different objects. The author feels that this feature precisely reflects the underlying motivation as outlined in Section 6.1, and it establishes the difference of the proposed approach to a number of related models.

6.2.3 Fundamental Insights into the Model

We now discuss some basic properties of descriptive generalization without considering a specific class of pattern languages determining the hypothesis space. This discussion is provided solely to give a more extensive view of the model, and to provide a wider context for the results presented in Section 6.3. The proof of Theorem 6.12 in this section and its extension Theorem 6.14 in the following section are due to Reidenbach and not part of the present thesis. These proofs can be found in [34] (and the upcoming full version of [34]).

At first glance, the definitions of descriptive generalization and of the LIM-TEXT model are closely related, and our first observation states that they are indeed equivalent if they are applied to any class of pattern languages:

Proposition 6.9. *Let* $\mathrm{PAT}_{\star,\Sigma}$ *be a class of pattern languages. Then* $\mathrm{PAT}_{\star,\Sigma} \in$ LIM-TEXT *if and only if* $\mathrm{PAT}_{\star,\Sigma} \in \mathrm{DG}_{\mathrm{PAT}_{\star,\Sigma}}$.

Proof. Directly from the definitions of LIM-TEXT and $\mathrm{DG}_{\mathrm{PAT}_{\star,\Sigma}}$. □

While descriptive generalization and inductive inference from positive data, thus, seem to be very similar, there are major differences between these two models. In fact, there are classes that can be descriptively generalized, although neither the class nor the hypothesis space can be exactly inferred from positive data:

Proposition 6.10. *There exists a class* $\mathcal{L}$ *of languages and a class* $\mathrm{PAT}_{\star,\Sigma}$ *of pattern languages satisfying* $\mathcal{L} \notin$ LIM-TEXT, $\mathrm{PAT}_{\star,\Sigma} \notin$ LIM-TEXT, *and* $\mathcal{L} \in \mathrm{DG}_{\mathrm{PAT}_{\star,\Sigma}}$.

Proof. The statement follows from Corollaries 6.27 and 6.32 in Section 6.3 and the fact that $\mathrm{ePAT}_{\mathrm{tf},\Sigma} \notin$ LIM-TEXT for $|\Sigma| = 2$ (cf. Theorem 6.8). □

Since the definition of descriptive generalization allows any class of pattern languages to be chosen as a hypothesis space, we can even devise a maximally powerful (yet utterly useless) generalization strategy:

Proposition 6.11. *Let Σ be an alphabet. There exists a class $\mathrm{PAT}_{\star,\Sigma}$ of pattern languages such that every class $\mathcal{L}$ of languages over Σ satisfies $\mathcal{L} \in \mathrm{DG}_{\mathrm{PAT}_{\star,\Sigma}}$.*

Proof. Let $\mathrm{PAT}_{\star,\Sigma} := \{L_{\mathrm{E},\Sigma}(x_1)\}$. Since x_1 is $\mathrm{PAT}_{\star,\Sigma}$-descriptive of every language $L \subseteq \Sigma^*$, a strategy S that constantly outputs x_1 generalizes $\mathcal{L}$. □

Obviously, the substantial gap between the LIM-TEXT model and descriptive generalization illustrated by Proposition 6.11 is based on a proof that uses a trivial notion of descriptiveness. In Section 6.3, we shall demonstrate that there are similarly deep differences between both models if a natural and nontrivial class of pattern languages, namely $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$, is used as admissible hypotheses for the generalization process.

The main result of the present section is the following characterization of descriptively generalizable indexed families of recursive languages. While the model of descriptive generalizability as well as our studies in Section 6.3 consider descriptive generalizations of arbitrary classes of languages, this restriction facilitates an interesting comparison of this result to Angluin's characterization of those indexed families that are inferrable in the LIM-TEXT model (see Angluin [5]). It is also worth noting that the subsequent argument cannot be based on strong insights into the descriptiveness of patterns, since we deal with arbitrary classes of pattern languages.

Theorem 6.12. *Let Σ be an alphabet, let $\mathcal{L} = (L_i)_{i=0}^{\infty}$ be an indexed family of nonempty recursive languages over Σ, and let $\mathrm{PAT}_{\star,\Sigma}$ be a class of pattern languages. $\mathcal{L} = (L_i)_{i=0}^{\infty} \in \mathrm{DG}_{\mathrm{PAT}_{\star,\Sigma}}$ if and only if there are effective procedures d and f satisfying the following conditions:*

(i) For every $i \geq 0$, there exists a $\delta_{d(i)} \in D_{\mathrm{PAT}_{\star,\Sigma}}(L_i)$ such that d enumerates a sequence of patterns $d_{i,0}, d_{i,1}, d_{i,2}, \ldots$ satisfying, for all but finitely many $j \in \mathbb{N}_0$, $d_{i,j} = \delta_{d(i)}$.

(ii) For every $i \geq 0$, f enumerates a finite set $F_i \subseteq L_i$ such that, for every $j \in \mathbb{N}_0$ with $F_i \subseteq L_j$, if $\delta_{d(i)} \notin D_{\mathrm{PAT}_{\star,\Sigma}}(L_j)$, then there is a $w \in L_j$ with $w \notin L_i$.

The author wishes to stress again that the proof of this theorem (as presented in [34]) is due to Reidenbach, and therefore omitted from the present thesis.

We shall briefly discuss in Section 6.2.4 how some observations on Theorem 6.12 and its proof can be used to define a more general model of inference.

6.2.4 A More General View

While an application of Theorem 6.12 might require profound knowledge on the descriptiveness of patterns, a closer look confirms our above remark that the actual characterization and its proof do not at all. More precisely, neither the Theorem nor the reasoning in its proof deal with the properties of the descriptive patterns $\delta_{d(i)}$, $i \geq 0$, but they merely make use of a notion of the *validity* of a hypothesis for a given language, i. e., a hypothesis is acceptable for a language if it is descriptive, but we do not check for

descriptiveness. This view is quite convenient to study the difference between descriptive generalization and inductive inference from positive data. In the LIM-TEXT model when applied to indexed families, a hypothesis i – i. e., the index of the language L_i – is valid for a language L_j, $j \neq i$, if and only if the hypothesis j is valid for the language L_i (if and only if $L_i = L_j$). In the model of descriptive generalization, this symmetry does not necessarily exist, as demonstrated by the following example:

Example 6.13. *Let* $\Sigma{:=}\{\mathtt{a}, \mathtt{b}\}$. *Let*

$$\begin{aligned} L_1 &:= \{\mathtt{ababa}, \mathtt{babab}\}, \text{ and} \\ L_2 &:= \{\mathtt{ababa}, \mathtt{babab}, \mathtt{abaaba}\}. \end{aligned}$$

We state without proof that $\delta_1{:=}x_1\,\mathtt{abab}\,x_2$ *is* ePAT_Σ*-descriptive of* L_1 *and* $\delta_2{:=}x_1x_2x_1x_2x_1$ *is* ePAT_Σ*-descriptive of* L_2. *While* δ_2 *is also* ePAT_Σ*-descriptive of* L_1, δ_1 *is not* ePAT_Σ*-descriptive of* L_2. *Hence, a strategy* S *that* ePAT_Σ*-descriptively generalizes a class including* L_1 *and* L_2 *can output* δ_1 *or* δ_2 *when reading a text for* L_1, *but it must not output* δ_1 *when reading a text for* L_2. ◇

Referring to this phenomenon and restricted to indexed families, we can now give a much more general model of inference than the one of descriptive generalization, and we can still characterize those indexed families that can be inferred according to this model in exactly the same way as we have done in Theorem 6.12. Hence, let $\mathcal{L} = (L_i)_{i=0}^{\infty}$ be an indexed family. Furthermore, for any $i \geq 0$, let HYP be a function that maps i to a subset of $\mathbb{N}_0$ that consists of all *valid hypotheses* for L_i. Here it is important to note that the numbers in $\mathrm{HYP}(i)$ do normally not refer to indices of the indexed family $\mathcal{L} = (L_i)_{i=0}^{\infty}$; e. g., in the model of descriptive generalization they would stand for indices in an arbitrary enumeration of a set of patterns. We then say that $\mathcal{L} = (L_i)_{i=0}^{\infty}$ is *inductively inferrable with hypotheses validity relation* HYP if and only if there exists a computable function $S : (\Sigma \cup \{\nabla\})^* \to \mathbb{N}_0$ such that, for every $i \geq 0$ and for every $t \in \mathrm{text}(L_i)$,

1. $S(t^n)$ is defined for every $n \geq 0$ and
2. there is a $j \in \mathrm{HYP}(i)$ and there is an $m \geq 0$ with $S(t^n) = j$ for every $n \geq m$.

Our notion of descriptive generalization demonstrates that there are natural instances of the model of inductive inference with hypotheses validity relation HYP. Nevertheless, to the best of the author's knowledge, its properties have not been explicitly studied so far.

As announced above, we now rephrase Theorem 6.12 so that it characterizes those indexed families that are inductively inferrable with hypotheses validity relation HYP:

Theorem 6.14. *Let* Σ *be an alphabet, let* $\mathcal{L} = (L_i)_{i=0}^{\infty}$ *be an indexed family of nonempty languages over* Σ, *and let* $\mathrm{HYP} : \mathbb{N}_0 \to \mathcal{P}(\mathbb{N}_0)$ *be a function.* $\mathcal{L} = (L_i)_{i=0}^{\infty}$ *is inductively inferrable with hypotheses validity relation* HYP *if and only if there are effective procedures* h *and* f *satisfying the following conditions:*

(i) For every $i \geq 0$, *there exists a* $\eta_i \in \mathrm{HYP}(i)$ *such that* h *enumerates a sequence of natural numbers* $i_0, i_1, i_2, \ldots$ *satisfying, for all but finitely many* $k \in \mathbb{N}_0$, $i_k = \eta_i$.

(ii) For every $i \geq 0$, f enumerates a finite set $F_i \subseteq L_i$ such that, for every $j \in \mathbb{N}_0$ with $F_i \subseteq L_j$, if $\eta_i \notin \mathrm{HYP}(j)$, then there is a $w \in L_j$ with $w \notin L_i$.

Proof (Reidenbach). Minor and straightforward editing of the proof of Theorem 6.12 – mainly substituting h for d, i_k for $d_{i,k}$, η_i for $\delta_{d(i)}$, and $\mathrm{HYP}(i)$ for $D_{\mathrm{PAT}_{\star,\Sigma}}(L_i)$ – turns it into a reasoning suitable for Theorem 6.14. □

To conclude this section on basic properties of the proposed model, we note that descriptive generalization can alternatively be interpreted as inductive inference of classes of pattern languages from *partial texts*. Hence, we can understand any language L_i as a tool to define texts that do not contain all words in $L(\delta_{d(i)})$, but nevertheless can be used to infer $\delta_{d(i)}$. Within the scope of this chapter, we do not explicitly discuss such a view, but the author expects that it might be a worthwhile topic for further studies. He anticipates that its analysis might involve substantial conceptual challenges that cannot be solved using established insights into related approaches (see Fulk and Jain [37]).

6.3 Inferring $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-Descriptive Patterns

We now study the model of descriptive generalizability for a fixed hypothesis space, namely the class $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$. The decidability of the inclusion problem for this class (see Theorem 2.4) allows us to develop a set of powerful tools.

This section is divided into three parts. In the first part, we consider some questions on the existence of $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive patterns for various classes of languages and develop a set of tools in order to simplify proofs on the existence and nonexistence of $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive patterns.

The second part deals with a generalization strategy that is based on the procedure that is described in Proposition 5.32, which the author deems so natural that we call it the canonical strategy Canon for $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive generalizations. Most importantly, we give a characterization of the class $\mathcal{TSL}_\Sigma$ of languages that can be descriptively generalized with Canon.

In the final part of this section, we examine the relationship of various classes of languages to $\mathcal{TSL}_\Sigma$ in order to gain further insights into $\mathrm{DG}_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}$ and the power of Canon.

6.3.1 Basic Tools

Before we proceed to an examination of $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive generalization in the next part of this section, we develop some tools and techniques that simplify the work with $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive patterns, and gather some results on the existence and nonexistence of such patterns for some classes of languages. We begin with the following result:

Lemma 6.15. *Let Σ be an alphabet with $|\Sigma| \geq 2$, and let $L_1, L_2 \subseteq \Sigma^*$ with $L_1 \supseteq L_2$. If there is a $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L_2)$ with $L_{\mathrm{E},\Sigma}(\delta) \supseteq L_1$, then $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L_1)$.*

Proof. Assume there is a $\gamma \in X^+$ with $L_{\mathrm{E},\Sigma}(\delta) \supset L_{\mathrm{E},\Sigma}(\gamma) \supseteq L_1$. Due to $L_1 \supseteq L_2$, this would imply $L_{\mathrm{E},\Sigma}(\delta) \supset L_{\mathrm{E},\Sigma}(\gamma) \supseteq L_2$ and contradict $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L_2)$. Therefore, δ is descriptive of L_1. □

This observation might seem to be elementary, but together with Lemma 6.18, it forms the fundament of the proof of almost every result in this section. The technical base of that Lemma derives from a phenomenon that often arises when dealing with ePAT$_{\mathrm{tf},\Sigma}$-descriptive patterns. We consider the following example:

Example 6.16. *Let* $\Sigma{:=}\{\mathtt{a},\mathtt{b}\}$ *and let*

$$\begin{aligned}L_1&{:=}\{\mathtt{a}^2\},\\ L_2&{:=}\{(\mathtt{a}\,\mathtt{b}^1\,\mathtt{a}\;\mathtt{a}\,\mathtt{b}^2\,\mathtt{a}\ldots\mathtt{a}\,\mathtt{b}^n\,\mathtt{a})^2 \mid n\geq 2\},\\ L_3&{:=}L_{\mathrm{E},\Sigma}(x_1^2)\setminus\{\mathtt{a}^2,\mathtt{b}^2\}.\end{aligned}$$

It is easy to see that all three languages are included in $L_{\mathrm{E},\Sigma}(x_1^2)$*. However, in addition to this, for every* $\alpha\in X^+$ *with* $L_{\mathrm{E},\Sigma}(\alpha)\supseteq L_i$ *(with* $1\leq i\leq 3$*),* $L_{\mathrm{E},\Sigma}(\alpha)\supseteq L_{\mathrm{E},\Sigma}(x_1^2)$ *holds as well. For* L_1*, this is obvious. For* L_2*, assume that* $L_{\mathrm{E},\Sigma}(\alpha)\supseteq L_2$ *for some* $\alpha\in X^+$*, let* $n{:=}|\operatorname{var}(\alpha)|$ *and* $w=(\mathtt{a}\,\mathtt{b}^1\,\mathtt{a}\;\mathtt{a}\,\mathtt{b}^2\,\mathtt{a}\ldots\mathtt{a}\,\mathtt{b}^n\,\mathtt{a})^2\in L_2$*, and choose any morphism* ϕ *with* $\phi(\alpha)=w$*. As* w *contains* n *distinct factors of the form* $\mathtt{a}\,\mathtt{b}^+\,\mathtt{a}$*, each occurring exactly twice, there must be an* $x\in\operatorname{var}(\alpha)$ *that contains at least one complete occurrence of such a segment, which implies* $|\alpha|_x\in\{1,2\}$*. In both cases, we can construct a morphism* ψ *with* $\psi(\alpha)=x_1^2$ *(by mapping* x *to* x_1 *or* x_1^2 *and erasing all other variables), which (according to Theorem 2.4) leads to* $L_{\mathrm{E},\Sigma}(\alpha)\supseteq L_{\mathrm{E},\Sigma}(x_1^2)$*. Finally, as* $L_3\supset L_2$*, this also proves the claim for* L_3*.*

As $L_{\mathrm{E},\Sigma}(x_1^2)$ *and all three* L_i *have exactly the same superpatterns, we are able to conclude that, for every* $i\in\{1,\ldots,3\}$*,* $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L_{\mathrm{E},\Sigma}(x_1^2))=D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L_i)$*. Although the four languages might seem rather different, they have exactly the same sets of* ePAT$_{\mathrm{tf},\Sigma}$*-descriptive patterns.* ◇

When generalizing languages using ePAT$_{\mathrm{tf},\Sigma}$-descriptive patterns, every language has a certain superset that is covered by every descriptive generalization of this language, and cannot be avoided. In order to formalize this line of reasoning (and in order to use this phenomenon), we introduce the set of *superpatterns* $\operatorname{Super}(L)$, and the *superpattern hulls* $\text{S-Hull}_\Sigma(L)$, which are defined as

$$\begin{aligned}\operatorname{Super}(L)&{:=}\{\alpha\in X^+ \mid \text{for every } w\in L, \text{ there is a morphism } \phi \text{ with } \phi(\alpha)=w\},\\ \text{S-Hull}_\Sigma(L)&{:=}\bigcap_{\alpha\in\operatorname{Super}(L)} L_{\mathrm{E},\Sigma}(\alpha)\end{aligned}$$

for all alphabets Σ,Σ' and any language $L\subseteq(\Sigma')^*$. Note that, by Theorem 2.4, for every pair of patterns $\alpha,\beta\in X^+$ and every Σ with $\Sigma\geq 2$, the following three conditions are equivalent:

1. $L_{\mathrm{E},\Sigma}(\alpha)\subseteq L_{\mathrm{E},\Sigma}(\beta)$,
2. $\beta\in\operatorname{Super}(L_{\mathrm{E},\Sigma}(\alpha))$,
3. $\beta\in\operatorname{Super}(\{\alpha\})$.

This allows us to state the following corollary:

Corollary 6.17. *Let* Σ,Σ' *be alphabets with* $|\Sigma|,|\Sigma'|\geq 2$*. Then* $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)=D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma'}}(L)$ *for every* $L\subseteq(\Sigma\cap\Sigma')^*$*.*

Although $\mathrm{Super}(L)$ and $\text{S-Hull}_\Sigma(L)$ might appear to be rather simple concepts, they can be used to establish most of the results in this section. Using Lemma 6.15, we can develop one of our main tools:

Lemma 6.18. *Let Σ be an alphabet with $|\Sigma| \geq 2$. For every $L \subseteq \Sigma^*$, $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L) = D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(\text{S-Hull}_\Sigma(L))$.*

Proof. Let $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$. Then $L_{\mathrm{E},\Sigma}(\delta) \supseteq L$, and $L_{\mathrm{E},\Sigma}(\delta) \supseteq \text{S-Hull}_\Sigma(L)$ by definition of S-Hull_Σ. Thus, $L_{\mathrm{E},\Sigma}(\delta) \supseteq \text{S-Hull}_\Sigma(L) \supseteq L$, and

$$\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(\text{S-Hull}_\Sigma(L))$$

follows by Lemma 6.15.

Assume $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(\text{S-Hull}_\Sigma(L))$, and there is a $\gamma \in X^+$ with $L_{\mathrm{E},\Sigma}(\delta) \supset L_{\mathrm{E},\Sigma}(\gamma) \supseteq L$. Then, $L_{\mathrm{E},\Sigma}(\gamma) \supseteq \text{S-Hull}_\Sigma(L)$ holds (again by definition of S-Hull_Σ), and this contradicts the initial assumption. □

In a sense, $\text{S-Hull}_\Sigma(L)$ captures the whole essence of L with respect to $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive patterns, as every $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive generalization of L is unable to distinguish between these two languages. This is illustrated by the following example:

Example 6.19. *Let $|\Sigma| \geq 2$ and define $L{:=}L_{\mathrm{E},\Sigma}(x_1^2) \cup L_{\mathrm{E},\Sigma}(x_1^3)$. Furthermore, let*

$$\begin{array}{lllll} \delta_1{:=}x_1^2x_2^3, & \delta_2{:=}x_1x_2x_1x_2^2, & \delta_3{:=}x_1x_2^2x_1x_2, & \delta_4{:=}x_1x_2^3x_1, & \delta_5{:=}x_1x_2^2x_1^2, \\ \delta_6{:=}x_1x_2x_1x_2x_1, & \delta_7{:=}x_1x_2x_1^2x_2, & \delta_8{:=}x_1^2x_2^2x_1, & \delta_9{:=}x_1^2x_2x_1x_2, & \delta_{10}{:=}x_1^3x_2^2. \end{array}$$

Recalling Theorem 2.4, it is easy to see that, for every $\alpha \in \mathrm{Super}(L)$, there is a δ_i, $1 \leq i \leq 10$, with $L_{\mathrm{E},\Sigma}(\alpha) \supseteq L_{\mathrm{E},\Sigma}(\delta_i)$ (as, for every α, there must be morphisms mapping α to both x_1^2 and x_1^3). By a convention common in the literature, all patterns are given in canonical form *(cf. [90]), where variables names are introduced in increasing lexicographic order.*

This example illustrates two important phenomena. First, note that $\delta_i \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$ for $1 \leq i \leq 10$, and for every $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$, there is a δ_i with $L_{\mathrm{E},\Sigma}(\delta) = L_{\mathrm{E},\Sigma}(\delta_i)$, but $L_{\mathrm{E},\Sigma}(\delta) \neq L_{\mathrm{E},\Sigma}(\delta_j)$ for every $j \neq i$. Thus, L has ten distinct $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive patterns.

