Universidad Autónoma de Madrid
Escuela politécnica superior

Double Degreee in Computer Engineering and Mathematics

B. Sc. Thesis

MATCHING REGULAR EXPRESSIONS ON UNCERTAIN DATA

Author: Arturo Gil Carvajal
Advisor: Simone Santini

June 2018

Universidad Autónoma de Madrid
Escuela politécnica superior

Doble Grado en Ingeniería Informática y Matemáticas

Trabajo Fin de Grado

BÚSQUEDA DE PATRONES SOBRE DATOS CON INCERTIDUMBRE

Autor: Arturo Gil Carvajal
Tutor: Simone Santini

Junio 2018

MATCHING REGULAR EXPRESSIONS ON UNCERTAIN DATA

Author: Arturo Gil Carvajal
Advisor: Simone Santini

Department of Comp. Engineering
Escuela Politécnica Superior
Universidad Autónoma de Madrid
June 2018

Resumen

Resumen

El presente trabajo examina una extensión de expresiones regulares sobre cadenas con incertidumbre. En el modelo estándar de expresiones regulares, los datos recibidos son una secuencia de símbolos bien posicionados y bien definidos, pertenecientes a un alfabeto Σ, mientras que la salida es un indicador que nos dice si la cadena cumple, o no, el patrón de la expresión, es decir, si la cadena pertenece al lenguaje generado por la expresión. En este trabajo, asumiremos la incertidumbre de los datos recibidos por la expresión regular.

Propondremos un modelo con incertidumbre, basado en la recepción de símbolos corruptos por el ruido del canal por donde son transmitidos. Como consecuencia de este ruido en el canal, durante el proceso de reconocimiento de símbolos, nosotros no veremos simplemente un símbolo, si no que obtendremos una distribución de probabilidad $\nu : \Sigma \to [0, 1]$, donde $\nu(a^{(n)})$ es la probabilidad de que el símbolo que estemos observando sea $a^{(n)}$. Después, estos símbolos $a^{(n)}$ serán enviados al algoritmo de comprobación de la expresión regular, para operar sobre ellos.

En este trabajo, desarrollaremos algoritmos que puedan encontrar la cadena mas probable que cumpla el patrón de la expresión regular ϕ, basándonos en las observaciones ν. Ademas, extenderemos este método para que sea capaz de considerar *flujos de datos infinitos con incertidumbre*, y estudiaremos sus propiedades. Después, expondremos y exploraremos ejemplos prácticos concretos, que puedan ser extrapolados a escenarios similares, donde aplicaremos nuestro modelo y demostraremos que nuestros algoritmos aseguran la decibilidad con probabilidad 1, bajo ciertas circunstancias, pueden calcular justificadamente la probabilidad de cumplimiento de un patrón en una cadena de símbolos, y son capaces de recobrarse de errores de lectura.

Palabras Clave

stream infinita, datos con incertidumbre, expresiones regulares

Abstract

The present work examines an extension of regular expressions to the case in which the string data is uncertain. In the standard embodiment of regular expressions, the input is a sequence of well positioned and well defined symbols from a finite alphabet Σ, and the output is an indication of whether the string matches the expression, that is, whether the string belongs to the language generated by the expression. In this work, we will assume uncertainty on the received input of the regular expressions.

We will propose an uncertainty model based on receiving symbols corrupted by the noise of the channel over which they are emitted. As a consequence of this channel noise, during the process of symbol recognition, we won't just see a symbol, but a probability distribution $\nu : \Sigma \to [0, 1]$, where $\nu(a^{(n)})$ gives the probability that the symbol we are observing is $a^{(n)}$. Then, these symbols $a^{(n)}$, are submitted to the regular expression matching algorithm which operates over them.

In this work, we shall develop algorithms to find the most likely match of a regular expressions ϕ based on the observations ν. In addition, we shall further extend the method to consider *infinite streams of uncertain data*, and we shall study their properties. Then, we will set and explore concrete practical examples, which could be extrapolated to similar scenarios, where we will apply our model and we will show that our algorithms ensure the decidability with probability 1 under determinate circumstances, can calculate justifiably the matching probability of a string of symbols, and are able to recover from reading errors.

Key words

infinite stream, uncertain data, regular expression

Agradecimientos / Подтверждение

Gracias a todas aquellas personas que me han apoyado durante la carrera, ya sea durante un periodo concreto, o durante todos estos años. Gracias especialmente a mis padres, que lo han hecho de manera incondicional, incluso cuando he estado estudiado o trabajando en el extranjero, lo cual no hubiera sido posible sin su apoyo

Спасибо тоже Есветана за ее поддержку. Через пересмотреть мой проект, и поощрять меня. Но особенно, спасибо через последний год, через путешествие и через научи меня наслаждаться природи.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and goal of the project

This work is dedicated to continue the preliminary study of an extension of one of the most important element in computer science, regular expressions or *regexs*, as proposed on [1]. Regular expressions are a very powerful tool for searching over strings of symbols, and checking the compliance of these symbols with a pattern. Their widespread use is one to their efficiency and to their very well understood properties. Regular expressions are equivalent to finite automaton [2] and consequently, they recognize regular language [2]. Exploting this equivalence, efficient matching algorithm can be defined [3]. In the standard embodiment of finite regular expressions, the input is a string of symbols from a finite alphabet Σ, and the output is an indication of whether the string *matches* the expression, that is, whether the string belongs to the language generated by the expression (formal definition of these concepts is shown on next chapter).

The standard model relies on three assumptions on the input for the matching algorithm:

1. **The input is a *finite* sequence of symbols:** $\omega = a_1...a_n$ where $a_i \in \Sigma$
2. **The symbols are well positioned:** If we distribute the symbols on a timeline based on their emission, symbol a_{i+1} is situated on that timeline after symbol a_i
3. **The symbols are well defined:** The identity of the i^{th} symbols is $a_i \in \Sigma$, without uncertainty

Regular expressions have been extended to situations in which assumptions 1) and 2) fail. In [4] regular expressions were applied to streams, that is, to infinite list of symbols, in contraposition to point 1), while in [5, 6] they were applied to partially ordered data, in contraposition to point 2).

In this work, we shall address the situation in which 3) is no longer valid, that is, we will assume that the identity of the i^{th} symbols received is subject of uncertainty. Having this uncertainty on the input may entail a problem for meaningful string matching, and we should realize that as a consequence, this can lead also to a loss of accuracy on any syntactical scanners which will be dependent of the output of the automaton, thus increasing the information entropy. Recent research has tried to recover information from the received symbols by studying the use

of context-aware lexical scanners [7, 8], and even prototyping them [9], or developing extensible regular expressions [10, 11]. These solutions try to approach most of the consequences which may be derived from not having full confidence on the received symbols, but don't actually address the real problem rather they bypass it. In order to give a solution to this problem, we will need first to fully understand it by properly defining the situation we are facing in a more precise way.

Our uncertainty model postulates a ground truth (the symbol emitted by M) corrupted by noise. This noise can be produced either by contingent (produced by the limitations of the symbol recognizer [12]) or by intrinsic causes derived from the symbol definition (produced when defining symbols which are trying to grasp subjective or complex concepts [13]), this noise can be defined in a generic way as a *channel noise*. As a consequence of this channel noise, during the process of symbol recognition, we won't just see a symbol, but a probability distribution $\nu : \Sigma \to [0, 1]$, where $\nu(a)$ gives the probability that the symbol we are observing is a. Then, symbols are emitted at a discrete rate over a channel which contains noise. These symbols, corrupted by the channel noise, are recognized by the detector with certain degree of accuracy. That means, when any symbol a is emitted, it gets corrupted by the channel noise, then the detector associate the received corrupted symbol to n symbols ($a^{(n)}$) with different probabilities ($\nu(a^{(n)})$), according to the likelihood of these symbols to the corrupted one. Then these symbols $a^{(n)}$, are submitted to the regular expression matching algorithm which operates over them. After L iterations of this process, the algorithm then determine whether it has find or not a matching string. This model is shown in Figure 1.1.

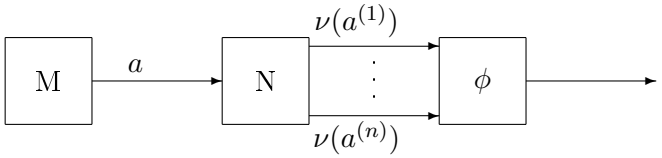


Figure 1.1: The model of string production and detection. The original string, of length L, is generated by a Markov chain over the alphabet Σ; a *channel noise* corrupts this initial chain with probabilities $p(a^{(i)}|a) = \nu(a^{(i)}) \forall a^{(i)} \in \Sigma$. The marginal probabilities $p(a^{(i)})$ resulting from these are the observations that are fed to the regular expression matching algorithm which must determine whether the string contains a sub-string that matches the expression.

Note that the model is not limited to finite lists: the module M may be assumed to produce a stream of symbols. This situation forces us to propose this as an additional problem that our model must give solution, and additionally being able to make the final problem definition which must be addressed by our model, this is, the handling of *infinite streams of uncertain data*.

In this work, we shall develop algorithms to find the most likely match of a regular expressions ϕ based on the observations ν and we shall study their properties. Our purpose is to ensure these algorithms will have a stop condition, and to measure how accurate they will be at the time of matching words, by calculating their error recovery at the time of associating symbols with either high or low probabilities. We will carry out this task by setting and exploring concrete practical examples, which could be extrapolated to similar scenarios. On this study we will find the necessary conditions that these scenarios should have, and what kind of tools our model should use on these scenarios, in order to reduce the error of applying regular expressions over uncertain data as much as possible.

Recent studies have pointed out the importance of developing algorithms for using regular expressions over uncertain data, and have tried to address similar problems related to streams of data and regular expressions [12, 13]. Those studies have proposed practical systems which work under those circumstances, trying to give a solution to the most common problems which arises on these situations. Still these studies have been focused on how to detect complex events

on real time from a large number of possible similar events and how to calculate its correlated probability [13], little has been done in what refers to stop conditions of their algorithms, and no studies have been done referring to that situation. The scientific novelty of this approach is such that the work is the only one of its kind in this scientific area and creating a theoretical basis for future researches.

1.2 Structure of the document

The remaining of this work is structured as follows. Chapter 2 describes concepts that will be used throughout the document, and define theorems on which our model is based; Chapter 3 shows our model definition and study its behavior when dealing with finite strings on different scenarios; whereas Chapter 4 present an extension of the model to infinite lists, on this extension additional scenarios are presented and studied. Finally in Chapter 5 we conclude the document with a brief analysis of obtained results, and proposing a direction for future studies on this field.

2 Background

2.1 Regular expressions

We present here a brief reminder of the relevant facts about regular expressions, limited to what we shall use in the remainder of the paper. The interested reader may find more detailed information in the many papers and texts of the subject, such as [14, 15, 2].

Let Σ be a finite alphabet. We shall denote with Σ^* the set of finite sequences of symbols of Σ, including the empty string ϵ. A *word*, or *string* on Σ is an element of Σ^*, that is

$$\omega = a_1 a_2 \cdots a_n \in \Sigma^n \subset \Sigma^* \tag{2.1}$$

We shall use exponents to represent repetitions of the same sub-string, so if $x, y, z \in \Sigma^*$, we shall set

$$xy^k z \equiv x \underbrace{yy \cdots y}_{k} z \tag{2.2}$$

We shall indicate with $|\omega|$ the number of symbols of the string ω. Also, we shall indicate sub-strings of ω using indices in square brackets, that is

$$\omega[i:j] = a_i \cdots a_j \quad 1 \leq i \leq j \leq |\omega| \tag{2.3}$$

If $i = j$ then $\omega[i:j] = \varepsilon$, the empty string. We shall indicate $\omega[:j] = \omega[0, j]$.

A *regular expression* ϕ is a "word recognizer", that is, a structure that defines a subset $L(\phi) \subseteq \Sigma^*$. The set $L(\phi)$ is called the *language* recognized by ϕ. Syntactically, regular expressions are defined as:

$$\phi ::= a|\phi\phi|\phi^*|\phi + \phi|(\phi)|\varepsilon|\eta \tag{2.4}$$

with $a \in \Sigma$. The symbol ϵ represents the expression that only recognizes the empty string, while the symbol η is the expression that doesn't recognize any string, that is

$$\begin{aligned} L(\epsilon) &= \{\varepsilon\} \\ L(\eta) &= \emptyset \end{aligned} \tag{2.5}$$

In order to define the semantics of an expression ϕ, we define the *modeling* relation $\models$. Given a word ω and an expression ϕ, we say that ω models ϕ ($\omega \models \phi$) if the following conditions are met.

$$\begin{aligned} \omega \models a &\text{ iff } \omega = a \\ \omega \models \phi_1\phi_2 &\text{ iff there are } \omega_1, \omega_2 \text{ such that } \omega = \omega_1\omega_2,\ \omega_1 \models \phi_1,\ \omega_2 \models \phi_2 \\ \omega \models \phi_1 + \phi_2 &\text{ iff } \omega \models \phi_1 \text{ or } \omega \models \phi_2 \\ \omega \models \phi^* &\text{ iff } \omega = \varepsilon \text{ or there is } k > 0 \text{ such that } \omega \models \phi^k \end{aligned} \tag{2.6}$$

The language $L(\phi)$ is then defined as $L(\phi) = \{\omega | \omega \in \Sigma^* \wedge \omega \models \phi\}$. Two expressions are equivalent if they recognize the same language, that is, we define $\phi \equiv \psi$ if $L(\phi) = L(\psi)$.

Given an expression ϕ its *length* $|\phi|$ is the number of symbols it contains.

A *regular set* (or *regular language*) is a set $R \subseteq \Sigma^*$ such that there is a finite expression ϕ ($|\phi| < \infty$) such that $L(\phi) = R$.

One important aspect of regular expressions is their connection with finite state automata.

Definition 2.1.1. *A finite state automaton is a 5-tuple $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ where Q is a finite set of states, Σ is the input alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ is the state transition relation*

A transition of the form $\delta(q, \varepsilon)$ is called an ε-transition. The previous definition refers to a *non-deterministic* finite state automaton (NFA(, if δ is a function $\delta : Q \times \Sigma \to Q$, the automaton is called *deterministic* (DFA).

An automaton is also a word recognizer. Let $\omega = a_0 a_1 \cdots a_{n-1} \in \Sigma^n$ a word. A *run* r of $\mathcal{A}$ over ω is a sequence of states $q_0, q_1, \ldots, q_n \in Q^{n+1}$ such that q_0 is the initial state and for all $i = 0, \ldots, n-1$ it is $\delta(q_i, a_i, q_{i+1})$.

