Beyond regular : pattern matching with extended regular expressions

Benjamin Robert Carle
University at Albany, State University of New York, bencarle@gmail.com

The University at Albany community has made this article openly available. Please share how this access benefits you.

Follow this and additional works at: https://scholarsarchive.library.albany.edu/legacy-etd

Part of the Computer Sciences Commons

Recommended Citation
https://scholarsarchive.library.albany.edu/legacy-etd/156

This Dissertation is brought to you for free and open access by the The Graduate School at Scholars Archive. It has been accepted for inclusion in Legacy Theses & Dissertations (2009 - 2024) by an authorized administrator of Scholars Archive. Please see Terms of Use. For more information, please contact scholarsarchive@albany.edu.
BEYOND REGULAR
PATTERN MATCHING WITH EXTENDED REGULAR EXPRESSIONS

by

Benjamin Carle

A Dissertation
Submitted to the University at Albany, State University of New York
in Partial Fulfillment of
the Requirements for the Degree of
Doctor of Philosophy

College of Computing and Information
Department of Computer Science
University at Albany
Albany, New York

May 1, 2010
BEYOND REGULAR
PATTERN MATCHING WITH EXTENDED REGULAR EXPRESSIONS

by

Benjamin Carle

© Copyright 2010
To my parents
ABSTRACT

Regular expression pattern matching or some variant thereof is present in almost all software that deal with textual data. In this work we have considered three types of extensions to the traditional regular expression, or regex pattern matching format. Regex pattern matching tools such as those in the Unix utility egrep and the popular scripting language Perl commonly include extended functionality beyond the classical definition of regular expressions. The main focus is on Extended Regular Expressions as studied by Câmpeanu, Salomaa and Yu [1]. We offer an additional pumping lemma that will show that a new class of languages is not recognizable by extended regular expressions and discuss closure properties, decidability, and complexity issues relating to these languages.

Another extension of the regular languages considered is the class of Extended Multi-Pattern Languages (EMPL), introduced by Nagy in [2]. We show that this class is a strict subclass of the family of languages recognized by extended regular expressions and discuss the decidability of the equivalence problem for these languages.

Synchronized Regular Expressions, as introduced by Penna, Intrigila, Tronci and Zilli [3], extend the EREG languages by allowing exponent variables that range over the natural numbers. We offer some clarifications of the matching semantics and a surprising result regarding linear synchronized regular expressions. A simplified proof of the language of palindromes is provided, showing that the family of languages defined by synchronized regular expressions is not a superset of the context-free languages.

Finally, we provide an overview of the relative expressive power of these three classes of languages and relate them to other significant language classes. In conclusion we discuss some current and potential future applications of extended pattern matching and some open problems.
ACKNOWLEDGMENT

During the years I have been working on my doctorate and this dissertation, I have been extremely fortunate to be in the company of brilliant friends, teachers, and researchers. First and foremost is my advisor, Paliath Narendran. I am immensely grateful for his guidance and support throughout the entire course of my graduate study. This work would not have been possible without him. He taught me to be a good researcher, reviewer, presenter, writer, teacher, and thinker. He is truly a positive force in teaching and research, an opinion that is shared by his many other students and colleagues. I owe him my sincere thanks and appreciation for all that I have learned and accomplished.

I would like to thank Neil Murray, not only for being a member of my committee, but for trusting me with so much teaching responsibility in his courses over the years. I have become a confident and capable instructor because of this opportunity. I would also like to thank Chittibabu Guda for serving on my committee and providing valuable insight into potential applications of my research.

I would especially like to thank Kai Salomaa for pioneering this field of research and for his encouragement and time spent discussing this work with me.

I would like to thank George Berg for giving me the opportunity to be an instructor in the department while I finished my degree. The responsibility of developing and teaching courses on my own has enabled me to build a valuable teaching portfolio with progressive learning ideas. I would like to thank Sekharipuram S. Ravi for being an excellent teacher and inspiring me to strive for the same. I would also like to thank Patricia Keller for being endlessly helpful throughout the long string of problems that inevitably arise during the process of earning a graduate degree.

Finally, I dedicate this work to my parents, who have always been unquestioningly supportive of everything I’ve done. Without their example and encouragement over the course of my life, I never would have accomplished as much as I have. To my parents and my sister, I am forever grateful.
## CONTENTS

ABSTRACT ................................................................. iv

ACKNOWLEDGMENT .................................................. v

LIST OF FIGURES ..................................................... vii

1. Introduction .......................................................... 1

2. Extended Regular Expressions ..................................... 4
   2.1 Definitions ..................................................... 4
   2.2 Match Trees for Extended Regular Expressions ............ 5

3. Pumping Lemmas ..................................................... 7
   3.1 Pumping Lemma Background .................................... 7
   3.2 The CSY Pumping Lemma ....................................... 8
   3.3 The CNS Pumping Lemma ....................................... 10
   3.4 Comparing Pumping Lemmas .................................... 11

4. Decidability and Complexity Results for Extended Regular Expressions .. 14
   4.1 Emptiness of Intersection of Extended Regular Expressions .... 14
   4.2 Matching Problem Over a Unary Alphabet .................... 16

5. Extended Multi-Pattern Languages ................................. 19
   5.1 Definitions ..................................................... 19
   5.2 Erasing and Non-Erasing Varieties of EMPL .................. 22
   5.3 Comparing EMPL, EREG and CFL .............................. 25
   5.4 The Equivalence Problem for EMSPL ........................... 29

6. Remarks on Synchronized Regular Expressions .................... 31
6.1 Introduction and Basic Definitions . . . . . . . . . . . . . . . . . . . . 31
6.2 Semantics of SRE-matching . . . . . . . . . . . . . . . . . . . . . . . 33
6.3 Linear SRE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
6.4 Exponent-Free SRE . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
6.5 On the Language of Palindromes . . . . . . . . . . . . . . . . . . . . 38
6.6 NP-completeness of SRE-matching without Backreferences . . . . . . 41
6.7 Unrestricted SRE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
7. Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45

APPENDICES

A. A Construction of Cube-Free Strings . . . . . . . . . . . . . . . . . . . . 48
Glossary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
Acronyms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
LIST OF FIGURES

7.1 Relationship between language classes. ............................ 46
CHAPTER 1

Introduction

Regular expressions allow us to describe and search text through pattern matching. One regular expression, or regex, can match a finite or possibly infinite set of strings. We call this set the language of that regular expression. In formal language theory we typically study regular expressions over a small alphabet, e.g., $\Sigma = \{a, b\}$, an alphabet of only two characters. In practice, regular expressions are commonly used over larger alphabets, e.g., ASCII, made up of letters, numbers, and special characters used in text documents and databases.

Regex pattern matching or some variant thereof is present in almost all software that deals with textual data. Examples include unix command line programs such as Grep, Egrep, and Awk, text editors such as Emacs, vi, and Microsoft Word, programming languages such as Perl, PHP, Java, C++, Python, and .NET, database servers such as MySQL, SQL Server, and Oracle, and even searching with Google.

While each of these programs have their own implementation details, most of them are modeled after the classical regular expression paradigm, based on the three main operators union, concatenation, and Kleene star. Other features such as character classes, wildcards, and line start/end anchors can be replicated with these three operators and therefore do not change the functionality of the regex system. However, many of these tools also contain additional features that increase the expressive power of the pattern matching system. The capabilities and limitations of these extensions are the focus of this work.

The addition of backreferences to the regular expression syntax is perhaps the most significant and prevalent extension available in modern regex tools. The earliest appearance of backreferences in a pattern matching utility was in SNOBOL [4], developed at Bell Labs by David Farber, Ralph Griswold, and Ivan Polonsky around 1964. Backreferences allow users to relate part of an expression to a previously matched substring, effectively creating a memory in an extended regular expression, or eregex.
This inherent memory enables extended regular expressions to define many additional languages, such as the language of repeated words, a well known non-regular language. The presence of backreferences is widespread in regex utilities. In fact, the modern versions of each of the tools mentioned above has the ability to match some non-regular languages through some form of backreferencing.

Backreferences are commonly employed by computer users to find repeated words or phrases in text. This could be for detecting errors such as typos or for finding relationships in documents. They are also used in find and replace operations to modify text in a specific way, though this use does not necessarily imply non-regular functionality.

Despite the availability and frequency of extended regular expression usage, not much was formally known about the expressive power of these tools until the pioneering work of Câmpeanu, Salomaa and Yu [1]. This thesis aims to extend their work by offering additional results to better understand this class of languages. In this work we show that the family of languages recognizable by extended regular expressions is not closed under intersection, thereby settling a previously open problem. Furthermore, we introduce a different pumping lemma and use that lemma to show a class of languages that satisfy the pumping property of [1] but are not expressible by extended regular expressions. We also give some new results on the decidability of the emptiness of the intersection problem and the complexity of the matching problem for the extended regular languages.

In addition to the extended regular expressions as defined in [1] we consider other extensions to classical regular expressions. All of these extensions are similar in that they relate different parts of a pattern to each other through the use of memory or synchronization in the matching scheme, but differ in their approach and expressive power. The work of Nagy [2] is considered, which extends the multi-pattern languages (MPL) defined by Kari, Mateescu, Paun and Salomaa [5]. We show that the class Nagy defines, which we call extended multi-pattern languages (EMPL), is a strict subclass of the family of languages recognized by extended regular expressions. We discuss the relationship between erasing and non-erasinig varities of extended multi-pattern languages and the equivalence problem for extended multi-star-pattern
languages, a subclass of EMPL formed without using the union operator.

For a third class of languages we discuss synchronized regular expressions (SRE), which, in addition to backreferences, allow exponent variables which range over the natural numbers. They were defined by G. Della Penna, B. Intrigila, E. Tronci and M. Venturini-Zili in [3]. We offer a clarification of the matching semantics for SRE as well as an analysis of how the exponent matching works that provides a surprising result relating to linear SRE. We also provide a simplified version of a proof showing that SRE cannot define the language of all palindromes, which is critical in relating SRE to the context-free languages.

