SAT based learning Algorithms

Author:
Rajarshi Roy

Supervisor:
Dr. Daniel Neider

A thesis submitted for the degree of

Masters degree in Computer Science
at Chennai Mathematical Institute

June 15, 2019

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

CHENNAI
MATHEMATICAL
INSTITUTE

Abstract

In this thesis, we present algorithms for learning regular-expressions, omega-regular expressions, LTL fomulas and PSL formulas, from a given sample of positive and negative examples. The precise learning problem studied in this thesis—is to find a model consistent with a given sample, that is, a model that accepts the positive examples and rejects the negative examples in the sample. SAT-solver based algorithms have been efficient in solving these problems in certain models such as Linear Temporal Logic (LTL).

Property Specification Language (PSL) is an extension of LTL, with increased expressive power. Now, PSL relies heavily on regular expressions. Hence, it is necessary to look at the learning problem for regular expressions and $\omega-$regular expressions for solving the learning problem of PSL. While it is straightforward to apply the existing techniques for solving the problem for regular expressions, it is challenging to translate the same ideas to $\omega-$regular expression. This is because the sample given in the case of ω-regular expressions contains infinite words. The same can be said about PSL formulas as well.

Acknowledgements

I would like to thank my supervisor, Daniel Neider, for presenting me with such a wonderful problem for my master thesis and being a great mentor while I worked on it. I would also, express my gratitude to my professors at CMI for teaching me courses, which made me competent enough to tackle a problem of this kind. Last but not the least, I would like to thank all my colleagues and friends at MPI and CMI for their constant assistance in helping me complete my thesis.

Contents

Chapter 1

Introduction

Constructing a model of system behavior from observation traces, in a form that is understandable and meaningful to a human, is central to human interpretability of complex systems. Naturally, ability to learn human-interpretable models is quite crucial in order to understand general behaviour of a system. But, various scenarios require us to come up with different models that suitably represent it. To this end, we have presented learning techniques for a variety of models, which could find applications in diverse spheres.

The techniques used in this thesis have been motivated by ideas from the work of Neider and Gavran[6], where they present novel algorithms for learning formulas in Linear Temporal Logic (LTL)[7]. LTL has a natural syntax useful in specifying properties of programs for verification. But, there are certain program behaviour which cannot be characterized by LTL formulas. For instance, the property "Event e_1 happens after even number of steps of e_2" cannot be expressed in LTL[8]. This led us to look at Property Specification Language (PSL)[4], which extends the expressive power of LTL and captures a wider range of program behaviour. PSL is the IEEE standard for temporal logic (IEEE). In terms of expressiveness, PSL is as powerful as $\omega-$regular expressions while LTL can express only star-free $\omega-$regular expressions. Consequently, formulating learning algorithms for PSL formulas is the natural next step.

The increased expressive power of PSL is due to the incorporation of regular expressions in its syntax. Thus, in an attempt to solve the aforementioned problem, it is necessary to understand how the same problem could be solved when the model being learnt is a regular expression. Moreover, since PSL formula argue about infinite strings, it is also useful to devise algorithms for learning $\omega-$regular expression as well.

The main goal of this thesis is to devise algorithms for learning high-level descriptions of system behaviour provided in the form of a sample consisting of some classified examples. Moreover, the description needs to be succints, it is not quite chalenging to construct arbitrary large models which adhere to the given examples. To be precise, given a sample $\mathcal{S}$ consisting of two finite sets of positive and negative examples, we want to learn the 'smallest' model $\mathcal{M}$ that is consistent with $\mathcal{S}$—in the sense that all positive examples satisfy $\mathcal{M}$, whereas all negative examples violate $\mathcal{M}$. The model being learnt could be—a regular expression (Chapter 3), $\omega-$regular expressions (Chapter 4), LTL fomulas (Chapter 5) or PSL formulas (Chapter 6).

The general strategy that has been employed for solving the learning problems is encoding the problem in a series of satisfiability formulas and then using highly optimized SAT-solver to search for a solution.

1.1 Possible Applications

Most of the learning problems for various models emerge out of some software verification problem. But, by extending the ideas to solve learning problems for regular expressions and $\omega-$regular expressions, there could be potential applications in diverse fields. This is because patterns and sequences are abundant in nature and identifying them could prove to be quite crucial. Here, we present some possible applications.

- *Bug Detection:* From a buggy program, it is possible to procure the data of desirable and undesirable program behaviour. In order to have a deeper understanding of the root of the bug, it could be helpful to apply the learning algorithms using the data of the program traces as the sample. High level description of the bug, could benefit bug-detection. Since, there exists learning algorithms for several models, the model which captures the bug more appropriately, can be learned.

- *Classifying biological sequences*: Consider the following table which represents biological sequences that have been associated with a group of diseases called amyloidosis.

String	Class
stviil	positive
ktvive	negative
stviie	positive
st piie	negative

  Now, from this data it can be deduced that any sequence satisfying the regular expression $\Sigma^*(stv)\Sigma^*$ could potentially be harmful. Hence, identifying such a pattern could be beneficial in this case. This problem has been discussed in [1].

Chapter 2

Preliminaries

Words and Languages An alphabet Σ is a nonempty, finite set of symbols. A *finite word* $u = a_0 \cdots a_{n-1}$ is a finite sequence of symbols $a_i \in \Sigma$ for $i \in \{1, \ldots, n\}$. The empty sequence is called the empty word and is denoted by ε. The length of a word u, denoted by $|u|$, is the number of symbols in u. For two words $u = a_0 \cdots a_{m-1}$ and $v = b_0 \cdots b_{n-1}$, the concatenation of u and v is the word $u \circ v = uv = a_0 \cdots a_{m-1} b_1 \cdots b_{n-1}$. The set of all finite words over an alphabet Σ is denoted by Σ^*. Let $w[i, j)$ refer to the subword of w starting at position i and ending at position $j - 1$. Also, $w[i, i) = \varepsilon$ in this definition. Assume, here that the indexing of word starts from position 0.

An *infinite word* $\alpha = a_0 a_1 \ldots$ is an infinite sequence of symbols $a_i \in \Sigma$ for each $i \geq 1$. Given a word $v \in \Sigma^+$, the infinite repetition of v is the infinite word $v^\omega = vv \ldots \in \Sigma^\omega$. We say that a word $\alpha \in \Sigma^\omega$ is *ultimately periodic* if it can be written as uv^ω with $u \in \Sigma^*$ and $v \in \Sigma^+$. Moreover, let $\alpha[i, j) = a_i a_{i+1} \cdots a_{j-1}$ be the finite infix of the infinite word $\alpha = a_0 a_1 \cdots \in \Sigma^\omega$. Similarly, let $\alpha[i, \infty)$ be the infinite suffix $a_i a_{i+1} \ldots \in \Sigma^\omega$.

A subset $L \subseteq \Sigma^\omega$ of infinite words is called a $\omega-$language. Concatenation can be also be extended to account for $\omega-$languages as well and has been referred to as $\circ_\omega$ in this thesis. But, $L \circ_\omega L'$ is well-defined only when L is a language of finite words and L' is a $\omega-$language. Also, the $\omega-$iteration of a language $L \subseteq \Sigma^*$ is the $\omega-$language $L^\omega = \{w_1 w_2 w_3 \cdots \mid w_i \in L \backslash \{\epsilon\}\}$.

Deterministic Finite Automata A *Deterministic Finite Automaton* (DFA) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, where Σ is a finite alphabet, Q is a non-empty, finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to Q$ is the transition function, and $F \subseteq Q$ is the set of final (or accepting) states. We can extend δ to a function $\delta : Q \times \Sigma^* \to Q$ by $\delta(q, \varepsilon) = q$ and $\delta(q, wa) = \delta(\delta(q, w), a)$ for all $a \in \Sigma^*$ and $w \in \Sigma^*$. The language accepted by $\mathcal{A}$ is the set $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$. The run of a DFA on a word $w = a_0 \cdots a_n$ is defined as a sequence of states of the DFA q_0, q_1, $\cdots$, q_n, $q_n + 1$, such that $\delta(q_i, a_1) = q_{i+1}$ for all $i = 1 \cdots n$. A word w is accepted if the run of the DFA on w terminates in a final state. A language L is regular iff there exists a DFA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = L$

Propositional Boolean Logic: Let AP be a set of propositional variables, which take Boolean values from $\mathbb{B} = \{0, 1\}$ (0 representing false and 1 representing true). Formulas in propositional (Boolean) logic have the following grammar:

$$\varphi ::= \mathfrak{p} \mid \neg\varphi \mid \varphi \vee \varphi$$

, where $\mathfrak{p} \in$ AP Moreover, we add syntactic sugar and allow the formulas true, false, $\varphi \wedge \psi$, $\varphi \implies$

ψ, $\varphi \iff \psi$. A propositional valuation is a mapping $v : \mathsf{AP} \to \mathbb{B}$, which maps propositional variables to Boolean values.

Semantics: The semantics of propositional logic is given by a satisfaction relation $\vDash$ that is inductively defined as follows:

$$v \vDash \mathfrak{p} \text{ iff } v(\mathfrak{p}) = 1; v \vDash \neg\varphi \text{ iff } v \nvDash \varphi; v \vDash \varphi \vee \psi \text{ iff } v \vDash \varphi \text{ or } v \vDash \psi$$

In the case that $v \vDash \varphi$, we say that v satisfies φ and call it a model of φ. A propositional formula φ is satisfiable if there exists a model v of φ. The size of a formula is the number of its subformulas (defined in the usual way).