A run is *accepting* if $q_{n+1} \in F$; the word ω is *accepted* by $\mathcal{A}$ (written $\mathcal{A} \vdash \omega$) if there is an accepting run for it. The language recognized by $\mathcal{A}$ is then defined as

$$L(\mathcal{A}) = \{\omega | \omega \in \Sigma^* \wedge \mathcal{A} \vdash \omega\} \tag{2.7}$$

Deterministic and non-deterministic automata recognize the same class of languages. The relation between automata and regular expressions is given by the following theorem [16]:

Theorem 2.1.1. *Given a regular expression ϕ, there is an automaton $\mathcal{A}$ such as $L(\phi) = L(\mathcal{A})$; conversely, given an automaton $\mathcal{A}$, there is a regular expression ϕ such that $L(\phi) = L(\mathcal{A})$.*

2.2 Regular expressions and infinite streams of events

As we mentioned in the introduction, regular expressions matching can be extended to streams, that is, to infinite lists. In the case of finite lists, we have a match if the whole list satisfies the conditions defined on the equations 2.6. The extension to infinite lists can be done in several ways. The first is a direct extension that requires that the whole infinite list matches the expression. This kind of match is recognized by Büchi automata [17], and despite its great theoretical interest, its practical applications are, for obvious reasons, limited.

A second option is to look for finite sub-lists of the stream that match the expression in the standard sense. This is the framework used in [4], and the one we shall consider here.

In this case, it is convenient to couch our argument in the language of *events*. We shall say that the emission of the element a_i corresponds to the occurrence of the elementary event a at the time i. The events are not observed directly, but the values $\nu_i(a^{(1)})...v_i(a^{(n)})$ give us

the probabilities of the simple events at the time i. If a finite sequence of events matches ϕ ($\omega[i:j] \models \phi$), we shall say that we have observed a complex event defined by ϕ. Our aim in this case is to estimate the probability of occurrence of complex events to choose the most likely.

Note that this event definition give a discrete definition of time. Without a discrete definition of time, either there will be needed additional mechanisms to be able to apply regular expressions, or there will no additional mechanism and regular expressions won't be possible to apply at all [18, 19]. The definition of event, allows us to proceed to the definition of *bounded stream*:

Definition 2.2.1. *We call bounded stream S of size n to the concatenation of n events, where a_1 is the first element of the stream, and a_n the last one.*

We can see that the bounded stream definition is the same to the string definition done in section 2.1, when having same alphabet Σ. So after this point we won't make a distinction between a bounded stream and a string, and we will just use the term *string* or *word*. Still this definition will be useful to have a better understanding of the definition of *unbounded stream*:

Definition 2.2.2. *Having a stream S, we call that stream unbounded if for each stream $\omega \subset S$, there exist $\Omega \subset S$ such as $\omega \subset \Omega$*

A consequence of this definition is the following theorem:

Theorem 2.2.1. *$\forall a_i$ of an unbounded stream S there exists a_{i-1} and a_{i+1}*

Proof. Let's consider $\omega = a_i$ contained in a unbounded stream S. If $\omega \subset S \implies \exists \Omega \subset S$ such as $\omega \subset \Omega$. That means that necessarily there must exist either a_{i-1}, a_{i+1} of both. Let's consider it only exists a_{i+1}, and therefore $a_{i+1} \in \Omega$. If we take $\Omega = \cup^{j \geq i} a_j$, it is obvious $\Omega \subset S$, and necessarily there must exists $\Omega_B \subset S$ such as $\Omega \subset \Omega_B$. This new Ω_B must contain a_{i-1}. The proof for a_{i+1} is analogous to this one

□

Without loss of generality we will refer to a_1 as the first considered symbol of the unbounded stream, but it is important to know that there have existed symbols previous to this one. These previous symbols is what we will call *previous context* of a string. It is automatically to see that generally regular expressions are not decidable over a finite time on unbound streams, since unbounded streams are not bounded to any finite time. Still, there is a remarkable subset inside the set of regular expressions which is actually decidable over infinite strings:

Theorem 2.2.2. *If the expression ϕ is limited ($\omega \models \phi \implies |\omega| < N < \infty$), ϕ is decidable over unbounded streams, but it is intractable (NP-complete)*

This theorem was proven in [4].

2.3 Matching as path finding

The recognition algorithm that we use in this paper is a modification of an indirect method known as the DB-graph of the *intersection graph* [4]. Let $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ be the (nondeterministic) automaton that recognizes a regular expression ϕ, and let $\omega = a_1 \cdots a_n$ be a string. We build the intersection graph $I(\phi, \omega) = (V, E)$ as follows:

i) V is the set of pairs (q, k) with $q \in Q$ and $k \in [0, \ldots, n]$;

ii) $(u, v) \in E$ if either:

- **a)** $u = (s, k-1)$, $v = (s', k)$ and $s' \in \delta(s, a_k)$, or
- **b)** $u = (s, k)$, $v = (s', k)$ and there is an ε-transition between s and s', that is $s' \in \delta(s, \varepsilon)$

In order to simplify the notation, when no confusion arises, in the figures, algorithms, and some formulas, we shall indicate the vertex (s_i, k) as s_i^k. Recognition using the graph is based on the following result:

Theorem 2.3.1. *$\omega \models \phi$ iff $I(\phi, \omega)$ has a path $(q_0, 0) \xrightarrow{*} (s, n)$ with $s \in F$.*

Proof. $\omega \models \phi$ iff the automaton has an accepting run, that is, a sequence of states $q_0 q_1 \cdots q_n$ such that $q_i \in \delta(q_{i-1}, a_i)$ and $q_n \in F$. It is immediate to see from the definition of the graph that such a run exists iff there is a path

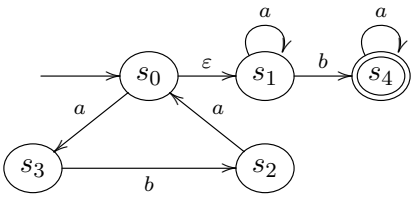
$$(q_0, 0) \to (q_1, 1) \to \cdots \to (q_n, n) \tag{2.8}$$

in $I(\phi, \omega)$. □

In some cases we are interested in determining whether there is a sub-string $\omega[i:j]$ of ω that matches ϕ. To this end, it is easy to verify the following result:

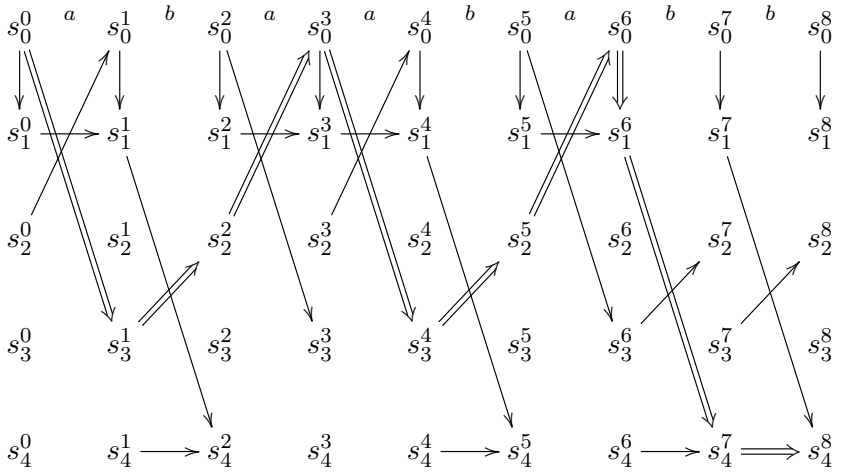
Corollary 2.3.1. *$\omega[i:j] \models \phi$ iff $I(\phi, \omega)$ has a path $(q_0, i-1) \xrightarrow{*} (s, j)$ with $s \in F$.*

Example I:
Consider the expression $\phi \equiv (aba)^* a^* bb^*$, corresponding to the NFA



(2.9)

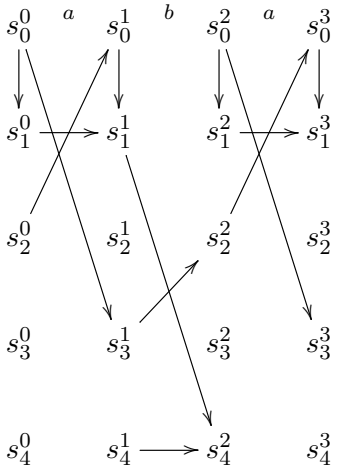
and the string $\omega = abaababb$. The graph $I = (\phi, \omega)$ is



(2.10)

The double edges show a path from s_0^0 to s_4^8 corresponding to the accepting run $s_0 s_3 s_2 s_0 s_3 s_2 s_0 s_1 s_4 s_4$, which shows that the string matches the expression. Note that the sub-string $\omega[4:7] = abab$ also matches the expression, corresponding to the path $s_0^3 \to s_3^4 \to s_2^5 \to s_0^6 \to s_1^6 \to s_4^7$.

Example II:
Consider the same expression and the string $\omega = aba$; the graph $I(\phi, \omega)$ is now



(2.11)

The graph has no path from s_0^0 to s_4^3, indicating that the string doesn't match the expression. However, there is a path from s_0^0 to s_4^2, indicating that the sub-string $\omega[1:2] = ab$ does match the expression

3

Formal model definition

3.1 Formal model definition

In this chapter, we shall give a formal definition of the matching problem on the model of Figure 1.1, and we shall develop the optimal matching algorithm. For the time being, we shall consider only the finite case, that is, we shall assume that M emits a string ω with $|\omega| = L$. Our goal is to determine the most likely sub-string $\omega[i:j]$, observing the different values of $\nu_1..\nu_L$, such that $\omega[i:j] \models \phi$

The symbols are emitted over a channel which contains noise, so that when the symbol a is emitted what we really observe is a discrete occurrence probably distribution over Σ, that is, we observe the quantities $\nu(a)$ where $a \in \Sigma$ and with $\sum_{a\in\Sigma} \nu(a) = 1$ [1]. Here, the detector is also giving us alphabet's definition, $a \in \Sigma \Leftrightarrow p(a) \neq 0$. Also, at any time something is observed, this implies that $\exists a^{(i)} \in \Sigma$ such that for a received $a \in \Sigma$ $p(a^{(i)}|a) \neq 0$. Due to this alphabet's definition, and under the premise of having a well defined alphabet (all possible events are considered), the set of defined events are collectively exhaustive and mutually exclusive ("one and only one of the events occur" [20]), and therefore the union of all possible events must always occur, if we think of the probability as a weight [21]. The string ω is generated by a Markov chain with transition probabilities $q(a_k|a_{k-1})$. Here symbols, as representation of real life events, are not understand as an absolute truth of probability 1, but as a truth with certain grade uncertainty. Let $\omega = a_1 a_2 \cdots a_L$ the (unknown) string emitted by M, and $\omega' = a'_1 \cdots a'_L$ the string composed by the symbols a'_j, where $a'_j = a_j^{(i)}$ such as $\omega' \models \phi$. Note that this ω', doesn't have to be unique, as there may exist several combination of $a_j^{(i)}$ which match ϕ.

We are receiving L symbols, but we are looking for possible *substrings* that match the expression. This entails that we have somehow to compensate for the bias towards shorter expressions. The posteriori probability of an expression ω will be given as the product of the probabilities of its constituting symbols, and shorter expressions will correspond to multiplications with a smaller number of terms, which tend to generate higher probabilities.

[1] Note that this is a formal model not corresponded to what actually happens in regular expression standard applications: we generally have a detector that attempts to detect a symbol (which is, in this case, an event), and stops considering any other similar events which may have occurred. On those models symbols are taken as certain whether they are or not

To avoid this, we consider the *a priory* information carried by a string. If we have no a priory information on the string that is being emitted, it being revealed to us would carry an information equal to $i_i(\omega) = -\log P(\omega)$. If we have at our disposal the string ω', we have an estimation of the strings ω', so we already have some information about it. The new information that we would have by knowing the actual string ω is $i_2(\omega) = -\log P(\omega|\omega')$.

The information that ω' gives us input ω is the difference of these two values:

$$I(\omega, \omega') = i_1(\omega) - i_2(\omega) = \log P(\omega|\omega') - \log P(\omega) = \log \frac{P(\omega|\omega')}{P(\omega)} \tag{3.1}$$

The $P(\omega)$ at the denominator avoids the bias towards short strings: given two strings ω'_1 and ω'_2 with the same value of posteriori probability, we shall select the longest one: the longest string is the one for which the a priory information was greater and therefore the one for which the information given by the estimation is greater.

The probability $P(\omega|\omega')$ is given by

$$\begin{aligned} P(\omega|\omega') &= P(a_n, \omega[:n{-}1] \,|\, \omega') \\ &= P(a_n, \omega[:n{-}1] \,|\, \omega'[:n{-}1]) \\ &= P(a_n \,|\, \omega[:n{-}1], \omega'[:n{-}1]) P(\omega[:n{-}1] \,|\, \omega[:n{-}1]) \\ &= P(a_n \,|\, a_{n-1}, a'_n) P(\omega[:n{-}1] \,|\, \omega[:n{-}1]) \end{aligned} \tag{3.2}$$

The last equality derives from the dependencies of a_n: it depends only on a_{n-1} via the Markov chain and on a'_n via the estimation: it is independent on all the other symbols of ω and ω'. The first term of the last equation can be rewritten using Bayes's theorem as

$$\begin{aligned} P(a_n|a_{n-1}) &= \frac{P(a'_n \,|\, a_n, a_{n-1}) P(a_n \,|\, a_{n-1})}{P(a'_n \,|\, a_{n-1})} \\ &= \frac{p(a'_n \,|\, a_n) q(a_n \,|\, a_{n-1})}{P(a'_n)} \end{aligned} \tag{3.3}$$

This is a recursive equation that gives as a result

$$P(\omega|\omega') = \frac{1}{P(\omega')} \prod_{k=1}^{n} p(a'_k \,|\, a_k) q(a_k \,|\, a_{k-1}) \tag{3.4}$$

where we assume that $q(a_1|a_0) = P(a_1)$ is the a priory probability that the string begin with a_1.