Second, the previous observation leads to $\text{S-Hull}_\Sigma(L) = \bigcap_{i=1}^{10} L_{\mathrm{E},\Sigma}(\delta_i)$. For every $n \geq 2$, there are $j, k \geq 0$ with $n = 2j + 3k$, and therefore, $\text{S-Hull}_\Sigma(L) \supseteq \bigcup_{n=2}^{\infty} L_{\mathrm{E},\Sigma}(x_1^n)$. Thus, every $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive generalization of L is unable to exclude any language $L_{\mathrm{E},\Sigma}(x_1^n)$ with $n \geq 2$. In this sense, $\text{S-Hull}_\Sigma(L)$ provides information on the coarseness of all descriptive generalizations. ◇

Observe that L in the previous example is a finite union of languages from $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$ that has a descriptive pattern, and recall that, according to Proposition 5.32, every finite set of words has an $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive pattern, while (by Theorem 5.18), there are infinite unions of languages from $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$ that have no descriptive pattern.

Using Lemma 6.18, we can extend Proposition 5.32 to show that not only every finite set of words, but every finite union of languages from $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$ has an $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive pattern:

Proposition 6.20. *Let Σ be an alphabet with $|\Sigma| \geq 2$, let $A = \{\alpha_1, \ldots, \alpha_n\} \subset X^+$ and let $L = \bigcup_{i=1}^{n} L_{\mathrm{E},\Sigma}(\alpha_i)$. Then $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L) \neq \emptyset$.*

Proof. Let $A = \{\alpha_1, \ldots, \alpha_n\} \subset X^+$ and $L = \bigcup_{i=1}^{n} L_{\mathrm{E},\Sigma}(\alpha_i)$. By Theorem 2.4, the equation $\mathrm{Super}(\{\alpha\}) = \mathrm{Super}(L_{\mathrm{E},\Sigma}(\alpha))$ holds for every $\alpha \in X^+$. Thus,

$$\mathrm{Super}(\{\alpha_1, \ldots, \alpha_n\}) = \mathrm{Super}(L_{\mathrm{E},\Sigma}(\alpha_1) \cup \ldots \cup L_{\mathrm{E},\Sigma}(\alpha_n)),$$

and therefore $\text{S-Hull}_\Sigma(A) = \text{S-Hull}_\Sigma(L)$. This is equivalent to $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(A) = D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$ (by Lemma 6.18). As A is a finite set, according to Proposition 5.32, $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(A)$ is nonempty, and thus, $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$ is nonempty as well. □

Basically, Example 6.19 and Proposition 6.20 are based on the fact that words in languages from ePAT$_{\mathrm{tf},\Sigma}$ and the generating patterns of these languages can often be used interchangeably by defining a morphism that maps the words back to their generating pattern. We proceed to develop this approach into another tool that allows us to make further statements on the (non-)existence of ePAT$_{\mathrm{tf},\Sigma}$-descriptive patterns. Let $\nu : \Sigma^* \to X^*$ be an arbitrary renaming. We define

$$\text{V-Hull}_\Sigma(L) := \bigcup_{w \in L} L_{\mathrm{E},\Sigma}(\nu(w)).$$

Like $\text{S-Hull}_\Sigma(L)$, $\text{V-Hull}_\Sigma(L)$ is equivalent to L with respect to Super and $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}$:

Lemma 6.21. *Let Σ be an alphabet, $|\Sigma| \geq 2$. For every L over Σ,*

$$\mathrm{Super}(L) = \mathrm{Super}(\text{V-Hull}_\Sigma(L)), \text{ and}$$
$$D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L) = D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(\text{V-Hull}_\Sigma(L)).$$

Proof. We first show $\mathrm{Super}(L) = \mathrm{Super}(\text{V-Hull}_\Sigma(L))$.

Let $L \subseteq \Sigma^*$ and let $\nu : \Sigma^* \to X^*$ be a renaming. Naturally, there is an inverse renaming $\nu^{-1} : (\nu(\Sigma))^* \to \Sigma^*$ with $(\nu^{-1} \circ \nu)(w) = w$ for every $w \in L$. Thus, $w \in L_{\mathrm{E},\Sigma}(\nu(w))$ holds for every $w \in L$, which implies $L \subseteq \text{V-Hull}_\Sigma(L)$, and $\mathrm{Super}(L) \supseteq \mathrm{Super}(\text{V-Hull}_\Sigma(L))$.

For the other direction, consider any $\alpha \in \mathrm{Super}(L)$. This is equivalent to $L_{\mathrm{E},\Sigma}(\alpha) \supseteq L$, and thus, for every $w \in L$, there is a morphism $\phi : X^* \to \Sigma^*$ with $\phi(\alpha) = w$. Accordingly, $(\nu \circ \phi)(\alpha) = \nu(w)$, and thus, $L_{\mathrm{E},\Sigma}(\alpha) \supseteq L_{\mathrm{E},\Sigma}(\nu(w))$ for every $w \in L$. This immediately implies $L_{\mathrm{E},\Sigma}(\alpha) \supseteq \text{V-Hull}_\Sigma(L)$, and therefore, $\alpha \in \mathrm{Super}(\text{V-Hull}_\Sigma(L))$.

As $\mathrm{Super}(L) = \mathrm{Super}(\text{V-Hull}_\Sigma(L))$, $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L) = D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(\text{V-Hull}_\Sigma(L))$ follows by definition of $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}$. □

This leads us to the following insight into the existence of ePAT$_{\mathrm{tf},\Sigma}$-descriptive patterns for infinite unions of languages from ePAT$_{\mathrm{tf},\Sigma}$:

Proposition 6.22. *Let $|\Sigma| \geq 2$. Then there is a set of patterns $A \subset \{x_1, x_2\}^*$ such that no pattern is* ePAT$_{\mathrm{tf},\Sigma}$*-descriptive of $\bigcup_{\alpha \in A} L_{\mathrm{E},\Sigma}(\alpha)$.*

Proof. Choose any alphabet $\Sigma' \subseteq \Sigma$ with $|\Sigma'| = 2$. By Theorem 5.18, there exists a language $L' \subset (\Sigma')^*$ such that $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma'}}(L') = \emptyset$. Furthermore, let $\nu : (\Sigma')^* \to \{x_1, x_2\}^*$ be a renaming. We claim that $A := \nu(L')$ fulfils the given requirements. As $A \subset \{x_1, x_2\}^*$

holds by definition, we only need to show that no pattern in X^+ is $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive of the language $L := \bigcup_{\alpha \in A} L_{\mathrm{E},\Sigma}(\alpha)$.

Assume to the contrary that $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L) \neq \emptyset$. Note that $L = \text{V-Hull}_\Sigma(L')$ holds, which allows us to use Lemma 6.21 to conclude $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L) = D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L')$. Furthermore, Corollary 6.17 implies $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L') = D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma'}}(L')$. Combining these two equations and the initial assumption of $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L) \neq \emptyset$, we arrive at $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma'}}(L') \neq \emptyset$, which contradicts our choice of L'. □

Thus, unlike in the case of *finite* unions of languages from $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$ (cf. Proposition 6.20), even restricting the number of variables in the generating patterns does not ensure that *infinite* unions of languages from $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$ have a descriptive pattern. The renaming ν that maps terminals to variables can also be used to obtain the following technical result:

Lemma 6.23. *Let Σ be an alphabet with $|\Sigma| \geq 2$. For every nonempty language $L \subseteq \Sigma^*$ with $L \neq \{\lambda\}$, $\text{S-Hull}_\Sigma(L)$ is infinite.*

Proof. Let $\nu : \Sigma^+ \to X^+$ be a nonerasing morphism and choose any $w \in L \setminus \{\lambda\}$. For every $\alpha \in \mathrm{Super}(L)$, there is a morphism $\psi : X^+ \to \Sigma^*$ with $\psi(\alpha) = w$. Thus, $\nu(\psi(\alpha)) = \nu(w)$, and therefore $\text{S-Hull}_\Sigma(L) \supseteq L_{\mathrm{E},\Sigma}(\nu(w))$. As $\nu(w) \in X^+$, this language is infinite. □

This insight shall be used in Section 6.4. We conclude the present part of Section 6.3.1 with a short remark illustrating that there are finite classes of languages which are not contained in $\mathrm{DG}_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}$:

Proposition 6.24. *Let Σ be an alphabet, $|\Sigma| \geq 2$. There exists a class $\mathcal{L}$ of nonempty languages over Σ with $|\mathcal{L}| = 1$ and $\mathcal{L} \notin \mathrm{DG}_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}$.*

Proof. As stated in Theorem 5.18, there is a language L_Σ with $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L_\Sigma) = \emptyset$. If we choose $\mathcal{L} = \{L_\Sigma\}$, then $|\mathcal{L}| = 1$ holds, and no strategy will be able to compute any hypothesis that is $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive of L_Σ. □

6.3.2 The Canonical Strategy and Telling Sets

According to Proposition 5.32, every finite set has a computable $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive pattern. The author considers it the canonical strategy of descriptive inference on any text t of a given language L to compute a descriptive pattern of every initial segment t^n, in the hope that the hypotheses will converge to a pattern that is descriptive of L. As evidenced by the language $L{:=}L_{\mathrm{E},\Sigma}(x_1^2) \cup L_{\mathrm{E},\Sigma}(x_1^3)$ (cf. Example 6.19), there are languages with more than one descriptive pattern. Furthermore, this applies also to finite languages, as for the set $S{:=}\{\mathtt{a}^2, \mathtt{b}^3\}$ (for arbitrary letters $\mathtt{a}, \mathtt{b} \in \Sigma$), $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(S) = D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$ holds. Although S already contains all the information that is needed to compute a descriptive generalization of L, the ten distinct patterns δ_1 to δ_{10} from Example 6.19 are all valid hypotheses. In order to allow our strategy to converge to one single hypothesis, we impose a total and well-founded order $<_{\mathrm{LLO}}$ on X^+ and let our strategy return the $<_{\mathrm{LLO}}$-minimal hypothesis.

Let $<_{\mathrm{LLO}}$ denote the length-lexicographic order[2] on X^+. Note that $<_{\mathrm{LLO}}$ is total and does not contain infinite decreasing chains. Thus, every set has exactly one element that is minimal with respect to $<_{\mathrm{LLO}}$.

The strategy Canon : $(\Sigma \cup \{\nabla\})^* \to (\Sigma \cup X)^+$ is defined by, for every text t,

$$\mathrm{Canon}(t^n) := \delta, \text{ where } \delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(t[n]) \text{ and } \delta <_{\mathrm{LLO}} \gamma \text{ for every other } \gamma \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(t[n]).$$

The computability of Canon follows immediately from the proof of Proposition 5.32, as all that remains is to sort the finite search space by $<_{\mathrm{LLO}}$. We say that Canon *converges* on a text $t \in \mathrm{text}(L)$ (of some language L over some alphabet Σ) if there is a pattern $\alpha \in X^+$ with $\mathrm{Canon}(t^n) = \alpha$ for all but finitely many values of n. If, in addition to this, $\alpha \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$, Canon is said to *converge correctly* on t. Now, when considering the languages L and S given in the example above, for every text $t \in \mathrm{text}(L)$, there is an $n \geq 0$ with $S \subseteq t[n]$. From this point on, $\mathrm{Canon}(t[n])$ will return the pattern $\delta_{10} = x_1^3 x_2^2$, as δ_{10} is an element of $(D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(S) \cap D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L))$ and the $<_{\mathrm{LLO}}$-minimum of the canonical forms of the δ_i. This phenomenon leads to the definition of what we call *telling sets*, which are of crucial importance for the study of descriptive generalizability with the strategy Canon:

Definition 6.25. *Let $L \subseteq \Sigma^*$. A finite set $S \subseteq L$ is a* telling set *for L if $(D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(S) \cap D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)) \neq \emptyset$.*

Note that telling sets have some similarity to the concept of telltales that is used in the model of learning in the limit. For a comparison of telltales and telling sets, see the comments after Corollary 6.33.

Using Lemma 6.15, we are now able to show that the existence of a telling set is characteristic for the correct convergence of Canon on any text:

Theorem 6.26. *Let Σ an alphabet with $|\Sigma| \geq 2$. For every language $L \subseteq \Sigma^*$, and every text $t \in \mathrm{text}(L)$,* Canon *converges correctly on t if and only if L has a telling set.*

Proof. We begin by proving the *only if* direction via its contraposition. Assume that L has no telling set. Then, for every text t of L and every $n \geq 0$, $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(t[n])$ and $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$, by definition, are disjoint, as otherwise $t[n]$ would be a telling set. Therefore, $\mathrm{Canon}(t^n) \notin D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$, which means that even if Canon converges on t to a pattern δ, this pattern is not ePAT$_{\mathrm{tf},\Sigma}$-descriptive of L.

For the *if* direction, assume that S is a telling set of L, and consider any text t of L such that Canon does not converge correctly on t. We first show that if Canon converges, it always converges correctly. Assume to the contrary that there exist some pattern δ and some $n \geq 0$ such that $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(t[m])$ for every $m \geq n$, but $\delta \notin D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$. As $L_{\mathrm{E},\Sigma}(\delta) \supseteq t[m]$ for every $m \geq n$, it follows that

$$\begin{aligned} L_{\mathrm{E},\Sigma}(\delta) &\supseteq \bigcup_{m \geq n} t[m] \\ &= \{t[i] \mid i \geq 0\} = L. \end{aligned}$$

As $L_{\mathrm{E},\Sigma}(\delta) \supseteq L \supseteq t[n]$ and $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(t[n])$, Lemma 6.15 gives $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$, which contradicts the initial assumption.

[2] I. e., $\alpha <_{\mathrm{LLO}} \beta$ if $|\alpha| < |\beta|$, or if $|\alpha| = |\beta|$, and α precedes β in the lexicographic order.

Next, assume that Canon does not converge on t at all; i. e., there is an infinite sequence $(\delta_n)_{n\geq 0}$ over X^+ with

1. $\delta_n = \text{Canon}(t^n)$ for every $n \geq 0$,
2. for every $n \geq 0$, there is an $m > n$ with $\delta_m \neq \delta_n$.

We first show that at most one pattern occurs twice in $(\delta_n)_{n\geq 0}$. Assume that there are $m_1 < n_1 < m_2 < n_2$ with $\delta_{m_1} = \delta_{m_2} \neq \delta_{n_1} = \delta_{n_2}$, and observe that

$$t[m_1] \subseteq t[n_1] \subseteq t[m_2] \subseteq t[n_2]$$

holds. As δ_{m_1} is $\text{ePAT}_{\text{tf},\Sigma}$-descriptive of $t[m_1]$, and

$$L_{\text{E},\Sigma}(\delta_{m_1}) = L_{\text{E},\Sigma}(\delta_{m_2}) \supseteq t[m_2] \supseteq t[n_1] \supseteq t[m_1],$$

Lemma 6.15 implies $\delta_{m_1} \in D_{\text{ePAT}_{\text{tf},\Sigma}}(t[n_1])$. However, due to $\text{Canon}(t[n_1]) = \delta_{n_1} \neq \delta_{m_1}$, $\delta_{n_1} <_{\text{LLO}} \delta_{m_1}$ must hold. Analogously, one can use Lemma 6.15 and $L_{\text{E},\Sigma}(\delta_{n_1}) \supseteq t[n_2] \supseteq t[m_2] \supseteq t[n_1]$ to conclude $\delta_{m_1} <_{\text{LLO}} \delta_{n_1}$, which leads to the contradictory statement $\delta_{m_1} <_{\text{LLO}} \delta_{n_1} <_{\text{LLO}} \delta_{m_1}$.

We now consider two cases. First, assume there is some δ such that $\delta_n = \delta$ for infinitely many $n \geq 0$ (as we have seen, there can be at most one such δ). Then $L_{\text{E},\Sigma}(\delta) \supseteq t[n]$ holds for infinitely many $n \geq 0$, which implies $L_{\text{E},\Sigma}(\delta) \supseteq t[m]$ for every $m \geq 0$. As above, this leads to

$$L_{\text{E},\Sigma}(\delta) \supseteq \{t(i) \mid i \geq 0\} = L.$$

In particular, if $\delta_n = \delta$,

$$L_{\text{E},\Sigma}(\delta) \supseteq L \supseteq t[n],$$

and the initial assumption $\delta \in D_{\text{ePAT}_{\text{tf},\Sigma}}(t[n])$ allow us to use Lemma 6.15 to conclude $\delta \in D_{\text{ePAT}_{\text{tf},\Sigma}}(L)$. But only a finite number of δ_m can satisfy $\delta_m <_{\text{LLO}} \delta$, which means that Canon converges to δ, and this contradicts our initial assumption.

For the other case, assume that no pattern occurs infinitely often in $(\delta_n)_{n\geq 0}$. Then for every $\delta \in X^+$, there is a $k \geq 0$ such that $\delta <_{\text{LLO}} \delta_n$ for all $n \geq k$. This holds in particular for that pattern $\delta_S \in (D_{\text{ePAT}_{\text{tf},\Sigma}}(S) \cap D_{\text{ePAT}_{\text{tf},\Sigma}}(L))$ that is minimal with respect to $<_{\text{LLO}}$ (such a pattern has to exist, as we required S to be a telling set of L). Choose some k such that

1. $\delta_S <_{\text{LLO}} \delta_n$ for all $n \geq k$, and
2. $S \subseteq t[k]$,

and observe that, due to Lemma 6.15,

$$(D_{\text{ePAT}_{\text{tf},\Sigma}}(S) \cap D_{\text{ePAT}_{\text{tf},\Sigma}}(L)) \subseteq D_{\text{ePAT}_{\text{tf},\Sigma}}(t[n])$$

for every $n \geq k$. Thus, $\delta_S = \text{Canon}(t[n])$ for every $n \geq k$. This contradicts our initial assumption that Canon does not converge correctly on t. □

In the final part of this section, we shall demonstrate that this is a strong result, by investigating the existence and nonexistence of telling sets for various languages.

6.4 Examination of the Class $\mathcal{TSL}_\Sigma$

As stated by Theorem 6.26, the existence of telling sets is a strong sufficient criterion for $\text{ePAT}_{\text{tf},\Sigma}$-descriptive generalizability. Furthermore, generalizability of a class $\mathcal{L} \subseteq \mathcal{P}(\Sigma^*)$ using Canon does not depend on the properties of the whole class, but only on the existence of a telling set for every single language $L \in \mathcal{L}$. Thus, we consider the largest possible class that can be generalized by Canon and define

$$\mathcal{TSL}_\Sigma := \{L \subseteq \Sigma^* \mid L \text{ has a telling set}\}.$$

Theorem 6.26 immediately leads to the following corollary:

Corollary 6.27. *For every alphabet Σ with $|\Sigma| \geq 2$, $\mathcal{TSL}_\Sigma \in \text{DG}_{\text{ePAT}_{\text{tf},\Sigma}}$.*

Thus, by examining $\mathcal{TSL}_\Sigma$, we gain insights into the power of Canon and of the whole model of descriptive generalization. Before we proceed to an examination of the relation of various classes of languages to $\mathcal{TSL}_\Sigma$, we show that it is not required to choose Σ as small as possible, a result that is similar to Corollary 6.17, which states that $D_{\text{ePAT}_{\text{tf},\Sigma}}(L)$ is largely independent of the choice of Σ. The same holds for telling sets:

Corollary 6.28. *Let Σ, Σ' be alphabets with $|\Sigma|, |\Sigma'| \geq 2$. Then $L \in \mathcal{TSL}_\Sigma$ if and only if $L \in \mathcal{TSL}_{\Sigma'}$ for every $L \subseteq (\Sigma \cap \Sigma')^*$.*

Proof. As $L \subseteq (\Sigma \cap \Sigma')^*$, the same holds for every telling set $S \subseteq L$. According to Corollary 6.17, $D_{\text{ePAT}_{\text{tf},\Sigma}}(S) = D_{\text{ePAT}_{\text{tf},\Sigma'}}(S)$. □

This also implies that, for every $\Sigma' \supseteq \Sigma$, $\mathcal{TSL}_{\Sigma'} \supseteq \mathcal{TSL}_\Sigma$. We begin our examination of $\mathcal{TSL}_\Sigma$ by expanding finite languages without losing their telling set properties. The next result follows immediately from Lemmas 6.15 and 6.18:

Lemma 6.29. *Let Σ be an alphabet with $|\Sigma| \geq 2$. Every nonempty $S \in \text{FIN}_\Sigma$ is a telling set of $\text{S-Hull}_\Sigma(S)$ and of every L with $S \subseteq L \subseteq \text{S-Hull}_\Sigma(S)$.*

Proof. Due to Lemma 6.18, $D_{\text{ePAT}_{\text{tf},\Sigma}}(S) = D_{\text{ePAT}_{\text{tf},\Sigma}}(\text{S-Hull}_\Sigma(S))$ holds; therefore S is a telling set of $\text{S-Hull}_\Sigma(S)$. Now choose any L with $S \subseteq L \subseteq \text{S-Hull}_\Sigma(S)$ and any $\delta \in D_{\text{ePAT}_{\text{tf},\Sigma}}(S)$. According to Lemma 6.15, $\delta \in D_{\text{ePAT}_{\text{tf},\Sigma}}(L)$, which means that S is a telling set of L. □

In addition to showing that $\text{FIN}_\Sigma \subseteq \mathcal{TSL}_\Sigma$, this result allows us (in conjunction with Lemma 6.23) to make the following statement on the cardinality of $\mathcal{TSL}_\Sigma$:

Proposition 6.30. *$\mathcal{TSL}_\Sigma$ is uncountable for every alphabet Σ with $|\Sigma| \geq 2$.*

Proof. Select a finite nonempty language $S \subseteq \Sigma^+$. Let

$$\begin{aligned}\mathcal{U} &:= \{L \mid S \subseteq L \subseteq \text{S-Hull}_\Sigma(S)\} \\ &= \{L \cup S \mid L \in \mathcal{P}\left(\text{S-Hull}_\Sigma\left(S\right) \setminus S\right)\}.\end{aligned}$$

Due to Lemma 6.29, S is a telling set of every $L \in \mathcal{U}$, and thus, $\mathcal{U} \subseteq \mathcal{TSL}_\Sigma$.