Chapter 3

Learning Regular Expressions

3.1 Regular Expressions

Regular expression consist of a finite alphabet Σ along with three operators $+$ (union), $\circ$ (concatenation) $*$ (Kleene star) with the following grammar

$$r ::= \varepsilon \mid a \in \Sigma \mid r + r \mid r \circ r \mid r^*$$

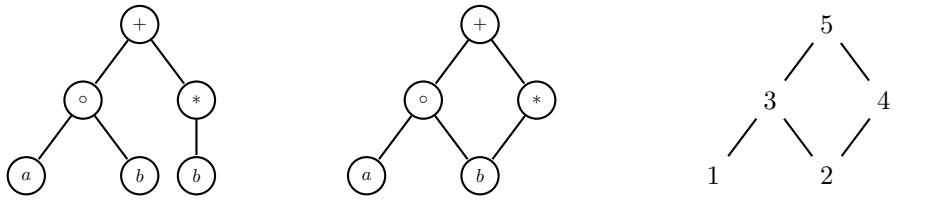
The set of all regular expressions over alphabet set Σ is referred to as $\mathcal{R}_\Sigma$. With slight abuse of notation, another commonly used expression is r^+ which refers to the expression $r \circ r^*$.

A regular expression can also be represented as a *syntax tree* where the nodes are labelled by elements from Λ_R, where $\Lambda_R = \Sigma \cup \{\varepsilon, +, \circ, *\}$-a combination of alphabets and operators. Leaf nodes are labelled by only alphabets or ε whereas the internal nodes are labelled by operator with the operands being their children.

A *syntax DAG* can also be created by merging the common subtrees in different branches of a syntax tree. A syntax DAG essentially eliminates redundant subexpressions from a syntax tree.

The *size* of a regular expression is the number of unique subexpression present. It coincides with the number of nodes in the syntax DAG representation of the expression.

The nodes of syntax DAG with n nodes can be indexed by natural numbers $1, \ldots, n$. The indexing that we follow in this thesis, satisfies the property that children of a node are always indexed by numbers smaller than the node itself and only a leaf node can have the index 1. Note that the indexing need not be unique. Fig. 3.1 shows an example of syntax tree and a syntax DAG for the regular expression $(a \circ b) + b^*$.



(a) Syntax tree (b) Syntax DAG (c) Indexing of syntax DAG

Figure 3.1: Syntax tree, syntax DAG with indexing of node, of the regular expression $(a \circ b) + b^*$

Semantics The semantics of regular expression is generally defined in terms of the language they define:

$$[\![\epsilon]\!] = \{\epsilon\};\ [\![a]\!] = \{a\};\ [\![r_1 + r_2]\!] = [\![r_1]\!] \cup [\![r_2]\!];\ [\![r_1 \circ r_2]\!] = [\![r_1]\!] \circ [\![r_2]\!];\ [\![r^*]\!] = [\![r]\!]^*$$

Definition 1. *Matching relation* $\vdash \subseteq \Sigma^* \times \mathcal{R}_\Sigma$ *is defined inductively on the structure of a regular expression* r *in the following manner:*

$$w[i,j) \vdash \epsilon \iff i = j$$
$$w[i,j) \vdash a \text{ where } a \in \Sigma \iff w[i,j) = a$$
$$w[i,j) \vdash r_1 + r_2 \iff w[i,j) \vdash r_1 \text{ or } w[i,j) \vdash r_2$$
$$w[i,j) \vdash r_1 \circ r_2 \iff \exists k \in \mathbb{N} \text{ such that } i \leq k \leq j \text{ and } w[i,k) \vdash r_1 \text{ and } w[k+1,j) \vdash r_2$$
$$w[i,j) \vdash r^* \iff \begin{cases} i = j \text{ or} \\ \exists k \in \mathbb{N} \text{ such that } i < k \leq j \text{ and } w[i,k) \vdash r \text{ and } w[k,j) \vdash r^* \end{cases}$$

The matching relation $\vdash$ holds true when a word belongs to the language defined by the regular expression. More precisely, we have that $w[i,j) \vdash r \iff w[i,j) \in [\![r]\!]$. This statement can be proved using a simple induction on the structure of r.

Motivated by definition of the matching relation, it is possible to design a *dynamic programming* algorithm for checking whether a given word w matches a regular expression r. Let T_w be a three dimensional table with $|w| + 1$ number of rows and columns both indexed using $0 \cdots n$ and the third index is used to denote the subexpression of the expression r for which we are computing the matching relation. The entries of the table have the following meaning.

$$T_w(i,j,r) = 1 \iff w[i,j) \vdash r$$

Given, a regular expression r, the table T_w is constructed in a iterative manner starting from the portion of the table which refers to the leaf elements of the syntax DAG. These table entries are used for propagating the results to table entries for the internal nodes.

- For the regular expression ε: $T_w(i,j,\varepsilon) = 1$ iff $i = j$

- For a leaf node say a, $T_w(i,j,a) = 1$ iff $w[i,j) = a$

- For $+$ (union) node, with children r_1 and r_2:
  $T_w(i,j,r_1 + r_2) = 1$ if $T_w(i,j,r_1) = 1$ or $T_w(i,j,r_2) = 1$

- For $\circ$ (concat) node, with children r_1 and r_2:
  $T_w(i,j,r_1 \circ r_2) = 1$ iff $\exists k, i \leq k \leq j$ such that $T_w(i,k,r_1) = 1$ and $T_w(k,j,r_2) = 1$

- For $*$ (Kleene star) node, with a child r:
  $T_w(i,j,r^*) = 1$ iff $i = j$ or $\exists k, i \leq k \leq j$ such that $T_w(i,k,r) = 1$ and $T_w(k,j,r^*) = 1$
  In this case, T_w is filled starting from $(n,n)-$th entry up along the columns.

3.2 The Learning Problem

We assume that the data to learn from is given as a pair $\mathcal{S} = (\mathcal{P}, \mathcal{N})$ consisting of two finite, disjoint sets $(\mathcal{P}, \mathcal{N}) \subset \Sigma^*$ of finite words such that $(\mathcal{P} \cap \mathcal{N}) \neq \emptyset$. We call this pair a *sample*. Moreover, we say that a regular expression r over Σ is *consistent* with a sample $\mathcal{S} = (\mathcal{P}, \mathcal{N})$ if

1. $r \vdash u$ for each $u \in \mathcal{P}$; and

2. $r \nvdash u$ for each $u \in \mathcal{N}$.

SAT solver based solution: A possible solution could be to use SAT-solvers to find the regular expression consistent with the given sample. The formula $\Phi_n^{\mathcal{S}}$ that we feed into the solver has the following structure:

$$\Phi_n^{\mathcal{S}} = \Phi_n^{\text{structure}} \wedge \Phi_n^{\text{consistency}} \tag{3.1}$$

Here, satisfying assignment of $\Phi_n^{\text{structure}}$ provides encoding of a syntax DAG which describes expression of size n, while the $\Phi_n^{\text{consistency}}$ checks whether the guessed expression is consistent with the given sample. $\Phi_n^{\text{structure}}$ uses the following variables:

- $x_{p,\lambda}$ where $p \in \{1, \cdots, n\}$ and $\lambda \in \Lambda_R$
- $l_{p,q}$ where $p \in \{2, \cdots, n\}$ and $q \in \{1, \cdots, p-1\}$
- $r_{p,q}$ where $p \in \{2, \cdots, n\}$ and $q \in \{1, \cdots, p-1\}$

Now, this is how the formula looks like. It is formed by taking conjunction of the following subformulas.

$$[\bigwedge_{1\leq p\leq n} \bigvee_{\lambda\in\Lambda_R} x_{p,\lambda}] \wedge [\bigwedge_{1\leq p\leq n} \bigwedge_{\lambda\neq\lambda'\in\Lambda_R} \neg x_{p,\lambda} \vee \neg x_{p,\lambda'}] \tag{3.2}$$

$$[\bigwedge_{2\leq p\leq n} \bigvee_{1\leq q\leq p} l_{p,q}] \wedge [\bigwedge_{2\leq p\leq n} \bigwedge_{1\leq q\leq q'\leq n} \neg l_{p,q} \vee \neg l_{p,q'}] \tag{3.3}$$

$$[\bigwedge_{2\leq p\leq n} \bigvee_{1\leq q\leq p} r_{p,q}] \wedge [\bigwedge_{2\leq p\leq n} \bigwedge_{1\leq q\leq q'\leq n} \neg r_{p,q} \vee \neg r_{p,q'}] \tag{3.4}$$

$$x_{1,\epsilon} \vee \bigvee_{a\in\Sigma} x_{1,a} \tag{3.5}$$

Subformula (3.2) says that each node would be uniquely labelled by an element from Λ_R. Subformulas (3.3) and (3.4) say that each node would have a unique left and right child respectively. Last but not the least, Subformula (3.5) says that the first node can have only ϵ or an alphabet.