The criterion that we try to maximize is therefore:

$$\mathcal{L}(\omega) = \log \left(\frac{1}{P(\omega)P(\omega')} \prod_{k=1}^{n} p(a'_k \,|\, a_k) q(a_k \,|\, a_{k-1}) \right) = \sum_{k=1}^{n} \log \left(\frac{p(a'_k \,|\, a_k) q(a_k \,|\, a_{k-1})}{P(\omega)P(\omega')} \right) \tag{3.5}$$

The a priory probabilities will be taken as those that result in the maximum information, that is, it will be assumed that the two strings ω and ω' will be composed by assembling independently generated, uniformly distributed symbols. This results in

$$P(\omega) = P(\omega') = \frac{1}{|\Sigma|^n} \tag{3.6}$$

Therefore

$$\mathcal{L}(\omega) = \sum_{k=1}^{n} \log \left(|\Sigma|^2 p(a'_k \,|\, a_k) q(a_k \,|\, a_{k-1}) \right) \tag{3.7}$$

We are interested not only in detecting maximum length strings, but in detecting sub-strings $\omega[i:j]$ as well. To this end, we define the partial information difference:

$$\mathcal{L}_{i,j}(\omega) = \sum_{k=i}^{j} \log\left(|\Sigma|^2 p(a'_k \,|\, a_k) q(a_k \,|\, a_{k-1})\right) = (j-i+1)\log|\Sigma|^2 + \sum_{k=i}^{j} \log\left(p(a'_k \,|\, a_k) q(a_k \,|\, a_{k-1})\right) \tag{3.8}$$

The first expression is the one we actually use. We have written down the second expression to highlight the effect of considering prior information: the term $(j-i+1)\log|\Sigma|^2$ is the bias that, all else being equal, favors the detection of longer strings. This bias derives from the fact that a long string is, a priory, less likely than a short one.

Our problem can therefore be expressed as finding the sub-string:

$$\bar{\omega} = \arg\max \omega[i:j] \models \phi\mathcal{L}_{i,j}(\omega) \tag{3.9}$$

Two simplified cases are of importance in applications. The first is when the generation of the symbols has no temporal dependency, in which case $q(a_k \,|\, a_{k-1}) = q(a_k)$, and

$$\mathcal{L}_{i,j}(\omega) = \sum_{k=i}^{j} \log\left(|\Sigma|^2 p(a'_k \,|\, a_k) q(a_k)\right) \tag{3.10}$$

The second is when the symbols are generated with uniform probability, in which case $q(a_k) = 1/|\Sigma|$ and

$$\mathcal{L}_{i,j}(\omega) = \sum_{k=i}^{j} \log\left(|\Sigma| p(a'_k \,|\, a_k)\right) \tag{3.11}$$

We have already seen that strings that match the regular expressions generate paths from sources to final states in the intersection graph. In this case, however, we have an intersection graph "on steroids", so to speak. At each step we do not receive just one symbol but (in principle) *all* the symbols of $|\Sigma|$, albeit with different probabilities associated.

Example III:
Consider again the expression of example 1. We build the intersection graph considering that, at each step, *all* symbols are received. In this case, there are only two symbols in the alphabet so the structure of the intersection graph for an input string of length 4 would be that of Figure 3.1. In this case, we haven't considered the probabilities associated to the symbols.

* * *

The algorithm for finding the optimal match to the expression is a form of maximum weight graph, with some modification. In a typical shortest path algorithm, each edge (u, v) has associated a weight $w(u, v)$ and, given a path $\pi = [u_1, u_2, \ldots, u_n]$, the weight of the path is the sum of the weights of the edges that compose it, that is $\sum_{i=1}^{n-1} w(u_i, u_{i+1})$. Each vertex u has associated a distance value $d[u]$. When one of the vertices is considered[2], its in-edges are analyzed. If the

[2]The order in which the vertices are analyzed, that is, which vertex will be considered next, depends on the algorithm that one is applying. The algorithm that we develop shall use its own criterion, so this detail is not relevant in this context. The reader may refer to the standard textbooks on algorithms for details.

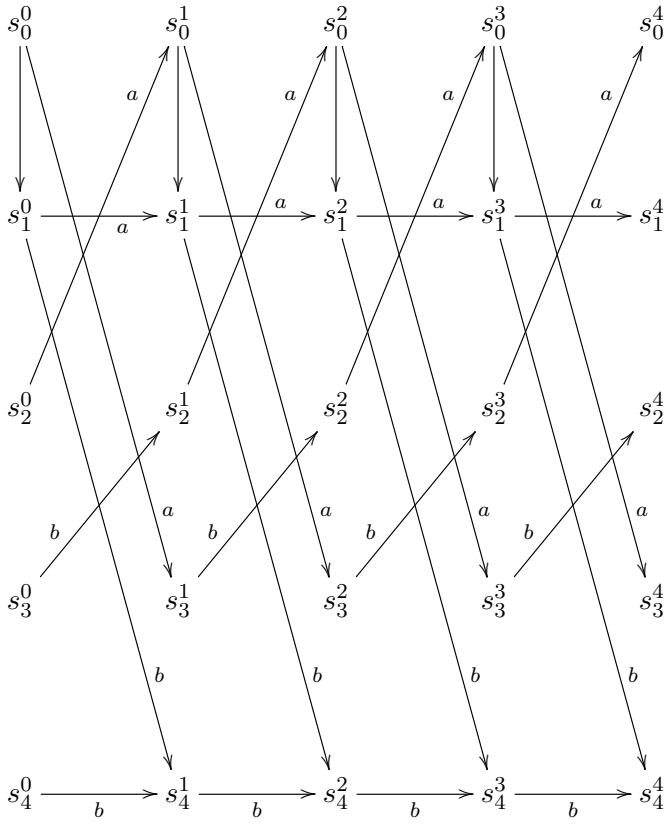
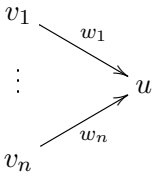


Figure 3.1: The structure of the graph of example 1 under the assumption that at each time all the symbols of Σ are received. The probability associated to each symbol has been ignored in this figure.

situation is the following:



then the distance value of u us updated as

$$d[u] \leftarrow \min\{d[v_i] + w_i, i = 1 \ldots, n\} \tag{3.12}$$

In our case, we mark each edge with a pair $(a, \mu(a))$, where $a \in \Sigma$ is the symbol that causes the traversal of that edge, and $\mu(a)$ is the probability that the symbol emitted was a. Given an edge e, we shall indicate the two elements of the associated pair as $\sigma[e]$ and $\mu[e]$.

In our problem we have a small difference and an additional complication. The small difference is that we look for the path with the *largest* weight, rather than the smaller. The additional complication is that, in the general case, the estimation that we have to minimize for the node u depends not only on the in-neighbors $v_1, \ldots, v_n$, but also on the label of the edge of the optimal path that enters the nodes $v_1, \ldots, v_n$. To clarify this point, given a node u, let $\mathcal{L}[u]$ be the estimate of the criterion that we are maximizing for the paths through u. Suppose we are evaluating a path entering u:

$$q \xrightarrow{a} v \xrightarrow{b} u$$

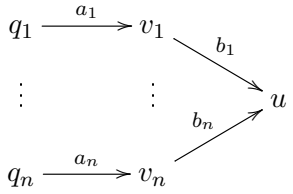
vertices		edges	
$\mathcal{L}[u]$	Estimation of the criterion for vertex u	$\sigma[e]$	Symbol of the alphabet that caused the edge to be crossed, $\sigma[e] \in \Sigma$
$\pi[u]$	Vertex that precedes u in the path built through u	$\mu[e]$	Probability of detection of the symbol $\sigma[e]$
general			
$\mu(a,k)$	Probability that the symbol a has been detected as the k-th symbol of the series. Note that given an edge e from (s,k) to $(s',k+1)$, it is $\mu[e] = \mu(a[e], k+1)$.		
$\lvert\Sigma\rvert$	Number of characters in the input alphabet.		
m	Number of states (rows of the graph)		
n	Number of inputs detected (columns of the graph)		

Table 3.1: Symbols used in the path finding algorithm.

The estimation $\mathcal{L}[u]$ for this path is given by

$$\mathcal{L}[u] = \mathcal{L}[v] + \log\big(|\Sigma|^2 \mu(b) q(b \,|\, a)\big) \tag{3.13}$$

So, in order to update the estimate $\mathcal{L}[u]$ we need to look at one edge further back than we would for a normal path-finding algorithm. In general, given the edges entering u:



we have

$$\mathcal{L}[u] = \max\left\{\mathcal{L}[v_i] + \log\big(|\Sigma|^2 \mu(b_i) q(b_i \,|\, a_i)\big)\right\} \tag{3.14}$$

It wouldn't be hard to adapt one of the standard path algorithms to work on this case, but it is more efficient to take advantage of the structure of the intersection graph. In the graph, we have two kinds of edges: forward edges of the kind $(s,k) \to (s',k+1)$ and ϵ-edges, corresponding to the ϵ-transitions, of the type $(s,k) \to (s',k)$; ϵ-edges can be executed at any time without changing the objective function. Table 3.1 shows the symbols used in the algorithm. The algorithm is composed of two parts: the first is the top level function that receives the graph $I(\phi)$ with the edges marked by the probability of having received the corresponding symbol. The second is a local *relaxation* function that checks whether the criterion estimation of some nodes can be improved by traversing some ϵ-edges. This function also check whether it is convenient to initialize a new path: if the state s_0 has a negative estimation, then its estimation is set to 0 and a new path is initialized. The function *relax* is shown in Figure 3.2. For the sake of simplifying notation, we shall use the notation $\mathcal{L}[s,k]$ in lieu of $\mathcal{L}[(s,k)]$ (the same for π), and we shall indicate the edge from (s_i,k) to $(s_j,k+1)$ as (s_i^k, s_i^{k+1}), using the compact notation for states. The main function of the algorithm is shown in Figure 3.3. The algorithm proceeds time-wise from the first symbol received to the last. The main loop adjusts the objective taking into account the edges from the previous time step, and then calls the function *relax* to take into account ϵ-edges and possible re-initializations of the path. At the end, s_f contains the final state that represents the end of the most likely path. The path can be reconstructed following the predecessor pointers until an initial path.

```
relax((V,E), k)
1.   if L[s_0, k] <0 then
2.     L[s_0, k] ← 0
3.     π[s_0, k] ← ⊥
4.   fi
5.   do
6.     changed ← false
7.     forall s:  (s, k) ∈ V  do
8.       forall t:  (t, k) ∈ V  do
9.         if (t^k, s^k) ∈ E  and  L[t, k] > L[s, k]  then
10.          L[s, k] ← L[t, k]
11.          π[s, k] ← (t, k)
12.          changed ← true
13.        fi
14.      od
15.    od
16.  while changed
```

Figure 3.2: The relaxation function. The function initializes a new path (lines 2, 3) if the start state has a negative value for $\mathcal{L}$ and attempts to traverse all ϵ-edges to check whether some value can be improved by traversing them.

k	1	2	3	4	5	6	7	8
$\mu(a)$	0.8	0.5	0.01	0.8	0.1	0.8	0.2	0.2
$\mu(b)$	0.2	0.5	0.99	0.2	0.9	0.2	0.8	0.8

Table 3.2: The sequence of symbols detected in input, with the probabilities of detection of each one.

Example IV:
Consider, once more, the regular expression of example 1. We detect eight symbols from the alphabet $\Sigma = \{a, b\}$, with the probabilities as in table 3.2. We assume that the symbols are independent and equiprobable, so we apply the formula (3.11). Before the first iteration, the first column of the algorithm has been initialized as:

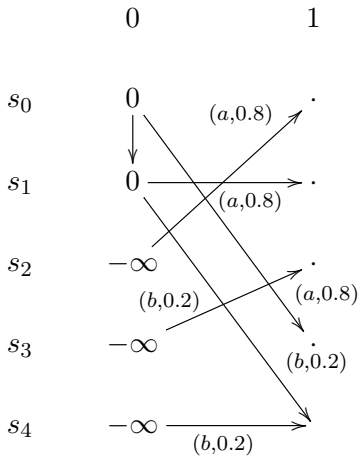
$$\begin{array}{cc} & 0 \\ s_0 & 0 \\ & \downarrow \\ s_1 & 0 \\ s_2 & -\infty \\ s_3 & -\infty \\ s_4 & -\infty \end{array}$$

Note that the state s_1 has value 0 as the ϵ-edge that joins it to s_0 has been relaxed. During the

```
match((V,E))
1.   for i←0 to n do
2.     for j←0 to m do
3.       L[s_j, i] ← −∞
4.       π[s_j, i] ← ⊥
5.     od
6.   od
7.   relax((V,E), 0)
8.   for i←1 to n do
9.     for j←0 to m do
10.      π[s_j, i] ← arg max_{s:(s^{i−1}, s_j^i)∈E} L[s, i − 1] + log(|Σ|² μ[s^{i−1}, s_j^i] q(σ[s^{i−1}, s_j^i] | σ[π[s]^{i−2}, s^{i−1}]))
11.      e_1 ← (π[s_j, i], s_j^i)
12.      e_2 ← (π[π[s_j, i]], π[s_j, i])
13.      L[s_j, i] ← L[π[s_j, i], i − 1] + log(|Σ|² μ[e_1] q(σ[e_1] | σ[e_2]))
14.    od
15.    relax((V,E), i)
16.  od
17.  s_f ← arg max_{(s,k):s∈F} {L[s, k]}
```

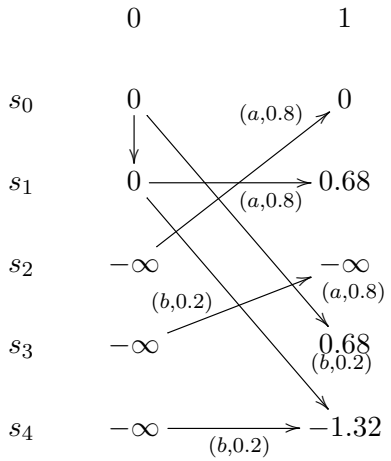
Figure 3.3: The main matching algorithm. The main loop of lines 9–16 proceeds time-wise updating at each step the objective estimation for all the states after symbol i. One the "best" predecessor of a state has been found (line 10), the edges e_1 and e_2 are, respectively, the edge from the predecessor to the current state, and that from the predecessor of the predecessor to the predecessor of the current state. At the end, s_f contains the final state that represents the end of the most likely path. The path can be reconstructed following the predecessor pointers up to the initial node.

second iteration, we consider the edges to the second column, with the following weights:



At this point, each node in the second column computes its value adding $|\Sigma|\mu[e]$ to the value of the predecessor for each incoming edge, and taking the maximum. If the start state has a negative value or if some value may be increased by traversing an ϵ-edge, this is done in the

function *relax*. The graph is now:



Note that the state s_0 had a negative value of $-\infty$, so it has been reinitialized to 0, making it possible to start a new path. Continuing until all the symbols have been processed, we arrive to the graph of Figure 3.4 The state at the bottom-right of the Figure, with a value $\mathcal{L} = 3.56$ is the final state where the optimal path ends. The path is highlighted using double arrows. Note that it doesn't extend to the whole input, but it begins at step 3, and corresponds to the input *ababb*. The reason is the very low probability of the symbol a in step 3, a symbol that would be necessary to continue the sequence ab that began in the first step. At the third step, the state that precedes s_0 and that would have to transition to s_0 to permit the continuation (state s_2) has a value 0.68, given that a has a probability of only 0.01, this gives:

$$\mathcal{L}[s_0, 3] = \mathcal{L}[s_2, 2] + \log|\Sigma|\mu(a) = 0.68 + \log 2 \cdot 0.01 = -4.96 \tag{3.15}$$

This negative value causes $(s_0, 3)$ to be reset to 0 in the function *relax* and, therefore, a new path is started.