Finally, this thesis concludes with a discussion of the relationship of the three classes of languages covered to each other and other established language classes such as the regular languages, context-free languages, and context-sensitive languages. We also mention potential application areas for these pattern matching paradigms and review some open problems.
CHAPTER 2
Extended Regular Expressions

2.1 Definitions

The syntax of extended regular expressions as in egrep and Perl is defined in [1].
Standard regular expressions, as specified in formal language theory, are extended
using backreferences. The backreference \n stands for the string previously matched
by the regular expression between the \nth left parenthesis and the corresponding right
parenthesis. These regular expressions with the addition of backreferences are called
extended regular expressions or eregex. As is well-known, this significantly increases
expressive power; for instance, the expression ((aa)+a)\1* specifies the language

\{ a^i \mid i > 0 \text{ and } i \text{ is not a power of 2} \}

which is not even context-free. Similarly (a+)(b+)\1\2 specifies

\{ a^ib^ja^ib^j \mid i, j > 0 \}

which is not context-free either.

For clarity, let us number left parentheses, starting with 1, from the left. Give
the same numbers to the corresponding (matching) right parentheses.

\[
\left( \left( \left( \left( \ldots \right) \ldots \right) \ldots \right) \ldots \right)
\]

\[
1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 1
\]

As in [1] we assume that any occurrence of a backreference \m in an extended
regular expression is preceded by )\m.

Matching a string with an extended regular expression (eregex-matching) is
often defined as follows: (paraphrasing [6])
1. If \( a \) is a symbol in the alphabet, then \( a \) matches \( a \).

2. If \( r \) matches a string \( x \), then \( \left( \frac{r}{i} \right) \) matches \( x \) and the value \( x \) is assigned to \( \backslash i \).

3. \( \backslash j \) matches the string that has been assigned to it.

4. If \( r_1 \) and \( r_2 \) are eregexes, then \( r_1 \cup r_2 \) matches any string matched by either \( r_1 \) or \( r_2 \).

5. If \( r_1 \) and \( r_2 \) are eregexes, then \( r_1 r_2 \) matches any string of the form \( xy \) where \( r_1 \) matches \( x \) and \( r_2 \) matches \( y \).

6. If \( r \) is an eregex, then \( r^* \) matches any string of the form \( x_1 \ldots x_n \), \( n \geq 0 \), where \( r \) matches each \( x_i \) \((1 \leq i \leq n)\).

2.2 Match Trees for Extended Regular Expressions

A more precise definition of a match is given in [1] using ordered trees. Here we give that definition with a slight modification. Positions in an ordered tree are denoted by sequences of positive integers, with the empty sequence denoting the root position. (See [7] for a formal definition.) Note that left-to-right lexicographic order \( \prec_{lex} \) among positions corresponds to pre-order traversal.

An ordered tree \( T \) is a valid match-tree for \( w \) and \( \alpha \) if and only if:

1. The root of \( T \) has the label \( (w, \alpha) \).

2. For every node \( u \in \text{dom}(T) \),

   (a) if \( T(u) = (w, a) \) for some \( a \in \Sigma \), then \( u \) is a leaf node and \( w = a \).

   (b) if \( T(u) = (w, \beta_1 \beta_2) \), then \( u \) has two children labeled, respectively, by \( (w_1, \beta_1) \) and \( (w_2, \beta_2) \) where \( w_1w_2 = w \).

   (c) if \( T(u) = (w, \beta_1|\beta_2) \), then \( u \) has one child labeled by either \( (w, \beta_1) \) or \( (w, \beta_2) \).
(d) if $T(u) = (w, \beta^*)$, then either $u$ is a leaf node and $w = \lambda$ or $u$ has $k \geq 1$ children labeled by $(w_1, \beta), \ldots, (w_k, \beta)$ where each $w_i \in \Sigma^+$, and $w = w_1 \ldots w_k$.

(e) if $T(u) = (w, (\gamma_i))$, then it has one child labeled by $(w, \gamma)$.

(f) if $T(u) = (w, \backslash m)$, then $u$ is a leaf node, $(\beta)^m$ is a subexpression of $\alpha$, and there is a node $v$ to the left of $u$ such that $T(v) = (w, (\beta)^m)$ and no node between $v$ and $u$ has $(\beta)^m$ in its label. In other words, $w$ is the string previously (in the left-to-right pre-order) matched by $(\beta)^m$.

The difference between this definition and the one in [1] is that unassigned back-references are not set to the empty string $\lambda$ as default in our definition. Thus there is no valid match-tree for $b$ and $((aa) | \backslash 2b)$.

The language denoted by an extended regular expression $\alpha$ is defined as

$$\mathcal{L}(\alpha) = \{ w \in \Sigma^* \mid (w, \alpha) \text{ is the label at the root of a valid match-tree} \}.$$ 

Let EREG be the family of languages defined by extended regular expressions. A language $L$ is an EREG language if and only if there is an extended regular expression $\alpha$ such that $L = \mathcal{L}(\alpha)$. In relation to the regular languages (REG), it can be seen that

$$REG \subset EREG$$
CHAPTER 3
Pumping Lemmas

3.1 Pumping Lemma Background

Pumping lemmas have long been used as a tool to show that a given language does not belong to a certain class of languages. The pumping lemma for regular languages is the most common and widely known. This theorem takes advantage of the fact that all regular languages are recognizable by deterministic automata with a finite number of states. If a string in the language is longer than the number of states, then some state must be repeated in recognizing the string. Therefore the path between that state and itself represents a loop, and that loop can be repeated, or pumped, to create other strings that must be in the language. If the pumped string is not in the language then that language is not regular.

This lemma can also be viewed from the perspective of regular expressions. If a regular expression matches a string that is longer than the regular expression then it must contain a star operator. This star operator can be pumped to obtain other strings that must be in the language. In extended regular expressions we have a similar situation with the added consideration of backreferences. An eregex has a pumping length, based on the length of the expression and the number of backreferences it contains. Any string longer than this pumping length must contain a star and can be pumped. The key difference is that the pump can occur multiple times in the string since the star operator can be backreferenced multiple times in the string.

The first such pumping lemma of this kind was introduced by Câmpeanu, Salomaa and Yu [1]. This result is a valuable tool in proving that certain languages are not in EREG. This chapter will also present an alternate pumping lemma that will allow us to prove a different class of languages is not in EREG.
3.2 The CSY Pumping Lemma

Câmpeanu, Salomaa and Yu [1] proved the following pumping lemma for EREG languages. To the best of our knowledge, this was the first pumping lemma of its kind.

**Lemma 3.1 (The CSY Pumping Lemma) [1]** Let $\alpha$ be an extended regular expression. Then there is a constant $N > 0$ such that if $w \in L(\alpha)$ and $|w| > N$, then there is a decomposition $w = x_0y_1x_1 \cdots y_1x_m$ for some $m \geq 1$, such that

1. $|x_0y| < N$,

2. $|y| \geq 1$, and

3. $x_0y^jx_1y^j \cdots y^jx_m \in L(\alpha)$ for all $j > 0$.

This pumping lemma can be used on the following language to settle the previously open question of closure under intersection.

**Lemma 3.2** The language

$$S = \{ a^i b a^{i+1} b a^k \mid k = i(i + 1)k' \text{ for some } k' > 0, i > 0 \}$$

is not an EREG language.

**Proof:** Assume $S$ is expressed by an eregex and let $N$ be the constant given by the CSY pumping lemma. Consider $w = a^N b a^{N+1} b a^N(N+1)$. Then there is a decomposition $w = x_0y_1x_1y_2 \cdots y_1x_m$ for some $m \geq 1$ and from the pumping lemma $y = a^p$ for some $p \geq 1$. Since $|x_0y| < N$ there must be at least one occurrence of $y$ in $a^N$. Assume there are $q \geq 1$ occurrences of $y$ in $a^N$. By (3) from the CSY pumping lemma there must also be $q$ occurrences of $y$ in $a^{N+1}$ as otherwise $x_0y^2x_1y^2 \cdots y^2x_m \notin S$. Let $r$ be the number of occurrences of $y$ in $a^{N(N+1)}$ and note that $N(N + 1) \geq rp \geq 0$. 

8
Now consider $x_0y^2x_1y^2 \ldots y^2x_m = a^{N+qp}ba^{N+1+qp}ba^{N(N+1)+rp} \in S$. Then

$$k_2(N + qp)(N + 1 + qp) = N(N + 1) + rp$$

for some $k_2$. Since $rp \leq N(N+1)$, we know that $N(N+1)+rp \leq 2(N(N+1))$. Since $qp \geq 1$, we know that $(N + qp)(N + 1 + qp) \geq (N + 1)(N + 2) > N(N + 1)$. Thus $k_2(N + qp)(N + 1 + qp) \geq k_2(N + 1)(N + 2) > k_2N(N + 1)$. Note that $k_2N(N + 1) < 2(N(N+1))$ is only true for $k_2 = 1$. Thus we have $(N + qp)(N + 1 + qp) = N(N + 1) + rp$, so

$$rp = q^2p^2 + qp(2N + 1) \quad (3.1)$$

Now consider $x_0y^3x_1y^3 \ldots y^3x_m = a^{N+2qp}ba^{N+1+2qp}ba^{N(N+1)+2rp} \in S$. Then it must be that

$$k_3(N + 2qp)(N + 1 + 2qp) = N(N + 1) + 2rp$$

for some $k_3$. But note that $(N + 2qp)(N + 1 + 2qp) = N(N + 1) + 4q^2p^2 + 2qp(2N + 1)$, whereas $N(N + 1) + 2rp = N(N + 1) + 2q^2p^2 + 2qp(2N + 1)$ by (1). Hence such a $k_3$ cannot exist. \hfill \Box

**Theorem 3.1** EREG languages are not closed under intersection.

**Proof:** The language $S$ of the previous lemma is the intersection of $L((a+)b(\backslash a)b(\backslash 1)a)\backslash 1+)$ and $L((a+)b(\backslash a)b(\backslash 2)+)$. \hfill \Box
3.3 The CNS Pumping Lemma

While the CSY pumping lemma can be used to show a large class of languages are not in EREG, it is not sufficient to prove this result for all non-EREG languages. For example, the language of all palindromes satisfies the CSY pumping property, but is not an EREG language as we will show. A key feature of the CSY pumping lemma is the localization of the pump. That is, the pump $y$ must occur within the first $N$ characters of the string. This feature is key to many pumping lemma proofs, but not helpful for the language of palindromes. We offer an alternative pumping lemma proofs, but not helpful for the language of palindromes. We offer an alternative pumping lemma that trades off the localization of the pump for an upper bound on the number of pumps that can occur. This bound illustrates a key limitation of extended regular expressions: they have a finite number of backreferences. It also exploits the key non-EREG characteristics of languages such as palindromes in that to be successfully defined they would require an infinite number of backreferences.