As S is nonempty, $\text{S-Hull}_\Sigma(S)$ must be infinite according to Lemma 6.23. Therefore, $\mathcal{U}$ is uncountable, which means that $\mathcal{TSL}_\Sigma$ is uncountable as well. □

This is an uncommon property, as inference from positive data is normally considered for classes consisting of countably many languages from some countable domain. Nonetheless, inferrability of uncountable classes has been studied before, see [49].

Next, we shall see that $\mathcal{TSL}_\Sigma$ contains a rich and natural class of languages, the DTF0L languages (cf. Section 5.2.2). We denote the class of all DTF0L languages over Σ by $\mathrm{DTF0L}_\Sigma$.

Proposition 6.31. *Let Σ be an alphabet with $|\Sigma| \geq 2$. Then $\mathrm{DTF0L}_\Sigma \subseteq \mathcal{TSL}_\Sigma$.*

Proof. Let $L \subseteq \Sigma^*$ be a DTF0L-language, where F is a nonempty set of axioms and Φ a nonempty set of morphisms generating L from F. It suffices to show that $\mathrm{Super}(F) = \mathrm{Super}(L)$, as this shall allow us to use Lemma 6.18 to obtain the desired result.

First, observe that, by definition, $F \subseteq L$, and thus, $\mathrm{Super}(F) \supseteq \mathrm{Super}(L)$. For the other direction, consider any $\alpha \in \mathrm{Super}(F)$. For every $w \in L$, there is a $v \in F$ and a finite sequence of morphisms in Φ that can be composed to a single morphism ϕ with $\phi(v) = w$. As $v \in F$ and $\alpha \in \mathrm{Super}(F)$, there is a morphism ρ with $\rho(\alpha) = v$. The composition of these morphisms leads to $(\phi \circ \rho)(\alpha) = w$, and (as w was chosen arbitrarily) $\alpha \in \mathrm{Super}(L)$.

Now, $\mathrm{Super}(F) = \mathrm{Super}(L)$ results in $\text{S-Hull}_\Sigma(F) = \text{S-Hull}_\Sigma(L)$ and, by Lemma 6.18, in $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(F) = D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$. As F is finite, it has at least one $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive pattern (according to Proposition 5.32), and is a telling set of L. □

Lemma 6.29 and Proposition 6.31 both imply that $\mathrm{FIN}_\Sigma \subseteq \mathcal{TSL}_\Sigma$. Furthermore, Proposition 6.30 and Proposition 6.31 both demonstrate that $\mathcal{TSL}_\Sigma$ contains at least one infinite language, which leads to the following observation:

Corollary 6.32. *The class $\mathcal{TSL}_\Sigma$ is superfinite for every alphabet Σ with $|\Sigma| \geq 2$.*

Together with Proposition 6.24, this allows us to describe the relation between $\mathrm{DG}_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}$ and LIM-TEXT:

Corollary 6.33. *Let Σ be an alphabet, $|\Sigma| \geq 2$. Then $\mathrm{DG}_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}$ and* LIM-TEXT *are incomparable.*

Proof. Directly from Proposition 6.24 and Corollary 6.32, as LIM-TEXT contains every finite class, but no superfinite classes (cf. Gold [39] and Theorem 6.2). □

We now briefly discuss the relation between telling sets and the notion of telltales. As already mentioned above in Theorem 6.4, an indexed family $\mathcal{L} = (L_i)_{i=0}^\infty$ of nonempty recursive languages is in LIM-TEXT if and only if there exists an effective procedure which, for every $j \geq 0$, enumerates a set T_j such that

- T_j is finite,
- $T_j \subseteq L_j$, and
- there does not exist a $j' \geq 0$ with $T_j \supseteq L_{j'} \supset L_j$.

If there exists a set T_j satisfying these conditions, it is called a *telltale* for L_j with respect to $\mathcal{L} = (L_i)_{i=0}^{\infty}$. Thus, the concepts telltales and telling sets are incomparable, as the former refers to a language and the class of languages it is contained in, whereas the latter relates to a language and certain properties of the class $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$. Nevertheless, for every language L in $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$, a set S is a telling set for L if and only if S is a telltale for L with respect to $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$ (for more details on the existence of telltales for languages in $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$, see [88]).

As Proposition 6.34 and Proposition 6.35 below show, Reidenbach's insights into the nonexistence and existence of telltales lead to the corresponding results for telling sets:

Proposition 6.34. *Let Σ be an alphabet with $|\Sigma| \geq 3$. For every $\alpha \in X^+$, $L_{\mathrm{E},\Sigma}(\alpha)$ has a telling set.*

Proof. This follows immediately from Lemma 25 in Reidenbach [88]. □

On the other hand, it is impossible to encode the structure of comparatively simple patterns in their languages with only two letters, which leads to the following negative result:

Proposition 6.35. *Let Σ be an alphabet with $|\Sigma| \geq 2$, and let* a, b *be two distinct letters from Σ. Then $L_{\mathrm{E},\{\mathtt{a},\mathtt{b}\}}(x_1^2x_2^2x_3^2) \notin \mathcal{TSL}_\Sigma$.*

Proof. This follows immediately from Lemma 7 in Reidenbach [86]. □

In contrast to this, Lemma 6.21 can be used to show that restricting the number of variables in the patterns leads to telling sets not only for languages from $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$, but also for their finite unions:

Proposition 6.36. *Let $\alpha_1, \ldots, \alpha_n \in \{x_1, \ldots, x_{|\Sigma|}\}^+$, and let $L{:=}\bigcup_{i=1}^{n} L_{\mathrm{E},\Sigma}(\alpha_i)$. Then $L \in \mathcal{TSL}_\Sigma$.*

Proof. Let Σ be an alphabet, $|\Sigma| \geq 2$. Let $X_{|\Sigma|} = \{x_1, \ldots, x_{|\Sigma|}\}$, let $\alpha_1, \ldots, \alpha_n \in X_{|\Sigma|}^+$, and let $L{:=}\bigcup_{i=1}^{n} L_{\mathrm{E},\Sigma}(\alpha_i)$. Choose any renaming $\nu : X_{|\Sigma|}^* \to \Sigma^*$, let ν^{-1} the corresponding inverse renaming and let

$$S{:=}\{\nu^{-1}(\alpha_1), \ldots, \nu^{-1}(\alpha_n)\}.$$

It is easily seen that $\text{V-Hull}_\Sigma(S) = L$ holds. By Lemma 6.21, we conclude $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(S) = D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(\text{V-Hull}_\Sigma(S))$, and therefore $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(S) = D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$. As S is finite, Proposition 5.32 leads to $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(S) \neq \emptyset$. Therefore, S is a telling set for L. □

Proposition 6.36 is particularly interesting when compared to Proposition 6.22, which tells us that infinite unions of languages from $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$ might not only have no telling set, but not even a descriptive pattern.

Furthermore, we state that the infinite sequence $(\beta_n)_{n\geq 0}$ that is used in the definition of the languages L_Σ for the proof of Theorem 5.18 describes an infinite ascending chain of languages from $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$; i. e., $L_{\mathrm{E},\Sigma}(\beta) \subset L_{\mathrm{E},\Sigma}(\beta_{n+1})$ for every $n \geq 0$. Although the presence of such a chain in $\text{S-Hull}_\Sigma(L)$ for a language L does not necessarily imply emptiness of $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$, it is a sufficient criterion for $L \notin \mathcal{TSL}_\Sigma$ (again, the proof relies on Lemma 6.18):

Lemma 6.37. *Let Σ be an alphabet with $|\Sigma| \geq 2$ and let $L \subseteq \Sigma^*$. If there is an infinite chain $(\beta_n)_{n\geq 0}$ over X^+ with $L_{\mathrm{E},\Sigma}(\beta_n) \subseteq \text{S-Hull}_\Sigma(L)$ for every $n \geq 0$, $L_{\mathrm{E},\Sigma}(\beta_n) \subset L_{\mathrm{E},\Sigma}(\beta_{n+1})$ for every $n \geq 0$, and $\bigcup_{n\geq 0} L_{\mathrm{E},\Sigma}(\beta_n) \supseteq L$, then L has no telling set.*

Proof. Let $L \subseteq \Sigma^*$ and $(\beta_n)_{n\geq 0}$ a strictly ascending infinite chain over X^+ that satisfies the above criteria. Assume to the contrary there are a finite set $S \subseteq L$ and a pattern $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L) \cap D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(S)$.

As $S \subseteq L$, there is an $m \geq 0$ such that $L_{\mathrm{E},\Sigma}(\beta_m) \supseteq S$, and therefore,

$$\text{S-Hull}_\Sigma(L) \supseteq L_{\mathrm{E},\Sigma}(\beta_m) \supseteq S.$$

As, due to Lemma 6.18, $\delta \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(\text{S-Hull}_\Sigma(L))$, we conclude

$$L_{\mathrm{E},\Sigma}(\delta) \supseteq \text{S-Hull}_\Sigma(L) \supseteq L_{\mathrm{E},\Sigma}(\beta_m) \supseteq S.$$

By definition, β_m is part of an infinite ascending chain, and therefore

$$\text{S-Hull}_\Sigma(L) \supseteq L_{\mathrm{E},\Sigma}(\beta_{m+1}) \supset L_{\mathrm{E},\Sigma}(\beta_m) \supseteq S$$

holds. This contradicts $\delta \notin D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(S)$. Thus, L either has no $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive pattern, or it has an $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive pattern, but no telling set. □

As a direct application of this result, we can prove that there are regular languages that have no telling set:

Proposition 6.38. *For every alphabet Σ with $|\Sigma| \geq 2$, there is a regular language $L \subseteq \Sigma^*$ with $L \notin \mathcal{TSL}_\Sigma$.*

Proof. Let $\mathtt{a}, \mathtt{b}$ two distinct letters from Σ and define $L = (\mathtt{aa} + \mathtt{bb})^*$. Next, we shall show that $\text{S-Hull}_\Sigma(L) = \Sigma^*$. For $\alpha \in \text{Super}(L)$, let $n{:=}|\alpha|$ and

$$w{:=}\,\mathtt{aabbaa}\;\mathtt{aa}(\mathtt{bb})^2\,\mathtt{aa}\ldots\mathtt{aa}(\mathtt{bb})^{n+1}\,\mathtt{aa}\,.$$

As $w \in L$, there is a morphism ϕ with $\phi(\alpha) = w$. Furthermore, as α has at most n distinct variables, there is at least one variable $x \in \text{var}(\alpha)$ such that $\phi(x)$ contains a factor $\mathtt{aa}(\mathtt{bb})^i\,\mathtt{aa}$ for some i with $1 \leq i \leq n+1$. As this factor occurs exactly once in w, x occurs exactly once in α, and thus, $L_{\mathrm{E},\Sigma}(\alpha) = \Sigma^*$. □

Note that this language is also an example of a language L that has no telling set, although $\text{S-Hull}_\Sigma(L)$ has a telling set. Furthermore, this is an example of a language L with $x_1 \in D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L)$, but no $w \in L$ contains a singular letter.

Chapter 7

On a Conjecture on $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-Descriptive Patterns

After developing the proof of Theorem 5.18, the author tried to extend the technique employed therein to a characterization of all languages that have no $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive pattern. Guided by an elegant but illusive conjecture, he spent six months on this attempt, only to discover a comparatively compact counterexample. The present chapter contains the results of this work. As it is mainly a collection of tools and examples, it diverges in tone and structure from the other chapters in this thesis.

We discuss in Section 7.1 a few technical concepts and the conjectures that were the main focus of the author's work on the existence of $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive patterns. Most of these conjectures are disproved by the *Loughborough Example*, which is explained in Section 7.4. In order to define and understand this example, we develop various concepts and tools in Sections 7.2 and 7.3.

Here, Section 7.2 formalizes a few observations on languages that are generated by strictly decreasing chains of terminal-free E-pattern languages, and introduces so called *chain systems*. These chain systems are used in Section 7.3 to define a family of languages $L_{\Sigma}^{(k)}$, and to define the languages used in the Loughborough Example, as well as the *Wittenberg Examples*[1] in Section 7.5, two examples that further illustrate certain phenomena that are related to strictly decreasing chains and the (non)-existence of $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive patterns.

Although the author considers these named examples in Sections 7.4 and 7.5 the main results of the present chapter, the tools in Section 7.2 and in Section 7.3 are not only the fundament on which the examples build, but also a generalization of the phenomena the author noticed during his work. As such, they might serve as a valuable starting point for further work on strictly decreasing chains.

As this Chapter is an extension of the work in Chapter 5 and, although far less so, Chapter 6, it uses various tools and techniques that can be found in Sections 5.2 and mentions a few concepts used in 6.3.1. Moreover, it is assumed that the reader is familiar with the material presented in Section 5.3.

[1] Explanations for these choices of names can be found in the appropriate sections.

7.1 Technical Preliminaries and Various Conjectures

As explained in Section 5.3, the non-existence of a descriptive pattern for a language L is inherently related to the existence of a chain of pattern languages that decreases strictly towards that language – in fact, the existence of such a chain is a necessary condition for the non-existence of a descriptive pattern. Naturally, every necessary condition leads to the question which further conditions might lead to a characterization. As inclusion for general E-pattern languages is undecidable (and thus, hard), we focus on terminal-free E-pattern languages, as in the proof of Theorem 5.18. Similarly to Section 6.3, the results in this chapter can be immediately adapted to all classes of E-pattern languages for which inclusion is characterized by the existence of a terminal-preserving morphism between the patterns (for some examples, see the list in Section 3.3, after Theorem 3.5).

When working with terminal-free E-patterns, we can restrict the considerations of Section 5.3, and refine the condition on chains of pattern languages. Due to the fact that equivalence for terminal-free E-pattern languages is comparatively straightforward and well-understood (cf. Section 5.2.1), we can consider chains of *patterns* instead of chains of *pattern languages*. Thus, a sequence $(\alpha_i)_{i=0}^{\infty}$ over X^+ is

1. a *decreasing chain* if $L_{\mathrm{E},\Sigma}(\alpha_i) \supseteq L_{\mathrm{E},\Sigma}(\alpha_{i+1})$ for every $i \geq 0$,
2. an *increasing chain* if $L_{\mathrm{E},\Sigma}(\alpha_i) \subseteq L_{\mathrm{E},\Sigma}(\alpha_{i+1})$ for every $i \geq 0$.

Furthermore, the chain is *strictly decreasing* (or *strictly increasing*) if it is decreasing (or increasing, respectively), and $L_{\mathrm{E},\Sigma}(\alpha_i) \neq L_{\mathrm{E},\Sigma}(\alpha_{i+1})$ for every $i \geq 0$.

For every Σ with $|\Sigma| \geq 2$, we say that a decreasing chain $(\alpha_i)_{i=0}^{\infty}$ over X^+ *generates* the language

$$L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty}) := \bigcap_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(\alpha_i).$$

Hence, one can understand $L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty})$ to be the limit of $L_{\mathrm{E},\Sigma}(\alpha_i)$, where i grows towards infinity.

We say that a language $L \subseteq \Sigma^*$ *is covered by* $(\alpha_i)_{i=0}^{\infty}$ if $L \subseteq L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty})$.

As explained in Section 5.3, we do not want to focus on strictly decreasing chains that achieve their decrease using redundant variables (e. g., as seen in Example 5.11, some x_1 that is mapped to x_1^2, x_1^4, x_1^8 and, in the limit, is utterly useless).

In order to formalize this notion, we develop the concept of patterns that are *reduced* with respect to some language L. To this end, we define, for every $A \subseteq X$, the *projection morphism* $\pi_A : X^* \to A^*$ by

$$\pi_A(x) := \begin{cases} x & \text{if } x \in A, \\ \lambda & \text{otherwise.} \end{cases}$$

Let Σ be an alphabet and $L \subseteq \Sigma^*$. A pattern $\alpha \in X^*$ with $L \subseteq L_{\mathrm{E},\Sigma}(\alpha)$ is *L-reduced* (or *reduced with respect to L*) if, for every $A \subset \mathrm{var}(\alpha)$, $L \not\subseteq L_{\mathrm{E},\Sigma}(\pi_A(\alpha))$. Intuitively, α does not contain any redundant letters. Obviously, a morphically primitive pattern that is ePAT$_{\mathrm{tf},\Sigma}$-descriptive of some language L must be L-reduced.

We now extend this concept from patterns to decreasing chains of patterns: If L is covered by a decreasing chain $(\alpha_i)_{i=0}^{\infty}$, that chain is L-reduced if every α_i is L-reduced.

Furthermore, a decreasing chain $(\alpha_i)_{i=0}^{\infty}$ is *self-reduced* if it is reduced with respect to $L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty})$.

By definition, L-reduced patterns are related to morphically primitive patterns, as is evidenced by the following observation:

Observation 7.1. *Let Σ be an alphabet with $|\Sigma| \geq 2$, let $L \subseteq \Sigma^*$ and let $L \subseteq L_{\mathrm{E},\Sigma}(\alpha)$ for some $\alpha \in X^+$. If α is not morphically primitive, then α is not L-reduced.*

This follows immediately from the fact that morphically primitive patterns are succinct, and thus the shortest generators of their pattern languages (cf. Section 5.2.1). On the other hand, not every morphically primitive pattern is L-reduced; e. g., consider $L{:=}L_{\mathrm{E},\Sigma}(x^2)$ and $\alpha{:=}x^2y^2$. In a way, one might consider self-reduced chains as an extension of succinct patterns, as both use the smallest number of variables to generate their respective language.

After some experimentation, the author developed the following conjecture:

Conjecture 7.2. *Let Σ be an alphabet with $|\Sigma| \geq 2$ and let $L \subseteq \Sigma^*$. If a strictly decreasing chain $(\alpha_i)_{i=0}^{\infty}$ over X^+ is self-reduced, then $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty})) = \emptyset$.*

Of course, according to Lemma 6.18, this would also extend to all languages L for which $\text{S-Hull}_{\Sigma}(L)$ is covered by a strictly decreasing chain. In either case, this conjecture is too weak to be used to obtain a characterization. As demonstrated by Example 6.19, for some languages $L_1, L_2 \subseteq \Sigma^*$ and patterns $\alpha_1, \alpha_2 \in X^+$ with $L_{\mathrm{E},\Sigma}(\alpha_i) \supseteq L_i$, there are multiple ways how α_1 and α_2 can be merged together to obtain a pattern β with $L_{\mathrm{E},\Sigma}(\beta) \supseteq L_1 \cup L_2$. Likewise, if one chooses L_1 and L_2 as languages where both do not have an $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive pattern (for example, by changing some exponents of the x_i used in the proof of Theorem 5.18), it might be possible that the two respective chains can be merged in a multitude of different ways (we revisit this in Section 7.5).

After some experimentation with various examples, the author arrived at the following conjecture:

Conjecture 7.3. *Let $|\Sigma| \geq 2$ and $L \subseteq \Sigma^*$. Then $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L) = \emptyset$ if and only if L is covered by a strictly decreasing L-reduced chain.*

Considering how strongly chains and the languages they cover appear to be related, it seemed natural to assume that the proof of Theorem 5.18 could be adapted to all these languages. First, the descending chain could lead to an ascending chain which every pattern from $\mathrm{Super}(L)$ has to cover (similarly to Lemma 5.24), and then, we could force every pattern from $\mathrm{Super}(L)$ somehow into this descending chain (similarly to the main part of the proof of Theorem 5.18).