In order to show in a clearer way how our model would work, we will illustrate its behavior with some practical examples:

3.2 Examples

In this section we will shall clarify the operation of our algorithm by applying it to some examples. As always in this chapter, we shall just consider finite streams. Before proceeding with the examples, we will define some notation.

As we stated on the model, for each symbol $a \in \Sigma$ received, we will read a set of symbols $a^{(i)} \in \Sigma$ each one associated with a value $\nu(a^{(i)})$. This value is the probability $p(a^{(i)}|a)$, or in other words, the probability of being the symbol emitted. For each symbol a we will have a $p(a^{(i)}|a)$.

In these examples, we will assume $\max(p(a^{(i)}|a_j)) = \nu_j(a^{(k)}) = c_1$ where c_1 is a constant $\forall j$. We will also assume that $p(a^{(i)}|a_j) = c_2 = \frac{1-c_1}{|\Sigma|-1}$ when $a^{(i)} \neq (a^{(k)}$, and therefore c_2 will be also a constant $\forall j$. In other words, we will consider that for each emitted symbol a_j, we will read all elements from alphabet $(a^{(i)})$ with probability c_2, except from $a^{(k)}$, that we will read with probability c_1. This definition of c_1 and c_2 requires from us to make an event distinction. Given ϕ, let p_1 and p_2 be the following events in the i^{th} position:

- "**High informative event**" or "$\boldsymbol{p_1}$ **event**": A symbol $a^{(i)}$ that is the j^{th} symbol of a string that matches ϕ, observed with $\nu_j(a^{(i)}) = c_1$. This events increase the value of $\mathcal{L}$.

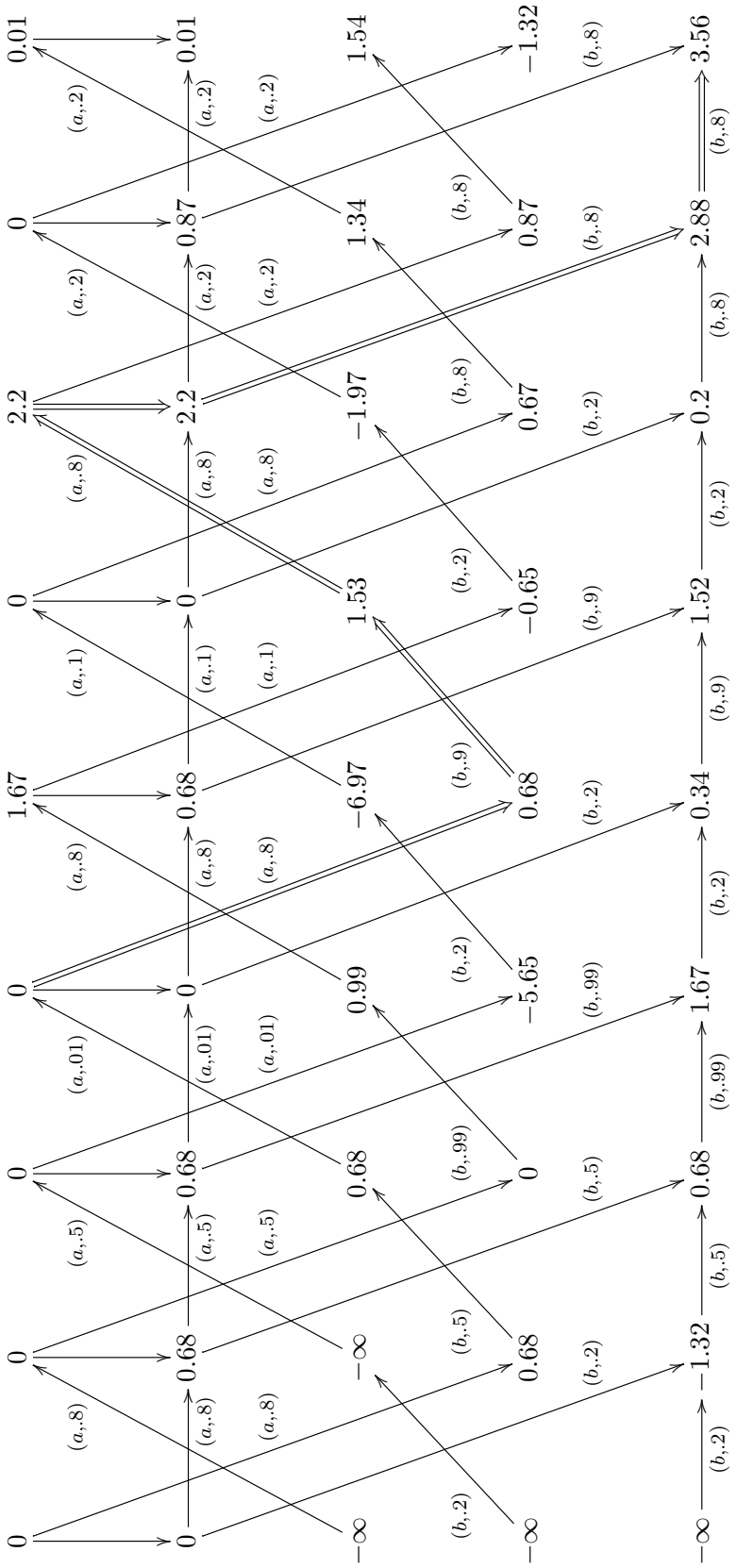


Figure 3.4: The graph corresponding to the expression of example 1 with the input probabilities of example 4. The state at the bottom-right of the Figure, with a value $\mathcal{L} = 3.56$ is the final state where the optimal path ends. The path is highlighted using double arrows. Note that it doesn't extend to the whole input, but it begins at step 3, and corresponds to the input *ababb*.

- "**Low informative event**" or $\boldsymbol{p_2}$ **event**: A symbol $a^{(i)}$ that is the j^{th} symbol of a string that matches ϕ, observed with $\nu_j(a^{(i)}) = c_2$. This events decrease the value of $\mathcal{L}$.

In this subsection, we will see two examples where we have received a finite stream ω, and we have read this emitted string as different potential ω' due to channel noise, having special consideration for ω'_{c_1}, as the string composed by the elements read with probability c_1. We will also have a regular expression ϕ, which will be used to match that read string. We will work as well with an alphabet Σ and a known probability c_1.

Example V:
Consider the regular expression $\phi \equiv a^*$, the alphabet $\Sigma = \{a, b\}$ ($|\Sigma| = 2$) and $c_1 > 0.5$. Now, let's consider we have read the following $\omega'_{c_1} = a^n b a^m$. With the considered premises we have $c_2 = 1 - c_1 < 0.5$. As we can see, in this example, we can find a matching string $\omega'_{c_1}[0 : n-1] = \omega'_1$. But there is also a remarkable matching string we can find, if we take in account we have read symbol a on position n, with probability c_2. This matching string would be $\omega'_2 = a^{n+m+1}$. Now, let's apply the weight formula to both matching strings and compare them, in order to find out which will be the most probable to take place.

$$n * \log(2 * c_1) < (n+m) * \log(2 * c_1) + \log(2 * (1 - c_1)) \tag{3.16}$$

After operating on this inequation, we arrive to the following expression:

$$m > \frac{\log(2 * (1 - c_1))}{\log(2 * c_1)} \tag{3.17}$$

So in order to take ω'_2 over ω'_{c_1}, for high values of c_1 we will require from higher values of m.

* * *

If we take some time to analyze this example we will realize that it correspond quite well to the reality. It has sense to think that if we have a serie of related consecutive events, followed by a singularity, and later another serie of related events, there is a huge chance that the singularity didn't happen at all, and that we have commit a error at the time of reading it. Meanwhile, if we have a serie of related consecutive events, followed by a singularity, and later isolated events which we are not sure about their relation with the first ones, the chance of the singularity not happening is lower [22]. We could say that the matching string supposes the context, and using this context, we can evaluate the singularities as an error or not, depending of that context. Let's try to show this with another example

Example VI:
Consider the regular expression $\phi \equiv a^*$, the alphabet $\Sigma = \{a, b\}$ and $c_1 = 0.75$. Now, let's consider we have read the following $\omega'_{c_1} = aaaababababab$. With the considered premises we also have $c_2 = 0.25$. As we can see on Figure 3.5, where we study how the weight function behaves, the more singularities, the most likeness of having read them correctly. Or in other words, the more singularities we find, the more that the context looses meaning.

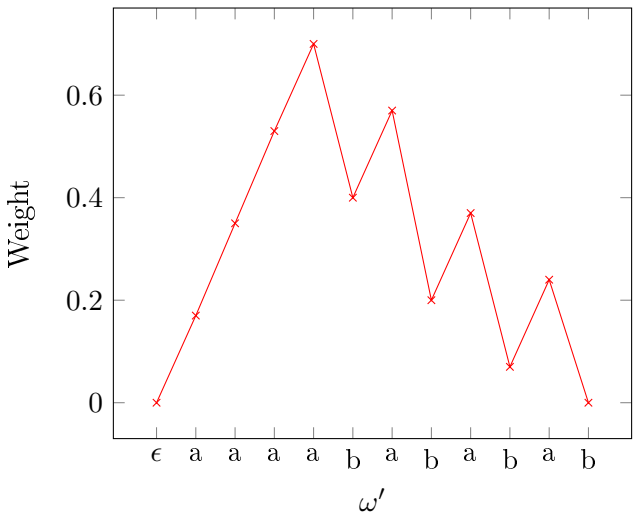


Figure 3.5: Weight when matching $\phi \equiv (a)^*$

3.3 Analysis of typical scenarios

In the previous section we have seen the behavior of our model in two practical examples. In this section, we would like to extend the previous analysis to some generic scenarios, based on the channel noise behavior, in order to have a deeper understanding of the model. We will analyze different scenarios where we will study how the channel noise affects the weight formula, in order to point out the necessary assumptions which our matching algorithm should consider. We will continue this scenario analysis in the following chapter, where we will introduce the unbounded streams to the scenarios.

3.3.1 Baseline: zero information

In this scenario, we are going to consider observations have minimal information, that is $p(a_k^{(i)} \mid a_k)$ will be constant $\forall\ a_k^{(i)}, a_k \in \Sigma$. Since we have defined events as collectively exhaustive and mutually exclusive, the union of all possible events must always occur (taking probabilities as a weight function [21]), then we will have $p(a_k^{(i)} \mid a_k) = \frac{1}{|\Sigma|}$, which will result on following weight formula:

$$\mathcal{L}_{i,j}(\omega) = (i-j)(\log(1) - C) + (i-j) * C = 0 \tag{3.18}$$

This result has sense if we think what are the consequences of having a channel noise homogeneous. We can't distinguish what symbols have been emitted, since the channel noise affects them equally, letting us with no information. The main consequence will be that on this scenario all the matching strings will have the same probability.

* * *

With these results, our model is telling us that, if we have uniform probabilities for all elements of a certain ω, we won't have any information about this ω ($\mathcal{L}(\omega) = 0$). This is the situation in which no prediction is possible. From now on we shall assume, that $\forall a_k \exists a^{(i)}, a^{(j)}$ such that $p(a^{(i)} \mid a_k) < p(a^{(j)} \mid a_k)$, and therefore $\exists a \max(a^{(i)})$.

3.3.2 Channel noise as a constant error when reading a symbol

In this scenario we shall consider that we have a probability c_1 of reading the correct symbol (independently of the symbol) and a probability c_2 of observing the wrong symbol (also independently of the symbol). That is, we have for all k:

$$p(a^{(k)} \mid a_k) = c_1 \tag{3.19}$$

and

$$p(a^{(h)} \mid a_k) = c_2 \; h \neq k \tag{3.20}$$

The normalization condition entails:

$$c_2 = \frac{c}{|\Sigma| - 1} \tag{3.21}$$

We are also considering $c_2 \neq c_1$ in order not to fall in the scenario described in section 3.3.1. Now, with this additional assumptions, we can rewrite the weight formula (4.15):

$$\mathcal{L}_{i,j}(\omega) = \max \left\{ \sum_{k=i}^{j-e} \log(c_1) + \sum_{k=j-e}^{j} \log(c_2) \right\} + (i-j) * C \tag{3.22}$$

Where e shall be the value over which the objective $\mathcal{L}$ is to be maximized. If $c_1 < c_2$ then the value of e which will maximize the weight function will be j; if $c_2 < c_1$ then the value of e which will maximize the weight function will be 0. So our weight function will look as follows:

$$\mathcal{L}_{i,j}(\omega) = \max\{(i-j)\log(c_1), (i-j)\log(c_2)\} + (i-j) * C = (i-j) * (\max\{\log(c_1), \log(c_2)\} + C) \tag{3.23}$$

Figure 3.6 shows the values we get from this formula when we make modifications on the different involved parameters.

As we can see on Figure 3.6, when $c_2 < c_1$ the string ω_{c_1} will have positive value, and any other string will have a negative value. Meanwhile, when $c_1 < c_2$, the relation is the opposite. And we can also see that this situation will occur independently of the string size. This situation is produced since the string size appears in our formula as a multiplier of the main result. The translation of this mathematical affirmation is that the size will only change the speed of growth (or the speed of decrease) of the weight, but not the place where it gets canceled.