**Lemma 3.3** Let $\alpha$ be an eregex. Then for any $k > 0$ there are positive integers $N(k)$ and $m$ such that if $w \in \mathcal{L}(\alpha)$ and $|w| > N(k)$ then $w$ has a decomposition $w = x_0yx_1y \cdots yx_m'$ ($1 \leq m' \leq m$) such that

1. $|y| \geq k$, and
2. $x_0y^jx_1y^j \cdots y^jx_m' \in \mathcal{L}(\alpha)$ for all $j > 0$.

**Proof:** Let $N(k) = |\alpha|2^tk$ where $t$ is the number of backreferences in $\alpha$. Then if $|w| > N$ there is a substring of $w$ of length $\geq k$ that matches a Kleene star in $\alpha$. (Each backreference can at most double the length of the word it matches.) Let $m = 2^t + 1$.

Let $w = x_0yz$ where $y$ is the rightmost largest substring of $w$ that matches a Kleene star. Then clearly $|y| \geq k$. Let $t'$ equal the number of (direct or indirect) backreferences to any expression that contains this star. Let $m' = t' + 1$. Let $z = x_1yx_2y \cdots x_m$ where the multiple instances of $y$ correspond to these backreferences. Then $w = x_0yx_1y \cdots yx_m'$ and clearly $x_0y^jx_1y^jx_2y^j \cdots y^jx_m' \in \mathcal{L}(\alpha)$ for all $j \geq 1$. □
3.4 Comparing Pumping Lemmas

With two pumping lemmas for extended regular languages it becomes important to understand their relative capabilities in proving that languages are not in EREG.

Definition: Let $L$-CSY be the family of languages that satisfy the CSY pumping property. Let $L$-CNS be the family of languages that satisfy the CNS pumping property.

If either $L$-CSY $\subseteq$ $L$-CNS, or $L$-CNS $\subseteq$ $L$-CSY, then only one of the pumping lemmas would be required. We will show in this section that this is not the case.

Lemma 3.4  The language $\mathcal{P} = \{wcw^R \mid w \in \{a,b\}^*\}$ satisfies the CSY pumping property. ($w^R$ stands for the reverse of $w$.)

Proof: Given any $w \in \mathcal{P}$ with $|w| > N$ we can let $m = 2$ and form the decomposition $w = x_0yx_1yx_2$. If we let $x_0 = \epsilon$, $x_2 = \epsilon$, and $|y| = 1$, then $y \in \Sigma$ and we can easily satisfy the three conditions as follows:

1. $|x_0y| = 1 < N$,
2. $|y| = 1 \geq 1$, and
3. $x_0y^jx_1y^jx_2 = y^{j-1}wy^{j-1} \in L(\alpha)$ for all $j > 0$.

Lemma 3.5  The language $\mathcal{P} = \{wcw^R \mid w \in \{a,b\}^*\}$ is not an EREG language.

Proof: Assume $\mathcal{P}$ is an EREG language. Let $k = 5$. Let $N(k)$ and $m$ be the constants given by Lemma 5. Consider $w = (abaabb)^{N(k)}c(bbaaba)^{N(k)}$. Then there is a decomposition $w = x_0y_1y_2 \cdots y_1y_{m'}$ for some $m' < m$. By Lemma 3.3, $|y| \geq 5$. 

(k = 5) Observe that $(abaabb)^{N(k)}$ and $(bbaaba)^{N(k)}$ do not share any common substrings of length ≥ 5. Therefore, y must occur to the left or the right of c, but not both. Consequently, $x_0y^2x_1y^2 \ldots y^2x_m' \notin \mathcal{P}$.

Lemma 3.6 The language

$$Q = \{ w_1cw_2 \ldots w_n c \mid w_1 \ldots w_n \in \{a,b\}^*, \ |w_1| = |w_2| = \ldots = |w_n|, \ n \geq 0 \}$$

satisfies the CNS pumping property.

Proof: Given any $k > 0$ and any $w \in Q$ with $|w| > N(k)$ we can let $m' = 1$ and form the decomposition $w = x_0yx_1$. If we let $x_0 = \epsilon$, $x_1 = \epsilon$, and $y = w$ then we can easily satisfy the two conditions as follows:

1. $|y| = |w| \geq 1$ since $N(k) \geq k$, and
2. $x_0y^jx_1 = w^j \in \mathcal{L}(\alpha)$ for all $j > 0$. □

Lemma 3.7 The language

$$Q = \{ w_1cw_2 \ldots w_n c \mid w_1 \ldots w_n \in \{a,b\}^*, \ |w_1| = |w_2| = \ldots = |w_n|, \ n \geq 0 \}$$

is not an EREG language.

Proof: Assume Q is expressed by an eregex and let $N$ be the constant given by the CSY pumping lemma. Consider $w = a^N cb^N c$. Then there is a decomposition $w = x_0yx_1y \ldots yx_m$ for some $m \geq 1$. From the CNS pumping lemma $|x_0y| < N$, so $y$ must be a substring of $a^N$. Let $y = a^p$ for some $p \geq 1$. It can be seen that all occurrences of $y$ must be within $a^N$, since there are no other $a$s in $w$. Consider $w' = x_0y^2x_1y^2 \ldots y^2x_m = a^{N+p|m} cb^N c$. Since $p \geq 1$ and $m \geq 1$ we know that
\[ N + pm > N. \text{ Consequently, } w' \notin \mathcal{Q}. \]

**Theorem 3.2** \textit{L-CSY and L-CNS are incomparable.}

**Proof:** This result follows directly from the previous lemmas.

It remains an open problem whether \( \text{L-CSY} \cap \text{L-CNS} = \text{EREG} \).
CHAPTER 4
Decidability and Complexity Results for Extended Regular Expressions

4.1 Emptiness of Intersection of Extended Regular Expressions

We can show, by a reduction from the membership problem for phrase structured grammars, that

Theorem 4.1 The following problem is undecidable:

Emptiness of Intersection of Extended Regular Expressions (EIERE):

Instance: Two eregexes α and β.
Question: Is \( L(\alpha) \cap L(\beta) \) empty?

Proof: The reduction is from the membership problem for phrase-structure grammars (MPSG), a known undecidable problem, as mentioned earlier. A phrase structure grammar is specified as \( G = (V, \Sigma, P, S) \) where \( V \) is a finite nonempty set called the total vocabulary, \( \Sigma \subseteq V \) is a finite nonempty set called the terminal alphabet, \( N = V - \Sigma \) is the nonterminal alphabet, \( S \in N \) is the start symbol and \( P \) is a finite set of rules (or productions) of the form \( l \rightarrow r \) where \( l \in V^* NV^* \) and \( r \in V^* \). The membership problem MPSG is defined as follows:

Instance: Phrase-structure grammar \( G = (V, \Sigma, P, S) \) and a string \( w \in \Sigma^* \)
Question: Is \( w \in \mathcal{L}(G) \)?

Given an instance of MPSG, we construct an instance of EIERE as follows:

For each production \( l_i \rightarrow r_i \in P \) let \( \alpha_i = \#((\Sigma)^*) l_i ((\Sigma)^*) \# \backslash 1 r_i \backslash 3 \), for some \( \# \notin V \), for \( 1 \leq i \leq |P| \). Let \( \alpha = (\alpha_1|\alpha_2|\ldots|\alpha_n)^* \).

Note that the backreferences will have to be renumbered, replacing each \( \backslash j \) in \( \alpha_i \) with \( \backslash j' \) where \( j' = 4(i - 1) + j + 1 \).
So, $\mathcal{L}(\alpha)$ is the language of sequences of derivation steps (though not necessarily continuous).

$$\#w_1\#w'_1 \#w_2\#w'_2 \#w_3\#w'_3 \ldots \#w_n\#w'_n$$

where each $w_i = xly$ and $w'_i = xry$ for $1 \leq i \leq n$ for some $x, y \in \Sigma^*$ and some $l \rightarrow r \in P$.

We now define $\beta$ to enforce derivation continuity. Consider $\beta_0 = \#((\Sigma)^*) \#1$ which matches strings of the form $\#w_i\#w_i$ for $w_i \in \Sigma^*$. Let $\beta = \#S(\#((\Sigma)^*) \#2) * \#w$. Then $\mathcal{L}(\beta)$ contains all strings of the form

$$\#S\#w_1\#w_1\#w_2\#w_2 \ldots \#w_n\#w_n\#w$$

for some $n \geq 0$

where each underbraced segment matches $\beta_0$.

Thus $\mathcal{L}(\beta)$ is the language of all continuous steps. (Not necessarily derivation steps from $G$.)