As we shall see in Section 7.4, Conjecture 7.3 is disproved by a counterintuitive example, which we call the Loughborough Example.

Before we continue to Section 7.2, where we develop the tools that we use in the definition of the Loughborough Example, we take a closer look at another concept related to the (non-)existence of $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive patterns.

Both when finding $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive patterns for, or constructing L-reduced strictly decreasing chains over a language L, it is helpful to consider the set of all patterns that are potentially suitable for any of these two roles. To this end, we combine the concept of reduced patterns with $\mathrm{Super}(L)$, the set of all patterns in X^+ that generate a superset

of L (cf. Section 6.3.1): For every alphabet Σ and every language $L \subseteq \Sigma^*$, let the set of *reduced superpatterns* $\mathrm{RedSuper}(L)$ be defined as

$$\mathrm{RedSuper}(L) := \{\alpha \in \mathrm{Super}(L) \mid \alpha \text{ is } L\text{-reduced and in canonical form}\}.$$

As explained above, every L-reduced pattern is necessarily morphically primitive. The additional requirement that all patterns in $\mathrm{RedSuper}(L)$ must be in canonical form ensures that for all $\alpha, \beta \in \mathrm{RedSuper}(L)$ with $\alpha \neq \beta$, $L_{\mathrm{E},\Sigma}(\alpha) \neq L_{\mathrm{E},\Sigma}(\beta)$ must hold.

This allows us to state the following observation:

Proposition 7.4. *Let $|\Sigma| \geq 2$ and $L \subseteq \Sigma^*$. For every $\alpha \in \mathrm{RedSuper}(L)$, there are only finitely many $\beta \in \mathrm{RedSuper}(L)$ with $L_{\mathrm{E},\Sigma}(\beta) \supset L_{\mathrm{E},\Sigma}(\alpha)$.*

Proof. If $L = \Sigma^*$, then $\mathrm{RedSuper}(L)$ contains exactly one pattern (e. g., x, depending on the ordering used in the canonical normal form), and the claim follows immediately.

Assume $L \subset \Sigma^*$ and let $\alpha \in \mathrm{RedSuper}(L)$. For every $\beta \in \mathrm{RedSuper}(L)$ with $L_{\mathrm{E},\Sigma}(\beta) \supset L_{\mathrm{E},\Sigma}(\alpha)$, according to Lemma 5.1, every morphism $\phi : X^* \to X^*$ with $\phi(\beta) = \alpha$ is neither a renaming, nor an imprimitivity morphism. Furthermore, as β is L-reduced by definition, every such ϕ must be nonerasing.

Thus, $|\beta| \leq |\alpha|$ must hold. As $\mathrm{RedSuper}(L)$ contains only patterns that are of canonical form, it contains only finitely many patterns of any given length. More specifically, there are only finitely many $|\beta| \leq |\alpha|$, which proves the claim. □

Using Proposition 7.4, we can observe the following:

Observation 7.5. *For every $L \in \mathrm{ePAT}_{\mathrm{tf},\Sigma}$, $\mathrm{RedSuper}(L)$ is finite.*

This follows immediately from Proposition 7.4, as for every $L \in \mathrm{ePAT}_{\mathrm{tf},\Sigma}$, there is $\alpha \in X^+$ with $L_{\mathrm{E},\Sigma}(\alpha) = L$ that is morphically primitive (and, thusly, $L_{\mathrm{E},\Sigma}(\alpha)$-reduced).

Considering Proposition 7.4 and Observation 7.5, one might think that RedSuper could be used to obtain a necessary criterion for the existence of $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive patterns, e. g., something like the following conjecture:

Conjecture 7.6. *Let $|\Sigma| \geq 2$ and $L \subseteq \Sigma^*$. If $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L) \neq \emptyset$, then $\mathrm{RedSuper}(L)$ is finite.*

We shall see in Section 7.4 that the Loughborough Example disproves not only Conjecture 7.3, but also Conjecture 7.6.

As a side note, Proposition 7.4 allows us to define the *inclusion depth (with respect to L)* of a pattern $\alpha \in \mathrm{RedSuper}(L)$ as the length n of the longest sequence $(\alpha_i)_{i=0}^n$ over $\mathrm{RedSuper}(L)$ with $\alpha_0 = x$, $\alpha_n = \alpha$ and $L_{\mathrm{E},\Sigma}(\alpha_i) \supset L_{\mathrm{E},\Sigma}(\alpha_{i+1})$ for $0 \leq i < n$. Due to Proposition 7.4, the inclusion depth is always finite and well-defined. In general, E-patterns do not necessarily have a finite inclusion depth, while the inclusion depth of NE-patterns is always finite (cf. Luo [66]).

7.2 Chains, Chain Systems, and Their Languages

Given a decreasing chain $(\alpha_i)_{i=0}^\infty$ over X^+, we know due to Theorem 2.4 that, for every $i \geq 0$, there is a morphism $\phi_i : X^* \to X^*$ with $\phi_i(\alpha_i) = \alpha_{i+1}$. Moreover, as we consider self-reduced chains, all involved ϕ_i are nonerasing.

In order to study the languages that are generated by decreasing chains, it is useful to consider not only the patterns α_i in the chains, but the morphisms ϕ_i that connect these patterns. Note that, even if all involved patterns are morphically primitive, we cannot assume a priori that ϕ_i is unique, as there might be more than one morphism that maps some α_i to its successor α_{i+1}.

To avoid any confusion which morphism we consider for a given chain, we fix the starting patterns and the morphism, and derive the chain accordingly. As this concept is very similar to an extension of D0L systems (cf. Section 5.2.2) to infinite alphabets, we call this concept a chain system:

Definition 7.7. *A* chain system *is a tuple* $(\alpha_0, (\phi_i)_{i=0}^{\infty})$*, where* $\alpha_0 \in X^+$*, and every* ϕ_i *is a nonerasing morphism* $\phi_i : X^+ \to X^+$*. The corresponding decreasing chain* $(\alpha_i)_{i=0}^{\infty}$ *is defined by* $\alpha_{i+1} := \phi_i(\alpha_i)$*.*

For all $i, j \geq 0$*, we define the morphism* $\phi_{i,i+j} : X^+ \to X^+$ *inductively through*

$$\phi_{i,i} := \mathrm{id},$$
$$\phi_{i,i+j+1} := \phi_{i+j} \circ \phi_{i,i+j},$$

where id *is the identity morphism on* X^+ *(i. e.,* id *is defined by* $\mathrm{id}(x) := x$ *for all* $x \in X$*).*

It is easily seen by induction that $\phi_{i,i+j}(\alpha_i) = \alpha_{i,j}$ holds for all $i, j \geq 0$. In other words, $\phi_{i,i+j}$ is that morphism mapping α_i to α_{i+j} that is obtained by

$$\phi_{i,i+j} = \phi_{i+j-1} \circ \phi_{i+j-2} \circ \ldots \circ \phi_{i+1} \circ \phi_i$$

for sufficiently large j. Moreover, note that $\phi_i = \phi_{i,i+1}$, as (by definition), $\phi_i(\alpha_i) = \alpha_{i+1}$.

The author hopes that the use of the symbol ϕ in two different roles (with one index in the definition of chain systems, and with two indices for the resulting combined morphisms) is not confusing to the reader. In the current chapter, we use ϕ with one index mostly solely in the definitions of the chain systems we consider, and mainly work with the morphism ϕ with two indices, which we assume are defined according to Definition 7.7.

We use these morphisms to derive the following classification of variables:

Definition 7.8. *Let* $(\alpha_0, (\phi_i)_{i=0}^{\infty})$ *be a chain system. For every* $i \geq 0$*, we partition* $\mathrm{var}(\alpha_i)$ *into the three sets* $\mathcal{S}_i$*,* $\mathcal{E}_i$ *and* $\mathcal{H}_i$ *by defining for every* $x \in \mathrm{var}(\alpha_i)$*:*

1. $x :\in \mathcal{S}_i$ *if* $\exists c \geq 1 : \forall j \geq 0 : [|\phi_{i,i+j}(x)| \leq c]$*,*
2. $x :\in \mathcal{H}_i$ *if* $x \notin \mathcal{S}_i$ *and*

$$\forall c \geq 1 : \exists j \geq 0 : \forall y \in \mathrm{var}(\phi_{i,i+j}(x)) : [y \in \mathcal{S}_{i+j} \vee |\alpha_{i+j}|_y > c]\,,$$

3. $x :\in \mathcal{E}_i$ *if* $x \notin \mathcal{S}_i$ *and* $x \notin \mathcal{H}_i$*; i. e.,* $x \notin \mathcal{S}_i$ *and*

$$\exists c \geq 1 : \forall j \geq 0 : \exists y \in \mathrm{var}(\phi_{i,i+j}(x)) : [y \notin \mathcal{S}_{i+j} \wedge |\alpha_{i+j}|_y \leq c]\,.$$

We call variables from $\mathcal{S}_i$ stagnant*, from* $\mathcal{H}_i$ hyper-expanding*, and from* $\mathcal{E}_i$ expanding*.*

Note that by definition, for every $i \geq 0$, $\mathcal{S}_i$, $\mathcal{H}_i$ and $\mathcal{E}_i$ form a partition of $\text{var}(\alpha_i)$.

The intuition behind the choice of the terms stagnant, hyper-expanding and expanding is as follows: Similarly to L Systems, the application of each morphism ϕ_i can be seen as the passage of one step in time, and every single pattern can be understood as having grown from its predecessors, such that the whole chain system describes the growth of the patterns α_i towards some unspecified pattern in the limit. Consequently, each subpattern $\phi_{i,i+j}(x)$ (with $x \in \text{var}(\alpha_i)$ can be understood as having grown from x, and furthermore, one might view the variables $\text{var}(\phi_{i,i+j}(x))$ as successors or children of $x \in \text{var}(\alpha_i)$.

As such, stagnant variables have a limit on their growth. No matter how much time passes, the length of their image is always bounded by some constant c. Hyper-expanding variables show a behavior that is totally opposite to stagnant variables: Not only is there no limit to their growth, each of their non-stagnant successors continues to multiply without bound. Finally, expanding variables fall somewhere in the middle between the other two classes. On the one hand, their growth is not bounded, on the other hand, they still have something that might be considered a unique successor.

Most examples the author considered prior to the formal definition of these partitions of $\text{var}(\alpha_i)$ used only stagnant and hyper-expanding variables. For example, we shall see in Section 7.3.1 that the patterns used in the proof of Theorem 5.18 did not use any $\mathcal{E}$-variables, and neither do any of the examples we consider later in this chapter. As we briefly discuss further down in the present section and also in Section 7.2.1, $\mathcal{E}$-variables introduce additional technical difficulties.

By their definition, chain systems can be understood as an extension of D0L systems to infinite alphabets (cf. Section 5.2.2). As we shall see in Theorem 7.19, the languages that are generated by chains that are defined through chain systems might be understood as multi-pattern languages (cf. Dumitrescu et al. [25]) that use a set of patterns that is generated by a restricted kind of HD0L system on an infinite alphabet. We observed a similar phenomenon in the proof of Theorem 5.18.

The following observation follows immediately when forming the complement of the definition of $\mathcal{S}_i$:

Observation 7.9. *Let $(\alpha_0, (\phi_i)_{i=0}^{\infty})$ be a chain system. For every $i \geq 0$, and every $x \in \text{var}(\alpha_i)$, $x \notin \mathcal{S}_i$ holds if and only if*

$$\forall c \geq 1 : \exists j \geq 0 : [|\phi_{i,i+j}(x)| > c] .$$

Furthermore, as all involved morphisms are nonerasing by definition, we observe the following:

Observation 7.10. *Let $(\alpha_0, (\phi_i)_{i=0}^{\infty})$ be a chain system, and let $c \geq 1$, $i \geq 0$ and $x \in \mathcal{H}_i$. If*

$$\forall y \in \text{var}(\phi_{i,i+j}(x)) : [y \in \mathcal{S}_{i+j} \vee |\alpha_{i+j}|_y > c]$$

holds for some $j \geq 0$, then

$$\forall y \in \text{var}(\phi_{i,i+k}(x)) : [y \in \mathcal{S}_{i+k} \vee |\alpha_{i+k}|_y > c]$$

holds for every $k \geq j$.

As mentioned above, if we view the application of each morphism ϕ_i as one step in time, we can consider the variables in $\phi_{i,i+j}(x)$ (in α_{i+j}) as successors of the variable x (in α_i). We can then conclude, almost directly from Definition 7.8, that $\mathcal{S}$-variables grow only into $\mathcal{S}$-variables, while $\mathcal{H}$-variables never grow into $\mathcal{E}$-variables:

Lemma 7.11. *Let $(\alpha_0, (\phi_i)_{i=0}^{\infty})$ be a chain system. For every $i \geq 0$, and every $x \in \mathrm{var}(\alpha_i)$, the following holds:*

1. *If $x \in \mathcal{S}_i$, then $\mathrm{var}(\phi_i(x)) \subseteq \mathcal{S}_{i+1}$,*
2. *if $x \in \mathcal{H}_i$, then $\mathrm{var}(\phi_i(x)) \subseteq (\mathcal{S}_{i+1} \cup \mathcal{H}_{i+1})$.*

Proof. We begin with the first claim. Let $i \geq 0$ and $x \in \mathcal{S}_i$. By definition, there is a $c_x \geq 1$ such that $|\phi_{i,i+j}(x)| \leq c_x$ for all $j \geq 0$. For the sake of contradiction, assume there exists a $y \in \mathrm{var}(\phi_i(x))$ with $y \notin \mathcal{S}_{i+1}$. Then, according Observation 7.9, there is a $j \geq 0$ such that $|\phi_{i+1,i+1+j}(y)| > c_x$. As y is a factor of $\phi_i(x)$, and as $\phi_{i,i+1+j} = \phi_{i+1,i+1+j} \circ \phi_i$, we observe that $\phi_{i+1,i+1+j}(y)$ is a factor of $\phi_{i,i+1+j}(x)$. Hence, and as every ϕ_i is nonerasing, it follows that

$$|\phi_{i,i+1+j}(x)| \geq |\phi_{i+1,i+1+j}(y)| > c_x,$$

which yields the intended contradiction.

The proof for the second claim can be executed analogously: Assume there are an $i \geq 0$, an $x \in \mathcal{H}_i$ and a $y \in \mathrm{var}(\phi_i(x))$ with $y \in \mathcal{E}_{i+1}$. According to the definition of $\mathcal{E}_{i+1}$, there is a $c_y \geq 1$ such that, for every $j \geq i+1$, there is a $z \in \mathrm{var}(\phi_{i+1,j})$ such that $z \notin \mathcal{S}_j$ and $|\alpha_j|_z \leq c_y$. By definition of $\mathcal{H}_i$ and Observation 7.10, we can choose $j \geq i+1$ large enough that, for every $z \in \mathrm{var}(\phi_{i,j}(x))$, $z \in \mathcal{S}_j$ or $|\alpha_j|_z > c_y$ holds. As $\mathrm{var}(\phi_{i,j}(x)) \supseteq \mathrm{var}(\phi_{i+1,j}(y))$, this is a contradiction. □

Note that $\mathcal{E}$-variables can be followed by all three variable types. On the other hand, we can observe that every $\mathcal{H}$-variable and every $\mathcal{E}$-variable must have a successor of the same type:

Lemma 7.12. *Let $(\alpha_0, (\phi_i)_{i=0}^{\infty})$ be a chain system. For every $i \geq 0$ and every $x \in \mathrm{var}(\alpha_i)$, the following holds:*

1. *If $x \in \mathcal{H}_i$, then $\phi_i(x)$ contains at least one variable from $\mathcal{H}_{i+1}$,*
2. *if $x \in \mathcal{E}_i$, then $\phi_i(x)$ contains at least one variable from $\mathcal{E}_{i+1}$.*

Proof. We begin with the first claim and consider any $i \geq 0$ and any $x \in \mathcal{H}_i$. Due to Lemma 7.11, we already know that $\mathrm{var}(\phi_i(x)) \subseteq (\mathcal{S}_{i+1} \cup \mathcal{H}_{i+1})$. Assume for the sake of contradiction that $y \in \mathcal{S}_{i+1}$ for every $y \in \mathrm{var}(\phi_i(x))$. According to Definition 7.8, for every such y, there is a $c_y \geq 0$ such that $|\phi_{i+1,i+1+j}(y)| \leq c_y$ for every $j \geq 0$. We choose

$$c := \max\{c_y \mid y \in \mathrm{var}(\phi_i(x))\}$$

and observe that, for every $j \geq 0$,

$$|\phi_{i,i+1+j}(x)| \leq |\phi_i(x)|c.$$

Therefore, $x \in \mathcal{S}_i$ holds, which contradicts our initial assumption of $x \in \mathcal{H}_i$.

For the second claim, assume there exist an $i \geq 0$ and an $x \in \mathcal{E}_i$ such that $(\text{var}(\phi_i(x)) \cap \mathcal{E}_{i+1}) = \emptyset$. Let $c \geq 1$ be a constant that satisfies the definition of $\mathcal{E}_i$ for x. Next, we consider the set

$$H_x := \{y \in \text{var}(\phi_i(x)) \mid y \in \mathcal{H}_i\}.$$

By Definition 7.8, $\text{var}(\phi_i(x))$ contains at least one variable that is not from $\mathcal{S}_{i+1}$; together with our assumption that no variable from $\phi_i(x)$ is in $\mathcal{E}_{i+1}$, it follows that H_x is not empty. Furthermore, for every $y \in H_x$, there is an $n_y \geq i+1$ such that, for every $j \geq 0$ and for all $z \in \text{var}(\phi_{i+1,n_y+j}(x))$, one of $z \in \mathcal{S}_{n_y+j}$ or $|\alpha_{n_y+j}|_z \geq c$ holds. We choose

$$n := \max\{n_y \mid y \in H_x\}$$

and observe that, due to our choice of all n_y and due to Lemma 7.11, for every $z \in \text{var}(\phi_{i,n}(x))$, $z \in \mathcal{S}_n$ or $|\alpha_n|_z > c$ holds, which contradicts our choice of c and the definition of $\mathcal{E}_i$. □

As mentioned above, the presence of $\mathcal{E}$-variables might lead to technical difficulties. Therefore, we mostly consider chain systems where no expanding variables are present. We say that the chain system $(\alpha_0, (\phi_i)_{i=0}^{\infty})$ is *$\mathcal{E}$-free* if $\mathcal{E}_i = \emptyset$ for every $i \geq 0$; and a decreasing chain is $\mathcal{E}$-free if it corresponds to a chain system that is $\mathcal{E}$-free.

The following lemma can be used to simplify the proof that a given chain system is $\mathcal{E}$-free:

Lemma 7.13. *A chain system $(\alpha_0, (\phi_i)_{i=0}^{\infty})$ is $\mathcal{E}$-free if and only if $\mathcal{E}_0 = \emptyset$.*

Proof. The *if* direction follows from Lemma 7.11, as for every $x \in (\mathcal{S}_0 \cup \mathcal{H}_0)$, and for every $i \geq 0$, $\text{var}(\phi_{0,i}(x)) \subseteq (\mathcal{S}_i \cup \mathcal{H}_i)$. Thus, if $\mathcal{E}_0 = \emptyset$, $\mathcal{E}_i = \emptyset$ for all $i \geq 0$.

We show the *only if* direction using its contraposition: Assume $\mathcal{E}_0 \neq \emptyset$. By Lemma 7.12, this immediately implies $\mathcal{E}_1 \neq \emptyset$, and by induction, $\mathcal{E}_i \neq \emptyset$ for all $i \geq 0$. □

One main advantage of $\mathcal{E}$-free chain systems is that in these systems, the behavior of $\mathcal{H}$-variables is more predictable. We can use this to show that the number of occurrences of any given $\mathcal{H}$-variable must grow with α_i:

Lemma 7.14. *Let $(\alpha_0, (\phi_i)_{i=0}^{\infty})$ be an $\mathcal{E}$-free chain system. For every $c \geq 1$, there is an $n \geq 0$ such that $|\alpha_{n+j}|_x > c$ for every $j \geq 0$ and every $x \in \mathcal{H}_{n+j}$.*

Proof. Let $c \geq 1$. By Definition 7.8, for every $x \in \mathcal{H}_0$, there is an $n_x \geq 0$ such that $y \in \mathcal{S}_{n_x}$ or $|\alpha_{n_x}|_y > c$ is satisfied for every $y \in \text{var}(\phi_{0,n_x}(x))$. Less formally, n_x is the point where every successor of x is a stagnant variable, or occurs more frequently than c times in α_{n_x}.