So, this scenario tell us that either we always take the symbol with probability c_1, or we always take any symbol of the alphabet, but the the symbol with probability c_1. This second option has no sense for the following reasons:

- If $|\Sigma| = 2$, we could simply rename c_2 as c_1 and vice versa.
- If $|\Sigma| > 2$, the scenario will be similar to the one presented on section 3.3.1. We would have constantly almost no information about the emitted symbol, since most of the symbols will have the same (high) probability. This situation let us without any information which could help us to decide to take a particular symbol over the rest of the read symbols, making most of the matching strings with same size to have same probability.

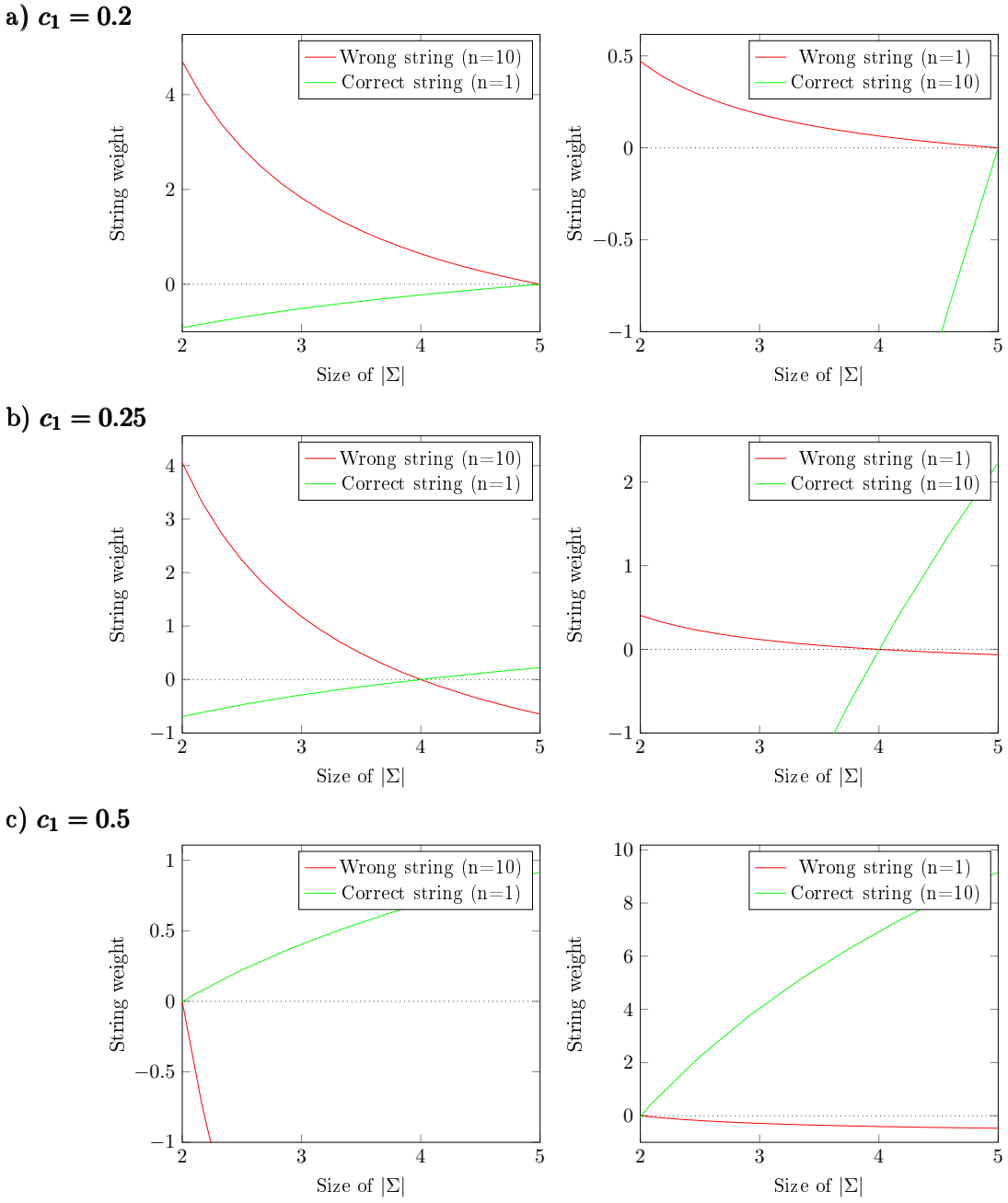


Figure 3.6: Weight of matching and not matching strings for different c_1 and n values.

Due to these reasons, from this point, we will always consider $c_1 > c_2$. The main consequence of this result is that in our model we will always prioritize to take the symbol read with probability c_1. This symbol will increase the string value, meanwhile any other symbol will reduce the value of the string. Right now, our model is the same of the classical one, so we still have to show the additional value of our proposed model. We will find this added value on the next scenario.

3.3.3 Spike errors on a matching string

In the previous scenario we studied how the weight formula (3.22) behaves as a function of the variables $n = |\omega| = j - i$, c_1, $|\Sigma|$, and the number of matching symbols read with probability c_2 (variable e). On that previous scenario, we also found that the behavior of our model was quite stable and predictable for similar situations to the one presented on the scenario. In this section we will consider a different situation. On this scenario the effect of noise on emitted symbols is considered as a spike lecture error of isolated symbols, therefore punctually under determinate circumstances, we may choose symbols read with probability c_2. We will illustrate this scenario on the following example:

Example VII:
Let us consider the expression $\phi \equiv a^*$ and an alphabet Σ with $a, b \in \Sigma$, and $|\Sigma| \geq 2$. We will also take a value of $c_1 > 0.5$, as a probability of making an correct symbol identification. Suppose we have read a string $\omega' = a^h b^e a^r$, where each symbol have been read with probability c_1. At first sight, it is easy to see that we will have inside this ω' at least two matching strings: $\omega'_{s1} = a^h$ and $\omega'_{s3} = a^r$. But we know there could be an additional matching string contained inside this ω, if instead of reading the elements of the sub-string $\omega_{s2} = b^e$ with probability c_1 we read them as elements a with probability c_2. This will give us a new sub-string $\omega'_{s2'} = a^e$, with each symbol read with a probability c_2. Now we can construct a new matching string $\omega'_{s4} = \omega'_{s1} + \omega'_{s2'} + \omega'_{s3} = a^{h+e+r}$.

Now we must determine under the hypothesis of constant uncertainty (c_1 and c_2 constant) whether the algorithm will choose ω'_{s4} over ω'_{s1}, that is whether $\mathcal{L}(a^h) > \mathcal{L}(a^h b^e a^r)$ as a function of e and r. Note that the effects of e and r are opposite: A higher value of e will make us less likely to choose $|\omega'_{s4}|$, as it will expose us to take more symbols with probability c_2, thus reducing the value of $\mathcal{L}$. Meanwhile a higher value of r will increase the value of $\mathcal{L}$, therefore making us to "recover" from the string of unlikely symbols.

So, in other words, the question that is posed to us, is if this new matching string ($|\omega'_{s4}|$) is providing us any additional information with respect to $|\omega'_{s1}|$. In order to answer this question, we are going to study the amount of information that ω_{s4} is providing us, compared to the information provided by ω_{s1} and ω_{s3}:

$$h * (\log(c_1) + C) = (h + r) * (\log(c_1)) + e * (\log(c_2)) + (h + r + e) * C \tag{3.24}$$

After operating on this equation, we arrive to the following expression:

$$T = -\frac{\log(c_1) + C}{\log(c_2) + C} = -\frac{\log(c_1 * |\Sigma|)}{\log(c_2 * |\Sigma|)} \tag{3.25}$$

This value T, is a relation between the number of events p_2 (e) and the number of events p_1 (r). Then, if we get a value of e and r such as $\frac{e}{r} < T$, on over eye, the algorithm will recognize the input as a valid one. The higher the value of T is, the bigger the tolerance we will have

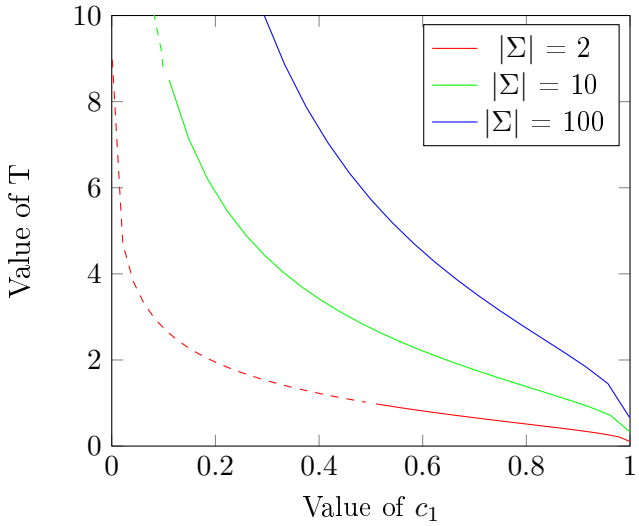


Figure 3.7: Value of Error Tolerance for values of $|\Sigma|$. The dashed line indicate the values where $c_2 \geq c_1$, which we are not considering. This values don't match the initial conditions

p_2. We will call this constant T the "Error Tolerance". In the Figure 3.7, we can see the values taken by this variable for different alphabet sizes.

The first thing we see in Figure 3.7 is the confirmation of what we already know from scenario 3.3.2. We can see that having $c_2 > c_1$, is an incongruence, since we will be considering most of the matching strings as the original emitted string.

As we can see, the error tolerance is fixed in a scenario with the specified conditions (channel noise homogeneous to all symbols). Once we stipulate the a constant value of c_1 and $|\Sigma|$, T value remains constant. This variable will be very useful at the time of deciding if we want to continue to explore a string with a few errors, or if we stop exploring it. We will define how we will use it in the next scenario.

4

Extension to infinite streams

In Chapter 3, we have developed an algorithm to find the optimal (in the sense of minimal residual information) match of a regular expression in the case of a bounded input. However, as we have mentioned in Chapter 1, many scenarios of practical interest involve detecting events in infinite streams of elementary events. This problem can be modeled as finding a finite sub-string matching an expression ϕ in an infinite list of symbols. This problem has been studied in [4], under the assumption that there is no uncertainty in the identity of the symbols. In this chapter we shall study the problem of matching a (*finite*) sub-string of an infinite list under the assumptions that the identity of the symbols is uncertain and the amount of emitted symbols is infinite.

First we will set up two examples where we will point out the challenges to overcome at the time of dealing with infinite streams of uncertain data, then we will make a formal definition of these challenges, and finally, we will define how our model will give answer to those problems.

4.1 Examples of infinite streams with uncertainty

Consider again the alphabet Σ and error probabilities c_1 and c_2, defined in section 3.2. In this examples a infinite string ω is being emitted. This string will be read as different possible matching strings ω' due to channel noise, having special consideration for ω'_{c_1}, as the string composed by the elements read with probability c_1. We will also have a regular expression ϕ, which will be used to match that string we are reading.

Example VIII:
Consider the regular expression $\phi \equiv a^*$, the alphabet $\Sigma = \{a, b\}$ and $c_1 = 1$. That means we are considering that there is no channel noise ($c_2 = 0$), and therefore $\omega = \omega'_{c_1}$.

In this example, we won't have any decidability problem because all matching sub-strings of the unbounded stream ω will be finite ($p(b) \neq 0$, by the alphabet definition), since those strings will be composed by a finite number of a and the weight function will have a value $\neq \infty$. If we try to add to any finite matching string ω_{match} a symbol b, we will have a string $\omega_{match+b}$ with weight $\mathcal{L}(\omega_{match+b}) = \mathcal{L}(\omega_{match}) + \log(2 * 0) = -\infty$, which will be automatically discarded.

* * *

As we could see on this example, when there is no noise on the channel, our model works the same way as classical regular expressions models. This makes our model not just valid for our specific conditions, but also valid for classic regular expressions problems.

Example IX:
Consider the regular expression $\phi \equiv a^*$, the alphabet $\Sigma = \{a, b\}$ and $c_1 = 0.75$. Now, let's take any sub-string from the unbounded stream ω'_{c_1} with length n. If we take $i \leq n$ such that $\omega'_{c_1}[i:n] > \omega'_{c_1}[j:n]\ \forall j \leq n, j \neq i$. We will call this string $\omega'_{c_1}[i:n]$ as $\omega'_{max} = a^{n-i}$. If the following symbol from ω_{c_1} (the symbol in the $(n+1)^{th}$ position) is a, we will add it to the matching string ω'_{max}, due to $\mathcal{L}(\omega'[i:n+1]) > \mathcal{L}(\omega'[i:n])$. This situation is analogous to the one exposed on example 6. But if the symbol in the $(n+1)^{th}$ position is b, then $\omega'[i:n+1] < \omega'[i:n]$. In order to be able to know if we accept this symbol as the end of the matching sub-string with probability c_1, or we interpret it as the symbol a with the complementary probability c_2, we will have to study the following k symbols. We will do so by comparing the weight of the new sub-string which we will analyze, with size $(n+1)+k$, to the previous one, the string we already have, with size n. This comparative brings as the following inequity:

$$\mathcal{L}(\omega'_{max}) < \mathcal{L}(\omega'_{max}) + \log(2 * 0.25) + \mathcal{L}_{n+2,n+k}(\omega') \tag{4.1}$$

If the k incoming symbols satisfy the following inequity, we will accept $\omega'[i:n+1+k] = a^{n+1+k}$ as the new ω'_{max}, and therefore, as the best candidate for being a matching string, and we will have to repeat the process. If the new k incoming symbols doesn't satisfy the inequity, we will discard them, and we will take $\omega'[i:n] = a^n$ as the matching string. Note that this process may continue indefinitely.

* * *

As we could see on this example, when working with unbounded streams, we doesn't have just to take in account the actual context, but the "future" context. This can suppose a problem, as we will see in the following sections.

4.2 Decidability of matching on an infinite list with noise

In classic regular expressions models, we can determine if any input string is accepted by any regular expression in linear time [23]. The reason is that, in a classic model we work with a finite and certain input. That is, the input is known once the matching algorithm starts, and the symbols we read are exactly the same symbols that have been sent to us (the input is certain). This two factors allow us to determine unequivocally if an input string matches or not a specific regular expression. This situation is the opposite to the one presented by our model, were we are working with infinite streams.

In the case of infinite streams, we are not interested in finding the one sub-string that best matches the expressions: in general, several events may be detected at the same time and there will be several string in different parts of the stream, possibly partially overlapping, that match the expressions. We are interested in catching them all. This multiplicity causes several

problems for the definition of a proper semantics for collecting matching strings (many problems arise out of having to decide what to do when matching strings overlap) which, in turn, may cause decidability issues [4]. We shall not consider those issues here, as they are orthogonal to the problems caused by uncertainty: if we can solve the basic problem of deciding whether $\omega \models \phi$ under uncertainty, then all the problems related to the definition of a proper semantics in a stream can be worked out using the theory in [4].