Now, it needs to be ensured that the regular expression guessed by the first part $\Phi_n^{\text{structure}}$ is consistent with the given sample. For each word w in the given sample, consider a three dimensional table whose entries are basically variables $y_{i,j,p}^w$, where the indices i and j refers to the substring of the word w and p refers to the sub-expression we are looking at. For simplicity, we assume $m = |w|$. Now, Φ_n^w is a conjunction of the following subformulas:

$$\bigwedge_{1\leq p\leq n} x_{p,\varepsilon} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq m} y^w_{i,j,p} \leftrightarrow [i=j] \Big] \tag{3.6}$$

$$\bigwedge_{1\leq p\leq n} \bigwedge_{a\in\Sigma} x_{p,a} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq m} \begin{cases} y^w_{i,j,p} \text{ if } w[i,j)=a \\ \neg y^w_{i,j,p} \text{ if } w[i,j)\neq a \end{cases} \Big] \tag{3.7}$$

$$\bigwedge_{\substack{1\leq p\leq n \\ 1\leq q,q'\leq p}} x_{p,+} \wedge l_{p,q} \wedge r_{p,q'} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq m} \Big[y^w_{i,j,p} \leftrightarrow y^w_{i,j,q} \vee y^w_{i,j,q'} \Big] \Big] \tag{3.8}$$

$$\bigwedge_{\substack{1\leq p\leq n \\ 1\leq q,q'\leq p}} x_{p,\circ} \wedge l_{p,q} \wedge r_{p,q'} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq m} \Big[y^w_{i,j,p} \leftrightarrow \bigvee_{i\leq k\leq j} y^w_{i,k,q} \wedge y^w_{k,j,q'} \Big] \Big] \tag{3.9}$$

$$\bigwedge_{\substack{1\leq p\leq n \\ 1\leq q,q'\leq p}} x_{p,*} \wedge l_{p,q} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq m} \Big[y^w_{i,j,p} \leftrightarrow [i=j] \vee \bigvee_{i<k\leq j} y^w_{i,k,q} \wedge y^w_{k,j,p} \Big] \Big] \tag{3.10}$$

The idea for each of these subformulas is motivated by the procedure for filling out the table T_w, which in turn essentially uses the way the matching relation has been defined. Using the formula Φ^w_n for various w, we construct $\Phi^{\text{consistency}}_n$ in the following manner. Notice, that variable $y^w_{0,m,n}$ encodes whether word w matches the guessed regular expression. Hence, $y^w_{0,m,n}$ should be true for positive words and should be false for negative words.

$$\Phi^{\text{consistency}}_n = \Big[\bigwedge_{w\in\mathcal{P}} \Phi^w_n \wedge y^w_{0,m,n} \Big] \wedge \Big[\bigwedge_{w\in\mathcal{N}} \Phi^w_n \wedge \neg y^w_{0,m,n} \Big] \tag{3.11}$$

Given that we have devised SAT formula for this problem, we utilise it to come up with a simple algorithm using SAT-solvers to find out the smallest regular expression consistent with the sample.

Algorithm 1: SAT-based learning algorithm for regular expressions

Input: A sample $\mathcal{S}$

```
n ← 0
repeat
    n ← n + 1
    Construct and solve the formula Φ^S_n using a SAT solver
until Φ^S_n is satisfiable, say with model v
R^v ← Extract(Φ^S_n, v)
return the regular expression R^v
```

In the above algorithm, if we can find a model v of $\Phi^{\mathcal{S}}_n$, we can extract a regular expression R^v using the procedure $\mathit{Extract}(\Phi^n_{\mathcal{S}}, v)$. The first part $\Phi^{\text{structure}}_n$ of the formula $\Phi^n_{\mathcal{S}}$ encodes the structure of the regular expression as a syntax DAG. The model v assigns valuations to the variables $x_{p,\lambda}$, $l_{p,q}$ and $r_{p,q}$ which determines the shape of the syntax DAG. Hence, the procedure $\mathit{Extract}(\Phi^n_{\mathcal{S}}, v)$ uses this information to derive the expression R^v from the syntax tree.

3.2.1 Proof of correctness

Claim 1. *Let $\mathcal{S} = (\mathcal{P}, \mathcal{N})$ be a sample, $n \in \mathbb{N} \setminus \{0\}$, and $\Phi^{\mathcal{S}}_n$ be the propositional formula defined above. Then, the following holds:*

1. *If there exists a regular expression of size n, $R^{\mathcal{S}}$ that is consistent with $\mathcal{S}$, then the propositional formula $\Phi_n^{\mathcal{S}}$ is satisfiable.*

2. *If $v \models \Phi_n^{\mathcal{S}}$, then R^v is a regular expression of size n that is consistent with $\mathcal{S}$.*

Proof. Using the syntax DAG of the expression $R^{\mathcal{S}}$ which is indexed using $1 \cdots n$, we formulate a valuation v for the propositional variables in $\Phi_n^{\mathcal{S}}$. We use $R_p^{\mathcal{S}}$ to refer to the regular expression rooted at the p^{th} node.

- We set $v(x_{p,\lambda}) = 1$ iff the p^{th} node is labelled by λ.
- We set $v(l_{p,q}) = 1$ iff q^{th} node is the left child of the p^{th} node and similarly, $v(r_{p,q}) = 1$ iff q^{th} node is the right child of the p^{th} node.
- We set $v(y^w_{i,j,p}) = 1$ iff $w[i,j) \vdash R_p^{\mathcal{S}}$

Firstly, it is easy to see that $v \models \Phi_n^{\text{structure}}$, since the formulated valuation ensures the uniqueness of the labels of the nodes as well as that of the left and right children. Moreover, $v \models \Phi_n^w$ for $w \in \mathcal{P} \cup \mathcal{N}$, since, the values of the variables $y^w_{i,j,p}$ correspond exactly to the valuation of each subexpression $R_i^{\mathcal{S}}$ on the subword $w[i,j)$. Finally, the fact that $R^{\mathcal{S}}$ is consistent with $\mathcal{S}$ implies $v \models y^w_{i,j,p}$ for each $w \in \mathcal{P}$ and $v \models \neg y^w_{i,j,p}$ for each $w \in \mathcal{N}$. Thus, $v \models \Phi_n^{\mathcal{S}}$, which proves that $\Phi_n^{\mathcal{S}}$ is satisfiable.

For the second statement, since, $v \models \Phi_n^S$, we have $v \models \Phi_n^{\text{structure}}$ as well. Hence, the valuation of the variables $x_{p,\lambda}$, $l_{p,q}$, and $r_{p,q}$ encode a syntax DAG from which we obtain the regular expression R^v. Again here, we consider R_p^v to be the subexpression of R^v rooted at the node p. Now, it needs to be shown that R^v is indeed consistent with the sample $\mathcal{S}$. For that we show, $v(y^w_{i,j,p}) = 1 \iff w[i,j) \vdash R_p^v$ for any i,$j \in \{0, \cdots, m\}$ where $m = |w|$ and $p \in \{0, \cdots, n\}$. This can be done using induction on the structure of R^v.

- *Base case* $R_p^v = \varepsilon$: In this case, $v(x_{p,\epsilon}) = 1$, which leads to the following implications:

$$\begin{aligned} v(y^w_{i,j,p}) = 1 &\iff \mathfrak{p} \in uv^\omega[i,j) && \text{[Using Subformula (3.6)]} \\ &\iff w[i,j) \vdash \varepsilon && \text{[Using Def. 1]} \end{aligned}$$

- *Base case* $R_p^v = \mathfrak{p}$: In this case, $v(x_{p,a}) = 1$ and this leads to the following implications:

$$\begin{aligned} v(y^w_{i,j,p}) = 1 &\iff w[i,j) = a && \text{[Using Subformula (3.7)]} \\ &\iff w[i,j) \vdash a && \text{[Using Def. 1]} \end{aligned}$$

- *Case* $R_p^v = R_q^v + R_{q'}^v$: In this case, $v(x_{p,+})$, $v(l_{p,q})$, $v(r_{p,q'})$ are all set to 1, and this leads to the following implications:

$$\begin{aligned} v(y^w_{i,j,p}) = 1 &\iff v(y^w_{i,j,q}) = 1 \text{ or } v(y^w_{i,j,q'}) = 1 && \text{[Using Subformula (3.8)]} \\ &\iff w[i,j) \vdash R_q^v \text{ or } w[i,j) \vdash R_{q'}^v && \text{[Using ind. hypothesis]} \\ &\iff w[i,j) \vdash R_q^v + R_{q'}^v && \text{[Using Def. 1]} \end{aligned}$$

- *Case* $R_p^v = R_q^v \circ R_{q'}^v$: In this case, $v(x_{p,\circ})$, $v(l_{p,q})$, $v(r_{p,q'})$ are all set to 1 and this leads to the

following implications:

$$\begin{aligned}
v(y^w_{i,j,p}) = 1 &\iff \exists k \in \mathbb{N},\ 1 \le k \le m, v(y^w_{i,k,q}) = 1 \text{ and } v(y^w_{k,j,q'}) = 1 && \text{[Using Subformula (3.9)]}\\
&\iff \exists k \in \mathbb{N},\ 1 \le k \le m, w[i,k) \vdash R^v_q \text{ and } w[k,j) \vdash R^v_{q'} && \text{[Using ind. hypothesis]}\\
&\iff w[i,j) \vdash R^v_q \circ R^v_{q'} && \text{[Using Def. 1]}
\end{aligned}$$