Finally, if we take the intersection of the two languages, namely $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$, we get strings of the form

$$\#S\#w_1\#w_1\#w_2\#w_2 \ldots \#w_n-1\#w_n-1\#w_n$$

where $w_i = xly$ and $w_{i+1} = xry$ for each $1 \leq i < n; x, y \in \Sigma^*$; and $l \rightarrow r \in P$ and $S \rightarrow w_1 \in P, w_n = w$. That is, we get sequences of continuous derivation steps beginning at $S$ and ending in $w$.

Therefore, $w \in \mathcal{L}(G)$ iff $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta) \neq \phi$
4.2 Matching Problem Over a Unary Alphabet

We now consider the Matching Problem for Extended Regular Expressions (MERE):

Instance: An eregex $\alpha$ and a string $w \in \Sigma^*$.

Question: Is $(w, \alpha)$ the label at the root of a valid match tree?

This has been shown to be NP-complete [6]. It turns out that the problem is NP-complete even if the target alphabet is unary:

**Theorem 4.2** The matching problem for extended regular expressions is NP-complete even when the target (subject) string is over a unary alphabet.

**Proof:** Membership in NP follows from the earlier result. NP-hardness can be proved by a reduction from the vertex cover problem.

Vertex Cover (VC)

Instance: A graph $G = (V, E)$ and a positive integer $k \leq |V|$.

Question: Is there a $V' \subseteq V$ such that $|V'| \leq k$ and 

$$\forall (u, v) \in E : u \in V' \lor v \in V'$$

Given an instance of VC, construct an instance of MERE as follows: Define $n = |V|$ and $m = |E|$. Without loss of generality, assume the vertices are numbered from 2 to $n + 1$, so $V = \{2,3,\ldots n+1\}$ and $E \subseteq \{(i,j) \mid 2 \leq i \leq n + 1, 2 \leq j \leq n + 1\}$.

(Note: The $n$ vertices are numbered from 2 to $n + 1$ to account for the shifting of backreferences caused by the outer parenthesis of $\alpha_0$, defined below.) Let $\Sigma = \{a\}$. Let $w = a^{k+|E|} = a^{k+m}$.

**Vertex Component:** Construct $\alpha$ as follows:

Let $\alpha_0 = (\begin{array}{cccc} a & a & a & \cdots & a & (a) & (a) & (a) & (a) & (a) \end{array})^*$

That is, $\alpha_0$ is $n$ copies of $(a)$ connected by $or$, and then starred. Note that $\alpha_0$ can be constructed in $O(|V|)$ time.
Edge Component: Assume the edges are ordered from 1 to \(m\): \(e_t \in E\) for \(1 \leq t \leq m\). For each \(e_t = (i, j) \in E\), let \(\alpha_t = (\{i\} \mid \{j\})\). That is, each \(\alpha_t\) represents the \(t^{th}\) edge via backreferences, with the backreference incremented by one to account for the outer parenthesis in \(\alpha_0\). Note that this can be done in \(O(|E|)\) time.

Finally, let \(\alpha = \alpha_0\alpha_1\alpha_2\ldots\alpha_m\). Note that \(\alpha\) can be constructed in \(O(|V| + |E|)\) time. We now show that \(\alpha\) matches \(w\) iff \(G\) has a vertex cover of size \(\leq k\). Suppose \(V' \subseteq V\) is a vertex cover for \(G\) with \(|V'| \leq k\). Then we can find a valid match tree for \((w, \alpha)\) as follows: We can safely assume that \(|V'| = k\), since additional vertices from \(V\) can always be added to make this true. Let us start by matching \(k\) iterations of \(\alpha_0\), one for each \(v \in V'\). For each \(v \in V' \subseteq \{2,3,\ldots,n+1\}\) match the \(v^{th}\) option of the or in \(\alpha_0\) on a different iteration. More specifically, if we let \(V' = \{v_1, v_2, \ldots, v_k\} \subseteq \{2,3,\ldots,n+1\}\) then on the \(v_i^{th}\) iteration of \(\alpha_0\) match the \(v_i^{th}\) option of the or in \(\alpha_0\), which will assign \(a\) to \(\{v_i\}\). Thus, \(\alpha_0\) matches \(a^k\). Recall that each \(\alpha_t = (\{i\} \mid \{j\})\) for \(1 \leq t \leq m\) represents edge \(e_t = (i, j)\). Since \(V'\) is a vertex cover for \(G\), at least one of \(\{i,j\}\) is in \(V'\). Therefore, at least one of \(\{\{i\},\{j\}\}\) is already defined in our match tree. If \(\{i\}\) is defined, match it. Otherwise match \(\{j\}\). Furthermore, any defined backreference \(\{2,\{3,\ldots\n+1\}\}\) can only match a single \(a\), so \(\alpha_1\alpha_2\ldots\alpha_m\) matches \(a^m\).

Therefore, \(\alpha = \alpha_0\alpha_1\alpha_2\ldots\alpha_m\) matches \(a^k a^m = a^{k+m} = w\).

Conversely, suppose \((w, \alpha) = (a^{k+m}, \alpha)\) is the root of a valid match tree. Then we can find a vertex cover \(V' \subseteq V\) for \(G\) with \(|V'| \leq k\) as follows: To begin, let \(V' = \phi\). Since there is a match for \(\alpha\), each of \(\alpha_1\alpha_2\ldots\alpha_m\) must be defined. Thus, each \(\alpha_t\) for \(1 \leq t \leq m\) matches a single \(a\). For each \(\alpha_t = (\{i\} \mid \{j\})\), if \(\{i\}\) is matched, let \(V' = V' \cup \{i\}\), else if \(\{j\}\) is matched, let \(V' = V' \cup \{j\}\). Recall that \(\alpha_1\alpha_2\ldots\alpha_m\) matches \(a^m\). Thus, \(V'\) is a vertex cover for \(G\) since it contains one vertex from each edge (from each corresponding \(\alpha_i\)).

Then \(\alpha_0\) must match the remaining \(a^k\). Therefore, there can be at most \(k\) unique backreferences defined, which means there can be at most \(k\) distinct vertices in \(V'\). Therefore, \(|V'| \leq k\) and \(G\) has a vertex cover of size \(\leq k\).
Thus $\alpha$ matches $w$ iff $G$ has a vertex cover of size $\leq k$. Furthermore, the reduction can be done in $O(|V|+|E|) = O(|V|^2)$ time. Therefore, MERE is NP-complete. \hfill $\square$

**Remark:** Notice that this proof crucially uses the (semantic) assumption that unassigned backreferences are not set to the empty string. The result can also be proved without using this ‘feature’. However, the proof is a bit more complicated and we omit it here.
5.1 Definitions

The languages defined by patterns consisting of variables and terminal letters have been considered in a number of ways. Dana Angluin [8] described the languages defined by single patterns in 1979. In 1995, Lila Kari, Alexandru Mateescu, Gheorghe Paun, and Arto Salomaa [5] defined multi-pattern languages (MPL), made up of sets of patterns.

Here we consider the extended multi-pattern languages (EMPL) introduced by Benedek Nagy in [2] as an alternate extension to classical regular expressions. To clarify the expressive power of this approach we show that the class Nagy defines is a strict subclass of the family of languages recognized by extended regular expressions. We also settle some questions that are left open in [2].

To begin we describe how patterns are constructed and interpreted and then extend that definition to EMPL as defined by Nagy [2]. Let $\Sigma$ be a finite set of terminals $\{a_1, \ldots, a_n\}$ and $V = \{x_1, x_2, \ldots\}$ be an infinite set of variables ($\Sigma \cap V = \emptyset$). Then a pattern is a non-null finite string over $\Sigma \cup V$. We use the terms erasing (E) and non-erasing (NE) pattern languages in the following sense. Let $H_{\Sigma, V}$ be the set of morphisms $h : (\Sigma \cup V)^* \rightarrow \Sigma^*$. The E pattern language generated by a pattern $\pi$ is defined as

$$L_E(\pi) = \{ w \in \Sigma^* \mid \exists h \in H_{\Sigma, V}((\forall a \in \Sigma : h(a) = a) \land w = h(\pi)) \}$$

The NE pattern language generated by a pattern $\pi$ is defined as

$$L_{NE}(\pi) = \{ w \in \Sigma^* \mid \exists h \in H_{\Sigma, V}((\forall a \in \Sigma : h(a) = a) \land$$
$$\quad (\neg \exists v \in V : h(v) = \lambda) \land$$
$$\quad w = h(\pi)) \}$$

19
Given a set of patterns \( \{ \pi_1, \pi_2, \ldots, \pi_n \} \), the \textit{E-multi-pattern language} (MPL-E) they define is \( \bigcup_{i=1}^{n} L_E(\pi_i) \). Similarly, the \textit{NE-multi-pattern language} (MPL-NE) defined by \( \{ \pi_1, \pi_2, \ldots, \pi_n \} \) is \( \bigcup_{i=1}^{n} L_{NE}(\pi_i) \).

This notion was extended by Nagy [2] to that of EMP expressions in the following way:

Let \( \{ \pi_1, \pi_2, \ldots, \pi_n \} \) be a set of patterns. Each pattern \( \pi_i \) (1 \( \leq \) i \( \leq \) n) is an EMP expression. If \( \gamma \) and \( \delta \) are EMP expressions then

\[ \gamma \lor \delta \] is also an EMP expression (using the operation union),

\[ \gamma \cdot \delta \] is also an EMP expression (using the operation concatenation),

\[ \gamma^* \] is also an EMP expression (using the operation Kleene star).

In other words, extended multi-pattern (EMP) expressions are obtained from the patterns \( \pi_1, \ldots, \pi_n \) by using finitely many regular operators. The EMP expressions which can be obtained \textit{without using union} (\( \lor \)) are called \textit{star-pattern} expressions (EMSP expressions).

The \textit{erasing extended multi-pattern language} defined by an EMP expression can be obtained from the E pattern languages in the following way:

\[ L_E(\gamma \lor \delta) = L_E(\gamma) \cup L_E(\delta) \] (using the operation union),

\[ L_E(\gamma \cdot \delta) = L_E(\gamma) \cdot L_E(\delta) \] (using the operation concatenation),

\[ L_E(\gamma^*) = (L_E(\gamma))^* \] (using the operation Kleene star).