Now let $n := \max\{n_x \mid x \in \mathcal{H}_0\}$. According to Lemma 7.11, for every $x \in \mathcal{S}_0$ and every $j \geq 0$, $\phi_{0,j}(x) \subseteq \mathcal{S}_j$ (as all successors of $\mathcal{S}$-variables are $\mathcal{S}$-variables as well). Thus, for every $y \in \mathcal{H}_n$, there is an $x \in \mathcal{H}_0$ with $y \in \text{var}(\phi_{0,n}(x))$, and by our choice of n, $|\alpha_n|_y \geq |\alpha_{n_x}|_y > c$ must hold.

As all morphisms $\phi_{i,i+j}$ are nonerasing, $|\alpha_{n+j}|_y > c$ holds for all $j \geq 0$ and all $y \in \mathcal{H}_{n+j}$. □

As an immediate consequence, we observe that if $(\alpha_i)_{i=0}^{\infty}$ is an $\mathcal{E}$-free chain, for every $w \in L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty})$, there is an $n \geq 0$ such that $L_{\mathrm{E},\Sigma}(\pi_{(\mathcal{S}_n)}(\alpha_n))$ (by choosing $c{:=}|w|$, and as every x with $|\alpha_n|_x > |w|$ must be erased when generating w from α_n).

Nonetheless, we cannot simply reconstruct $L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty})$ from $\mathcal{S}$-variables, as the chain might not always be 'tight' enough. This is illustrated by the following example:

Example 7.15. *Let $\alpha_0{:=}y^2$, $\alpha_{i+1}{:=}\phi(\alpha_i)$, where the morphism $\phi : X^+ \to X^+$ is defined by*

$$\phi(y){:=}y^2x_1^2, \qquad \phi(x_1){:=}x_2^2, \qquad \phi(x_{i+1}){:=}\phi(x_{i+2})$$

for every $i \geq 2$. We now define a second chain $(\alpha'_i)_{i=0}^{\infty}$ by $\alpha'_0{:=}y^2$ and $\alpha'_{i+1}{:=}\phi'(\alpha'_i)$, where

$$\phi'(y){:=}y^2x_1^4, \qquad \phi'(x_i){:=}x_{i+1}$$

for every $i \geq 1$. Consider the first few patterns of both chains:

$$\begin{aligned}
\alpha_0 &= y^2, & \alpha'_0 &= y^2,\\
\alpha_1 &= (y^2x_1^2)^2, & \alpha'_1 &= (y^2x_1^4)^2,\\
\alpha_2 &= ((y^2x_1^2)^2x_2^4)^2, & \alpha'_2 &= ((y^2x_1^4)^2x_2^4)^2,\\
\alpha_3 &= (((y^2x_1^2)^2x_2^4)^2x_3^4)^2, & \alpha'_3 &= (((y^2x_1^4)^2x_2^4)^2x_3^4)^2,\\
&\vdots & &\vdots\\
\alpha_i &= ((\dots((y^2x_1^2)^2x_2^4)^2\dots x_{i-1}^4)^2x_i^4)^2, & \alpha'_i &= ((\dots((y^2x_1^4)^2x_2^4)^2\dots x_{i-1}^4)^2x_i^4)^2.
\end{aligned}$$

It is easy to see that both chains generate the same language

$$L{:=}L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty}) = L_{\mathrm{E},\Sigma}((\alpha'_i)_{i=0}^{\infty}).$$

Notice that for both chains in every step, y is a $\mathcal{H}$-variable, while all x_i that occur are $\mathcal{S}$-variables. In Theorem 7.19, we shall see that L can also be expressed as

$$L = \bigcup_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(\pi_{\{x_1,\dots,x_i\}}(\alpha'_i)).$$

On the other hand, $\pi_{\{x_1\}}(\alpha_1) = x_1^4$, and we can easily see that $L \not\supseteq L_{\mathrm{E},\Sigma}(x_1^4)$, as, for example, $L_{\mathrm{E},\Sigma}(\alpha_2) \not\supseteq L_{\mathrm{E},\Sigma}(x_1^4)$, while $L_{\mathrm{E},\Sigma}(\alpha_i) \supseteq L$ must hold for every $i \geq 0$ by definition.

Intuitively, in every step of the first chain, each variable x_1 still needs to "grow" in another step, while the x_1 in the second chain have already reached their final form and reflect the structure of L. From this point of view, the first chain is not tight enough to truly reflect the structure of its language. ◇

As Example 7.15 shows, in order to obtain the language that is generated by an $\mathcal{E}$-free chain from that chain, we need to wait until the $\mathcal{S}$-variables have stopped growing. We formalize this using the concept of adult variables:

Definition 7.16. *Let $(\alpha_0, (\phi_i)_{i=0}^{\infty})$ be an $\mathcal{E}$-free chain system. For every $i \geq 0$, we say that $x \in \mathrm{var}(\alpha_i)$ is* adult (in α_i) *if $x \in \mathcal{S}_i$ and*

$$\forall j \geq 0 : \left[|\phi_{i,i+j}(x)| = 1 \wedge |\alpha_{i+j}|_{\phi_{i,i+j}(x)} = |\alpha_{i+j}|_x \right].$$

We use $\mathcal{A}_i$ to denote the set of all variables that are adult in α_i.

The term and notion of adult variables is inspired by the terminology of L Systems (cf. Section 5.2.2), where a word is called adult if it does not change under any application of the permitted rules (cf. Kari et al. [53]). Intuitively, adult variables can also be viewed as having stopped changing: By the first condition, they have reached their maximal expansion rate (every image is a single variable), and moreover, the second condition requires that this variable has reached its maximal number of occurrences (although it might be renamed in every step, like the x_i in the chain $(\alpha'_i)_{i=0}^{\infty}$ Example 7.15).

Adult variables play an important role in $\mathcal{E}$-free chain systems – as we shall see, every word in $L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty})$ is (ultimately) generated only by adult variables. We prove this using a version of Lemma 7.14 for $\mathcal{A}$-variables:

Lemma 7.17. *Let $(\alpha_0, (\phi_i)_{i=0}^{\infty})$ be an $\mathcal{E}$-free chain system. For every $c \geq 1$, there is an $n \geq 0$ such that for all $x \in \mathrm{var}(\alpha_n)$, $x \in \mathcal{A}_n$ or $|\alpha_n|_x > c$ holds.*

Proof. Let $c \geq 1$. According to Lemma 7.14, there is an $n_0 \geq 0$ such that $|\alpha_{n_0+j}|_x > c$ holds for every $j \geq 0$ and every $x \in \mathcal{H}_{n_0+j}$. Next, we choose an $n_1 \geq n_0$ such that $|\phi_{n_1,n_1+j}(y)| = 1$ holds for every $x \in \mathcal{S}_{n_0}$, every $y \in \mathrm{var}(\phi_{n_0,n_1}(x))$, and every $j \geq 0$. Such an n_1 must exist, as $|\phi_{n_1,n_1+j}(x)|$ is bounded (following directly from the definition of $\mathcal{S}_{n_1}$), and all involved morphisms are nonerasing by definition. Less formally, we find the point n_1 at which all variables from $\mathcal{S}_{n_0}$ have reached their maximal expansion.

We now define the set $S_{n_0,n_1} := \bigcup_{x \in \mathcal{S}_{n_0}} \mathrm{var}(\phi_{n_0,n_1}(x))$. In other words, S_{n_0,n_1} contains all those variables in $\mathcal{S}_{n_1}$ that have grown from variables in S_{n_0} and have only images of length 1; meaning that those variables satisfy the first criterion of adult variables.

Now, for every $y \in S_{n_0,n_1}$, we select an $n_y \geq n_1$ such that one of the following two conditions is met:

1. $|\alpha_{n_y}|_{\phi_{n_1,n_y}(y)} = |\alpha_{n_y+j}|_{\phi_{n_1,n_y+j}(y)}$ for every $j \geq 0$, or
2. $|\alpha_{n_y}|_{\phi_{n_1,n_y}} > c$.

Recall that, by our choice of n_1, $|\phi_{n_1,n_1+j}(y)| = 1$ for every $j \geq 0$ and every $y \in S_{n_0,n_1}$. Also, as all involved morphisms are nonerasing, $|\alpha_{n_1+j+1}|_{\phi_{n_1,n_1+j+1}(y)} \geq |\alpha_{n_1+j}|_{\phi_{n_1+j}(y)}$ holds for every $j \geq 0$, which means that such a n_y can always be found. We define

$$n := \max\{n_y \mid y \in S_{n_0,n_1}\},$$

and verify that this n satisfies the claim: If $x \in \mathcal{H}_n$, then $|\alpha_n|_x > c$, as $n \geq n_0$ and by our choice of n_0. If $x \in \mathcal{S}_n$, we distinguish two cases. First, if there is a $y \in \mathcal{H}_{n_0}$ with $x \in \mathrm{var}(\phi_{n_0,n}(y))$, $|\alpha_n|_x \geq |\alpha_{n_0}|_y > c$ follows. Otherwise, there is a $y \in \mathcal{S}_{n_0}$ with $x \in \mathrm{var}(\phi_{n_0,n}(y))$. Therefore, as $n \geq n_1$ and by our choice of n_1, $|\phi_{n,n+j}(x)| = 1$ holds for every $j \geq 0$. Finally, as $n \geq n_z$ for every $z \in S_{n_0,n_1}$, we know that x satisfies the first condition in the choice of the n_z, which means that $x \in \mathcal{A}_n$, or the second condition (and thus, $|\alpha_n|_x > c$). □

This implies that, for every $w \in L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty})$, there is an $n \geq 0$ such that $w \in L_{\mathrm{E},\Sigma}(\pi_{\mathcal{A}_n}(\alpha_n))$, simply by selecting $c{:=}|w|$. Thus, removing from each of the patterns of the chains all of its non-adult variables does not lead to a loss of expressive power, as long as all patterns of the chain are available.

As Example 7.15 shows, considering all $\pi_{\mathcal{S}_i}(\alpha_i)$ might generate words outside of $L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty})$. On the other hand, projecting the patterns only to adult variables leads to an ascending chain of patterns that is tight enough to stay inside of $L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty})$:

Lemma 7.18. *Let Σ be an alphabet with $|\Sigma| \geq 2$, and let $(\alpha_0, (\phi_i)_{i=0}^{\infty})$ be an $\mathcal{E}$-free chain system. For every $i \geq 0$, $L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty}) \supseteq L_{\mathrm{E},\Sigma}(\pi_{\mathcal{A}_i}(\alpha_i))$.*

Proof. Let $i \geq 0$. We assume that $\pi_{\mathcal{A}_i}(\alpha_i) \neq \lambda$, as otherwise, the claim follows immediately. In order to show that, for every $j \geq 0$, $L_{\mathrm{E},\Sigma}(\alpha_{i+j}) \supseteq L_{\mathrm{E},\Sigma}(\pi_{\mathcal{A}_i}(\alpha_i))$, we consider every $x \in \mathcal{A}_i$, and define a respective $y_x \in \mathrm{var}(\alpha_{i+j})$ in the following way:

As $x \in \mathcal{A}_i$, we know by Definition 7.16 that $|\phi_{i,i+j}(x)| = 1$, and define $y_x{:=}\phi_{i,i+j}(x)$. Note that then, also by definition, $|\alpha_{i+j}|_{y_x} = |\alpha_i|_x$ holds, which implies that $y_x \notin \mathrm{var}(\phi_{i,i+j}(z))$ for every $z \in \mathrm{var}(\alpha_i)$ with $z \neq x$.

We now define a morphism $\psi_m : X^* \to X^*$ by

$$\psi_j(y){:=}\begin{cases} x & \text{if } y = y_x \text{ for some } x \in \mathcal{A}_i, \\ \lambda & \text{otherwise.} \end{cases}$$

As there is a one-to-one correspondence between $\mathcal{A}_i$ and $\{y_x \mid x \in \mathcal{A}_i\}$, $\psi_j(\alpha_{i,i+j}) = \pi_{\mathcal{A}_i}(\alpha_i)$ follows, and according to Theorem 2.4, $L_{\mathrm{E},\Sigma}(\pi_{\mathcal{A}_j}(\alpha_j)) \subseteq L_{\mathrm{E},\Sigma}(\alpha_{i+j})$. As j was chosen freely, we conclude

$$\begin{aligned} L_{\mathrm{E},\Sigma}(\pi_{\mathcal{A}_i}(\alpha_i)) &\subseteq \bigcap_{j=0}^{\infty} L_{\mathrm{E},\Sigma}(\alpha_{i+j}) \\ &= \bigcap_{j=0}^{\infty} L_{\mathrm{E},\Sigma}(\alpha_j) = L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty}). \end{aligned}$$

□

By combining Lemmas 7.17 and 7.18, we are able to show that, for $\mathcal{E}$-free chains, adult variables allow us to obtain the language that is generated by the chain from the patterns of the chain:

Theorem 7.19. *Let Σ be an alphabet with $|\Sigma| \geq 2$, and let $(\alpha_0, (\phi_i)_{i=0}^{\infty})$ be an $\mathcal{E}$-free chain system. Then $L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty}) = \bigcup_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(\pi_{\mathcal{A}_i}(\alpha_i))$.*

Proof. First, note that $L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty}) \supseteq \bigcup_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(\pi_{\mathcal{A}_i}(\alpha_i))$ holds due to Lemma 7.18.

In order to show $L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty}) \subseteq \bigcup_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(\pi_{\mathcal{A}_i}(\alpha_i))$, consider any $w \in L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty})$. By definition of $L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty})$, $w \in L_{\mathrm{E},\Sigma}(\alpha_i)$ for all $i \geq 0$. According to Lemma 7.17, there is a $i \geq 0$ such that $|\alpha_i|_x > |w|$ for every $x \in \mathrm{var}(\alpha_i)$ that is not adult. Thus, $w \in L_{\mathrm{E},\Sigma}(\pi_{\mathcal{A}_i}(\alpha_i))$ must hold, as for every morphism $\sigma : X^* \to \Sigma^*$ with $\sigma(\alpha) = w$, $\sigma(x) = \lambda$ for every $x \in \mathrm{var}(\alpha_i)$ with $|\alpha_i|_x > |w|$. As n was chosen freely, this proves $w \in \bigcup_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(\pi_{\mathcal{A}_i}(\alpha_i))$, and, together with the fact that w was chosen freely, concludes the proof. □

As mentioned before, Theorem 7.19 demonstrates that the language that is generated by the descending chain that is associated to an $\mathcal{E}$-free chain system can be understood as a HD0L system over an infinite alphabet, where the final morphism (which puts the 'H' in 'HD0L') is restricted to be a very specific projection morphism[2]. First, note that, as we are dealing with infinite variable alphabets, all involved variables can be renamed, such that $\mathrm{var}(\alpha_i)$ and $\mathrm{var}(\alpha_j)$ are disjoint for all $i \neq j$. Thus, instead of using an infinite sequence $(\phi_i)_{i=0}^{\infty}$ of morphisms, we can use a single morphism ϕ for each step of the chain, and a single morphism π instead of each $\pi_{\mathcal{A}_i}$.

An application of Theorem 7.19 to the chain used in the proof of Theorem 5.18 in Section 5.4.2 can be found in Section 7.3.1. Generally, we observe that the phenomenon we used in Lemma 5.24 (that languages can be expressed using ascending or descending chain) can be generalized to all $\mathcal{E}$-free chains: A language that is generated by such a strictly descending chain of patterns can also be expressed as the union of languages generated by the patterns of an ascending chain. In Section 7.3.1, we briefly sketch how this result can be used to generalize the proof of Theorem 5.18 to a certain class of $\mathcal{E}$-free chains. Nonetheless, this does not immediately lead to a proof of Conjecture 7.2, as Theorem 7.19 does not deal with the more complicated case of chains that are not $\mathcal{E}$-free.

Before we continue in Section 7.3 (and thereafter) with our observations on the existence of ePAT$_{\text{tf},\Sigma}$-descriptive patterns for languages that are generated or covered by descending chains, we take a brief look at an example of a chain system that uses $\mathcal{E}$-variables.

7.2.1 A Chain System That Is Not $\mathcal{E}$-Free

The chains we considered up to now and the chains that are used in the following sections of the present chapter are mostly $\mathcal{E}$-free. In order to give a broader view of chain systems and their chains, the present section contains a chain system that is not $\mathcal{E}$-free. This short example is not necessary to understand any other part of the present chapter (or even the present thesis) and serves solely as additional background for those who are deeply interested in chain systems. All others are invited to skip the present section, and to continue in Section 7.3.

Example 7.20. *Let z, y and $x_1, x_2, \ldots$ be pairwise distinct variables. We define a chain system $(\alpha_0, (\phi_i)_{i=0}^{\infty})$ through $\alpha_0{:=}zy^2z$ and $\phi_i = \phi$ for all $i \geq 0$, where the morphism $\phi : X^+ \to X^+$ is defined by*

$$\phi(y){:=}y^2x_1^2, \qquad \phi(z){:=}zx_1^2, \qquad \phi(x_i){:=}x_{i+1}$$

[2]The most interesting part of this observation is that the final morphism is restricted. If this morphism is not restricted in any way, the observation becomes trivial: Let Σ be any terminal alphabet, and let $X = \{x_0, x_1, \ldots\}$ be disjoint from Σ. Consider any language $L \subseteq \Sigma^*$, and let $(w_i)_{i=0}^{\infty}$ be any sequence with $\{w_i\} = L$. Let the morphisms $\phi : X^+ \to X^+$ and $\psi : X^* \to \Sigma^*$ be defined by $\phi(x_i){:=}x_{i+1}$ and $\psi(x_i){:=}w_i$ for all $i \geq 0$. Then L is generated by the HD0L system with iterated morphism ϕ, final morphism ψ and axiom x_0. Thus, it is important to note that we restrict the final morphism to be a projection morphism.

for every $i \geq 1$. The resulting chain $(\alpha_i)_{i=0}^{\infty}$ starts as follows:

$$\begin{aligned}
\alpha_1 &= zx_1^2 \cdot (y^2x_1^2)^2 \cdot zx_1^2,\\
\alpha_2 &= zx_1^2x_2^2 \cdot ((y^2x_1^2)^2x_2^2)^2 \cdot zx_1^2x_2^2,\\
\alpha_3 &= zx_1^2x_2^2x_3^2 \cdot (((y^2x_1^2)^2x_2^2)^2x_3^2)^2 \cdot zx_1^2x_2^2x_3^2,\\
&\vdots
\end{aligned}$$

Generally, we observe that, for sufficiently large i,

$$\alpha_i = zx_1^2x_2^2 \dots x_{i-1}^2x_i^2 \cdot ((\dots((y^2x_1^2)^2x_2)^2 \dots x_{i-1}^2)^2x_i^2)^2 \cdot zx_1^2x_2^2 \dots x_{i-1}^2x_i^2.$$

Now, note that, for all $i, j \geq 0$ and every $i \geq 1$, $|\phi_{i,i+j}(x_i)| = |x_{i+j}| = 1$. Thus, if $x_i \in \mathrm{var}(\alpha_i)$, then $x_i \in \mathcal{S}_i$.

Next, we observe that, for all $i \geq 0$, $|\alpha_i|_y = |\phi_{0,i}(y)|_y = 2^{1+i}$. Therefore, for every $i \geq 0$ and every $c \geq 1$, we can select an appropriate $j \geq 0$ with $|\alpha_{i+j}|_y = 2^{1+i+j} > c$. Then $\mathrm{var}(\phi_{i+j}(y)) = \{y, x_1, \dots, x_j\}$, which means that $y \in \mathcal{H}_i$ for every $i \geq 0$.

In order to show that $z \in \mathcal{E}_i$ for every $i \geq 0$, observe that for every $j \geq 0$,

1. *$|\phi_{i,i+j}(z)| = 2j + 1$ (which implies $z \notin \mathcal{S}_i$), and*
2. *$z \in \phi_{i,i+j}(z)$, $z \notin \mathcal{S}_{i+j}$, and $|\alpha_{i+j}|_z = 2$.*

Hence, $z \in \mathcal{E}_i$ for every $i \geq 0$, and therefore, the chain system is not $\mathcal{E}$-free. ◇

If one uses the same definition of adult variables as for $\mathcal{E}$-free chain systems, the chain system in Example 7.20 would have $\mathcal{A}_i = \mathcal{S}_i = \{x_1, \dots, x_i\}$ for every $i \geq 0$. On the other hand, it is easy to see that $L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty}) = \bigcup_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(\pi_{\mathcal{A}_i \cup \{z\}}(\alpha_i))$. Accordingly, in order to extend Theorem 7.19 to general chain systems, one would need to extend the notion of adult variables. Unlike for $\mathcal{E}$-free chain system, this extension would probably need to include $\mathcal{H}$-variables, as Lemma 7.14 does not apply when $\mathcal{E}$-variables are allowed to introduce new $\mathcal{H}$-variables.