– *Case* $R^v_p = (R^v_q)^*$: In this case, $v(x_{p,*})$ and $v(l_{p,q})$ are set to 1 and this leads to the following implications:

$$\begin{aligned}
v(y^w_{i,j,p}) = 1 &\iff \begin{cases} i = j \text{ or} \\ \exists k \in \mathbb{N},\ 1 \le k \le m,\ v(y^w_{i,k,q}) = 1 \text{ and } v(y^w_{k,j,p}) = 1 \end{cases} && \text{[Using Subformula (3.10)]}\\
&\iff \begin{cases} i = j \text{ or} \\ \exists k \in \mathbb{N},\ 1 \le k \le m,\ w[i,k) \vdash R^v_q \text{ and } w[k,j) \vdash (R^v_q)^* \end{cases} && \text{[Using ind. hypothesis]}\\
&\iff w[i,j) \vdash (R^v_q)^* && \text{[Using Def. 1]}
\end{aligned}$$

Here, we assumed that $v(y^w_{k,j,p}) = 1$ iff $w[k,j) \vdash R^v_p$ in the second step, which cannot be deduced from the induction on the structure of R^v_p. This we obtained using another level of induction, which is on the length of the subword that matches R^v_p. More precisely, we induct on k which ranges from j to $i+1$, where by induction hypothesis, it is assumed that

$$v(y^w_{k,j,p}) = 1 \iff w[k,j) \vdash R^v_p \quad \forall k,\ i < k \le j$$

The base case occurs when $k = j$, and then, we have $v(y^w_{k,j,p}) = 1 \iff k = j \iff w[k,j) \vdash R^v_p$.

□

Chapter 4

Learning $\omega-$Regular Expressions

4.1 $\omega-$Regular expression

An ω-regular expression r_ω is an expression with the following grammar:

$$r_\omega ::= r^\omega \mid r \circ_\omega r_\omega \mid r_\omega +_\omega r_\omega$$

where, r is a regular expression as described in Section 3.1. Let the set of $\omega-$regular expressions be denoted by $\mathcal{R}^\omega_\Sigma$. Again here, an $\omega-$regular expression can be represented in the form of a syntax tree or a syntax DAG, similar to ones for regular expressions. The only difference here is that the nodes of the syntax tree and the syntax DAG are labelled by elements from $\Lambda_\omega = \Lambda_R \cup \{\omega, +_\omega, \circ_\omega\}$.

Semantics The semantics of an $\omega-$regular expression is as usual defined in terms of the language they define; obtained in the following manner:

$$[\![r^\omega]\!] = [\![r]\!]^\omega;\ [\![r \circ_\omega r_\omega]\!] = [\![r]\!] \circ_\omega [\![r_\omega]\!];\ [\![r_\omega +_\omega r'_\omega]\!] = [\![r_\omega]\!] \cup [\![r'_\omega]\!]$$

It should be noted that r^ω is well defined only when $\epsilon \notin [\![r]\!]$, since otherwise, $[\![r^\omega]\!]$ might contain invalid infinite words.

Now, let us look at some results regarding membership of ultimately periodic words in languages defined by $\omega-$regular expressions. In the following results, we assume $u \in \Sigma^*$ and $v \in \Sigma^+$ and r is a regular expression in $\mathcal{R}_\Sigma$, such that $\epsilon \notin [\![r]\!]$. Moreover, assume that the minimal DFA of $[\![r]\!]$ has size n.

Proposition 1. $uv^\omega \in [\![r^\omega]\!] \iff \exists i, j \in \mathbb{N},\ |u| < i < j,\ uv^\omega[0,i) \in [\![r^+]\!]$ *and* $uv^\omega[i,j) \in [\![r^+]\!]$ *where* $j \equiv i \mod |v|$.

Proof. ($\Rightarrow$) From the semantics of ω-regular expressions, we have $uv^\omega \in [\![r^\omega]\!] \implies uv^\omega \in [\![r]\!]^\omega$. Hence, there exists an infinite sequence of natural numbers $i_1, i_2, i_3, \ldots$ satisfying $|u| < i_1 < i_2 < \ldots$ such that $uv^\omega[0, i_1) \in [\![r^+]\!]$ and from then on $uv^\omega[i_1, i_2) \in [\![r]\!]$, $uv^\omega[i_2, i_3) \in [\![r]\!]$ and so on. Since, $i_0, i_1, i_2, \ldots$ is an infinite sequence, it is possible to find numbers i and j in the sequence, such that $j \equiv i \mod |v|$ using pigeonhole principle. Now, clearly, $uv^\omega[0,i) \in [\![r^+]\!]$ and $uv^\omega[i,j) \in [\![r^+]\!]$, since i and j belong to the sequence.

($\Leftarrow$) Let $uv^\omega[0,i) = v_1$ and $uv^\omega[i,j) = v_2$, where both $uv^\omega[0,i)$ and $uv^\omega[i,j)$ satisfy the properties mentioned in the lemma. It is an easy observation that $v_1 v_2^\omega = uv^\omega$. As we already know $v_1 \in [\![r^+]\!]$ and $v_2 \in [\![r^+]\!]$, we can conclude that $v_1 v_2^\omega \in [\![r^+]\!] \circ [\![r^+]\!]^\omega \implies v_1 v_2^\omega \in [\![r^\omega]\!] \implies uv^\omega \in [\![r^\omega]\!]$. □

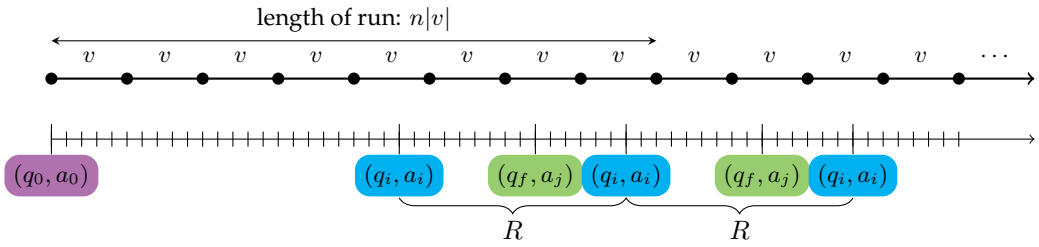


Figure 4.1: The run of the DFA of $[\![r]\!]$ on $v^\omega[i,j)$, where R is the repeating run staring at (l,q). The first occurrence of R happens within $n|v|+1$ steps and hence, the first occurrence of state q_f also happens within that many steps.

Lemma 1. $v^\omega \in [\![r^\omega]\!] \iff \exists i \in \mathbb{N},\ v^\omega[0,i) \in [\![r^{|v|+1}]\!]$.

Proof. ($\Rightarrow$) Since, $v^\omega \in [\![r]\!]^\omega$, for any natural number n, there is a prefix $v^\omega[0,i)$ of v^ω such that $v^\omega[0,i) \in [\![r^n]\!]$. Therefore, it holds true for $n = |v|+1$ as well.

($\Leftarrow$) Here, we have $v^\omega[0,i) \in [\![r^{|v|+1}]\!]$ for some $i \in \mathbb{N}$. Therefore, we can find a finite sequence of distinct natural numbers $i_1, i_2, \ldots, i_{|v|+1}$, where $i_{|v|+1} = i$ and also, $v^\omega[0,i_1) \in [\![r]\!]$, $v^\omega[i_1,i_2) \in [\![r]\!]$ and so on. Now, notice that it is possible to find numbers i', j' with $i' < j'$, in the sequence such that $j' \equiv i' \mod |v|$ using pigeonhole principle (Since, the sequence contains more than $|v|$ elements). Consequently, we have $v^\omega[0,i') \in [\![r^+]\!]$, $v^\omega[i',j') \in [\![r^+]\!]$ and $j' \equiv i' \mod |v|$, which gives us $v^\omega \in [\![r^\omega]\!]$ straightaway using Prop. 1. □

Lemma 2. *Let* $uv^\omega \in [\![r^\omega]\!]$ *such that for some* $i > |u|$, $uv^\omega[0,i) \in [\![r]\!]$ *and* $uv^\omega[i,\infty) \in [\![r^\omega]\!]$. *Then,* $\exists j \in \mathbb{N},\ j > i$, *such that* $uv^\omega[0,j) \in [\![r^{|v|+1}]\!]$

The proof for this proceeds exactly the same way as Lemma 1. The next lemma shows how the length of each match can be bounded.

Lemma 3. $v^\omega[i,j) \in [\![r]\!] \implies \exists k \in \mathbb{N},\ k - i \leq n|v|$ *such that* $v^\omega[i,k) \in [\![r]\!]$ *and* $k \equiv j \mod |v|$, *where* n *is the size of the minimal DFA for* $[\![r]\!]$.