We then define EMPL-E (Extended Multi-Pattern Languages – Erasing) to be the family of erasing extended multi-pattern languages \(^1\).

\(^1\)Note that this is not the same as the family PL(REG, REG) defined in [9]
The non-erasing extended multi-pattern language defined by an EMP expression can be obtained from the NE pattern languages in the following way:

\[ L_{NE}(\gamma \lor \delta) = L_{NE}(\gamma) \cup L_{NE}(\delta) \] (using the operation union),

\[ L_{NE}(\gamma \cdot \delta) = L_{NE}(\gamma) \cdot L_{NE}(\delta) \] (using the operation concatenation),

\[ L_{NE}(\gamma^*) = (L_{NE}(\gamma))^* \] (using the operation Kleene star).

Let EMPL-NE (Extended Multi-Pattern Languages – Non-Erasing) stand for the family of non-erasing extended multi-pattern languages.

It is not hard to see that EMPL-E (resp. EMPL-NE) is the regular closure \([10, 11]\) of the family of E pattern (resp. NE pattern) languages.

If \( \gamma \) is an EMSP expression then \( L_E(\gamma) \) is an erasing extended multi-star-pattern language. Let EMSPL-E be the family of erasing extended multi-star-pattern languages. Similarly, if \( \gamma \) is an EMSP expression then \( L_{NE}(\gamma) \) is a non-erasing extended multi-star-pattern language. Let EMSPL-NE be the family of non-erasing extended multi-star-pattern languages.

Finally, let EMPL = EMPL-E \( \cup \) EMPL-NE, and likewise, let EMSPL = EMSPL-E \( \cup \) EMSPL-NE.
5.2 Erasing and Non-Erasing Varieties of EMPL

As was introduced in the last section the language of a pattern can be interpreted as either erasing or non-erasing. Note that the pattern does not exhibit this property, but rather the language of that pattern does. A single pattern can generate two distinct languages for its E and NE varieties.

In attempting to better understand these erasing and non-erasing variants, two important questions arise:

**Question 1:** Does $\text{EMPL-}E = \text{EMPL-}NE (= \text{EMPL})$?

**Question 2:** Does $\text{EMSPL-}E = \text{EMSPL-}NE (= \text{EMSPL})$?

In this section we answer Question 1 affirmatively and Question 2 negatively.

**Lemma 5.1** The language $L = \{ xbx \mid x \in \{a, b\}^*\} \in \text{EMSPL-}E$.

**Proof:** Let $\Sigma = \{a, b\}$ and $v \in V$. Then $\alpha = v bv$ is an EMSP expression and $L_E(\alpha) = L$. 

**Lemma 5.2** The language $L = \{ xbx \mid x \in \{a, b\}^*\} \notin \text{EMSPL-}NE$.

**Proof:** (by contradiction). Assume $L$ is an NE star-pattern language. Then there is an NE star-pattern expression $\alpha$ such that $L_{NE}(\alpha) = L$. $\alpha$ cannot contain the union operator since it is an NE star-pattern expression.

Clearly $\alpha$ cannot have a star as its outer-most operator. Otherwise, $\alpha$ would match $\lambda$, which is not in $L$. Define a language to be non-trivial if and only if it is neither empty nor the singleton set $\{\lambda\}$. 
Claim: $L$ is not the concatenation of two non-trivial languages.

Proof: Assume the contrary and let $L = A \circ B$ with $A$ and $B$ non-trivial. Without loss of generality assume that $b \in B$. Hence every non-empty string in $A$ must be of the form $ubbu$ for some $u \in \{a, b\}^*$. Now consider the string $aba$ which belongs to $L$. $aba$ has to be in $B$ since no non-empty prefix of it can be in $A$. But then the word equation $ubbuaba = xbx$ has no solution.

Since $\alpha$ cannot be a single pattern either, the result follows.

Lemma 5.3 $EMSPLE \neq EMSPLEN$.

Proof: The language $L$ of Lemma 5.1 (and 5.2) is in EMSPL-E, but is not in EMSPL-NE.

Lemma 5.4 $EMPLNE = EMPL$.

Proof: This follows from the results of [5]. We omit the proof.

Lemma 5.5 Every EMPL language is semilinear.

Proof: Every language in MPL is semilinear [5]. The family of semilinear languages is closed under union, concatenation and star.

\footnote{Thus without backreferences the analogous result to Theorem 17 of [2] does not hold.}

\footnote{A language $L$ is semilinear iff its Parikh mapping $\psi(L)$ is a semilinear set of n-tuples over the natural numbers, where $\psi(w) = (#_{a_1}(w), \ldots, #_{a_n}(w))$, each $#_{a_i}(w)$ denoting the number of occurrences of $a_i$ in the word $w$ [12].}
Note that not all EREG languages are semilinear. Consider the language \{ a^i \mid i > 0 \text{ and } i \text{ is not a power of 2} \} defined by the eregex \(\text{aaa(aa)*}\)\(^1\). It is known that any semilinear language over a unary alphabet is regular. This is a well-known non-regular language. Therefore, it cannot be semilinear. This itself shows that EMPL \(\neq\) EREG. In the next section we show that EMPL is actually a proper subclass of EREG.
5.3 Comparing EMPL, EREG and CFL

To fully understand the capabilities of EMPL we would like to know its class relationship to REG, EREG, and CFL. To see that EMPL is a strict superclass of the regular languages is trivial, since an EMP expression without variables is a regular expression, and even a simple pattern like $xx$ generates a non-regular language.

In relating EMPL to EREG we exploit a weakness in EMPL that in defining patterns we have no control over what a variable matches. That is, any variable can match anything from $\Sigma^*$, so any pattern containing a variable is subject to containing unlimited numbers of characters from the alphabet. We will also use this feature to relate EMPL to CFL.

Lemma 5.6 $EMPL \subseteq EREG$.

**Proof:** Given an EMP expression $\gamma$ over terminal alphabet $\Sigma = \{a_1, \ldots, a_n\}$, we can construct an equivalent eregex $\alpha$. Let $\{\pi_1, \pi_2, \ldots, \pi_n\}$ be the set of patterns that occur within $\gamma$.

For each pattern $\pi_i$ $(1 \leq i \leq n)$, replace the first occurrence of a variable within $\pi_i$ with $((a_1 | a_2 | \ldots | a_n)^*)$, where $k$ is the index of the outer left parenthesis we are adding. That is, there are $k - 1$ left parentheses in $\gamma$ to the left of the first occurrence of this variable in pattern $\pi_i$. Replace each subsequent occurrence of the same variable in $\pi_i$ with $\backslash k$.

Lemma 5.7 The EREG language $R = \mathcal{L}((a^*)b(\backslash 1)) = \{a^i ba^i \mid i \geq 0\}$ is not in $EMPL$.

**Proof:** (by contradiction) Assume that $R \in EMPL$. Then there is an EMP expression $\gamma$ that defines $R$. Clearly $\gamma$ must contain at least one variable. If not, then $\mathcal{L}(\gamma)$ would be a regular language and $R$ is a well-known non-regular language. Let $x$ be
a variable occurring in $\gamma$. Consider the morphism $h \in H_{\Sigma V}$ with $h(x) = bb$. Thus $L(\gamma)$ will contain strings of the form $\Sigma^*bb\Sigma^*$ which are not in $R$. Therefore, no such $\gamma$ can exist and $R \not\in \text{EMPL}$. \hfill $\square$

**Theorem 5.1** EMPL $\subset$ EREG.

**Proof:** This result follows directly from the previous two lemmas. \hfill $\square$

**Theorem 5.2** EMPL is incomparable with CFL.

**Proof:** The family of languages defined by single patterns is incomparable with context-free languages [8]. In particular, the EMP expression $xx$ defines the language $\{ww \mid w \in \Sigma^*\}$ which is a well known non-context-free language (for $|\Sigma| > 1$). The CFG $\{ S \to aSa \mid b \}$ generates the language $R$ of Lemma 5.7 which is not in EMPL. \hfill $\square$

We also consider the question of whether the non-regular subsets of EMPL and CFL are disjoint. In other words, is REG = EMPL $\cap$ CFL? As it turns out, these subsets are not disjoint, and REG $\subset$ EMPL $\cap$ CFL.

**Lemma 5.8** The language

$$S = \{ xcy \mid x, y \in \{a, b\}^* \land x \neq y \} \cup \{ xycz \mid x, y, z \in \{a, b, c\}^* \}$$

is in EMPL.
Proof: Let $\alpha$ be an EMP expression over $\Sigma = \{a, b, c\}$ and $V = \{x, y, z\}$, defined as follows:

$$\alpha = xaycxbz \cup xycaxz \cup xayx \cup xyca \cup xcay \cup xcby \cup xcycz$$

Then we have $L_E(\alpha) = S$. \hfill $\Box$

**Lemma 5.9** The language

$$S = \{xy \mid x, y \in \{a, b\}^* \land x \neq y\} \cup \{xcyz \mid x, y, z \in \{a, b, c\}^*\}$$

is in CFL.