Also note that, when dealing with L-reduced or self-reduced chains, morphisms involving $\mathcal{E}$-variables are dangerously close to morphic imprimitivity (as briefly explained in Section 7.3 in the the comment after Lemma 7.23). As $\mathcal{E}$-variables are required to have a unique successor in every morphic image, it is quite easy to construct examples that generate chains of patterns that are morphically imprimitive and, thus, not reduced (cf. Observation 7.1 for a short discussion of reduced patterns and morphic primitivity). The author spent far too much time on chains of morphically imprimitive patterns without noticing their imprimitivity; realizing far too often, far too late that each of this chains was not a strictly descending chain, but a chain of patterns that all generated the same language.

7.3 The Languages $L_\Sigma^{(k)}$

This section defines the languages $L_\Sigma^{(k)}$, which we later use as building blocks for the examples in Sections 7.4 and 7.5. In a way, these languages can be viewed as simplified generalizations of the language L_Σ that is used in the proof of Theorem 5.18, as this language and the languages $L_\Sigma^{(k)}$ are generated by chains that arise from similar chain systems. We define the languages $L_\Sigma^{(k)}$ as follows:

Definition 7.21. *Let y and all x_i with $i \geq 0$ be pairwise distinct elements of X. For every $k \geq 2$, we define the chain system $(\hat{\alpha}_{k,0}, (\hat{\phi}_{k,i})_{i=0}^{\infty})$*

$$\hat{\alpha}_{k,0} := y^2,$$
$$\hat{\phi}_{k,i} := \hat{\phi}_k$$

for every $i \geq 0$, where the morphism $\hat{\phi}_k : X^+ \to X^+$ is defined through

$$\hat{\phi}_k(y) := y^2 x_1^k,$$
$$\hat{\phi}_k(x_i) := x_{i+1}$$

for every $i \geq 0$. Accordingly, we define the increasing chain $(\hat{\beta}_{k,i})_{i=0}^{\infty}$ through $\hat{\beta}_{k,i} := \psi(\hat{\alpha}_{k,i})$, where $\psi(y) := \lambda$ and $\psi(x_i) := x_i$ for every $i \geq 0$. Finally, for any alphabet Σ, let

$$L_{\Sigma}^{(k)} := \bigcup_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(\hat{\beta}_{k,i}).$$

We illustrate this definition using the following example:

Example 7.22. *For example, the chains $(\hat{\alpha}_{3,i})_{i=0}^{\infty}$ and $(\hat{\beta}_{3,i})_{i=0}^{\infty}$ start as follows:*

$$\begin{aligned}
\hat{\alpha}_{3,0} &= (y)^2, & \hat{\beta}_{3,0} &= \lambda,\\
\hat{\alpha}_{3,1} &= ((y)^2 x_1^3)^2, & \hat{\beta}_{3,1} &= (x_1^3)^2,\\
\hat{\alpha}_{3,2} &= (((y)^2 x_1^3)^2 x_2^3)^2, & \hat{\beta}_{3,2} &= ((x_1^3)^2 x_2^3)^2,\\
\hat{\alpha}_{3,3} &= ((((y)^2 x_1^3)^2 x_2^3)^2 x_3^3)^2, & \hat{\beta}_{3,3} &= (((x_1^3)^2 x_2^3)^2 x_3^3)^2,\\
&\vdots & &\vdots
\end{aligned}$$

◇

Generally, for any $k \geq 2$ and sufficiently large values of i, we observe

$$\hat{\alpha}_{k,i} = ((\ldots((((y^2 x_1^k)^2) x_2^k)^2 x_3^k)^2 \ldots x_{i-1}^k)^2 x_i^k)^2,$$
$$\hat{\beta}_{k,i} = ((\ldots((((x_1^k)^2) x_2^k)^2 x_3^k)^2 \ldots x_{i-1}^k)^2 x_i^k)^2.$$

Using the tools we developed in Section 7.2, it is easy to make the following straightforward observations on each chain system $(\hat{\alpha}_{k,0}, (\hat{\phi}_{k,i})_{i=0}^{\infty})$ and its associated language $L_{\Sigma}^{(k)}$:

Lemma 7.23. *For every $k \geq 2$, let the chain system $(\hat{\alpha}_{k,0}, (\hat{\phi}_{k,i})_{i=0}^{\infty})$ be defined as in Definition 7.21. The following holds:*

1. *For every $i \geq 0$, $\mathcal{S}_i = \{x_j \mid 1 \leq j \leq i\}$.*
2. *For every $i \geq 0$, $\mathcal{A}_i = \mathcal{S}_i$.*
3. *For every $i \geq 0$, $\mathcal{H}_i = \{y\}$.*

4. *For every* $i \geq 0$, $\mathcal{E}_i = \emptyset$.

5. $L_{\mathrm{E},\Sigma}((\hat{\alpha}_{k,i})_{i=0}^{\infty}) = L_\Sigma^{(k)}$.

6. $(\hat{\alpha}_{k,i})_{i=0}^{\infty}$ *is* $L_\Sigma^{(k)}$*-reduced.*

Proof. First, observe that one can easily verify by induction that, for all $i \geq 0$,

$$|\alpha_i|_y = 2^{i+2},$$

and, for every $j \geq 1$,

$$|\alpha_i|_{x_j} = \begin{cases} 0 & \text{if } j > i, \\ k|\alpha_{i-j}|_y & \text{if } j \leq i. \end{cases}$$

Moreover, for all $i \geq 0$ and all $x_j \in \mathrm{var}(\alpha_i)$, $\phi_{i+l}(x_j) = x_{j+l}$ holds for all $l \geq 0$, which immediately gives $|\phi_{i+l}(x_j)|_{x_{j+l}} = 1$ and thus proves claim 1. Furthermore, we can also conclude that $|\alpha_i|_{x_j} = |\alpha_{i+1}|_{x_{j+1}}$ for every $i \geq 0$ and every $x_j \in \mathcal{S}_i$, which proves claim 2.

Likewise, for all $i, j \geq 0$, the above equation gives $|\phi_{i,i+j}(y)|_y = |\phi^j(y)|_y = \frac{1}{2}|\hat{\alpha}_{k,j}|_y = 2^{j+1}$. Thus, for every $c \geq 1$ and every $i \geq 0$ we can choose an appropriate $j \geq 0$ such that $|\phi_{i,i+j}(y)|_y > c$. Together with claim 1, this proves claim 3.

As $\mathrm{var}(\hat{\alpha}_{k,0}) = \{y\}$, and as $y \in \mathcal{H}_0$ by claim 3, Lemma 7.13 proves claim 4. Hence, the chain system is $\mathcal{E}$-free, and claim 5 follows from Theorem 7.19 and claim 2.

Finally, to prove claim 6, consider any $i \geq 0$. By claim 5, we know

$$\begin{aligned} L_{\mathrm{E},\Sigma}(\hat{\alpha}_{k,i}) &\supseteq L_{\mathrm{E},\Sigma}(\hat{\beta}_{k,i+1}) \\ &= L_{\mathrm{E},\Sigma}(\pi_{\mathcal{A}_{i+1}}(\hat{\alpha}_{k,i+1})). \end{aligned}$$

With some straightforward effort, one can easily verify that any morphism $\tau : X^* \to X^*$ with $\tau(\hat{\alpha}_{k,i}) = \hat{\beta}_{k,i+1}$ must satisfy

$$\begin{aligned} \tau(y) &= x_1^k, \\ \tau(x_i) &= x_{i+1} \end{aligned}$$

for every i with $1 \leq i \leq n$. Especially, this means that no such τ can erase any variable from $\mathrm{var}(\hat{\alpha}_{k,i})$; in other words, $L_{\mathrm{E},\Sigma}(\pi_A(\hat{\alpha}_{k,i})) \not\supseteq L_{\mathrm{E},\Sigma}(\hat{\beta}_{k,i})$ for every $A \subset \mathrm{var}(\hat{\alpha}_{k,i})$. Thus, every $\hat{\alpha}_{k,i}$ is $L_\Sigma^{(k)}$ reduced, which proves claim 5. □

Note that we do not allow $k = 1$, as the corresponding $\hat{\phi}_1(y) = (y^2 x_1)$ is an imprimitivity morphism that would lead to $L_{\mathrm{E},\Sigma}(\hat{\alpha}_{1,i}) = L_{\mathrm{E},\Sigma}(x^2)$ for every $i \geq 0$. Furthermore, the resulting chain system would not be $\mathcal{E}$-free, as $y \in \mathcal{E}_i$ would hold for every n, with x_1 being the unique successor of y of bounded frequency.

In the following section, we briefly sketch how the languages $L_\Sigma^{(k)}$ can be used in the proof of Theorem 5.18 (from Chapter 6). After that, we use the tools we developed in the present section to construct the Loughborough Example and the Wittenberg Examples in Sections 7.4 and 7.5, respectively.

7.3.1 $L_\Sigma^{(k)}$ and the Proof of Theorem 5.18

Recall that the proof of Theorem 5.18 in Section 5.4.2 uses a language $L_\Sigma = L_{\text{E},\Sigma}((\alpha_i)_{i=0}^\infty)$, where the strictly decreasing chain $(\alpha_i)_{i=0}^\infty$ is defined as in Definition 5.17. In the terminology we developed in the present chapter, that chain is generated by the chain system $(\alpha_0, (\psi_i)_{i=0}^\infty)$, where $\alpha_0 = y^2z^2$, and $\phi_i = \phi$ for every $i \geq 0$, where the morphism ϕ is defined by

$$\phi(y){:=}y^2x_1, \qquad \phi(z){:=}x_1z^2, \qquad \phi(x_i){:=}x_{i+1}$$

for every $i \geq 1$. As we prove in Section 5.4.2, there is a strictly increasing chain $(\beta_i)_{i=0}^\infty$ with $\bigcup_{i=0}^\infty L_{\text{E},\Sigma}(\beta_i)$, and for every $i \geq 0$, $\beta_i = \psi(\alpha_i)$, where the morphism $\psi : X^* \to X^*$ is defined through

$$\psi(x_j){:=}x_j, \qquad \psi(y){:=}\psi(z){:=}x_0$$

for every $j \geq 1$. Using our newfound knowledge, it is easy to see that, $\mathcal{H}_i = \{y, z\}$, and $\mathcal{S}_i = \mathcal{A}_i = \{x_1, \ldots, x_i\}$ hold for every $i \geq 0$. We use this and Theorem 7.19 to conclude

$$L_\Sigma = \bigcup_{i=0}^{\infty} L_{\text{E},\Sigma}(\pi_{\mathcal{A}_i}(\alpha_i)).$$

At first this might seem like a contradiction, but a closer look reveals that, for every $i \geq 0$, $L_{\text{E},\Sigma}(\pi_{\mathcal{A}_i}(\alpha_i)) = L_{\text{E},\Sigma}(\beta_{i+1})$.

Furthermore, with a little effort, one can adapt the proof of Theorem 5.18 to use any language $L_\Sigma^{(k)}$ with $k \geq 2$ instead of L_Σ. As in the original proof, the main idea is to show that, for every $\delta \in \text{Super}(L_\Sigma^{(k)})$, there is an $i \geq 0$ such that $L_{\text{E},\Sigma}(\delta) \supseteq L_{\text{E},\Sigma}(\hat{\alpha}_{k,i})$, and consequently, $L_{\text{E},\Sigma}(\delta) \supset L_{\text{E},\Sigma}(\hat{\alpha}_{k,i+1}) \supseteq L_\Sigma^{(k)}$.

To prove this, one can define a sequence of morphisms $(\theta_{k,i})_{i=0}^\infty$ with $\theta_{k,i}(\delta) = \hat{\beta}_{k,i}$, and each $\theta_{k,i}$ erases as many variables of δ as possible. We then choose $m, i \geq 1$ such that $\theta_{k,i}$ and $\theta_{k,i+j}$ erase exactly the same variables of δ, and observe that Lemma 5.26 applies (mutatis mutandis). I.e., if for some $x \in \text{var}(\delta)$, $\text{var}(\theta_{k,i+j}(x))$ contains a variable x_i with $1 \leq i \leq m$, it must also contain a variable x_j with $m+1 \leq j \leq i+j$ (as the variables $x_{m+1}, \ldots, x_{i+j}$ in $\hat{\beta}_{k,i+j}$ are the images of the variables $x_1, \ldots, x_i$ in $\hat{\beta}_{k,i}$).

Thus, looking only at each $\theta_{k,i+j}(x)$ by itself, we can recognize for every occurrence of x_1 whether it is mapped to the leftmost occurrence of a block x_1^k in $\hat{\beta}_{k,i+j}$. If it is such a leftmost occurrence, we can modify $\theta_{k,i+j}(x)$ to include y^2 to the left of each such leftmost x_1, and obtain a morphism θ' with $\theta'(\delta) = \hat{\alpha}_{k,i+j}$.

7.4 The Loughborough Example

We now use the language $L_\Sigma^{(k)}$ to construct the so-called Lougborough Example[3], with which we then disprove Conjectures 7.3 and 7.6:

[3]On the name of this example: As mentioned a few times throughout this chapter, the author firmly believed in Conjecture 7.3, and deeply felt that a proof was within his reach. But six months of hard work passed, and although he had developed most of the material contained in the present chapter, no proof had been discovered. During a short research visit to Loughborough, Daniel Reidenbach remarked

Definition 7.24 (Loughborough Example)**.** *Let Σ be an alphabet with $|\Sigma| \geq 2$. Let x, y, z, z_1, z_2 and all x_i with $i \geq 1$ be pairwise distinct elements of X. We define $L_L \subset \Sigma^*$ as follows: First, let $\alpha_{L_1} := x^2y^3$. Next, we define the decreasing chain $(\alpha_i)_{i=0}^{\infty}$ through $\alpha_i := z^3 \cdot \hat{\alpha}_{2,i}$, where each pattern $\hat{\alpha}_{2,i}$ is defined as in Definition 7.21. Finally, let*

$$\begin{aligned} L_1 &:= L_{\mathrm{E},\Sigma}(\alpha_{L_1}) = L_{\mathrm{E},\Sigma}(x^2y^3), \\ L_2 &:= L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^{\infty}) = L_{\mathrm{E},\Sigma}(z^3)\, L_{\mathrm{E},\Sigma}((\hat{\alpha}_{2,i})_{i=0}^{\infty}), \\ L_L &:= L_1 \cup L_2. \end{aligned}$$

According to Lemma 7.23, $L_2 = L_{\mathrm{E},\Sigma}(z^3) \cdot L_{\Sigma}^{(2)}$ – in other words, L_2 consists of a concatenation of $L_{\mathrm{E},\Sigma}(z^3)$ and the language $L_{\Sigma}^{(2)}$ from Definition 7.21. Thus, we conclude that $L_2 = \bigcup_{i=0}^{\infty} L_{\mathrm{E},\Sigma}(z^3 \cdot \hat{\beta}_{2,i})$. Keeping this in mind, we are able to observe the following result:

Theorem 7.25. *Let L_L and $(\hat{\alpha}_{2,i})_{i=0}^{\infty}$ be defined as in Definition 7.24. The following holds:*

1. *The chain $(z_1^2\, z_2^3\, \hat{\alpha}_{2,i})_{i=0}^{\infty}$ is L_L-reduced and covers L, and*
2. *the pattern $\delta := x^3y^2z^3$ is $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive of L_L.*

Proof. First, note that as $L_L = L_1 \cup L_2$, for every $\alpha \in \sup(L)$, both $L_{\mathrm{E},\Sigma}(\alpha) \supseteq L_1$ and $L_{\mathrm{E},\Sigma}(\alpha) \supseteq L_{\mathrm{E},\Sigma}(\hat{\beta}_{2,i})$ for every $i \geq 0$ must hold.

We begin with the first claim. For $i \geq 0$, let $\gamma_i := z_1^2\, z_2^3\, \hat{\alpha}_{2,i}$. First, note that $L_{\mathrm{E},\Sigma}(\gamma_i) \supseteq L_L$ follows from the definition and from Theorem 7.19, and it is easy to see that the chain is a strictly decreasing chain. Therefore, we only need to prove that every γ_i is L_L-reduced. For this, fix any $i \geq 0$. By Theorem 2.4, there are morphisms $\tau_1, \tau_2 : X^* \to X^*$ such that $\tau_1(\gamma_i) = \alpha_{L_1}$ and $\tau_2(\gamma_i) = z^3\hat{\beta}_{2,i+1}$. Now, for all such morphisms, $\tau_1(z_2) = y$ and $\tau_2(z_2) = z$ must hold, as z_2 is the only variable in $\mathrm{var}(\gamma_i)$ that occurs exactly three times in γ_i (and as no variable from $\mathrm{var}(\gamma_i)$ occurs only once in γ_i), and there is no other way to obtain the y^3 in α_{L_1} or the z^3 in $z^3\hat{\beta}_{2,i+1}$ from γ_i. This immediately implies $L_{\mathrm{E},\Sigma}(\pi_{X\setminus\{z_2\}}(\gamma_i)) \not\supseteq L_L$.

Furthermore, $\tau_1(z_2) = y$ implies $\tau_1(z_1) = x$; thus, z_1 can also not be erased from γ_i. Finally, as $(\hat{\alpha}_{2,i})_{i=0}^{\infty}$ is self-reduced (according to Lemma 7.23), $\tau_1(z_2) = z$ implies that we cannot erase any other variable from $\gamma_i = z_1^2\, z_2^3\, \hat{\alpha}_{2,i}$. Hence, every γ_i is L_L-reduced, and therefore, $(z_1^2\, z_2^3\, \hat{\alpha}_{2,i})_{i=0}^{\infty}$ is L_L-reduced.

For the second claim, $L_{\mathrm{E},\Sigma}(\delta) \supseteq L_1 \cup L_2$ is obvious, as $x^3y^2z^3$ can be easily mapped to x^2y^3 and to $z^3y^2 = z^3\hat{\alpha}_{2,0}$, which proves $L_{\mathrm{E},\Sigma}(\delta) \supseteq L_1$ and $L_{\mathrm{E},\Sigma}(\delta) \supseteq L_2$, respectively. Next, note that no variable in δ is L_L-redundant: If we erase y, the resulting pattern

that, after all this time, the author should reconsider his convictions, and try equally hard to find a counterexample. Although he promised to follow this advise, the author also offered to express his sincere belief in the correctness of Conjecture 7.3 by means of a bet. Reidenbach refused. Fortunately; as in the very next morning, the author discovered the example that disproved the conjecture he had so firmly believed in for half a year. After this noteworthy experience, he decided that this example deserved a name, at least due to its personal importance, and decided on "Loughborough Example", with "The Example That Almost Earned Daniel Reidenbach the Easiest Money Ever" being a close but unwieldy and slightly ungrammatical second, and "That Annonying Example That I Could Have Found Far Earlier" being a contender.

can be mapped neither to $\alpha = x^2z^3$, nor to any $\hat{\beta}_{2,i}$ with $i \geq 1$. Similarly, $\pi_{y,z}(\delta)$ cannot be mapped to any $z^3\hat{\beta}_{2,i}$ with $i \geq 0$, while $\pi_{x,y}(\delta)$ cannot be mapped to α.

Accordingly, if there is a $\gamma \in X^+$ with $L_{\mathrm{E},\Sigma}(\delta) \supset L_{\mathrm{E},\Sigma}(\gamma) \supseteq L$, every morphism $\tau : X^* \to X^*$ with $\tau(\delta) = \gamma$ must be nonerasing. By Lemma 5.1, τ may be neither an imprimitivity morphism, nor a renaming. But as $L_{\mathrm{E},\Sigma}(\gamma) \supseteq L_1$ must hold, there must be an $x_y \in \mathrm{var}(\tau(y))$ such that $|\gamma|_{x_y} = 2$, and an $x_z \in \mathrm{var}(\tau(z))$ with $|\gamma|_{x_z} = 3$. Likewise, in order to generate the $z^3\hat{\beta}_{2,i}$ from γ, there must be an $x_x \in \mathrm{var}(\tau(x))$ with $|\gamma|_{x_x} = 3$. Hence, τ is an imprimitivity morphism, and $L_{\mathrm{E},\Sigma}(\gamma) = L_{\mathrm{E},\Sigma}(\delta)$, which means that δ must be ePAT$_{\text{tf},\Sigma}$-descriptive of L. □

Theorem 7.25 demonstrates that L_L disproves Conjecture 7.3. Moreover, as each $z_1^2 z_2^3 \hat{\alpha}_{2,i}$ (or its canonical form) is an element of RedSuper(L_L), RedSuper(L_L) is infinite, which disproves Conjecture 7.6.