Proof. If $j - i \leq n|v|$, we are done, since we simply take $k = j$. On the other hand, if $j - i > n|v|$, finding the suitable k is a bit more involved. Firstly, since, $v^\omega[i,j) \in [\![r]\!]$, there is an accepting run of the DFA $\mathcal{A}$ for $[\![r]\!]$ on $v^\omega[i,j)$, which is of length greater than $n|v|$. Now, we look at this run of $\mathcal{A}$ on $v^\omega[i,j)$. Fig 4.1 provides a pictorial depiction of the run. Here we consider the run to be a sequence of tuples of the form (q,m), where the first entry refers to the state of the automaton at that instant and the second entry refers to the position in v which will be read next by the automaton. Hence, it is easy to observe that any run longer than $n|v|$ would mean that there exists a tuple which repeats itself during the run, due to pigeonhole principle. Let (q,l) be the tuple which repeats itself and let the run starting at the first occurrence of (q,l) to the next, be referred to as R. Notice that due to the deterministic nature of the automaton, R repeats itself during the rest of the run. Hence, if a final state q_f occurs after $n|v|$ steps, there must be a tuple (q_f,m) which belongs to the run R. Clearly, (q_f,m) must have been also visited during the first occurrence of R, which happened within $n|v|$ steps. Thus, we get a prefix $v^\omega[i,k) \in [\![r]\!]$, which has length less than $n|v|$. Moreover, $v^\omega[i,j)$ and $v^\omega[i,k)$ terminate at the same position in v, meaning $k \equiv j \mod |v|$.

□

Lemma 4. $uv^\omega[i,j) \in [\![r]\!]$, *where* $i < |u| < j \implies \exists k \in \mathbb{N}, |u| < k \leq |u| + n|v|$ *such that* $uv^\omega[i,k) \in [\![r]\!]$ *and* $k \equiv j \mod |v|$.

The proof for this statement goes along the same lines as Lemma 3. Finally, having these results, it is possible to verify whether an infinite word (ultimately periodic word in this case) actually belongs to the language of regular expression by checking only a finite portion of the word. The following theorems come in handy when designing algorithms for checking matching of infinite word with $\omega-$regular expressions, by bounding the length on which checking is done.

Theorem 1. $v^\omega \in [\![r^\omega]\!] \iff \exists i,j \in \mathbb{N},\ 0 < i < j \leq n|v| + n|v|^2\ v^\omega[0,i) \in [\![r^+]\!]$ *and* $v^\omega[i,j) \in [\![r^+]\!]$ *where* $j \equiv i \mod |v|$

Proof. ($\Leftarrow$) This can be seen directly from Prop. 1, assuming $u = \epsilon$.

($\Rightarrow$) Using Lemma 1, there exists some i' such that $v^\omega[0,i') \in [\![r^{|v|+1}]\!]$. As a result, we can generate a sequence of distinct natural numbers, $i_1, \ldots, i_{|v|+1}$, with $i_{|v|+1} = i'$ such that $v^\omega[0,i_1) \in [\![r]\!]$, $v^\omega[i_1,i_2) \in [\![r]\!]$ and so on. Now, we can find another sequence $j_1, \ldots, j_{|v|+1}$ with $j_{|v|+1}$, which satisfy all the properties that the sequence of i's satisfy and additionally have $j_1 \leq n|v|$, $j_2 - j_1 \leq n|v|$, so on and also, $j_1 \equiv i_1 \mod |v|$, $j_2 \equiv i_2 \mod |v|$, so on. This is a direct consequence of Lemma 3. It is easy to see that $j_{|v|+1} \leq n|v| + n|v|^2$. Moreover, we can again find some i and j in this sequence such that $j \equiv i \mod |v|$ and clearly, $v^\omega[0,i) \in [\![r^+]\!]$ and $v^\omega[i,j) \in [\![r^+]\!]$. □

Theorem 2. $uv^\omega \in [\![r^\omega]\!] \iff \exists i,j \in \mathbb{N}, |u| < i < j \leq |u|+n|v|+n|v|^2$, $uv^\omega[0,i) \in [\![r^+]\!]$ *and* $uv^\omega[i,j) \in [\![r^+]\!]$ *where* $j \equiv i \mod |v|$

Proof. ($\Leftarrow$) This can be seen directly from Prop. 1.

($\Rightarrow$) When $uv^\omega \in [\![r^\omega]\!]$, there could be two possible cases arising from how uv^ω matches r^ω.

- First case would be one in which $u \in [\![r^+]\!]$ and $v^\omega \in [\![r^\omega]\!]$. In this case, Theorem 1 can be applied directly to obtain the result.

- In the second case, $\exists i', j' \in \mathbb{N}$ with $i' < |u| < j'$, such that, $uv^\omega[0,i') \in [\![r^+]\!]$, $uv^\omega[i',j') \in [\![r]\!]$, and $uv^\omega[j',\infty) \in [\![r^\omega]\!]$. Now, it can be seen that infix $uv^\omega[i',\infty)$ satisfies the conditions of Lemma 2. Hence, there is some $k' > j'$, such that $uv^\omega[i',k') \in [\![r^{|v|+1}]\!]$. As a result, we can generate a sequence of distinct natural numbers $i_1, \ldots, i_{|v|+1}$, with $i_1 = j'$ and $i_{|v|+1} = k'$ such that $uv^\omega[i',i_1) \in [\![r]\!], uv^\omega[i_1,i_2) \in [\![r]\!]$ and so on. Combining results of Lemma 4 and Lemma 3, we can find another sequence $j_1, j_2 \ldots j_{|v|+1}$ which satisfy the properties of the sequence of i's and additionally have $j_1 \leq |u| + n|v|$, $j_2 - j_1 \leq n|v|$, $j_3 - j_2 \leq n|v|$, so on and also, $j_1 = i_1 \mod |v|$, $j_2 = i_2 \mod |v|$, so on. It is easy to see that $j_{|v|+1} \leq |u| + n|v| + n|v|^2$. Moreover, in this sequence we can find some i and j such that $j \equiv i \mod |v|$ and clearly, $uv^\omega[0,i) \in [\![r^+]\!]$ and $uv^\omega[i,j) \in [\![r^+]\!]$.

□

Motivated by the above results, we define the matching relation for infinite words which would help us to algorithmically check the matching of a ultimately periodic word with an $\omega-$ regular expression.

Definition 2. *The matching relation $\vdash$ for finite words with regular expressions, could be extended to account for matching of ultimately periodic words with ω-regular expressions as well. Here, n refers to the size of the minimal DFA of $[\![r]\!]$ in all the cases.*

$$uv^\omega[i,\infty) \vdash r^\omega \iff \exists j,k \in \mathbb{N},\ max(i,|u|) < j < k \leq i+|u|+|v|+n|v|+n|v|^2,$$
$$uv^\omega[i,j) \vdash r^+ \textit{ and } uv^\omega[j,k) \vdash r^+ \textit{ where } k \equiv j \mod |v|$$
$$uv^\omega[i,\infty) \vdash r \circ r_\omega \iff \exists j \in \mathbb{N},\ i \leq j \leq |u|+|v|+n|v|,\ uv^\omega[i,j) \vdash r \textit{ and } uv^\omega[j,\infty) \vdash r_\omega$$
$$uv^\omega[i,\infty) \vdash r_\omega + r'_\omega \iff uv^\omega[i,\infty) \vdash r_\omega \textit{ or } uv^\omega[i,\infty) \vdash r'_\omega$$

It is necessary to check whether the matching relation satisfies $v^\omega[i,\infty) \vdash r_\omega \iff v^\omega[i,\infty) \in [\![r_\omega]\!]$. This check proceeds via structural induction on the expression r_ω.

Observation 4.1: Let $uv^\omega \in \Sigma^\omega$. Then, $uv^\omega[|u|+i,\infty) = uv^\omega[|u|+j,\infty)$ for $j \equiv i \mod |v|$. Because of this observation, we can say $r_\omega \vdash uv^\omega[|u|+i,\infty) \iff r_\omega \vdash uv^\omega[|u|+j,\infty)$. Hence, the matching relation has been defined only for suffix $uv^\omega[i,\infty)$, where $i < |uv|$.

Unlike regular expression, the construction of the tables for algorithmically checking matching of an $\omega-$regular expression with an $\omega-$word needs a slightly different approach. There are two types of tables that need to be maintained for each node. The first table keeps track of which infinite suffixes of the $\omega-$word matches which $\omega-$regular subexpressions. The other table maintains record of matching of finite infixes of the $\omega-$word with regular subexpressions. The entries of both types of tables are propagated to tables of higher indexed nodes using the matching relation.

4.2 The Learning Problem

We assume that the data to learn from is given as a pair $\mathcal{S} = (\mathcal{P}, \mathcal{N})$ consisting of two finite, disjoint sets $\mathcal{P}, \mathcal{N} \subset \Sigma^*$ of ultimately periodic words such that $\mathcal{P} \cap \mathcal{N} \neq \emptyset$.

Just like regular expressions, we say that an $\omega-$regular expression r_ω over Σ is *consistent* with a sample $\mathcal{S} = (\mathcal{P}, \mathcal{N})$ if

1. $r_\omega \vdash u$ for each $u \in \mathcal{P}$; and
2. $r_\omega \nvdash u$ for each $u \in \mathcal{N}$.