Proof: Let $G = (\{a, b, c\}, \{S, X, X_1, Y, Z, Z_1\}, P, S)$ be a context-free grammar with productions $P$:

- $S \rightarrow X \mid Y$
- $X \rightarrow X_1bZ \mid X_2aZ$
- $X_1 \rightarrow Z_1X_1Z_1 \mid aZc$
- $X_2 \rightarrow Z_1X_2Z_1 \mid bZc$
- $Y \rightarrow Z_1YZ_1 \mid cZ_1Z \mid Z_1Zc$
- $Z \rightarrow aZ \mid bZ \mid \epsilon$
- $Z_1 \rightarrow a \mid b$

Then we have $L(G) = \{xy \mid x, y \in \{a, b\}^* \land x \neq y\}$\textsuperscript{4}. Furthermore, $L((a|b|c)^*(a|b|c)^*(a|b|c)) = \{xcyz \mid x, y, z \in \{a, b, c\}^*\}$. The union of a CFL and a regular language is context-free. Therefore, $S$ is in CFL. \hfill $\Box$

\textsuperscript{4}This is given as Exercise 2.22 of [13]
Theorem 5.3 \( REG \subset EMPL \cap CFL \)

**Proof:** \( S \) is a well-known non-regular language. \( S \in EMPL \cap CFL \) follows from the previous two lemmas. \( \square \)

Corollary 5.1 \( REG \subset EREG \cap CFL \)

**Proof:** This follows directly from the previous theorem and the fact that \( EMPL \subset EREG \). \( \square \)
5.4 The Equivalence Problem for EMSPL

Lemma 5.10 The following problem is undecidable:

Instance: Two sets of patterns $P_1$ and $P_2$.
Question: Is $(L_E(P_1))^* \subseteq (L_E(P_2))^*$?

Proof: The problem of deciding, given two patterns $\alpha$ and $\beta$, whether $L_E(\alpha) \subseteq L_E(\beta)$ is known to be undecidable [14]. Let $#$ be a new symbol, not present in the alphabet $\Sigma$ of $\alpha$ and $\beta$. Let $\Omega = \Sigma \cup \{\#\}$. Now form the sets of patterns

$$
\Gamma = \{\#\alpha\#\} \quad \text{and} \quad \Delta = \{\#\beta\#, \#x_1\#x_2\#\}.
$$

Claim 1: $(L_E(\Gamma))^* \subseteq (L_E(\Delta))^*$ over $\Omega$ if and only if $L_E(\alpha) \subseteq L_E(\beta)$ over $\Sigma$.

Proof of Claim 1: Assume that $(L_E(\Gamma))^* \subseteq (L_E(\Delta))^*$ over $\Omega$. Then given any string $w \in L_E(\alpha)$, where $w \in \Sigma^*$, we must show that $w \in L_E(\beta)$. If $w \in L_E(\alpha)$ then $w#$ is in $L_E(\Gamma)$ and therefore in $(L_E(\Gamma))^*$. By our assumption $w#$ is also in $(L_E(\Delta))^*$. Now $w#$ must also be in $L_E(\Delta)$, since $w$ does not contain $. Furthermore, it must match $\#\beta#$ and so $w$ matches $\beta$. Thus $w \in L_E(\beta)$.

Now assume conversely that $L_E(\alpha) \subseteq L_E(\beta)$ over $\Sigma$. Then given any string $w \in (L_E(\Gamma))^*$ we must show that $w \in (L_E(\Delta))^*$. Let $w = w_1w_2\ldots w_k$ for $k \geq 0$, where each $w_i \in L_E(\Gamma)$, for $1 \leq i \leq k$. Thus each $w_i$ is of the form $\#w_i^\prime\#$ where $w_i^\prime \in L_E(\alpha)$ over $\Omega$. If $w_i^\prime$ contains a $#$, then $w_i$ matches $\#x_1\#x_2\#$ and is therefore in $L_E(\Delta)$. Otherwise, $w_i^\prime$ does not contain a $#$ and is in $L_E(\alpha)$ over $\Sigma$. By our assumption it is also in $L_E(\beta)$ over $\Sigma$ and therefore $w$ matches $\#\beta#$ and is in $L_E(\Delta)$.

Claim 2: $(L_E(\Gamma))^* \subseteq (L_E(\Delta))^*$ if and only if $(L_E(\Gamma \cup \Delta))^* = (L_E(\Delta))^*$.

Proof of Claim 2: Assume that $(L_E(\Gamma))^* \subseteq (L_E(\Delta))^*$ over $\Omega$. It is trivial to show that $w \in (L_E(\Delta))^*$ implies $w \in (L_E(\Gamma \cup \Delta))^*$. Given any string $w \in (L_E(\Gamma \cup \Delta))^*$ we must show that $w \in (L_E(\Delta))^*$. Let $w = w_1w_2\ldots w_k$ for $k \geq 0$, where each
\( w_i \in L_E(\Gamma \cup \Delta) \), for \( 1 \leq i \leq k \). Then each \( w_i \) is of the form \( \#w_i'\# \) and either \( w_i \in L_E(\Gamma) \) or \( w_i \in L_E(\Delta) \). If \( w_i \in L_E(\Gamma) \) then \( w_i \in (L_E(\Gamma))^* \). By our assumption \( w_i \) is also in \( (L_E(\Delta))^* \). Then there is a decomposition \( w_i = w_{i_1}w_{i_2} \ldots w_{i_k} \) where each \( w_{i_j} \in L_E(\Delta) \). Therefore, \( w \) is of the form \( w_1w_2 \ldots w_{i_1}w_{i_2} \ldots w_{i_k} \ldots w_k \), which is the concatenation of strings in \( L_E(\Delta) \). Thus \( w \in (L_E(\Delta))^* \).

Now assume conversely that \( (L_E(\Gamma \cup \Delta))^* = (L_E(\Delta))^* \). Given any string \( w \in (L_E(\Gamma))^* \), clearly \( w \in (L_E(\Gamma \cup \Delta))^* \), and by our assumption \( w \in (L_E(\Delta))^* \).

\[ \square \]

**Theorem 5.4** The equivalence problem for EMSPL-E is undecidable.

The same technique will work for EMSPL-NE, except that \( \Delta \) will have to be defined a little differently, as \( \{\# \beta \#, \#x_1\#x_2\#, \#\#x_2\#, \#\#x_1\#, \#\#\} \). It can also be shown that

**Lemma 5.11** For every EMP expression of the form \( \alpha^* \) (i.e., with star as the outermost operator), there is an EMSP expression \( \gamma \) such that \( L_{NE}(\gamma) = L_E(\alpha^*) \).

**Proof:** Since the families EMPL-E and EMPL-NE are the same, there must be an EMP expression \( \beta \) such that \( L_{NE}(\beta) = L_E(\alpha) \). It is known that the expressions \( (x \lor y)^* \) and \( (x^*y^*)^* \) are equivalent. Therefore, we can generate an EMSP expression \( \gamma \) by recursively replacing each occurrence of \( \lor \) in \( \beta^* \) with the equivalent union-free subexpression. The resulting expression \( \gamma \) does not contain any union operators and \( L_{NE}(\gamma) = L_{NE}(\beta^*) = L_E(\alpha^*) \).

\[ \square \]

**Theorem 5.5** The equivalence problem for EMSPL-NE is undecidable.

This settles an open problem given in [2].

---

30
CHAPTER 6
Remarks on Synchronized Regular Expressions

6.1 Introduction and Basic Definitions

As a third variant of extension of classical regular expressions we consider synchronized regular expressions (SRE) as defined by G. Della Penna, B. Intrigila, E. Tronci and M. Venturini Zilli in [3]. The core functionality of SRE is the same as extended regular expressions although SRE uses a different syntax, e.g., + instead of | and variable binding/referencing instead of numbered parenthesis backreferences.

In addition, synchronized regular expressions extend this by allowing exponent variables which range over the natural numbers. These exponent variables allow SRE to define a class of languages not in EREG, such as \{a^n b^n \mid n \geq 0\}.

Definition 6.1 The Synchronized Regular Expressions on an alphabet A, a set of variables V, and a set of exponents X are defined as follows:

- $\emptyset \in \text{SRE}$ (empty language)
- $\epsilon \in \text{SRE}$ (empty string)
- $\forall a \in A: a \in \text{SRE}$ (letters)
- $\forall v \in V: v \in \text{SRE}$ (variables)

If $e_1, e_2 \in \text{SRE}$ then:

1. $e_1^* \in \text{SRE}$ (star)
2. $\forall x \in X: e_1^x \in \text{SRE}$ (exponentiation)
3. $\forall v \in V: e_1 \% v \in \text{SRE}$ (variable binding)
4. $e_1e_2 \in SRE$ (concatenation)

5. $e_1 + e_2 \in SRE$ (union)

**Definition 6.2** Beyond these basic syntactic definitions, a synchronized regular expression must meet the following conditions to be considered valid. We call this the SRE validity test.

1. *Each variable occurs in a binding operation no more than once in the expression.*

2. *Each occurrence of a variable in the expression is preceded by a binding of that variable somewhere to the left of the occurrence in the expression.*

Unless otherwise specified, any mention of SRE will refer to valid SRE. SRE that do not pass the validity test are considered in Section 6.7.
6.2 Semantics of SRE-matching

This section provides a more concise and complete definition of the semantics of SRE matching. The definition of SRE semantics given in [3] include an ambiguous definition of union and contains no definition for the star operator. A precise definition for the star operator is important, since the star operator behaves differently than unsynchronized exponents, as is shown in Section 6.3. This definition corrects those issues, as well as removing the set of bound free variables $V_F$ that is unnecessary if the semantics are to match only valid SRE.

The function $Eval$ encodes the semantics of SRE using induction on the structure of expressions.

Let $Eval(PSL, V_B, E_B)$ be our evaluation function, where

- $PSL$ is an ordered list of (pattern, string) pairs;
- $V_B$ is a set of (variable, string) pairs, representing all the variables already associated to a string by a binding operation;
- $E_B$ is a set of (exponent, number) pairs, representing all the exponents already assigned.