In the case of streams, we are not typically interested in finding the sub-string which matches the regular expression with the biggest probability of all matching sub-strings, which represents a too strong condition for practical applications. Given a (finite) portion of the stream ω such that $\omega \models \beta\phi$, it is clearly undecidable in an infinite stream, whether there will be, at some future time, a portion ω' such that $\omega' \models \beta'\phi$ with $\beta' > \beta$. Moreover, in streams we are interested in determining a collection of finite strings that match the expression, so the use of an absolute criterion such as assuming only one string can match the expression in the strong sense is not very useful. We will try to illustrate this situation with an example:

Example X:
As in example 9, let's consider the regular expression $\phi \equiv a^*$, the alphabet $\Sigma = \{a, b\}$. On a classical model, we will be able to determine whether a string matches ϕ. For example, if we observe the string $\omega = aaab$ with $n = 4$, we can say we found a match with the sub-string $\omega[0:2] = aaa$. But in our model scenario, where we are working with channel noise, if we read the same string, as a sub-string $\omega'_{c_1}[i : i+4]$ from a unbounded stream ω'_{c_1}, we won't be able to affirm this that easily. The reason for this situation is that ω'_{c_1} have been altered by the channel noise, and therefore the original string $\omega[i : i+4]$ could be either *aaab*, *aabb*, *abab*... and so on, with all possible combinations of the alphabet symbols on the four positions. The election of the best candidate for sub-string $\omega[i : i+4]$ will be selected by using the weight formula stated on the section 3.1, using the different values of $\nu(a^{(i)})$. But still, this formula can't resolve our scenario in a total satisfying way. One of the possible candidates for the sub-string is $\omega[i : i+4] = aaaa$. For this combination of symbol, the stop condition has not be reached, and therefore, the algorithm will need to consider additional symbols subsequent to the position $i+4$. In concrete, as we expect to have infinite number of symbols after the one on the position $i+4$, the matching algorithm won't stop until $p(a|\omega[i])$ for $i \geq n$ is equal to 0.

We can conclude that the problem of the example is not decidable ("there is no algorithm that takes as input an instance of the problem and determines whether the answer to that instance is yes or no" [2])

* * *

As we have seen on the example, generally we won't be able to find the sub-string with the biggest probability of all matching sub-strings, when applying regular expressions over any infinite uncertain stream. We shall therefore find the sub-string with the biggest probability from a reduced set of matching sub-strings throughout this section. Since the stream is infinite and we are interested in chunks of it, we shall assume, without loss of generality, that the strings we are testing start at the beginning of the relevant part of the stream, that is, all the strings that we test are sub-strings of the stream.

The problem we are interested in is therefore the following:

> STREAM-MATCH: Given a string ω, and expression ϕ, and an infinite stream of observations ν, is it the case that $\omega \models \phi$?

As we mentioned, we assume that if $|\omega| = L$, the observations on which the recognition of ω is based are the first L of the stream ν.

Our first result is a negative one:

Theorem 4.2.1. STREAM-MATCH *is undecidable.*

Proof. Suppose the problem is decidable. Then there is an algorithm $\mathcal{A}$ such that, for each expression ϕ, observations ν and string ω, $\mathcal{A}(\phi, \nu, \omega)$ stops in finite time with "yes" if $\omega \models \phi$, and with "no" otherwise.

Consider the expression $\phi \equiv a^*$, and an alphabet Σ with $|\Sigma| > 1$ and $a \in \Sigma$. Suppose that the observations are such that $\mathcal{L}(a^L) = \beta$ and, for $k > L$, $\nu[k](a) = q < 1/|\Sigma|$. Then, for $N > L$:

$$\mathcal{L}(a^N) = \mathcal{L}(a^L) + (N - L) \log |\Sigma| q < \beta \tag{4.2}$$

Since the algorithm is correct, it will stop after M steps on "yes". Note that $\mathcal{L}(a^M) = \beta' < \beta$.

Consider now a new stream of observations ν' with $\nu'[k] = \nu[k]$ for $k \leq M$ and, for $k > M$, $\nu'[k](a) = c > 1/|\Sigma|$, that is, $\log |\Sigma| c > 0$. Take $Q > M$, then

$$\mathcal{L}(Q) = \mathcal{L}(a^M) + (Q - M) \log |\Sigma| c = \beta' + (Q - M) \log |\Sigma| c \tag{4.3}$$

If

$$Q > M + \frac{\beta - \beta'}{\log |\Sigma| c} \tag{4.4}$$

then $\mathcal{L}(a^Q) > \mathcal{L}(a^L) = \beta$, therefore $a^L \not\models \phi$. On the other hand $\mathcal{A}$ is working as in the previous case on the same data: it will only visit at most M elements on ν', so it will stop on "yes", contradicting the hypothesis that it is correct. □

Remark 1: Note that we have proven something stronger than undecidability: undecidability is related to Turing machines, while we have proven that with the available information no finite method can decide the problem, a notion known as *(un)realizability* [24].

* * *

The undecidability result depends on having $\nu[k](a) > 0$ for all k. If for some m we have $\nu[m](a) = 0$, then for all $k \geq m$ we have $\mathcal{L}(a^k) = -\infty$ independently of the values $\nu[k](a)$ for $k > m$ (remember that we are interested in matching *finite* sub-strings: things might be different for infinite string matching). This entails that, in the previous example, in order to ascertain whether $a^L \models \phi$ we only have to check strings a^k for $k < m$ and the problem is therefore decidable.

In terms of the Cartesian graph, a^L corresponds to a path π_L and decidability depends on the fact that in order to check matching we only have to extend the path up to m: after that the value of the objective in all paths that extend π_L is $-\infty$.

In the general case, the presence of zero-valued observations, even an infinite number of them, does not guarantee decidability.

Example XI:
Let $\phi \equiv (ab)^*$, $a, b \in \Sigma$, and $|\Sigma| > 2$. Suppose $\nu[k](a) = 0$ for k odd and $\nu[k](b) = 0$ for k even.

Then $\mathcal{L}((ab)^n) > -\infty$ for all n, and the same considerations of the theorem apply considering sequences of groups ab (for which $\nu[2k](a) \cdot \nu[2k+1](b) > 0$) in lieu of the symbol a of the demonstration.

If we have zero-valued observations on all observations, we can reduce the problem to one of infinite streams under the classical model, and still not being decidable:

Example XII:
Consider the expression $\phi \equiv a^*(a+b)^*bba$ and the alphabet $\Sigma = \{a, b\}$. Once again, on a classical model we will be able to affirm if a string matches or not ϕ and therefore, if the algorithm halts or not. Consider now that there is no channel noise ($c_1 = 1$), and that we have received $\omega = abba$. Because our expression contain $(a+b)^*$ any new input symbol will be accepted by the matching algorithm. In a classical scenario where the data is static, we will stop receiving input symbols at some point, and therefore we will be able to determine if there is a match with the whole input string. The problem is that in our scenario we don't expect to stop receive data at some point, so our matching algorithm won't stop.

* * *

However, if the values of k and a for which $\nu[k](a) = 0$ are in sufficient number and randomly distributed, then the problem is almost always decidable.

Theorem 4.2.2. *Suppose that for each $a \in \Sigma$, $\nu[k](a) = 0$ infinitely often. Then with probability 1* Stream-Match *can be decided in finite time.*

Before we prove the theorem, we need some additional constructions and results. For each $a \in \Sigma$, define the list

$$[a] = [k_1^a, k_2^a, \ldots] \tag{4.5}$$

with $k_i^a < k_{i+1}^a$ and for each $i \in \mathbb{N}$, $\nu[k_i^a](a) = 0$. That is, $[a]$ is the list of indices of the observations in which a has zero probability of occurrence. Because of our hypothesis, each $[a]$ is an infinite list of finite numbers.

Also, set

$$[a]_{|n} = [k | k \leftarrow a, k > n] \tag{4.6}$$

From these lists, we build a list Ξ as follows:

```
1.  Ξ ← []
2.  while true do
3.    a ← uniform(Σ)
4.    k ← k_1^a
5.    Ξ ← Ξ ++ [k]
6.    for a' in Σ do
7.      [a'] ← [a']_|k
8.    od
9.  od
```

where uniform(Σ) is a function that picks an element of Σ at random with uniform distribution. Note that the list

$$\Xi = [\xi_1, \xi_2, \ldots] \tag{4.7}$$

is also infinite, so it can never be built completely and, consequently, the algorithm never stops. However, in the proof of the theorem we shall only use finite parts of Ξ so one can imagine a lazy evaluation of the algorithm that only computes the portions that we need for the proof. From the construction of Ξ, it is easy to see that, for each ξ and a,

$$\mathbb{P}\Big[\nu[\xi_i](a) = 0\Big] = \frac{1}{|\Sigma|} \tag{4.8}$$

and, consequently,

$$\mathbb{P}\Big[\nu[\xi_i](a) > 0\Big] = \frac{|\Sigma| - 1}{|\Sigma|} \tag{4.9}$$

Lemma 4.2.1. *Let π be a path in a Cartesian graph. $\omega[\pi] \in \Sigma^*$ the string that causes it to be followed, and let $\mathcal{L}(\pi) > -\infty$. Then, with probability 1, π is finite.*

Proof. Suppose that the lemma is not true, then there is $\epsilon > 0$ such that

$$\mathbb{P}\Big[\pi \ \text{ finite}\Big] < 1 - \epsilon \tag{4.10}$$

That is, $\mathbb{P}\Big[\pi \ \text{ infinite}\Big] > \epsilon$, or

$$\forall n \in \mathbb{N} \ \ \mathbb{P}\Big[|\pi| > n\Big] > \epsilon \tag{4.11}$$

Consider the list Ξ, and the element ξ_k, with

$$k > \frac{\log 1/\epsilon}{\log \frac{|\Sigma|}{|\Sigma|-1}} \tag{4.12}$$

If $|pi| > \xi_k$ then, for each $i \leq k$, if $\pi[\xi_i] = a_i$, it must be $\nu[\xi_i](a_i) > 0$ (since, by hypothesis, $\mathcal{L}(\omega[\pi]) > -\infty$). This event has probability $(|\Sigma| - 1)/|\Sigma|$, therefore

$$\mathbb{P}\Big[\nu[\xi_1](a_1) > 0, \ldots, \nu[\xi_k](a_k) > 0\Big] = \left(\frac{|\Sigma| - 1}{|\Sigma|}\right)^k < \epsilon \tag{4.13}$$

which contradicts (4.11). □

After we have pointed out the main challenges our model will have to overcome, in a practical and in a formal way, we can proceed with our proposed solution to these challenges in the next section

4.3 Matching scenarios on infinite streams

In the previous section we have pointed out the decidability problems that working with unbounded uncertain streams could carry. In this section, we will look at some examples and the practical solutions afforded by theorem 2.2.2. First, we will make a short review of our model, and the assumptions we have make to work with it, then we will proceed with an analysis of some generic scenarios, were we will be able to have a better understanding of the problem we are trying to give solution, and finally we will define the tools our model will use to ensure the stop condition on these generic scenarios.

4.3.1 Previous assumptions and considerations

Let's consider a finite alphabet Σ where $a_k \in \Sigma \; if \; p(a_k) \neq 0$. We will consider as well $p(a^{(k)}|a_k) \neq 0 \; for \; a^{(k)} \in \Sigma$. Since $p(a_k) \neq 0 \; \forall a_k \in \Sigma$, we can assume that for long enough streams ($|\omega_{stream}| \to \infty$), all finite symbol combinations will appear (symbols are independently generated and uniformly distributed). We will also assume that all symbols occur with the same probability $p(a_k) = 1/|\Sigma|$. Our main goal being to maximize the weight function (3.11), it is important to see how this assumption affects us. Having the following weight formula:

$$\mathcal{L}_{i,j}(\omega) = \max \left\{ \sum_{k=i}^{j} \log \left(|\Sigma| p(a'_k \,|\, a_k) \right) \right\} \tag{4.14}$$

Now we will try to extend the weight function, and take out from the operator "max" as many terms as we can. Since $|\Sigma|$ is constant, we have:

$$\mathcal{L}_{i,j}(\omega) = \max \left\{ \sum_{k=i}^{j} \log \left(p(a'_k \,|\, a_k) \right) \right\} + (i-j) \log \left(|\Sigma| \right) \tag{4.15}$$

From this point we will refer to $(i-j)$ as $|\omega_{match}|$. For limited regular expressions ($\omega \models \phi \implies |\omega| < N < \infty$), we won't have problems with the value of $|\omega_{match}|$ since its value will be finite. But for the rest of regular expressions the situation will be different. In those cases, due to the characteristics of the problem, for any $M \geq 0$ there will be matching strings with $|\omega_{match}| > M$. On those matching strings, we will always have the largest weight and therefore will always be priority, making as discard all shorter matching strings, even if they are composed just by high probability symbols. On the following section we will make an analysis of this problem, and we will define a solution.

4.3.2 Analysis of typical scenarios extended to infinite streams

We will continue the study of scenarios that we began on section 3.3. On this section we will see how our matching algorithm will work with unbounded streams.

Constant channel noise at the time of reading a symbol with a string rejection value

In the scenario 3.3.3 we have found a value T (the "Error Tolerance"). This value T is determining the statistical equilibrium between p_1 and p_2 events that cause us to consider the whole string as a matching or not. But what is more important, is that we will be able to determine this in a countable set of events, which makes the stated problem as decidable.

As we have point out on section 4.2, one of the challenges of the problem we want to address, is that we may not be able to know when to stop our algorithm at the time to apply it to some regular expressions. We can solve these situations with the variable found on the previous scenario. If in the first incoming symbols we detect that the proportion between "low informative elements" and "high informative elements elements" surpass the value T, we can reject that incoming symbols to be part of the previous string. This is due, there is a bigger probability that the first string matched as its own, rather than as a matching sub-string, inside a bigger string.

The definition of the number of new incoming symbols we will need to read (R) in order to decide if the string belong to a bigger string, will vary depending on the system. The only

restriction for this variable is that $R > T$. Otherwise, we will be loosing part of the information provided by the variable T. As we saw, the variable T tell us the number of p_2 events we can accept, for each p_1 events we have, and still have a higher information string (with a higher value of $\mathcal{L}$) that we had. With a lower value of R, what we are effectively doing is to reducing this number of "low symbols" we can read, and therefore, potentially loosing higher information strings.