A solution similar to the one used for regular expression can be used for this problem as well. In fact, the Algorithm 1 would work here and would return a $\omega-$regular expression. The formula $\Phi_n^{\mathcal{S}}$ used in the algorithm is designed slightly differently for this case. The formula has two parts as described in Equation 3.1. The first part $\Phi_n^{\text{structure}}$ which encodes a syntax DAG of an ω-regular expression has the following variables.

- $x_{p,\lambda}$ where $p \in \{1, \cdots, n\}$ and $\lambda \in \Lambda_\omega$
- $l_{p,q}$, $r_{p,q}$ where $p \in \{2, \cdots, n\}$ and $q \in \{1, ..., p-1\}$

Now, the variables should satisfy certain constraints in order to form a valid syntax DAG. But, these constraints are precisely the ones that needed to be satisfied by the syntax DAG of regular expressions. Thus, the exact description of the formulas have been skipped.

While the first part of the formula guesses an $\omega-$regular expression r, the second part $\Phi_n^{\text{consistency}}$ checks whether words in the sample matches r. This formula makes use of the following variables. Here, we assume $b = n + n|v| + 1$ and $c = n + 1$.

- $y_{i,p}^{u,v}$ where $0 \leq i \leq |uv| - 1$
- $z_{i,j,p}^{uv^b}$, where $0 \leq i \leq j \leq |uv^b|$.

- $\bar{z}_{i,j,p}^{uv^b}$, where $0 \leq i \leq j \leq \left|uv^b\right|$

The variable $y_{i,p}^{u,v}$ basically checks the matching of the infinite suffix $uv^\omega[i,\infty)$ with the subexpression of the guessed expression r rooted at an infinite operator indexed by p. The variable $z_{i,j,p}^{uv^b}$ checks the matching of finite infix $uv^b[i,j)$ with the subexpression of r rooted at a finite operator indexed by p. $\bar{z}_{i,j,p}^{uv^b}$ is an auxiliary variable which captures that whether $uv^b[i,j)$ matches $\bar{r}^+$ if $\bar{r}$ is the subexpression rooted under the p^{th} node. Φ_n^w is a disjunction of several formulas as mentioned below.

$$\bigwedge_{1\leq p\leq n} x_{p,\varepsilon} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq \left|uv^b\right|} z_{i,j,p}^{uv^b} \leftrightarrow [i=j]\Big] \tag{4.1}$$

$$\bigwedge_{1\leq p\leq n}\bigwedge_{a\in\Sigma} x_{p,a} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq \left|uv^b\right|} \begin{cases} z_{i,j,p}^{uv^b} \text{ if } w[i,j)=a \\ \neg z_{i,j,p}^{uv^b} \text{ if } w[i,j)\neq a \end{cases}\Big] \tag{4.2}$$

$$\bigwedge_{\substack{1\leq p\leq n\\ 1\leq q,q'\leq p}} x_{p,+}\wedge l_{p,q}\wedge r_{p,q'} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq \left|uv^b\right|} \Big[z_{i,j,p}^{uv^b} \leftrightarrow z_{i,j,q}^{uv^b}\vee z_{i,j,q'}^{uv^b}\Big]\Big] \tag{4.3}$$

$$\bigwedge_{\substack{1\leq p\leq n\\ 1\leq q,q'\leq p}} x_{p,\circ}\wedge l_{p,q}\wedge r_{p,q'} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq \left|uv^b\right|} \Big[z_{i,j,p}^{uv^b} \leftrightarrow \bigvee_{i\leq k\leq j} z_{i,k,q}^{uv^b}\wedge z_{k,j,q'}^{uv^b}\Big]\Big] \tag{4.4}$$

$$\bigwedge_{\substack{1\leq p\leq n\\ 1\leq q,q'\leq p}} x_{p,*}\wedge l_{p,q} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq \left|uv^b\right|} \Big[z_{i,j,p}^{uv^b} \leftrightarrow [i=j]\vee \bigvee_{i<k\leq j} z_{i,k,q}^{uv^b}\wedge z_{k,j,p}^{uv^b}\Big]\Big] \tag{4.5}$$

The above formulas are for the finite operators in the guessed expression. They are motivated by Formulas 3.6 to 3.10 for regular expressions. The formulas for the infinite operators are as follows.

$$\begin{aligned}\bigwedge_{\substack{1\leq p\leq n\\ 1\leq q\leq p}} x_{p,\omega}\wedge l_{p,q} \rightarrow \Big[\bigwedge_{0\leq i<|uv|} \Big[y_{i,p}^{u,v} \leftrightarrow \bigvee_{\substack{max(i,|u|)<j<k\leq \left|uv^b\right|\\ k\equiv j \mod |v|}} \Big[\bar{z}_{i,j,q}^{uv^b}\wedge \bar{z}_{j,k,q}^{u,v}\Big]\Big]\Big] \\ \wedge \Big[\bigwedge_{0\leq i\leq j\leq \left|uv^b\right|} \Big[\bar{z}_{i,j,q}^{u,v} \leftrightarrow \bigvee_{i<k\leq j} z_{i,k,q}^{uv^b}\wedge \bar{z}_{k,j,q}^{uv^b}\Big]\Big]\Big]\end{aligned} \tag{4.6}$$

$$\bigwedge_{\substack{1\leq p\leq n\\ 1\leq q,q'\leq p}} x_{p,\circ_\omega}\wedge l_{p,q}\wedge r_{p,q'} \rightarrow \Big[\bigwedge_{0\leq i<|uv|} \Big[y_{i,p}^{u,v} \leftrightarrow \bigvee_{i\leq j\leq |uv^c|} \Big[z_{i,j,q}^{uv^b}\wedge y_{M(j),q'}^{u,v}\Big]\Big]\Big] \tag{4.7}$$

$$\bigwedge_{\substack{1\leq p\leq n\\ 1\leq q,q'\leq p}} x_{p,+_\omega}\wedge l_{p,q}\wedge r_{p,q'} \rightarrow \Big[\bigwedge_{0\leq i<|uv|} y_{i,p}^{u,v} \leftrightarrow y_{i,q}^{u,v}\vee y_{i,q'}^{u,v}\Big] \tag{4.8}$$

The formulas for the infinite operators are motivated by how the matching relation is defined. In the Formula 4.6, the second disjunct computes the matching relation of $\bar{r}^+$ for the subexpression $\bar{r}$ rooted at the q^{th} node. $M(j)$ used in Formula 4.7 for $\circ_\omega$ is defined as follows:

$$M(j) = \begin{cases} j & \text{for } j<|uv| \\ |u| + ((j-|u|)\%|v|) & \text{for } j\geq |uv| \end{cases} \tag{4.9}$$

$a\%b = r$ if r is the remainder when a is divided by b. $\%$ is a 'remainder' operation commonly used

in programming languages such as C. Finally, all the Formulas 4.1 to 4.8 are put together in Φ_n^w. $\Phi_n^{\text{consistency}}$ is constructed in the following manner:

$$\Phi_n^{\text{consistency}} = \Big[\bigwedge_{w \in \mathcal{P}} \Phi_n^w \wedge y_{0,n}^w \Big] \wedge \Big[\bigwedge_{w \in \mathcal{N}} \Phi_n^w \wedge \neg y_{0,n}^w \Big] \tag{4.10}$$

Claim 2. *Let $\mathcal{S} = (\mathcal{P}, \mathcal{N})$ be a sample, $n \in \mathbb{N} \setminus \{0\}$, and $\Phi_n^{\mathcal{S}}$ be the propositional formula defined above. Then, the following holds:*

1. *If there exists a $\omega-$regular expression of size n, $R_\omega^{\mathcal{S}}$ that is consistent with $\mathcal{S}$, then the propositional formula $\Phi_n^{\mathcal{S}}$ is satisfiable.*
2. *If $v \models \Phi_n^{\mathcal{S}}$, then R^v is a $\omega-$regular expression of size n that is consistent with $\mathcal{S}$.*

The proof for this is similar to the proof for Claim 1. Appendix B has the complete proof.

Chapter 5

Learning of Linear Temporal Language

The learning algorithm in this chapter has been taken from the paper by Neider and Gavran[6].

5.1 Linear Temporal Logic

Linear Temporal Logic (LTL)[3] is an extension of propositional Boolean logic with modalities that allow expressing temporal properties. The syntax of LTL is given by the following grammar:

$$\varphi ::= \mathfrak{p} \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi$$

where $\mathfrak{p} \in \mathsf{AP}$, where AP be a set of atomic propositions. The set of all LTL formulas over AP is referred to as $\mathcal{F}_{LTL}$

LTL is allowed to have formulas commonly used in propositional logic such as $true$, $false$, $\varphi \wedge \psi$, $\psi \rightarrow \varphi$ which are defined as usual. Moreover, additional temporal formulas are also allowed— $\mathbf{F}\varphi := true\mathbf{U}\varphi$ ("finally") and $\mathbf{G}\varphi := \neg F \neg \varphi$ ("globally"). For sake of the learning problem, it is enough to consider the primary operators. Hence, let $\Lambda_L = \mathsf{AP} \cup \{\vee, \neg, \mathbf{X}, \mathbf{U}\}$

Syntax trees and syntax DAGs are appropriate representation for LTL formulas as well. Here, the labels of the syntax tree or syntax DAG come from the set Λ_L. The size of an LTL formula is naturally defined as the number of unique subformulas in it, which is equivalent to number of nodes in the syntax DAG representation of the formula.