The function is true iff, for every (pattern, string) pair in $PSL$, pattern matches string using the binding rules given in $V_B$ and $E_B$. 
The rules for $Eval$ are the following:

1. $Eval((a, \alpha) :: PSL, V_B, E_B) = Eval(PSL, V_B, E_B) \land (\alpha = a)$

2. $Eval((\varepsilon_1 \varepsilon_2, \alpha) :: PSL, V_B, E_B) = \\
\lor_{\{\alpha_1, \alpha_2 = \alpha\}} Eval((\varepsilon_1, \alpha_1) :: ((\varepsilon_2, \alpha_2) :: PSL), V_B, E_B)$

3. $Eval((\varepsilon_1 + \varepsilon_2, \alpha) :: PSL, V_B, E_B) = \\
Eval((\varepsilon_1, \alpha) :: PSL, V_B, E_B) \lor Eval((\varepsilon_2, \alpha) :: PSL, V_B, E_B)$

4. $Eval((\varepsilon^*, \alpha) :: PSL, V_B, E_B) = \\
\begin{cases} 
Eval(PSL, V_B, E_B) & \text{if } (\alpha = \varepsilon) \\
\lor_{k \in \mathbb{N}} Eval((\varepsilon^k, \alpha) :: PSL, V_B, E_B) & \text{otherwise}^1
\end{cases}$

5. $Eval(((\varepsilon)^{\%v}, \alpha) :: PSL, V_B, E_B) = \\
Eval((\varepsilon, \alpha) :: PSL, (V_B \setminus \{(v, \alpha') \mid \alpha' \in A^*\}) \cup \{(v, \alpha)\}, E_B)$

6. $Eval((v, \alpha) :: PSL, V_B, E_B) = \\
Eval(PSL, V_B, E_B) \land \exists (v, \alpha) \in V_B$

7. $Eval((\varepsilon^x, \alpha) :: PSL, V_B, E_B) = \\
\begin{cases} 
Eval((\varepsilon^x, \alpha) :: PSL, V_B, E_B) & \text{if } (x, n) \in E_B \\
\lor_{n \in \mathbb{N}} Eval((\varepsilon^n, \alpha) :: PSL, V_B, E_B \cup \{(x, n)\}) & \text{otherwise}
\end{cases}$

8. $Eval(((), V_B, E_B) = true$

---

1Note that the value of $k$ here is ephemeral and is not added to $E_B$. Thus, the star operator behaves differently than exponents, even in linear SRE.
6.3 Linear SRE

Definition: A Linear SRE (LSRE) is an SRE in which each exponent occurs at most once.

Given that exponents occur only once in linear SRE it might be expected that these exponents behave in the same manner as the star operator and, as such, linear synchronized regular expressions define the same class of languages as extended regular expressions. In this chapter we will show that this is not the case. In fact if an exponent occurs nested inside a star operator or another exponent it has an inherent synchronization when the nested sub-expression is repeated.

Lemma 6.1 The language

\[ Q = \{ w_1cw_2c \ldots w_nc \mid w_1 \ldots w_n \in \{a, b\}^*, |w_1| = |w_2| = \ldots = |w_n|, \ n \geq 0 \} \]

is a LSRE language.

Proof: The LSRE \((a+b)^n c^*\) defines the language \(Q\). \qed

Lemma 6.2 \(LSRE \not\subseteq EREG\).

Proof: The language \(Q\) of Lemma 6.1 (and 3.7) is in LSRE, but is not in EREG. \qed
Lemma 6.3 \( EREG \subseteq LSRE. \)

**Proof:** Given an eregex, an equivalent LSRE can be constructed by the following:

1. Replace each right parenthesis \( \) with \( \text{)^\%v_i} \) where \( i \) is the number (in a left to right ordering) of the right parenthesis.

2. Replace each backreference \( \text{\textbackslash {}i} \) with \( v_i \).

The resulting expression is a valid LSRE matching the same language. \( \square \)

Theorem 6.1 \( EREG \subset LSRE. \)

**Proof:** This result follows directly from the previous two lemmas. \( \square \)
6.4 Exponent-Free SRE

Definition: An Exponent-Free SRE (EFSRE) is an SRE that does not contain any exponents.

The relationship between exponent-free synchronized regular expressions and extended regular expressions seems obvious since they provide the same features. However, because of the different semantics between the two approaches and the surprising result regarding linear SRE it is worth clarifying this point.

Theorem 6.2 \( \text{EFSRE} = \text{EREG} \).

Proof: Because of the syntactic restrictions on SRE enforced by the validity test all backreferences in SRE follow the same restrictions inherent in eregex.

Given an eregex, an equivalent EFSRE can be constructed by the same method used in the proof of Lemma 6.3.

Given an EFSRE, an equivalent eregex can be constructed by the following, where \( V_I \) is a set of (variable, integer) pairs. To begin, let \( V_I = \{\} \).

1. Replace each variable binding operation \( \%v \) with \( ) \). Let \( V_I = V_I \cup \{(v, i)\} \) where \( i \) is the number (in a left to right ordering) of the right parenthesis.

2. Replace each variable occurrence \( v \) with \( \backslash i \) where \( (v, i) \in V_I \).

The resulting expression is an eregex matching the same language.

\[ \square \]
6.5 On the Language of Palindromes

In order to provide a simpler proof that SRE is incomparable with CFL we consider the following language of palindromes over the alphabet \( \{a, b, c\} \).

\[
P = \{ wcw^R \mid w \in \{a, b\}^* \}
\]

**Lemma 6.4** \( P \not\in SRE \).

**Proof:** Suppose to the contrary that \( P \) is an SRE language. Then there is a synchronized regular expression that defines this language. We consider strings of the form \( ucu^R \) where \( u \) is a cube-free string such that the largest common substring between \( u \) and \( u^R \) is of length at most 7. (A construction of such strings is given in the appendix.) For the remainder of the proof the term cube-free shall refer to strings of this form. Clearly, if \( u \) is cube-free then \( u^R \) is also cube-free.

First off, we see that for any palindrome in the language, no subexpression matching a substring of its right half, i.e., the string to the right of the \( c \), can contain a \( * \). If it does, then adding repetitions to the \( * \) would not affect the variables or exponents matching the left half. Thus, the expression would match an infinite number of distinct strings on the right half to a single string on the left half, which would clearly match non-palindromes.

Now consider then the rightmost-innermost subexpression that matches infinitely many cube-free strings. This subexpression must be of the form \( e^y \) since it matches infinitely many strings and cannot contain a \( * \). We know that \( e \) can only match finitely many cube-free strings since \( e^y \) is the innermost expression to match infinitely many cube-free strings. Furthermore, no proper subexpression of \( e^y \) used in matching cube-free words can contain the exponent \( y \). Otherwise it would contradict our hypothesis about \( e^y \).

Clearly for any palindrome in the language \( e^y \) will match a substring of its right half. Let \( V \) be the set of variables in \( e \) whose values get defined in the left half. Since \( e \) matches only finitely many cube-free strings, the values of these variables that
contribute to cube-free strings are from a finite set, and we can effectively eliminate these variables from consideration by suitable instantiation\(^2\). Similarly, we can also eliminate exponent variables that do not take infinitely many values while matching these cube-free strings. Now, note that \(e^y\) must match a single cube-free string. Otherwise it could match multiple strings without affecting the left half which would clearly lead to non-palindromes.

Since \(e\) can only match finitely many cube-free strings, every cube-free string matched by \(e^y\) must be from \((w_1+\ldots+w_n)^*\) where \(n > 1\) and \(w_1, \ldots, w_n\) are cube-free strings. Let \(\alpha = \alpha_1\alpha_2 \ldots \alpha_n\) be a cube-free string matched by \(e^y\) for the exponent value \(y = n\), where each \(\alpha_i\) is matched by \(e\) for \(1 \leq i \leq n\). Now \(e^{n+1}\) matches

\[
\begin{align*}
\alpha' &= \alpha_1\alpha_1\alpha_2\alpha_3 \ldots \alpha_n \\
\alpha'' &= \alpha_1\alpha_2\alpha_2\alpha_3 \ldots \alpha_n \\
\alpha''' &= \alpha_1\alpha_2\alpha_3\alpha_3 \ldots \alpha_n
\end{align*}
\]

Now, \(\alpha'\) and \(\alpha''\) are the same only if \(\alpha_1\) and \(\alpha_2\) are the same. In addition, \(\alpha''\) and \(\alpha'''\) are the same if and only if \(\alpha_2\) and \(\alpha_3\) are the same. This clearly contradicts the assumption that \(\alpha\) is cube-free.

Therefore, a synchronized regular expression that defines the language \(\mathcal{P}\) of palindromes does not exist. \(\square\)

It is worth noting that the complement of \(\{ww^R \mid w \in \{a,b\}^*\}\) is indeed an SRE language: the expression

\[(a+b)^x a(a+b)^* b(a+b)^x + (a+b)^x b(a+b)^* a(a+b)^x\]

will only match non-palindromes.

\(^2\)Variables that take values from a finite set can be instantiated by creating a new expression that is the finite union of the original expression with the variable replaced by each of its possible values.
In addition, the expression

\[(a + b)^x(a + b)(a + b)^x c(a + b)^x + \]
\[(a + b)^x c(a + b)^x (a + b)(a + b)^x + \]
\[(a + b)^x a(a + b)^y c(a + b)^y b(a + b)^x + \]
\[(a + b)^x b(a + b)^y c(a + b)^y a(a + b)^x \]

will match non-palindromes of the form \( \{ wcw^R \mid w \in \{ a, b \}^* \} \), which is the complement of \( \mathcal{P} \). This shows that SRE-languages are not closed under complement.

**Theorem 6.3** SRE is incomparable with CFL.

**Proof:** This follows from the result of the previous lemma and the fact that SRE is a proper superclass of EREG, and EREG is incomparable with CFL. \( \square \)
6.6 NP-completeness of SRE-matching without Backreferences

We show in this section that SRE-matching without backreferences is NP-complete even when the alphabet is unary. This improves on a result of [3] where the same complexity result is obtained for an alphabet with 2 letters.

We show this by a sequence of reductions from 1-in-3 3SAT. Our starting point is the following problem (call it \textbf{P1}) which can be shown to be NP-complete by an easy reduction from 3SAT.

\textbf{Input:} A set of variables \{x_1, \ldots, x_n\} and a set of equations \(E\) of the form \(x_i x_j x_k = ^? 2\).