Even if currently we have little knowledge of what value should R take, still it is easy to see that the most appropriate number of new incoming symbols that we should take, will be the one that maximizes the number of "reading errors" from which our model can recover, minimizing the number of incorrect matching. In order to see how the variation of the number of analyzed incoming symbols may affect the number of accepted strings, we are going to propose two practical examples. In the first example, we will study the minimum value R should take for our model to recover from the maximum number of reading errors, meanwhile in the second example, we will study the maximum value R should take to reduce the amount of false string matching. So from this examples we expect to get the optimum value range of new incoming symbols (R) we should read before rejecting a new string, when applying our model to scenarios with constant c_1 and c_2. In order to do so, we will take an scenario where we will be receiving just one symbol, but the scenario conditions generate a noise which will make us read two different symbol, with different probabilities. In this scenario we will also see a practical case of how a change on the conditions may affect the "reading noise" of the elements emitted (and therefore, we will see an example of defining conditions for c_1 and c_2)

Before proceeding with the examples, we are going to explain their background. The following examples will apply our model on a particular case of a classic physics experiment, the "Double-slit experiment", originally performed by Davisson and Germer in 1927 [25]. The objective of the Davisson and Germer experiment was to know the nature of the elemental particles, by studying their behavior. This experiment consists of making elementary particles go through a screen, one by one. In this screen there are two narrow slits that let the particle go trough it, in order to arrive to an "*observation screen*". In this observation screen, we will be able to see the pattern of arrival of the particles which have go thought the first screen. The experiment is illustrated on the Figure 4.1.

Even though, we are not particularly interested in the conclusions obtained in this experiment, one of its variants represents a suitable scenario for our model. One of these variants from the original experiment consist on having a detector (a mask) on each slit of the first screen, in order to know from which one the elemental particle go through. One of this variant outcomes is that with these conditions the pattern arrival of the particles to the observation screen is similar to the one we will get if we would had a single slit, but duplicated, each one aligned with the corresponding nearest slit [26]. The pattern obtained when having a single slit, will consist on the majority of particles arriving to an area centered on the closest point of the observation screen to the slit. The probability of finding then a collision on the observation screen, will diminish as we get farther in any direction from this area. Then, if we take any straight direction which goes through the closest point to the slit of the observation screen, we will have a normal distribution $\mathcal{N}(\mu\, 1)$ of collisions on the observable screen, where μ will be the this closest point. Note we have take 1 as the value of deviation of the normal distribution, without loss of generality, to simplify the calculations. This value may vary if the units of measurement or the distance between the screens were changed. The stated experiment variant has been illustrated on Figure 4.2

So if we take the line that goes thought the closest point to the slit A and the closest point to the slit B, we will have two symmetrical independent normal distributions of collisions, $\mathcal{N}_A$ aligned with slit A and $\mathcal{N}_B$ aligned with slit B. First of all, we have to determine for these normal distributions the values of μ_A and μ_B, which will be respectively the closest part of the

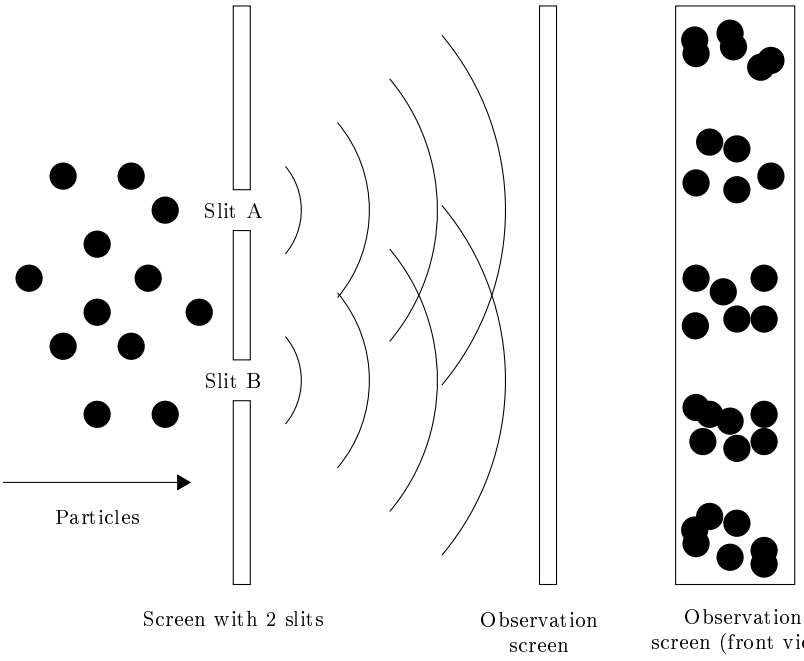


Figure 4.1: The particles go through a screen with 2 slits (slit A and slit B), and arrive to an observation screen were they display a disposition similar to the one expected from a wave, instead to the one expected from classic physics model

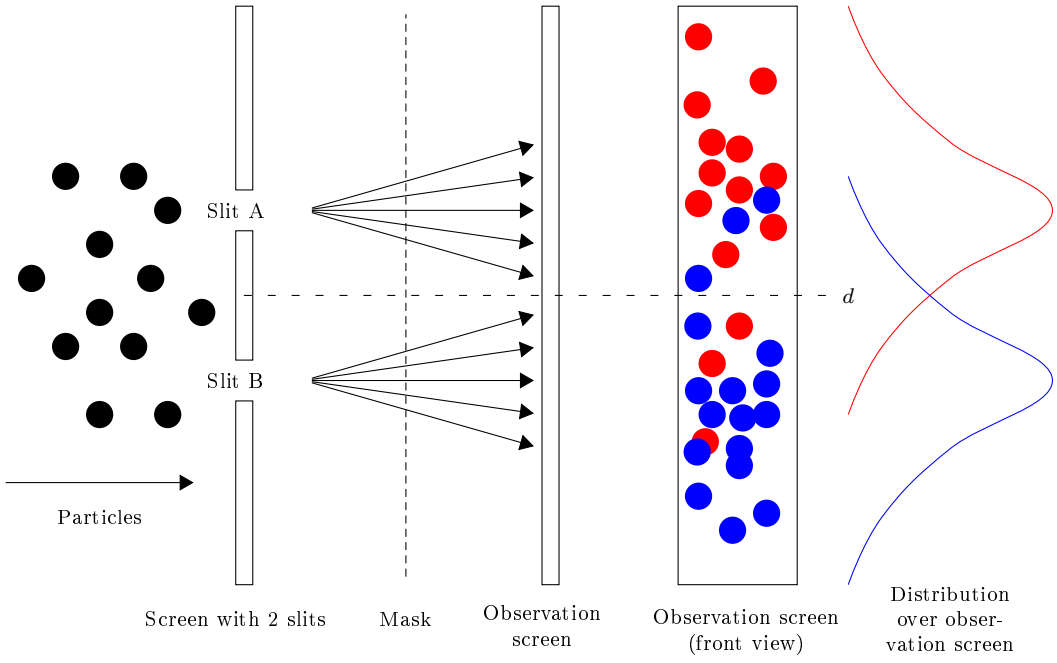


Figure 4.2: The particles go through a screen with 2 slits (slit A and slit B) and through a mask which is used as a detector, to determine from which slit elementary particles came from. Now particles arrive to an observation screen and display a classic physics model distribution. The particles which came from slit A (colored red) have a normal distribution $\mathcal{N}_A$ over observation screen, meanwhile particles which came from slit B(colored blue) have a normal distribution $\mathcal{N}_B$ over observation screen. Line d begins exactly in the middle of slit A and slit B and divide the observation screen in two parts

observation screen to slit A and to slit B. We will also divide the observation screen in two parts. In order to do so, we will determine a point d, equidistant to μ_A and μ_B. The upper part, S_A, will be the one where the particles which went trough slit A will collide with probability c_1 and the particles from slit B with probability c_2, meanwhile on the lower part, S_B, we will have the opposite situation. This probability c_1 will be $P_{\mathcal{N}_A}(z < d) = P_{\mathcal{N}_B}(z > d)$ and $c_2 = 1 - c_1$. It is easy to see on the Figure 4.2, that we may vary this channel noise if we modify the distance between the slits. We we get the slits closer, c_1 will diminish and c_2 will increase, and the opposite if we separate the screens. This definition also comes from common sense, if we have the slits very close to each other, it will be more difficult to distinguish from which one the element went through. Note that still, this problem definition has an uncomfortable restrain regarding c_1, which is that its value can't be less than 0.5, as we can see in Figure 4.2. This restriction could affect the generality of the stated case, but we will overcome this restrain with the redefinition of the problem we will do in the next paragraphs.

Example XIII:

Consider an alphabet $\Sigma = a, b$ where the emission of a symbols a corresponds to the event "*particle went trough slit A*" and that of b will refer to the event "*particle went trough slit B*". On this scenario we will work with the regular expression $\phi \equiv a^*$.

First we are going to try to avoid the stated restriction of $c_1 > 0.5$, to gain generality. To do so, we can adjust the expected frequency of the particles taking one or another slit, in order to be able to decrease c_1 under the value of 0.5. If we decrease the frequency of the particles to go through slit A, it will decrease the value of c_1. So, from now on, the value of c_1 could be different from the value of $P_{\mathcal{N}_A}(z < d)$.

Now, we will proceed to close slit B, so that all particles will go thought slit A, and we will be receiving only symbols a. This entails that the "*ground truth*" of our experiment will match the expression (for more complex expressions one, can think of a device that alternatively closes A and B, so the events that take place will match with ϕ). But, due to the uncertainty (in this case, the normal distribution of particle arrival on a divided "*observation screen*") we will read all symbols with different probabilities.

This scenario, as we have defined it, has $|\Sigma| = 2$, and therefore, using the notation of the previous examples $c_2 = 1 - c_1$, where c_1 is the probability of detecting the correct symbol, and c_2 the probability of selecting the wrong one. In order to gain flexibility in the definition of c_2, we introduce an new symbols associated with additional events.To do so, we will define events related to the event "*particles that went thought slit B*", as b_1, b_2, b_m, so that $\Sigma = \{a, b_1, b_2, ..., b_m\}$. Assuming that the probability of observing b_i, when the particle went through slit A is constant and equal to c_2. This means:

$$c_2 = \nu(b_i) = \frac{1 - c1}{|\Sigma| - 1} \tag{4.16}$$

In order to define these elements, we can think in different types of events, for example, the particle going between a range of speeds, when going through slit B (supposing all particles can have any range of speeds with same probability), or the charge of the particle (supposing we use any type of particle with same probability).

After we have set our scenario, we need to find a number of events (R) over which we will apply our defined threshold T on it. We should find the value of R, that will minimize the string rejected for this scenario, and we will have found a value of R which will minimize the number of correct string rejection. In order to be able able to define this minimum value of R,

we need to find out what considerations we should have to carry out this task. As we showed at the beginning of this section, it must be $R > T$, so for the sake of simplicity, and as first approximation, we will take $R = nT$ where $n \in \mathbb{N}$ and $n > 1$. The value of T obtained from the definition of the problem (after knowing values of c_1 and $|\Sigma|$) represents a maximum threshold of the number of additional p_2 events we can accept regarding the additional p_1 events we have already get. Therefore, we must find a value of $R = e + r$ where we will be maximizing the number e of p_2 events we can accept without rejecting the string, for a number r of p_1 events. In order to do so, these two values should meet $\frac{e}{r} < T$ for any T given. Remember that for this particular example e will be the number of symbols a read with probability c_2 and r will be the number of symbols a read with probability c_1.

Now, using the knowledge we have about R and T, we will be able to study this maximization of R. Considering $R = nT$ then we have $e = nT - r$, and we will have the following inequality:

$$T > \frac{e}{r} = \frac{(nT - r)}{r} = \frac{n}{r}T - 1 \tag{4.17}$$

This inequality is satisfied *iff* $r \geq \frac{nT}{T+1}$, so we must calculate the probability of $r \geq \frac{nT}{T+1}$ for a set of symbols of size $R = nT$. Since we know that the probabilities of having a p_1 event is $P(p_1) = P_{\mathcal{N}_A}(z < d)$, we will not just be able to do that, but we will also be evaluating how good is our estimation of R for this case. The bigger the probability of $r \geq \frac{nT}{T+1}$ the better our value of R will be, because that will mean that we will be reducing the probability of rejecting a string. It is easy to see that the probability of $r = i$ has a Binomial $\mathcal{B}(nT, P(p_1))$, and therefore, the probability of $r \geq n$ is a summation of this Binomial:

$$P(r \geq \lceil \frac{nT}{T+1} \rceil) = 1 - \sum_{i=0}^{\lceil \frac{nT}{T+1} \rceil} \binom{nT}{i} P(p_1)^i * (1 - P(p_1))^{nT-i} \tag{4.18}$$

Due to the difficulty of calculating this value, we will try to approximate this Binomial using a Normal distribution. We know $\mathcal{B}(n, p)$ behaves as a normal distribution $\mathcal{N}_T(np, \sqrt{np(1-p)})$ when certain requirements are met [27]. These requirements depend on the value of n, p and the value of some inequalities (3-standard-deviation, skewness...) which contains these variables, but a good generalization of this requirements is that when $n > 30$ and $0.15 < p < 0.85$ we can make the approximation of a Binomial distribution to a Normal Distribution. So for the shake of modeling, we will consider $nT > 30$ and $0.5 < P(p_1) < 0.85$, and we will calculate $P(r \geq n)$ with normal distribution $\mathcal{N}_T(n * T * P(p_1), \sqrt{n * T * P(p_1) * (1 - P(p_1))})$

$$P(r \geq \frac{nT}{T+1}) = 1 - P(r < \frac{nT}{T+1}) \approx 1 - P(z < \frac{\frac{nT}{T+1} - nTP(p_1)}{\sqrt{nTP(p_1)(1 - P(p_1))}}) \tag{4.19}$$

Note that due we are working now with a normal distribution, no rounding will be necessary $\frac{nT}{T+1}$. So, in order to understand how the value of n affects the probability, we should study the behavior of the expression contained in the probability:

$$\mathcal{X}(n) = \frac{\sqrt{nT}(1 - P(p_1)(T+1))}{(T+1)\sqrt{P(p_1)(1 - P(p_1))}} \tag{4.20}$$

In order to study its behavior, we should study its derivative ($\mathcal{X}'$) to know the growth of $\mathcal{X}$, and its limit when $n \to \infty$. Then, it will be easy to see that $\mathcal{X}$ is either continuous, monotonic increasing, or monotonic decreasing depending of the values of $(1 - P(p_1)(T+1))$ (here is

important to remember that the values of T and $P(p_1)$ are constant). So, from this point we have three different case of study, depending on the values of $P(p_1)(T+1)$:

1. $P(p_1)(T+1) > 1$, having $\mathcal{X}' < 0$, and therefore $\mathcal{X}$ is monotonic decreasing. Here when $n \to \infty$, $\mathcal{X} \to -\infty$
2. $P(p_1)(T+1) < 1$, having $\mathcal{X}' > 0$, and therefore $\mathcal{X}$ is monotonic increasing. Here when $n \to \infty$, $\mathcal{X} \to \infty$
3. $P(p_1)(T+1) = 1$, and $\mathcal{X} = 0$

The first case is the most favorable to solve the example's problem. On this scenario when $n \to \infty$ the probability $P(r \geq \frac{nT}{T+1}) = 1 - P(z < -\infty) = 1$. What this is telling us, next to the function being monotonic decreasing, is that the bigger the n we take, the lesser the chance of rejecting a string, what also means, the lesser the cases of false negatives we will get. Also, we will be able to measure how good our value of R is. We will be able to calculate the confidence of R, by calculating $P(z < \mathcal{X}(n))$, which should be relatively easy.