Semantics: Given a word $w \in (2^{\mathsf{AP}})^\omega$ and a LTL formula φ, we define the relation $w \models_{\text{LTL}} \varphi$ (read "w satisfies φ").

$w \models_{\text{LTL}} p$ iff $p \in w[0,0)$
$w \models_{\text{LTL}} \neg\varphi$ iff $w \not\models_{\text{LTL}} \varphi$
$w \models_{\text{LTL}} \varphi \vee \psi$ iff $w \models_{\text{LTL}} \varphi$ or $w \models_{\text{LTL}} \psi$
$w \models_{\text{LTL}} \mathbf{X}\varphi$ iff $w[1,\infty) \models_{\text{LTL}} \varphi$
$w \models_{\text{LTL}} \varphi \mathbf{U} \psi$ iff $\exists i \geq 0$ such that, $w[i,\infty) \models_{\text{LTL}} \psi$ and $\forall k\ 0 < k < i,\ w[k,\infty) \models \varphi$

Just as in the case of regular expressions and $\omega-$regular expressions, a satisfaction relation needs to be defined for LTL formulas, for checking whether an infinite word over 2^{AP} satisfies an LTL

formula φ. Here also, we restrict infinite words to only ultimately periodic words, because, they have certain structural properties, which aid the process of checking satisfaction.

Due to observation 4.1, it is enough to define the satisfaction relation for only those infinite suffix $uv^\omega[i,\infty)$ of uv^ω, where $0 \leq i \leq |uv| - 1$.

Definition 3. *The satisfaction relation* $\vDash \subseteq (2^{\mathsf{AP}})^\omega \times \mathcal{F}_{LTL}$ *is defined inductively on the structure of an LTL formula* φ *in the following manner:*

$$uv^\omega[i,\infty) \vDash p \iff p \in uv^\omega[i,i)$$

$$uv^\omega[i,\infty) \vDash \neg\varphi \iff uv^\omega[i,\infty) \nvDash \varphi$$

$$uv^\omega[i,\infty) \vDash \varphi \vee \psi \iff uv^\omega[i,\infty) \vDash \varphi \text{ or } uv^\omega[i,\infty) \vDash \psi$$

$$uv^\omega[i,\infty) \vDash \boldsymbol{X}\varphi \iff \begin{cases} uv^\omega[i+1,\infty) \vDash \varphi, & \text{for } 0 \leq i < |uv| - 1 \\ uv^\omega[|u|,\infty) \vDash \varphi, & \text{for } i = |uv| - 1 \end{cases}$$

$$uv^\omega[i,\infty) \vDash \varphi \boldsymbol{U} \psi \iff \begin{cases} \exists j,\ i \leq j \leq |uv| - 1,\ uv^\omega[j,\infty) \vDash \psi \text{ and } \forall k,\ i < k < j\ uv^\omega[k,\infty) \vDash \varphi & \text{for } i < |u| \\ \exists j,\ |u| \leq j \leq |uv| - 1,\ uv^\omega[j,\infty) \vDash \psi \text{ and } \forall k \in \mathcal{I}_{u,v}(i,j)\ uv^\omega[k,\infty) \vDash \varphi & \text{for } i \geq |u| \end{cases}$$

The set $\mathcal{I}^{u,v}(i,j)$ used in defining the satisfaction relation is defined as follows:

$$\mathcal{I}^{u,v}(i,j) = \begin{cases} \{i, \cdots, j-1\} & \text{for } i \leq j \\ \{|u|, \cdots, j-1\} \cup \{i, \cdots, |uv-1|\} & \text{for } i > j \end{cases}$$

Now, it needs to be shown that the satisfaction relation so defined is actually consistent with the semantics of LTL. Precisely, we need to prove, $uv^\omega[i,\infty) \vDash \varphi \iff uv^\omega[i,\infty) \models_{\text{LTL}} \varphi$. But, this is not difficult to see and requires an induction on the structure of φ.

5.2 The Learning Problem

The problem statement for this case remains almost the same as in $\omega-$regular expressions. Given a sample $\mathcal{S} = (\mathcal{P}, \mathcal{N})$ consisting of two finite, disjoint sets $\mathcal{P}, \mathcal{N} \subset \Sigma^*$ of ultimately periodic words such that $\mathcal{P} \cup \mathcal{N} \neq \emptyset$, we need to find a LTL formula φ over AP *consistent* with a sample $\mathcal{S}$.

The solution for this learning problem is not much different from the ones described so far in this thesis. In fact, the same Algorithm 1 can be used to learn a LTL formula from the given sample. The only difference is in how the formula $\Phi_n^{\mathcal{S}}$ is formulated. The formula has two parts as usual.

$$\Phi_n^{\mathcal{S}} = \Phi_n^{\text{structure}} \wedge \Phi_n^{\text{consistency}}$$

The first part $\Phi_n^{\text{structure}}$ which encodes a syntax DAG of a LTL formula has the following variables.

- $x_{p,\lambda}$ where $p \in \{1, \cdots, n\}$ and $\lambda \in \Lambda_L$
- $l_{p,q}$, $r_{p,q}$ where $p \in \{2, \cdots, n\}$ and $q \in \{1, ..., p-1\}$

Now, the variables should satisfy certain constraints in order to form a valid syntax DAG. But, these constraints are precisely the ones that needed to be satisfied by the syntax DAG of regular expressions. Thus, the exact description of the formulas have been skipped.

The second part of the formula checks whether the guessed syntax DAG of the LTL formula is consistent with the sample. Φ_n^w is a disjunction of following formulas.

$$\bigwedge_{1\leq p\leq n}\bigwedge_{\mathfrak{p}\in\mathsf{AP}} x_{p,\mathfrak{p}} \rightarrow \Big[\bigwedge_{0\leq i<|uv|}\begin{cases} y_{i,p}^{u,v} \text{ if } \mathfrak{p}\in uv[i,i) \\ \neg y_{i,p}^{u,v} \text{ if } \mathfrak{p}\notin uv[i,i)\end{cases}\Big] \tag{5.1}$$

$$\bigwedge_{\substack{1\leq p\leq n\\ 1\leq q,q'\leq p}} x_{p,\vee}\wedge l_{p,q}\wedge r_{p,q'} \rightarrow \Big[\bigwedge_{0\leq i<|uv|}\Big[y_{i,p}^{u,v}\leftrightarrow y_{i,q}^{u,v}\vee y_{i,j,q'}^{u,v}\Big]\Big] \tag{5.2}$$

$$\bigwedge_{\substack{1\leq p\leq n\\ 1\leq q\leq p}} x_{p,\mathbf{X}}\wedge l_{p,q} \rightarrow \Big[\bigwedge_{0\leq i<|uv|-1}\Big[y_{i,p}^{u,v}\leftrightarrow y_{i+1,q}^{u,v}\Big]\wedge\Big[y_{|uv|-1,p}^{u,v}\leftrightarrow y_{|u|,q}^{u,v}\Big]\Big] \tag{5.3}$$

$$\begin{aligned}\bigwedge_{\substack{1\leq p\leq n\\ 1\leq q,q'\leq p}} x_{p,\mathbf{U}}\wedge l_{p,q}\wedge r_{p,q'} \rightarrow &\Big[\bigwedge_{0\leq i\leq|u|}\Big[y_{i,p}^{u,v}\leftrightarrow\bigvee_{i\leq j<|uv|}\Big[y_{j,q}^{u,v}\wedge\bigwedge_{i\leq k<j}y_{k,q}^{u,v}\Big]\Big]\\ &\wedge\Big[\bigwedge_{|u|\leq i<|uv|}\Big[y_{i,p}^{u,v}\leftrightarrow\bigvee_{|u|\leq j<|uv|}\Big[y_{j,q}^{u,v}\wedge\bigwedge_{k\in\mathcal{I}^{u,v}(i,j)}y_{k,q}^{u,v}\Big]\Big]\end{aligned} \tag{5.4}$$

These formulas originate from how the satisfaction relation could be used to check whether word uv^ω satisfies the guessed formula. The final step is to collate all the formulas and ensure consistency with the sample $\mathcal{S}$.

$$\Phi_n^{\text{consistency}} = \Big[\bigwedge_{w\in\mathcal{P}}\Phi_n^w\wedge y_{0,n}^w\Big]\wedge\Big[\bigwedge_{w\in\mathcal{N}}\Phi_n^w\wedge\neg y_{0,n}^w\Big] \tag{5.5}$$

The proof of correctness for the learning algorithm for LTL formulas goes along the same lines as the proof of Claim 1. The complete proof can be be found in [6].