The basic idea behind L_L might be understood as follows: Note that while $L_1 = L_{\mathrm{E},\Sigma}(x^3y^2)$, we also have $L_2 \subset L_{\mathrm{E},\Sigma}(y^2x^3)$. Similar to Example 6.19 in Section 6.3.1, the patterns x^3y^2 and y^2x^3 can be merged in different ways, where each resulting pattern generates a superset of $L = L_1 \cup L_2$.

In $\delta = x^3y^2z^3$, the y^2 is used in two different roles. Here, y^2z^3 is used to generate L_1, while x^3y^2 is used to generate L_2. Seeing δ as the result of merging x^2y^3 and x^3y^2 (with a renaming of variables), the important fact is that the "squared part" of $L_1 = L_{\mathrm{E},\Sigma}(x^2y^3)$ (i. e., everything generated by the y^2 in α_{L_1}) covers the $\mathcal{H}$-variable of the chain that generates $L_\Sigma^{(2)}$ and 'blocks' this part of δ. Thus, every morphism that expands y^2 in a non-redundant way loses the expressivity that is needed to generate L_1.

In contrast to this, the patterns of the chain $z_1^2 z_2^3 \hat{\alpha}_{2,i}$ merge α_{L_1} and the chain $z_2^3 \hat{\alpha}_{2,i}$ on the cubed variable, which allows the chain's descent to continue undisturbed by L_1.

Even though one might argue that L_L and the chain that descends toward it can still be considered to be very similar, the Loughborough Example uses the fact that, although all variables in every pattern of the chain are indeed necessary to generate L_L, no word needs all of these variables of any of the patterns at once.

In the next section, we discuss two examples that utilize this phenomenon in order to highlight further difficulties that arise when dealing with chains that cover languages.

7.5 The Wittenberg Examples

One of the questions that arose during the author's work on decreasing chains that cover languages and their relation to the existence of descriptive patterns was whether there is a language that is covered by multiple 'different' chains. Of course, this question first depends on a suitable definition of 'different chains' (or, if defining through the complement, of 'equivalent chains') – for example, considering chains as in Example 7.15 as different would answer the question in a positive but not particularly illuminating way, as would removing an arbitrary number of patterns from any infinite properly descending chain (as long as infinitely many patterns remain).

As, in the context of the present thesis, chains themselves are of less central importance than the languages they generate (in the limit), we consider two decreasing chains $(\alpha_i)_{i=0}^\infty$ and $(\beta_i)_{i=0}^\infty$ over X^+ to be *equivalent* if $L_{\mathrm{E},\Sigma}((\alpha_i)_{i=0}^\infty) = L_{\mathrm{E},\Sigma}((\beta_i)_{i=0}^\infty)$ (for any non-unary alphabet Σ). If one views our definition of the language that is generated

by a chain of patterns as the limit of the sequence of pattern languages, this definition might even be considered canonical. Analogously, two chains are *different* if they are not equivalent.

On his way back to Frankfurt from Lutherstadt Wittenberg (after attending Theorietag 2009), the author realized that there are in fact languages L that are covered by multiple (pairwise) different L-reduced chains – in fact, even uncountably many such chains. We study an example of these languages (which we call "the first Wittenberg Example"[4]) in Definition 7.26 and Observations 7.28 and 7.29. Furthermore, not only do such languages exist, some of them even have an $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive pattern, as explained in Example 7.30 (accordingly, "the second Wittenberg Example").

Through the Loughborough Example (cf. Section 7.4), the author understood that the existence of $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive patterns is more complicated than he had thought during his work on Conjecture 7.3, but he still had hope that his proof idea could be saved. But as the two Wittenberg Examples show, a characterization in the spirit of Conjecture 7.3 that somehow generalizes the proof of Theorem 5.18 (i. e., $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L) = \emptyset$ if and only if $\text{S-Hull}_\Sigma(L)$ is covered by an $\text{S-Hull}_\Sigma(L)$-reduced chain that satisfies certain criteria) would require considerable effort (for example, some way to reconstruct all other possible chains from the given chain and L).

After discovering the Wittenberg Examples, and considering the six months he had already spent on his ill-fated attempt to prove Conjecture 7.3, the author gave up on finding such a characterization or sufficient criteria, and decided to direct his attention to different topics.

Like the Loughborough Example, the Wittenberg Examples use a technique that is similar to Example 6.19: When constructing a pattern that is $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive of the union of two languages L_1 and L_2, we can construct languages that cover $L_1 \cup L_2$ by merging patterns that cover L_1 or L_2, respectively. We begin with the first of these two examples:

Definition 7.26 (First Wittenberg Example)**.** *Let $L_{W_1} := L_\Sigma^{(2)} \cup L_\Sigma^{(3)}$ (which are defined as in Definition 7.21). We define the morphisms ϕ_2 and ϕ_3 through*

$$\begin{aligned} \phi_2(x) &:= (x)^2\, y_1^2\, z_1^3, & \phi_3(x) &:= (x)^2\, z_1^3\, y_1^2, \\ \phi_2(y_i) &:= y_{i+1}, & \phi_3(y_i) &:= y_{i+1}, \\ \phi_2(z_i) &:= z_{i+1}, & \phi_3(z_i) &:= z_{i+1} \end{aligned}$$

for all $i \geq 0$.

For $i \geq 1$ and $\vec{s} = (s_1, \ldots, s_n) \in \{2,3\}^n$, the morphism $\phi_{\vec{s}}$ is defined as follows: If $\vec{s}$ is of length 1, let $\phi_{\vec{s}} := \phi_{s_1}$. Otherwise, let $\phi_{\vec{s}} := \phi_{s_n} \circ \phi_{(s_1,\ldots,s_{n-1})}$.

This allows us to define the patterns $\alpha := x^2$ and $\alpha_{\vec{s}} := \phi_{\vec{s}}(\alpha)$ for every $i \geq 1$ and every $\vec{s} \in \{2,3\}^n$.

This definition is illustrated by Figure 7.1 and the following example:

[4] After all, a name in the spirit of "the first train back from Wittenberg Example" would have been unpleasantly ambiguous.

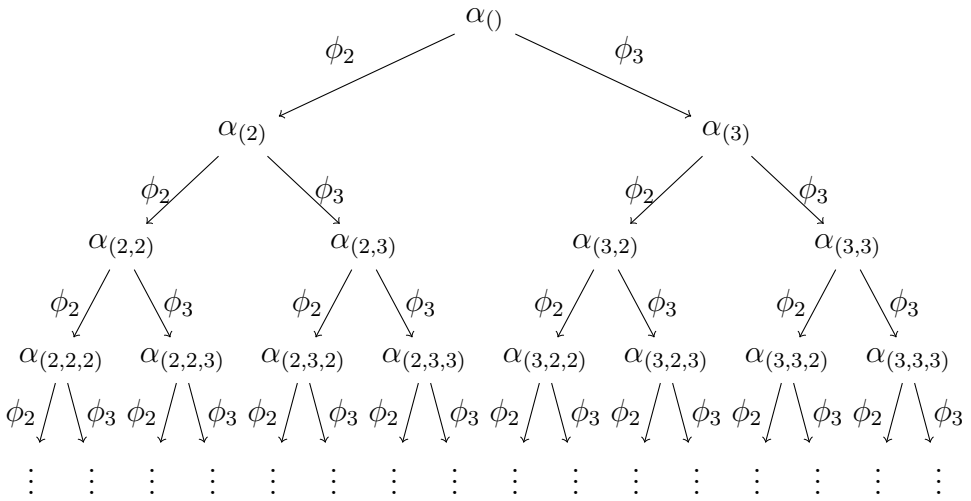

Figure 7.1: An illustration of the relation between the patterns defined in Definition 7.26. The patterns $\alpha_{\vec{s}}$ are generated from α using the morphisms $\phi_{\vec{s}}$, where $\vec{s}$ is a finite sequence over $\{2,3\}$. The patterns shown in this illustration are given explicitely in Example 7.27.

Example 7.27. *Figure 7.1 contains the first four levels of the resulting tree. The corresponding patterns are as follows:*

$$\begin{aligned}
\alpha_() &= x^2, \\
\alpha_{(2)} &= ((x)^2\, y_1^2 z_1^3)^2, & \alpha_{(3)} &= ((x)^2\, z_1^3 y_1^2)^2, \\
\alpha_{(2,2)} &= (((x)^2\, y_1^2 z_1^3)^2\, y_2^2 z_2^3)^2, & \alpha_{(2,3)} &= (((x)^2\, z_1^3 y_1^2)^2\, y_2^2 z_2^3)^2, \\
\alpha_{(3,2)} &= (((x)^2\, y_1^2 z_1^3)^2\, z_2^3 y_2^2)^2, & \alpha_{(3,3)} &= (((x)^2\, z_1^3 y_1^2)^2\, z_2^3 y_2^2)^2, \\
\alpha_{(2,2,2)} &= ((((x)^2\, y_1^2 z_1^3)^2\, y_2^2 z_2^3)^2\, y_3^2 z_3^3)^2, & \alpha_{(2,2,3)} &= ((((x)^2\, z_1^3 y_1^2)^2\, y_2^2 z_2^3)^2\, y_3^2 z_3^3)^2, \\
\alpha_{(2,3,2)} &= ((((x)^2\, y_1^2 z_1^3)^2\, z_2^3 y_2^2)^2\, y_3^2 z_3^3)^2, & \alpha_{(2,3,3)} &= ((((x)^2\, z_1^3 y_1^2)^2\, z_2^3 y_2^2)^2\, y_3^2 z_3^3)^2, \\
\alpha_{(3,2,2)} &= ((((x)^2\, y_1^2 z_1^3)^2\, y_2^2 z_2^3)^2\, z_3^3 y_3^2)^2, & \alpha_{(3,2,3)} &= ((((x)^2\, z_1^3 y_1^2)^2\, y_2^2 z_2^3)^2\, z_3^3 y_3^2)^2, \\
\alpha_{(3,3,2)} &= ((((x)^2\, y_1^2 z_1^3)^2\, z_2^3 y_2^2)^2\, z_3^3 y_3^2)^2, & \alpha_{(3,3,3)} &= ((((x)^2\, z_1^3 y_1^2)^2\, z_2^3 y_2^2)^2\, z_3^3 y_3^2)^2.
\end{aligned}$$

◇

Thus, we can use every infinite sequence over $\{2,3\}$ to obtain a chain system and the corresponding chain. From the definitions of the involved morphisms, it should be obvious that each of these chain systems is $\mathcal{E}$-free, and in all cases, the adult variables are exactly those y_i and z_i that occur in the pattern.

Using Theorem 7.19, we can then derive a language from each sequence s, and observe that different sequences lead to different languages:

Observation 7.28. *Let $S = (s_i)_{i=0}^{\infty}$ and $T = (t_i)_{i=0}^{\infty}$ be infinite sequences over $\{2,3\}$. For every $i \geq 1$, we define $\vec{s_i}{:=}(s_1, \ldots, s_i)$ and $\vec{t_i}{:=}(t_1, \ldots, t_i)$. Then*

$$L_{\mathrm{E},\Sigma}((\alpha_{\vec{s_i}})_{i=0}^{\infty}) = L_{\mathrm{E},\Sigma}((\alpha_{\vec{t_i}})_{i=0}^{\infty})$$

if and only if $S = T$.

Proof. As mentioned above, if we consider the chain systems that can be derived from each sequences S or T, and corresponding chains $(\alpha_{\vec{s_i}})_{i=0}^{\infty}$ and $(\alpha_{\vec{t_i}})_{i=0}^{\infty}$, we have $\mathcal{H}_i = \{x\}$ and $\mathcal{A}_i = \mathcal{S}_i = \{y_1, z_1, \ldots, y_i, z_i\}$ for every $i \geq 0$.

Thus, according to Theorem 7.19, we can derive the languages that are generated by the two chains simply by considering the union of all $\pi_{X\setminus\{x\}}(\alpha_{\vec{s_i}})$ and the union of all $\pi_{X\setminus\{x\}}(\alpha_{\vec{t_i}})$, and see that the claim follows immediately. □

We now show that each of these chains is L_{W_1}-reduced and covers L_{W_1}:

Observation 7.29. *For every $n \geq 1$ and every $\vec{s} \in \{2,3\}^n$, $L_{\mathrm{E},\Sigma}(\alpha_{\vec{s}}) \supseteq L_{W_1}$, and $\alpha_{\vec{s}}$ is L_{W_1}-reduced.*

Proof. Let $i \geq 0$ and $\vec{s} \in \{2,3\}^n$. The first part of the claim follows immediately from the definition of $\alpha_{\vec{s}}$. If $\vec{s} \in \{2,3\}^n$, we can define the morphisms $\psi_2, \psi_3 : X^* \to X^*$ through

$$\begin{aligned} &\psi_2(x) := y, && \psi_2(y_i) := x_i, && \psi_2(z_i) := \lambda, \\ &\psi_3(x) := y, && \psi_3(y_i) := \lambda, && \psi_3(z_i) := x_i \end{aligned}$$

for every $i \geq 1$. Then $\psi_k(\alpha_{\vec{s}}) = \hat{\alpha}_{k,n}$ holds for every $k \in \{2,3\}$, and thus, $L_{\mathrm{E},\Sigma}(\alpha_{\vec{s}}) \supseteq L_{\Sigma}^{(2)} \cup L_{\Sigma}^{(3)} = L_{W_1}$.

We can prove that $\alpha_{\vec{s}}$ is L_{W_1}-reduced by examining the morphisms that map $\alpha_{\vec{s}}$ to the patterns $\hat{\beta}_{2,i+1}$ and $\hat{\beta}_{3,i+1}$ (which must exist, as the patterns generate subsets of $L_{\Sigma}^{(2)}$ and $L_{\Sigma}^{(3)}$, respectively), using a reasoning that is similar to the corresponding reasoning in the proof of Theorem 7.25.

Let $\tau_2, \tau_3 : X^* \to X^*$ be morphisms with $\tau_k(\alpha_{\vec{s}}) = \hat{\beta}_{k,i+1}$ for $k \in \{2,3\}$. By comparing the order and frequency of the occurrence of the variables in $\alpha_{\vec{s}}$ and both $\hat{\beta}_{k,i+1}$, one can easily verify that

$$\begin{aligned} &\tau_2(x) = x_1^2, && \tau_2(y_i) = x_{i+1}, && \tau_2(z_i) = \lambda, \\ &\tau_3(x) = x_1^3, && \tau_3(y_i) = \lambda, && \tau_3(z_i) = x_{i+1} \end{aligned}$$

must hold for all such morphisms and all y_i, z_i with $1 \leq i \leq n$.

Thus, if we remove any y_i from $\alpha_{\vec{s}}$, the resulting pattern cannot be mapped to $\hat{\beta}_{2,n+1}$, and if we remove any z_i, the resulting pattern cannot be mapped to $\hat{\beta}_{3,n+1}$. Furthermore, if we remove y, we lose the possibility to map the resulting pattern to any of the two $\hat{\beta}_{k,n+1}$. Therefore, we cannot remove any variable from $\alpha_{\vec{s}}$, which means that the pattern is L_{W_1}-reduced. As n and $\vec{s}$ were chosen freely, this concludes the proof. □

Observation 7.29 shows that each infinite sequence over $\{2,3\}$ leads to a chain that is L_{W_1}-reduced and covers L_{W_1}. As there are uncountably many infinite sequences over $\{2,3\}$, and each of these sequences leads to a different language (cf. Observation 7.28), there are uncountably many L_{W_1}-reduced chains that cover L_{W_1}. Like the Loughborough Example, the first Wittenberg Example uses the fact that, although all chains are L_{W_1} reduced (which means that every variable of every pattern in the chain is needed to generate some word of L_{W_1}), no word uses all variables of a given pattern at once.

In fact, it is even possible to combine the approaches from the Loughborough Example and the first Wittenberg Example to show that there is a language L_{W_2} such that L_{W_2} has an $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive pattern, but is covered by uncountably many L_{W_2}-reduced strictly decreasing chains (which converge to pairwise different languages):

Example 7.30 (Second Wittenberg Example)**.** *For any Σ with $|\Sigma| \geq 2$, we define*

$$\begin{aligned} L_1 &:= L_{\mathrm{E},\Sigma}(x^2y^3), \\ L_2 &:= (L_{\mathrm{E},\Sigma}(x^3)L_\Sigma^{(2)}) \cup (L_{\mathrm{E},\Sigma}(x^3)L_\Sigma^{(3)}), \\ L_{W_2} &:= L_1 \cup L_2. \end{aligned}$$

Thus, L_{W_2} is defined almost as in the Loughborough Example (cf. Definition 7.24), the only notable difference being that L_2 is a modification of the language L_{W_1} from Definition 7.26 instead of $L_\Sigma^{(2)}$.

For every $i \geq 0$ and every $\vec{s} \in \{2,3\}^i$, let $\alpha_{\vec{s}}$ be defined as in Definition 7.26, and let x_1, x_2 be new variables from X that are pairwise distinct from x, all y_i and all z_i.

Using the same reasoning as for Observation 7.29 and for Theorem 7.25, one can easily verify that, for every $i \geq 0$ and every $\vec{s} \in \{2,3\}^i$, $L_{\mathrm{E},\Sigma}(x_1^2x_2^3\alpha_{\vec{s}}) \supset L_{W_2}$ holds, while $x_1^2x_2^3\alpha_{\vec{s}}$ is L_{W_2}-reduced.

On the other hand, one can easily see that $\delta := x^3y^2z^3$ is ePAT$_{\mathrm{tf},\Sigma}$*-descriptive of L_{W_2}, again using the same reasoning as for Theorem 7.25.* ◇

Chapter 8

Conclusions and Suggestions for Future Research

This chapter provides a summary of the contents of the thesis (with the obvious exemption of Chapters 1 and 2).

Chapter 3

As demonstrated in this chapter, inclusion for pattern languages remains undecidable even if we bound the number of terminal letters and the number of variables in the pattern. The number of variables where inclusion is provably undecidable that are given in Theorem 3.10 (and Bremer's extension to NE-pattern languages) remains comparatively high. Furthermore, Propositions 3.7 and 3.8 demonstrate that existence of a morphism between the patterns is not a characteristic inclusion criterion even when we apply comparatively small bounds. As suggested in Open Problem 3.9, one possible direction is to attempt to prove that inclusion is decidable for these classes.

On the other hand, undecidability might be established for far lower bounds than in Theorem 3.10 by finding a suitable undecidable problem that has a comparatively compact representation. Possible candidates are universal generalizations of the Collatz iteration (cf. Kurtz and Simon [60]) and one-state linear operator algorithms (cf. Kaščák [54]), which are both based on Conway iterations (cf. Conway [19]). Also note that it would suffice to find machines with a compact representation in patterns that are not universal, but have an undecidable halting problem. The existence of non-universal machines with an undecidable halting problem follows from the existence of undecidable but non-universal sets of natural numbers, but to the author's knowledge, there has been no examination of small machines with this property (see also Margenstern [68]).

Chapter 4

This chapter shows that extending regular expressions with only a single variable already leads to an immense increase in succinctness and expressive power. The good part of this news is that in certain applications, using the right extended regular expression instead of a proper regular expression can lead to far more efficient running times, all with the matching engine that is already present on the machine. The bad part of this news is that this additional power cannot be harnessed in full, due to the undecidability of the corresponding decision problems. Naturally, this greatly diminishes the usefulness of

extended regular expressions as more efficient alternative to proper regular expressions.

Due to these undecidable problems, some questions of designing "good" extended regular expressions are of difficulty comparable to designing good programs. For applied computer scientists, it could be worthwhile to examine heuristics and good practices to identify cases where the non-conventional use of extended regular expressions might offer unexpected speed advantages. For theoretical computer scientists, the results in this chapter highlight the need for appropriate restrictions other than the number of variables; restrictions that lead to large and natural subclasses with decidable decision problems. One possible approach that does not extend the expressive power of proper regular expressions beyond regular languages would be a restriction of the length of the words on which variables can be bound.

Furthermore, as a very vague proposal, the construction used in Theorem 4.14 could also offer insights on the undecidability of inherent ambiguity for RegEx(1), similar to results on context-free languages (cf. Kutrib [61]). Although, to the author's knowledge, ambiguity of extended regular expressions has not been examined, research in this direction has been proposed by Salomaa [99]. It is reasonable to expect that every finite and every cofinite language should not be inherently ambiguous for most reasonable definitions of ambiguity of extended regular expressions, while non-regular INVALC($\mathcal{X}$) might be inherently ambiguous for most definitions of this concept.

Chapter 5

Theorem 5.18 gives a positive answer to the longstanding open question whether there are infinite languages for which no pattern is E-descriptive. One possible continuation is the search for a characterization of these languages. Considering the fact that this problem is surprisingly hard even for terminal-free E-pattern languages where the inclusion is decidable, the author considers this problem very hard, and suggests to study it for that restricted case before attempting the full class of E-pattern languages. Further suggestions on this topic can be found in the comments on Chapter 7 further down in the present chapter.