\textbf{Question:} Does \(E\) have a solution over the natural numbers \(\{0, 1, 2, \ldots\}\)?

This can further be reduced to the following problem \textbf{P2}:

\textbf{Input:} A set of variables \{x_1, \ldots, x_n\}, a polynomial \(p \in \mathbb{Z}[x_1, \ldots, x_n]\) with coefficients from \(\{0, 1\}\) and a natural number \(m\), with all numbers expressed in unary.

\textbf{Question:} Does \(p = ^? m\) have a solution over the natural numbers \(\{0, 1, 2, \ldots\}\)?

The proof of this reduction is as follows: let \(E\) be an instance of \textbf{P1} and \(k\) be the number of equations in \(E\). Let \(d\) be the smallest natural number such that \(k \cdot 2^d < 3^d\). The number \(l = \lceil \log_2 k^2 \rceil = \lceil 2 \log_2 k \rceil\) is an upper bound for this \(d\) since \(k \cdot 2^l = k^3 < 3^l = (k^2)^{\log_2 3}\) and \(\log_2 3\) is greater than 1.5. Now form the polynomial \(p = \sum (x_i x_j x_k)^d\) summing up the \(d\)th powers of the left-hand sides in \(E\) and let \(m = k \cdot 2^d\). The equation \(p = ^? m\) has a solution over the natural numbers if and only if \(E\) has a solution over the natural numbers.
Reducing this problem to our SRE-matching problem is not difficult. The polynomial $p$ in any instance of $P2$ is merely a sum of monomials. Let \( \{a\} \) be a unary alphabet. Now for any monomial $m$ over a set of variables \( \{x_1, \ldots, x_n\} \), we can effectively form $a^m$ by a tower of exponents: for instance, if the monomial is $x_1x_2^2x_3$ then form

\[
((a^{x_1})^{x_2})^{x_3}
\]

Thus there is a match for this SRE iff there is a solution for $p =^? m$. \qed
6.7 Unrestricted SRE

**Definition:** Unrestricted SRE (USRE) are expressions defined by the SRE syntax rules but without regard for the validity test. That is, they may or may not be valid. The semantic definition of SRE given in [3], as well as the one given here, allows for the matching of patterns that would have otherwise been precluded by the validity check. Let UEFSRE denote unrestricted exponent-free synchronized regular expressions.

**Lemma 6.5** The language

\[ S = \{ a^{n^2} \mid n \geq 1 \} \]

is an UEFSRE language.

**Proof:** \( S \) is defined by the following UEFSRE: \((a)\%v((vaa)\%v)^*\).

**Lemma 6.6** The language

\[ S = \{ a^{n^2} \mid n \geq 1 \} \]

is not an EREG language.

**Proof:** Assume \( S \) is expressed by an earegex and let \( N \) be the constant given by the CSY pumping lemma. Consider \( w = a^{N^2} \). Then there is a decomposition \( w = x_0y_1x_2y_3 \ldots x_my_m \) for some \( m \geq 1 \) and from the pumping lemma \( y = a^p \) for some \( p \geq 1 \). Let \( k = pm \). Consider \( w' = x_0y^{k+1}x_1y^{k+1} \ldots y^{k+1}x_m = a^{N^2+k^2} \).

By the CSY pumping lemma \( w' \in S \), so \( N^2+k^2 = r^2 \) for some \( r \) and since \( N \geq 1 \) clearly \( r > k \). Now consider \( w'' = x_0y^{k+2}x_1y^{k+2} \ldots y^{k+2}x_m = a^{N^2+k^2+k} \); with \( |w''| = N^2+k^2+k = r^2+k \). Since \( |w'| = r^2 \), the smallest perfect square greater than \( r^2 \) is \((r+1)^2 = r^2 + 2r + 1\). Now \( |w''| > r^2 \) since \( k \geq 1 \) and \( |w''| < (r+1)^2 \) since \( k < 2r+1 \). Consequently, \( |w''| \) is not a perfect square and \( w'' \notin S \).
Theorem 6.4 \( EREG \subset UEFSRE \).

Proof: Clearly \( EREG \subseteq UEFSRE \), since \( EREG \subseteq EFSRE \). Since the language \( S \) of Lemma 6.5 (and 6.6) is in \( UEFSRE \), but is not in \( EREG \), it follows that \( UEFSRE \nsubseteq EREG \). \( \square \)
CHAPTER 7

Conclusion

As a combination of the work of Câmpeanu, Salomaa and Yu [1] and the results given here, we now have a much more robust understanding of the expressive power of extended regular expressions. In this chapter we summarize the main properties of the EREG languages and relate the three classes of languages discussed in this thesis to other major language classes and to each other.

The following closure properties of EREG languages are known:

EREG languages are closed under the following operations:

- Kleene star
- image under a homomorphism [1]
- concatenation
- union
- intersection with a regular language [15]

EREG languages are not closed under the following operations:

- complement [1]
- image under an inverse homomorphism [1]
- intersection [16, 17]
- difference
Considering the pumping lemmas for EREG languages, it is far from clear whether the two pumping lemmas are sufficient to prove that all non-ERE languages are not in EREG. Specifically, is $L-\text{CSY} \cap L-\text{CNS} = \text{EREG}$? We suspect that this is not the case and that there are non-ERE languages that satisfy both pumping lemmas. One such language might be $\{a^i b a^j \mid i^2 > j\}$. We are currently working on this problem.

The following figure describes the set relationship between the three classes of languages we have discussed and other major language classes.

\[ \begin{array}{c}
\subseteq \\
\subseteq \\
\subseteq \\
\subseteq \\
\subseteq \\
\subseteq \\
\end{array} \]

\textbf{Figure 7.1: Relationship between language classes.}

Furthermore, all three of EMPL, EREG, and SRE are incomparable with CFL. It remains an open problem whether there exists a language $\mathcal{L}$ such that $\mathcal{L} \in \text{EMPL}$ and $\mathcal{L} \not\in \text{REG}$.

As understanding of more complex and powerful pattern matching systems increases it is likely that significant application areas for these expressive powers will develop as well. One such area may be in computational biology problems in the field of bioinformatics. Many problems in this field deal with large volumes of data including amino acid sequences for proteins and nucleotide base sequences for DNA. The ability to perform sophisticated interpretations and classifications of these data could prove to be a viable application area of current and future pattern matching systems.

Another potential application area of extended pattern matching might be in plagiarism detection. All of the pattern matching models studied here share the common theme of containing a type of memory or synchronization enabling them to relate one part of a string to another. This characteristic might enable enhanced pattern matching systems to aid in detecting plagiarism.
We have discussed many pattern models in this thesis, but have not considered the different formats of the data we might need to work with. One potential area for future work in extended patterns is in pattern matching over compressed strings. Many data, especially large data sets, are stored in compressed format. One such compression format is a straight line program (SLP) [18], which stores a string as a restricted context-free grammar. Developing an approach to extended regular expression pattern matching over SLP would provide for the application of advanced pattern matching to large sets of compressed data.
We can generate infinitely many cube-free strings using the Prouhet-Thue-Morse sequence [19]. This sequence can be defined over \( \Sigma = \{0, 1\} \) as follows:

Let

\[
\begin{align*}
u_0 &= 0 \\
v_0 &= 1
\end{align*}
\]

Then for all \( i > 0 \), let

\[
\begin{align*}
u_{i+1} &= u_i v_i \\
v_{i+1} &= v_i u_i
\end{align*}
\]

This sequence generates the following values for \( u \) and \( v \):

\[
\begin{align*}u : \{0, 01, 0110, 01101001, 0110100110010110, \ldots \} \\
v : \{1, 10, 1001, 10010110, 1001011001101001, \ldots \}
\end{align*}
\]

Note that \( u_i = g(v_i) \) and \( v_i = g(u_i) \) for homomorphism \( g : g(0) = 1, g(1) = 0 \).

Now consider the following homomorphism:

\[
\begin{align*}h(0) &= aab \\
h(1) &= abb
\end{align*}
\]

Let \( u \) be a cube-free string over \( \Sigma = \{0, 1\} \) generated by the Prouhet-Thue-Morse sequence. Then it is not hard to show that \( h(u) \) is a cube-free string over \( \Sigma = \{a, b\} \), with the additional property that the largest common substring between \( u \) and \( u^R \) is of length at most 7.
Glossary

**EMPL** The family of erasing and non-erasing extended multi-pattern languages. 2, 19, 21

**EMPL-E** The family of erasing extended multi-pattern languages. 20

**EMPL-NE** The family of non-erasing extended multi-pattern languages. 21

**EMSPL** The family of erasing and non-erasing extended multi-star-pattern languages. 21

**EMSPL-E** The family of erasing extended multi-star-pattern languages. 21

**EMSPL-NE** The family of non-erasing extended multi-star-pattern languages. 21

**EREGR** The family of languages defined by extended regular expressions. 6

**eregex** An extended regular expression, i.e. a regular expression with the addition of backreferences. 4

**MPL** The family of multi-pattern languages. 19

**REG** The regular languages, i.e. the family of languages defined by regular expressions. 6

**SRE** The family of languages defined by synchronized regular expressions. 31
Acronyms

CNS Carle, Narendran and Scheriff [16, 17]. 10

CSY Cămpeanu, Salomaa and Yu [1]. 8

EFSRE Exponent-Free Synchronized Regular Expression. 37

EIERE Emptiness of Intersection of Extended Regular Expressions. 14

LSRE Linear Synchronized Regular Expression. 35

MERE Matching Problem for Extended Regular Expressions. 16

MPSG Membership Problem for Phrase-Structure Grammars. 14

SLP Straight Line Program. 47

SRE Synchronized Regular Expression. 3, 31

UEFSRE Unrestricted Exponent-Free Synchronized Regular Expression. 43

USRE Unrestricted Synchronized Regular Expression. 43
BIBLIOGRAPHY