Meanwhile the second case is the less favorable to solve the example's problem. On this scenario when $n \to \infty$ the probability $P(r \geq \frac{nT}{T+1}) = 1 - P(z < \infty) = 0$. What this is telling us, next to the function being monotonic increasing, is that the bigger the n we take, the higher the chance of rejecting a string. Still this case will be useful for our next scenario, where we will try to be avoiding false positives.

And the last scenario will require from a further analysis. On this scenario we have a constant probability $P(r \geq \frac{nT}{T+1}) = 1 - P(z < 0) = 0.5$. What this is telling us, is that it is irrelevant the value of R, because the probability of $r \geq n$ will remain constant. Still, at this point is important to remember that we are just calculating an approximation of $\frac{P(r \geq nT}{T+1)}$. If we are dealing with this scenario, a further analysis of the binomial summation 4.18 will be required, in order to study if a particular case falls in scenario one or two, or in a combination of both. On this further analysis most relevant values to study will be the values were a round of $\frac{nT}{T+1}$ will be necessary.

The conclusion we get from this example, is that when we have a constant value of c_1 and $|\Sigma|$ and we know how "uncertain" is our read symbol (in the previous example it was the value $P(p_1)$), we are able to calculate a threshold T which indicates us a proportion of p_2 events we can admit for each p_1 events on a read sub-string of R elements, in order to decide if we admit this sub-string as valid or not. This variable T, under certain conditions ($R > 30$, $0.5 < P(p_1) < 0.85$, $P(p_1)(T+1) > 1$) will be an useful tool to reject possible non-matching strings, reducing the amount of false negatives as much as we want. But it this example, we had a scenario where our variable didn't reduce the amount of false negatives, but increased them (when $P(p_1)(T+1) < 1$). In the following example, we will study deeply this scenario, and try to find if our variable T can be useful on it.

Example XIV:

Let's consider a scenario with similar considerations as the previous example, but reversed. Meanwhile we will be working with same considerations over adding events b_i and change the frequency of slits to get different c_1, to keep generality, this time we will be closing slit A, but still we will be working with regular expression $\phi \equiv a^*$. Therefore, in this scenario the probability of p_1 event will be $P(p_1) = P_{\mathcal{N}_B}(z > d)$. So, on this scenario we will continue studying how our model will recover from reading errors, but on this time, we will try to find a value of R which

will minimize the incorrect string acceptance. Now we will try to find a value of $R = e + r = nT$ such as $\frac{e}{r} > T$ for any T given, and we will have the following inequality:

$$T < \frac{e}{r} = \frac{(nT - r)}{r} = \frac{n}{r}T - 1 \tag{4.21}$$

This inequality only is satisfied *iff* $\frac{r \leq nT}{T+1}$, so now we must calculate the probability of $r < n$ for a set of symbols of size $R = nT$. Since we know the probabilities of having a p_1 event ($P(p_1)$), we will not only just be able to do that, but also, we will be evaluating how good is our estimation of R for this case. The bigger the probability of $\frac{r \leq nT}{T+1}$ the better our value of R will be, because that will mean that we will be reducing the probability of accepting an invalid string. It is easy to see that the probability of $r = i$ is a Binomial with parameters $\mathcal{B}(nT, P(p_1))$, and therefore, the probability of $r < n$ is a summation of this Binomial:

$$P(r < \lceil \frac{nT}{T+1} \rceil) = \sum_{i=0}^{\lceil \frac{nT}{T+1} \rceil - 1} \binom{nT}{i} P(p_1)^i * (1 - P(p_1))^{nT-i} \tag{4.22}$$

With similar conditions as previous example ($R > 30$, $0.15 < P(p_1) < 0.5$), and with the same purpose of reducing the complexity of this calculation, we will approximate this binomial probability to a normal distribution $\mathcal{N}_T(n * T * P(p_1), \sqrt{n * T * P(p_1) * (1 - P(p_1))})$. Now we will be able to calculate $\frac{P(r<nT}{T+1)}$ in an easier way:

$$P(r < \frac{nT}{T+1}) \approx P(z < \frac{\sqrt{nT}(1 - P(p_1)(T+1))}{(T+1)\sqrt{P(p_1)(1 - P(p_1))}}) = P(r < \mathcal{X}(n)) \tag{4.23}$$

Note that due we are working now with a normal distribution, no rounding will be necessary $\frac{nT}{T+1}$. The analysis of $\mathcal{X}$ and the obtained scenarios are exactly the same as the previously found out, but now the conclusions over the first and the second scenario have changed (the conclusion for the third scenario keep being the same). Now the first scenario is the less favorable to solve this new problem. On this scenario when $n \to \infty$ the probability $P(r < \frac{nT}{T+1}) = P(z < -\infty) = 0$. What this is telling us, next to the function being monotonic decreasing is that we are loosing confidence at the time of taking false positives as we increase the value of R.

Now the most favorable scenario is the second one, due to when $n \to \infty$ the probability $P(r < \frac{nT}{T+1}) = P(z < \infty) = 1$ and the function is being monotonic increasing. Now we are lessening the chances of accepting a string, and therefore, lessening the cases of false positives. Also, we will be able to measure how good our value of R is. We will be able to calculate the confidence of R, by calculating $P(z < \mathcal{X}(n))$, which should be relatively easy.

The conclusion we get from this example, is that when we have a constant value of c_1 and $|\Sigma|$ and we know how the noise affects our received symbol (in the previous example it was the value $P(p_1)$), we are able to calculate a threshold T which indicates us a proportion of p_2 events we can admit for each p_1 events on a read sub-string of R elements, in order to decide if we admit this sub-string as valid or not. This variable T, under certain conditions ($R > 30$, $0.15 < P(p_1) < 0.5$, $P(p_1)(T + 1) < 1$) will be an useful tool to reject possible non-matching strings, reducing the amount of false positives as much as we want.

5

Conclusions and future work

Regular expressions are powerful and well understood tool for matching finite strings of symbols drawn from a finite alphabet. In this work we have presented an extension of regular expression, to the case where the identity of the symbols we are receiving is uncertain. For this extension we have defined a model where the symbols emitted are affected by channel noise, so for each symbol a_i, we will observe a probability distribution $\nu_i : \Sigma \to [0, 1]$, where $\nu(a^{(n)})$ gives the probability that the i^{th} symbol that was sent to us is $a^{(n)}$. For this model we have analyses two different input types: finite and infinite strings.

In the case of finite strings we presented an algorithm based on minimizing the residual information, or in other words, an algorithm which ensures us the obtained matching string is the most informative of all possible matching strings. Then we studied the behavior of this algorithm on different scenarios, with different types of channel noises. On these scenarios we have consider that the probability distribution $\nu(a^{(n)})$ takes value c_1 for $a^{(k)}$, and c_2 for $a^{(i)}$ when $i \neq k$. We have show that when c_1 is not bigger than c_2, we can't have enough information to make any assumptions about the emitted symbols, and therefore from this finding, we have only considered $c_1 > c_2$ in the rest of the work. The analysis of these scenarios have taken us to define a variable T (the Error Tolerance), which help us to discard potential matching strings which do not contain enough information about the emitted symbols. This variable has been very helpful to define the model's behavior when having as input an infinite list.

For the case of infinite lists, we have shown that the problem is generally undecidable, but we have defined the necessary conditions such that problem is decidable with $P = 1$. We have found out that under the consideration of having a detector good enough to associate the majority (not all) of correct emitted symbols to a higher probability, and with static probabilities (c_1 and c_2 constant) our model has show up good levels of error recovery when taking enough amount of symbols to consider before making a match ($R > 30$), while having the following problem conditions:

- When we are having a high chance of associating the correct symbol to a high probability (having $P(p_1)$ between 0.5 and 0.85), and this probability satisfy $P(p_1)(T+1) > 1$, we can reduce the probability of false negatives as much as we want (when $n \to \infty$ the probability $P(r \geq \frac{nT}{T+1}) = 1$)
- When we are having a low chance of associating an incorrect symbol to a high probability

(having $P(p_1)$ between 0.15 and 0.5), and this probability satisfy $P(p_1)(T+1) < 1$, we can reduce the probability of false positives as much as we want (when $n \to \infty$ the probability $P(r < \frac{nT}{T+1}) = 1$)

Is important to point out that, even thought both scenarios can take place at the same time, they don't necessarily do so. Some of these conclusions have been presented in [1]

We leave the analysis of additional scenarios for future work, as the objective of this work has been to obtain some preliminary result about conditions were our model could be applied. Interesting scenarios were model could be studied is when having variable values of c_1 and c_2, and probabilistic studies will be needed to define the value of T, due its value won't keep static.

These results have shown that our model can be an interesting alternative to apply over situations were regular expressions are used over an unbounded uncertain stream of data, in order to improve the accuracy respect the classical model of regular expressions, even thought additional studies would be needed. We believe these results will have implications for tool designers and for future research aimed to improve and support the use of regular expressions.

Glossary

- **Unbounded stream**: An infinite sequence of symbols
- **Decidable**: The expression ϕ is decidable if there is an algorithm such that it takes as input an instance of the problem and determines whether the answer to that instance is yes no
- **Limited expression**: The expression ϕ is limited if all the words from the language defined by ϕ are shorter than $N < \infty$
- **Unrealizable**: The expression ϕ is unrealizable if it might require the computation of a non-recursive function, or it might be logically inconsistent
- **Elementary particle**: An elementary particle in physics is the one which is not composed by any other particle. These particles do not obey the laws of classic physics mechanics, but the laws of quantum mechanics

Bibliography

[1] Arturo Gil and Simone Santini. Querying uncertain data using regular expressions. *VLDB Journal (submitted)*, 2018.

[2] Ullman Hopcroft. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[3] Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

[4] Simone Santini. Querying streams using regular expressions: some semantics, decidability, and efficiency issues. *The VLDB Journal*, 24(6):801–821, 2015.

[5] Shailendra Bhonsle, Amarnath Gupta, Simone Santini, Marcel Worring, and Ramesh Jain. Complex visual activity recognition using a temporally ordered database. In *International Conference on Advances in Visual Information Systems*, pages 722–729. Springer, 1999.

[6] Simone Santini. Regular expression matching on finite posets. *Journal of Computer and System Sciences (in press)*, 2018.

[7] Eric R Van Wyk and August C Schwerdfeger. Context-aware scanning for parsing extensible languages. In *Proceedings of the 6th international conference on Generative programming and component engineering*, pages 63–72. ACM, 2007.

[8] Aggeliki Vlachostergiou, George Marandianos, and Stefanos Kollias. Context incorporation using context-aware language features. In *Signal Processing Conference (EUSIPCO), 2017 25th European*, pages 568–572. IEEE, 2017.

[9] Markus Walther. Context-aware linear time tokenizer, August 7 2003. US Patent App. 10/071,934.

[10] Matthias Anlauff. X asm-an extensible, component-based abstract state machines language. In *Abstract State Machines-Theory and Applications*, pages 69–90. Springer, 2000.

[11] Angel X Chang and Christopher D Manning. Tokensregex: Defining cascaded regular expressions over tokens. *Tech. Rep. CSTR 2014-02*, 2014.

[12] Hideyuki Kawashima, Hiroyuki Kitagawa, and Xin Li. Complex event processing over uncertain data streams. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2010 International Conference on*, pages 521–526. IEEE, 2010.

[13] YH Wang, Kening Cao, and XM Zhang. Complex event processing over distributed probabilistic event streams. *Computers & Mathematics with Applications*, 66(10):1808–1821, 2013.

[14] Ronald V Book. *Formal language theory: perspectives and open problems*. Academic Press, 2014.

[15] Jan Goyvaerts and Steven Levithan. *Regular expressions cookbook*. O'reilly, 2012.

[16] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2012.

[17] J Richard Büchi. On a decision method in restricted second order arithmetic. In *The Collected Works of J. Richard Büchi*, pages 425–435. Springer, 1990.

[18] Alexander Rabinovich. Automata over continuous time. *Theoretical Computer Science*, 300(1-3):331–363, 2003.

[19] Christian Eisentraut, Holger Hermanns, and Lijun Zhang. On probabilistic automata in continuous time. In *Logic in Computer Science (LICS), 2010 25th Annual IEEE Symposium on*, pages 342–351. IEEE, 2010.

[20] Daniel T Gillespie. A rigorous derivation of the chemical master equation. *Physica A: Statistical Mechanics and its Applications*, 188(1-3):404–425, 1992.

[21] Robert Schlaifer. Probability and statistics for business decisions. pages 8–11, 1959.

[22] Ian H Witten and Timothy C Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *Ieee transactions on information theory*, 37(4):1085–1094, 1991.

[23] Donald E Knuth, James H Morris, Jr, and Vaughan R Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

[24] Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In *International Colloquium on Automata, Languages, and Programming*, pages 1–17. Springer, 1989.

[25] Clinton Davisson and Lester H Germer. Diffraction of electrons by a crystal of nickel. *Physical review*, 30(6):705, 1927.

[26] Roger Bach, Damian Pope, Sy-Hwang Liou, and Herman Batelaan. Controlled double-slit electron diffraction. *New Journal of Physics*, 15(3):033018, 2013.

[27] George EP Box, William Gordon Hunter, J Stuart Hunter, et al. Statistics for experimenters. 1978.