Instead of trying to find such a characterization, it might be worthwhile to examine the existence of E-descriptive patterns for all languages from various prominent classes; e. g., the regular languages:

Open Problem 8.1. *Is there a regular language L of which no pattern is E-descriptive?*

Alternatively, further examination of Open Problem 5.35 could provide new insights into the equivalence problem for E-pattern languages, and such insights are gravely needed now that Ohlebusch and Ukkonen's conjectures (cf. [79]) have been disproven (cf. [32]). More importantly, it seems more promising to start with the more fundamental question whether E-descriptive patterns for finite sets can be computed effectively (Open Problem 5.31).

Chapter 6

In this chapter, we introduced a new inference paradigm, descriptive generalization, and showed that the loss of precision (in comparison to Gold's model) that comes with the use of descriptive patterns as hypotheses can lead to greater power. For inductive

inference, the author considers it most promising to examine the properties of the more general model HYP proposed in Section 6.2.4. For formal language theory, an alternative characterization of $\mathcal{TSL}_\Sigma$ or more sufficient criteria for the non-existence of telling sets as in Lemma 6.37 should be very interesting. Apart from this, the tools developed in Section 6.3.1 should be of use for further work on $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$-descriptive patterns.

Note that most of the results of this chapter can be adapted to all those classes of E-pattern languages where inclusion is characterized by the existence of a morphism between the patterns (for some examples, see the list in Section 3.3, after Theorem 3.5).

Chapter 7

The main insight of Chapter 7 is probably that the question of the existence of descriptive patterns is hard, even for classes (or at least one class) of pattern languages with a decidable inclusion problem.

Of the conjectures presented in Section 7.1, only Conjecture 7.2 has not been disproven. The author is still convinced that this conjecture holds, as languages that are generated by chains, and not simply covered by chains should show sufficient similarities between the language and the patterns of the chain to adapt the proof of Theorem 5.18 (note that the examples in Sections 7.4 and 7.5 use languages that are covered by chains, but not generated by chains).

The author thinks that a proof of Conjecture 7.2 might be obtained by extending Lemma 7.18 and Theorem 7.19 to chain systems with $\mathcal{E}$-variables. As mentioned in Section 7.2.1, this would require a formalization of the concept of adult $\mathcal{H}$-variables. If such extensions can be found, a proof of Conjecture 7.2 might still require technical effort, but be within reach.

Apart from chains and chain systems, the author considers the following question very interesting, not only due to its relation to Open Problem 5.35:

Conjecture 8.2. *Let Σ be an alphabet with $|\Sigma| \geq 2$, and let $L_1, L_2 \in \Sigma^*$. If $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L_1)$ and $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L_2)$ are nonempty, then $D_{\mathrm{ePAT}_{\mathrm{tf},\Sigma}}(L_1 \cup L_2)$ is nonempty.*

Although this conjecture seems fairly obvious, the author was unable to prove it. Moreover, if it should hold, it would be very interesting to see how the descriptive patterns for the single languages and the descriptive patterns for the union of these languages are related.

Bibliography

[1] A.V. Aho. Algorithms for finding patterns in strings. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 5, pages 255–300. Elsevier, 1990.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.

[3] J. Albert and L. Wegner. Languages with homomorphic replacements. *Theoretical Computer Science*, 16:291–305, 1981.

[4] D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

[5] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.

[6] Dana Angluin. Computational learning theory: Survey and selected bibliography. In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing, STOC 1992*, pages 351–369, 1992.

[7] H. Arimura, T. Shinohara, and S. Otsuki. Finding minimal generalizations for unions of pattern languages and its application to inductive inference from positive data. In *Proc. 11th Annual Symposium on Theoretical Aspects of Computer Science, STACS 1994*, volume 775 of *LNCS*, pages 649–660, 1994.

[8] P. Barceló, C.A. Hurtado, L. Libkin, and P.T. Wood. Expressive languages for path queries over graph-structured data. In *Proc. Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010*, pages 3–14, 2010.

[9] H. Bordihn, J. Dassow, and M. Holzer. Extending regular expressions with homomorphic replacements. *RAIRO Theoretical Informatics and Applications*, 44(2):229–255, 2010.

[10] A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5:279–305, 1998.

[11] J. Bremer. Inklusionsprobleme für nichtlöschende Patternsprachen. Diplomarbeit, Institut für Informatik, Johann-Wolfgang-Goethe-Universität, Frankfurt am Main, 2010. In German.

[12] J. Bremer and D.D. Freydenberger. Inclusion problems for patterns with a bounded number of variables. In *Proc. 14th International Conference on Developments in Language Theory, DLT 2010*, volume 6224 of *LNCS*, pages 100–111, 2010. Received the DLT 2010 Best Paper Award.

[13] C. Câmpeanu, K. Salomaa, and S. Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14:1007–1018, 2003.

[14] C. Câmpeanu and N. Santean. On the intersection of regex languages with regular languages. *Theoretical Computer Science*, 410(24–25):2336–2344, 2009.

[15] C. Câmpeanu and S. Yu. Pattern expressions and pattern automata. *Information Processing Letters*, 92(6):267–274, 2004.

[16] B. Carle and P. Narendran. On extended regular expressions. In *Proc. Language and Automata Theory and Applications, Third International Conference, LATA 2009*, volume 5457 of *LNCS*, pages 279–289, 2009.

[17] J. Cassaigne. Unavoidable patterns. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, chapter 3, pages 111–134. Cambridge University Press, Cambridge, New York, 2002.

[18] C. Choffrut and J. Karhumäki. Combinatorics of words. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 6, pages 329–438. Springer, 1997.

[19] J. H. Conway. Unpredictable iterations. In *Proc. 1972 Number Theory Conference*, pages 49–52, Boulder, Colorado, 1972. University of Colorado.

[20] James Currie. Open problems in pattern avoidance. *American Math. Monthly*, 100(8):790–793, 1993.

[21] N. Cutland. *Computability: An Introduction to Recursive Function Theory*. Cambridge University Press, 1980.

[22] E. Czeizler. The non-parametrizability of the word equation $xyz = zvx$: A short proof. *Theoretical Computer Science*, 345(2–3):296–303, 2005.

[23] G. Della Penna, B. Intrigila, E. Tronci, and M. Venturini Zilli. Synchronized regular expressions. *Acta Informatica*, 39(1):31–70, 2003.

[24] V. Diekert. Makanin's algorithm. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, chapter 12, pages 387–442. Cambridge University Press, Cambridge, New York, 2002.

[25] S. Dumitrescu, G. Păun, and A. Salomaa. Languages associated to finite and infinite sets of patterns. *Revue Roumaine de Mathématiques Pures et Appliquées*, 41:607–625, 1996.

[26] V.G. Durnev. Undecidability of the positive $\forall\exists^3$-theory of a free semigroup. *Siberian Mathematical Journal*, 36(5):917–929, 1995. Translated from Sibirskiĭ Matematicheskiĭ Zhurnal, Vol. 36, No. 5, pp. 1067–1080.

[27] A. Ehrenfeucht and G. Rozenberg. Finding a homomorphism between two words is NP-complete. *Information Processing Letters*, 9:86–88, 1979.

[28] G. Filè. The relation of two patterns with comparable language. In *Proc. 5th Annual Symposium on Theoretical Aspects of Computer Science, STACS 1988*, volume 294 of *LNCS*, pages 184–192, 1988.

[29] D.D. Freydenberger. Extended regular expressions: Succinctness and decidability. In *Proc. 28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011*, volume 9 of *LIPIcs*, pages 507–518, 2011. Invited to Special Issue.

[30] D.D. Freydenberger and D. Reidenbach. Bad news on decision problems for patterns. In *Proc. 12th International Conference on Developments in Language Theory, DLT 2008*, volume 5257 of *LNCS*, pages 327–338, 2008.

[31] D.D. Freydenberger and D. Reidenbach. Existence and nonexistence of descriptive patterns. In *Proc. 13th Conference on Developments in Language Theory, DLT 2009*, volume 5583 of *LNCS*, pages 228–239, 2009.

[32] D.D. Freydenberger and D. Reidenbach. Bad news on decision problems for patterns. *Information and Computation*, 208(1):83–96, 2010.

[33] D.D. Freydenberger and D. Reidenbach. Existence and nonexistence of descriptive patterns. *Theoretical Computer Science*, 411(34-36):3274 – 3286, 2010.

[34] D.D. Freydenberger and D. Reidenbach. Inferring descriptive generalisations of formal languages. In *Proc. 23rd Annual Conference on Learning Theory, COLT 2010*, pages 194–206, 2010.

[35] D.D. Freydenberger, D. Reidenbach, and J.C. Schneider. Unambiguous morphic images of strings. *International Journal of Foundations of Computer Science*, 17:601–628, 2006.

[36] J.E.F. Friedl. *Mastering Regular Expressions*. O'Reilly Media, Sebastopol, CA, 2nd edition, 2002.

[37] M. Fulk and S. Jain. Learning in the presence of inaccurate information. *Theoretical Computer Science*, 161:235–261, 1996.

[38] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, New York, 1979.

[39] E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[40] T. Harju and J. Karhumäki. Applications to problems of iterated morphisms. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 7.3, pages 490–492. Springer, 1997.

[41] J. Hartmanis. On Gödel speed-up and succinctness of language representations. *Theoretical Computer Science*, 26(3):335–342, 1983.

[42] T. Head. Fixed languages and the adult languages of 0L schemes. *International Journal of Computer Mathematics*, 10:103–107, 1981.

[43] Ju.I. Hmelevskiĭ. Equations in free semigroups. *Proceedings of the Steklov Institute of Mathematics*, 107, 1971. In Russian. English translation in American Mathematical Society Translations, 1976.

[44] Š. Holub. Polynomial-time algorithm for fixed points of nontrivial morphisms. *Discrete Mathematics*, 309(16):5069–5076, 2009.

[45] M. Holzer and M. Kutrib. The complexity of regular(-like) expressions. In *Proc. 14th Conference on Developments in Language Theory, DLT 2010*, volume 6224 of *LNCS*, pages 16–30, 2010.

[46] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.

[47] O.H. Ibarra, T.C. Pong, and S.M. Sohn. A note on parsing pattern languages. *Pattern Recognition Letters*, 16(2):179–182, 1995.

[48] S. Jain and E. Kinber. Learning and extending sublanguages. *Theoretical Computer Science*, 397:233–246, 2008.

[49] S. Jain, Q. Luo, P. Semukhin, and F. Stephan. Uncountable automatic classes and learning. In *Proc. 20th International Conference on Algorithmic Learning Theory, ALT 2009*, pages 293–307, 2009.

[50] T. Jiang, E. Kinber, A. Salomaa, K. Salomaa, and S. Yu. Pattern languages with and without erasing. *International Journal of Computer Mathematics*, 50:147–163, 1994.

[51] T. Jiang, A. Salomaa, K. Salomaa, and S. Yu. Decision problems for patterns. *Journal of Computer and System Sciences*, 50:53–63, 1995.

[52] J. Karhumäki, F. Mignosi, and W. Plandowski. The expressibility of languages and relations by word equations. *Journal of the ACM*, 47:483–505, 2000.

[53] L. Kari, G. Rozenberg, and A. Salomaa. L systems. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 5, pages 253–328. Springer, 1997.

[54] F. Kaščák. Small universal one-state linear operator algorithm. In *Proc. 17th International Symposium Mathematical Foundations of Computer Science, MFCS 1992*, volume 629 of *LNCS*, pages 327–335, 1992.

[55] S.C. Kleene. Representation of events in nerve nets and finite automata. In C.E. Shannon; J. McCarthy; W.R. Ashby, editor, *Automata Studies*, pages 3–42. Princeton University Press, Princeton, 1956.

[56] S. Kobayashi and T. Yokomori. On approximately identifying concept classes in the limit. In *Proc. 6th International Workshop on Algorithmic Learning Theory, ALT 1995*, volume 997 of *Lecture Notes in Artificial Intelligence*, pages 298–312, 1995.

[57] S. Kobayashi and T. Yokomori. Learning approximately regular languages with reversible languages. *Theoretical Computer Science*, 174:251–257, 1997.

[58] Takeshi Koshiba. Typed pattern languages and their learnability. In *Computational Learning Theory, Second European Conference, EuroCOLT '95*, volume 904 of *LNCS*, pages 367–379, 1995.

[59] G. Kucherov and M. Rusinowitch. Patterns in words versus patterns in trees: A brief survey and new results. In *Perspectives of System Informatics*, volume 1755 of *LNCS*, pages 283–296, 2000.

[60] S. A. Kurtz and J. Simon. The undecidability of the generalized collatz problem. In *Proc. 4th International Conference Theory and Applications of Models of Computation, TAMC 2007*, volume 4484 of *LNCS*, pages 542–553, 2007.

[61] M. Kutrib. The phenomenon of non-recursive trade-offs. *International Journal of Foundations of Computer Science*, 16(5):957–973, 2005.

[62] J.C. Lagarias. The 3x+1 problem: An annotated bibliography (1963–1999), Aug 2009. `http://arxiv.org/abs/math/0309224`.

[63] J.C. Lagarias. The 3x+1 problem: An annotated bibliography, II (2000–2009), Aug 2009. `http://arxiv.org/abs/math/0608208`.

[64] J.C. Lagarias, editor. *The Ultimate Challenge: The* $3x + 1$ *Problem*. American Mathematical Society, Providence, Rhode Island, USA, 2010.

[65] K.S. Larsen. Regular expressions with nested levels of back referencing form a hierarchy. *Information Processing Letters*, 65(4):169–172, 1998.

[66] W. Luo. Compute inclusion depth of a pattern. In *Proc. 18th Annual Conference on Learning Theory, COLT 2005*, volume 3559 of *Lecture Notes in Artificial Intelligence*, pages 689–690, 2005.

[67] G. S. Makanin. The problem of solvability of equations in a free semigroup. *Mat. Sb. (NS)*, 103(2):147–236, 1977. In Russian. English translation in Mathematics of the USSR-Sbornik, 32, 129–198, 1977.

[68] M. Margenstern. Frontier between decidability and undecidability: a survey. *Theoretical Computer Science*, 231(2):217–251, 2000.

[69] A. Mateescu and A. Salomaa. Patterns. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 4.6, pages 230–242. Springer, 1997.

[70] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Upper Saddle River, NJ, 1967.

[71] A.R. Mitchell. Learnability of a subclass of extended pattern languages. In *Proc. 11th Annual Conference on Computational Learning Theory, COLT 1998*, pages 64–71, 1998.

[72] V. Mitrana. Patterns and languages: An overview. *Grammars*, 2(2):149–173, 1999.

[73] Y. Mukouchi. Inductive inference of an approximate concept from positive data. In *Proc. 5th International Workshop on Algorithmic Learning Theory, ALT 1994*, volume 872 of *Lecture Notes in Artificial Intelligence*, pages 484–499, 1994.

[74] T. Nagell, editor. *Selected mathematical papers of Axel Thue*. Universitetsforlaget, Oslo, 1977.

[75] T. Neary and D. Woods. Four small universal turing machines. *Fundamenta Informaticae*, 91(1):123–144, 2009.

[76] Y.K. Ng and T. Shinohara. Developments from enquiries into the learnability of the pattern languages from positive data. *Theoretical Computer Science*, 397:150–165, 2008.

[77] P.G. Odifreddi. *Classical Recursion Theory*, volume I. Elsevier, Amsterdam, 1989.

[78] P.G. Odifreddi. *Classical Recursion Theory*, volume II. Elsevier, Amsterdam, 1999.

[79] E. Ohlebusch and E. Ukkonen. On the equivalence problem for E-pattern languages. *Theoretical Computer Science*, 186:231–248, 1997.

[80] T. Oliveira e Silva. `http://www.ieeta.pt/~tos/3x+1.html`. Retrieved Aug 28th, 2010.

[81] D. Perrin. Words. In M. Lothaire, editor, *Combinatorics on Words*, chapter 1. Addison-Wesley, Reading, MA, 1983.

[82] W. Plandowski. Satisfiability of word equations with constants is in PSPACE. *Journal of the ACM*, 51(3):483–496, 2004.

[83] W. Plandowski and W. Rytter. Application of Lempel-Ziv encodings to the solution of word equations. In *Proc. 25th International Colloquium on Automata, Languages and Programming, ICALP 1998*, volume 1443 of *LNCS*, pages 731–742, 1998.

[84] W. V. Quine. Concatenation as a basis for arithmetic. *Journal of Symbolic Logic*, 11(4):105–114, 1946.

[85] D. Reidenbach. *The Ambiguity of Morphisms in Free Monoids and its Impact on Algorithmic Properties of Pattern Languages*. PhD thesis, Fachbereich Informatik, Technische Universität Kaiserslautern, 2006. Logos Verlag, Berlin.

[86] D. Reidenbach. A non-learnable class of E-pattern languages. *Theoretical Computer Science*, 350:91–102, 2006.

[87] D. Reidenbach. An examination of Ohlebusch and Ukkonen's conjecture on the equivalence problem for E-pattern languages. *Journal of Automata, Languages and Combinatorics*, 12:407–426, 2007.

[88] D. Reidenbach. Discontinuities in pattern inference. *Theoretical Computer Science*, 397:166–193, 2008.

[89] D. Reidenbach and M.L. Schmid. A polynomial time match test for large classes of extended regular expressions. In *Proc. 15th International Conference on Implementation and Application of Automata, CIAA 2010*, volume 6482 of *LNCS*, pages 241–250, 2010.

[90] D. Reidenbach and J.C. Schneider. Morphically primitive words. *Theoretical Computer Science*, 410(21-23):2148–2161, 2009.

[91] H. Rogers. *Theory of Recursive Functions and Effective Computability*. MIT Press, Cambridge, MA, 1992. 3rd print.

[92] E. Roosendaal. `http://www.ericr.nl/wondrous/index.html`. Retrieved Aug 28th, 2010.

[93] G. Rozenberg and A. Salomaa. *The Mathematical Theory of L systems*. Academic Press, New Lork, London, 1980.

[94] G. Rozenberg and A. Salomaa. When L was young. In G. Rozenberg and A. Salomaa, editors, *The Book of L*, pages 383–401. Springer, Berlin, 1986.

[95] A. Salomaa. *Formal Languages*. Academic Press, New York, London, 1973.

[96] A. Salomaa. Patterns. *Bulletin of the EATCS*, 54:194–206, 1994.

[97] A. Salomaa. Return to patterns. *Bulletin of the EATCS*, 55:144–157, 1995.

[98] K. Salomaa. Patterns. In C. Martin-Vide, V. Mitrana, and G. Păun, editors, *Formal Languages and Applications*, number 148 in Studies in Fuzziness and Soft Computing, pages 367–379. Springer, 2004.

[99] K. Salomaa. Patterns, 2006. Lecture, 5th PhD School in Formal Languages and Applications, URV Tarragona.

[100] K. Salomaa. Personal communication to the author, 2008.

[101] M. Schaefer, E. Sedgwick, and D. Stefankovic. Recognizing string graphs in NP. *Journal of Computer and System Sciences*, 67(2):365–380, 2003.

[102] T. Shinohara. Polynomial time inference of extended regular pattern languages. In *Proc. RIMS Symposia on Software Science and Engineering, Kyoto*, volume 147 of *LNCS*, pages 115–127, 1982.

[103] A. Thue. Über unendliche Zeichenreihen. *Kra. Vidensk. Selsk. Skrifter. I Mat. Nat. Kl.*, 7, 1906. In German. Reprinted in [74].

[104] H.S. Wilf. *generatingfunctionology*. Academic Press, New York, second edition, 1994.

[105] K. Wright. Identification of unions of languages drawn from an identifiable class. In *Proc. 2nd Annual Workshop on Computational Learning Theory, COLT 1989*, pages 328–333, 1989.

Index

Lebenslauf

Name:	Dominik D. Freydenberger
Adresse:	Am Fels 14
	55234 Albig
Geburtstag:	31. Oktober 1979
Geburtsort:	Regensburg
Familienstand:	ledig

1986–1987 Kreuzbergschule Schwandorf

1987–1990 Grundschule Gau-Odernheim

1990–1999 Gymnasium am Römerkastell Alzey

1999–2000 Wehrdienst im FmRgt 920, Kastellaun

2000–2006 Studium der Informatik an der TU Kaiserslautern

März 2006 Abschluss als Diplom-Informatiker,
Diplomarbeit bei Prof. Dr. Rolf Wiehagen

2006–2007 Besuch von Vorlesungen der *5th International PhD School in Formal Languages and Applications*, Tarragona, Spanien

März 2007– Promotionsstudent und wissenschaftlicher Mitarbeiter an der Johann Wolfgang Goethe-Universität, Frankfurt am Main, in den Professuren von Prof. Dr. Detlef Wotschke (bis Oktober 2009) und Prof. Dr. Nicole Schweikardt (seit Oktober 2009)