Chapter 6

Learning Program Specification Language

6.1 Property Specification Language

Property Specification Language (PSL)[4] is a language for the formal specification of hardware. It is used to describe properties that are required to hold in the design under verification. PSL subsumes LTL and has an increased expressive power. While the expressive power of LTL is that of star-free ω-regular expressions, the expressive power of PSL is the same as the full class of ω-regular expressions[2]. As PSL is a widely used standard it contains a large amount of derived operators; we concentrate here only on a subset of operators that suffice to obtain the expressive power of ω-regular expressions. We call this subset corePSL. The syntax of corePSL is given by the following grammar:

$$\varphi ::= \mathfrak{p} \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi_1 \mathbf{U} \varphi_2 \mid r \mapsto \varphi$$

where $\mathfrak{p} \in \mathsf{AP}$, where Let AP be a set of atomic propositions and r is a regular expression over 2^{AP}. The operator $\mapsto$ is called *suffix implication* or *trigger*. The set of all PSL formulas over AP is referred to as $\mathcal{F}_{PSL}$. PSL uses operators from both regular expressions and from LTL. Hence, $\Lambda_P = \Lambda_R \cup \Lambda_L\{\mapsto\}$, where $\Sigma = 2^{\mathsf{AP}}$, that is the alpahbet for the regular expressions in PSL is the subsets of the propositions in AP.

Abbreviations We mention only the main abbreviations related to the suffix implication operator.

$$r \Mapsto \varphi ::= r \mapsto (true \cdot \varphi)$$
$$r \diamond\!\!\rightarrow \varphi ::= \neg(r \mapsto \neg\varphi)$$
$$r ::= r \diamond\!\!\rightarrow true$$

Semantics Given a word $w \in (2^{\mathsf{AP}})^\omega$ and a corePSL formula φ, we define the relation $w \models_{\text{PSL}} \varphi$ (read "w satisfies φ") similarly to the way it is defined for LTL. We provide the semantics only for the new operator, the suffix implication operator.

$$w \models_{\text{PSL}} r \mapsto \varphi \text{ iff } \forall j \in \mathbb{N},\ (w[0,j) \in [\![r]\!] \implies w[j-1,\infty) \models_{\text{PSL}} \varphi)$$

That is, w satisfies $r \mapsto \varphi$ if *for every* prefix u of w that matches the regular expression r, the suffix of w starting where u ends (with one letter overlap) satisfies the formula φ.

Definition 4. *The satisfaction relation* $\vDash \subseteq (2^{\mathsf{AP}})^{\omega} \times \mathcal{F}_{PSL}$ *is defined inductively on the structure of an PSL formula* φ*. Since, the satisfaction relation for the LTL operators in PSL remain the same—satisfaction relation is defined only for the* $\mapsto$ *operator.*

$$uv^{\omega}[i,\infty) \vDash r \mapsto \varphi \iff \forall j \in \mathbb{N},\ j < |u| + |v| + n|v|\,,\ (uv^{\omega}[i,j) \vdash r \implies uv^{\omega}[j-1,\infty) \vDash \varphi)$$

6.2 The Learning Problem

The learning problem in this case is same as LTL. Hence, the approach taken here is same as the one for LTL. The problem is solved by encoding it as a series of satisfiabilty problem and then using a SAT solver. The formula designed has two parts. The first part encodes the structure of the PSL formula and is similar to the one for LTL, with the added PSL operators and operators for regular expression.

The interesting part is to design $\Phi_n^{\text{consistency}}$, which essentially depends on how the satisfaction relation is defined. There are two type of variables used in the formula.

- $y_{i,p}^{u,v}$ where $0 \leq i \leq |uv| - 1$
- $z_{i,j,p}^{uv^b}$, where $0 \leq i \leq j \leq \left|uv^b\right|$.

Let, $b = n+1$ for simplicity. The variables $y_{i,p}^{u,v}$ ensure satisfaction of the guessed PSL formula with a word uv^{ω} present in sample while the variables $z_{i,j,p}^{uv^b}$ ensure matching of the regular expressions in the formula with certain finite infixes of the word uv^{ω}. Φ_n^w is a disjunction of the formulas described below.

$$\bigwedge_{\substack{1\leq p\leq n\\ 1\leq q,q'\leq p}} x_{p,\nmid} \wedge l_{p,q} \wedge r_{p,q'} \rightarrow \Big[\bigwedge_{0\leq i<|uv|} \Big[y_{i,p}^{u,v} \leftrightarrow \bigwedge_{i\leq j\leq \left|uv^b\right|} \Big[z_{i,j,q}^{uv^b} \rightarrow y_{M(j-1),q'}^{u,v} \Big] \Big] \Big] \tag{6.1}$$

$$\bigwedge_{1\leq p\leq n} x_{p,\varepsilon} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq \left|uv^b\right|} z_{i,j,p}^{uv^b} \leftrightarrow [i=j] \Big] \tag{6.2}$$

$$\bigwedge_{1\leq p\leq n} \bigwedge_{a\in 2^{\mathsf{AP}}} x_{p,a} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq \left|uv^b\right|} \begin{cases} z_{i,j,p}^{uv^b} \text{ if } w[i,j) = a \\ \neg z_{i,j,p}^{uv^b} \text{ if } w[i,j) \neq a \end{cases} \Big] \tag{6.3}$$

$$\bigwedge_{\substack{1\leq p\leq n\\ 1\leq q,q'\leq p}} x_{p,+} \wedge l_{p,q} \wedge r_{p,q'} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq \left|uv^b\right|} \Big[z_{i,j,p}^{uv^b} \leftrightarrow z_{i,j,q}^{uv^b} \vee z_{i,j,q'}^{uv^b} \Big] \Big] \tag{6.4}$$

$$\bigwedge_{\substack{1\leq p\leq n\\ 1\leq q,q'\leq p}} x_{p,\circ} \wedge l_{p,q} \wedge r_{p,q'} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq \left|uv^b\right|} \Big[z_{i,j,p}^{uv^b} \leftrightarrow \bigvee_{i\leq k\leq j} z_{i,k,q}^{uv^b} \wedge z_{k,j,q'}^{uv^b} \Big] \Big] \tag{6.5}$$

$$\bigwedge_{\substack{1\leq p\leq n\\ 1\leq q,q'\leq p}} x_{p,*} \wedge l_{p,q} \rightarrow \Big[\bigwedge_{0\leq i\leq j\leq \left|uv^b\right|} \Big[z_{i,j,p}^{uv^b} \leftrightarrow [i=j] \vee \bigvee_{i<k\leq j} z_{i,k,q}^{uv^b} \wedge z_{k,j,p}^{uv^b} \Big] \Big] \tag{6.6}$$

It can be observed that for operators of regular expressions, we have used formulas which are motivated by Formulas 3.6 to 3.10. Also, $M(j)$ has been defined in 4.9. Finally $\Phi_{\mathcal{S}}^{\text{consistency}}$ is defined as 5.5, as for LTL. In other words, given a sample $\mathcal{S}$, we construct a formula $\Phi_n^{\mathcal{S}}$

Claim 3. *Let $\mathcal{S} = (\mathcal{P}, \mathcal{N})$ be a sample, $n \in \mathbb{N} \setminus \{0\}$, and $\Phi_n^{\mathcal{S}}$ be the propositional formula defined above. Then, the following holds:*

1. *If there exists a PSL formula of size n, $\varphi^{\mathcal{S}}$ that is consistent with $\mathcal{S}$, then the propositional formula $\Phi_n^{\mathcal{S}}$ is satisfiable.*
2. *If $v \models \Phi_n^{\mathcal{S}}$, then φ^v is a PSL formula of size n that is consistent with $\mathcal{S}$.*

The proof has been presented in the Appendix B

Chapter 7

Conclusion

The main problem talked about in this thesis is the following: given a sample $\mathcal{S}$ consisting of two finite sets of positive and negative examples, learn a model $\mathcal{M}$ that is consistent with $\mathcal{S}$. The problem has been looked at for the following models: regular expressions, $\omega-$regular expressions, Linear Temporal Logic, Property Specification Language.

There are certain interesting directions of future work which emerge from the work presented in thesis. An implementation of the theoretical results presented in the thesis using SAT solvers such as Z3 could be of great use. Checking whether the bounds mentioned in the lemmas and theorems of Chapter 4 are actually tight—could have theoritical relevance. Moreover, the learning techniques used in this thesis could potentially be applied to many more models including timed regular expressions[5].

Appendix A

Proofs from Chapter 4

Lemma 5. $uv^\omega[i,\infty) \vdash r_\omega \iff uv^\omega[i,\infty) \in [\![r_\omega]\!]$

Proof. The proof goes via induction on the structure of the the $\omega-$regular expression.

- *Case* $r_\omega = r^\omega$: When $i < |u|$, let $u[i,|u|) = u'$. Now, we need to prove $u'v \vDash r^\omega$When $i \geq |u|$, we use Theorem 1 for the proof.
- *Case* $r_\omega = r \circ r'_\omega$: Consider the following implications:

$$\begin{aligned}
uv^\omega[i,\infty) \vDash r \circ r'_\omega &\iff \exists i \leq j \leq |u| + |v| + n|v|\,,\ uv^\omega[i,j) \vDash r \text{ and } uv^\omega[j,\infty) \vDash r'_\omega \\
&\iff \exists i \leq j \leq |u| + n|v|\,,\ uv^\omega[i,j) \in [\![r]\!] \text{ and } uv^\omega[j,\infty) \in [\![r'_\omega]\!] && \text{[Using ind. hypothesis]} \\
&\iff \exists i \leq j,\ uv^\omega[i,j) \in [\![r]\!] \text{ and } uv^\omega[j,\infty) \in [\![r'_\omega]\!] && \text{[Using Lemma 4]} \\
&\iff v^\omega[i,\infty) \in [\![r \circ r'_\omega]\!]
\end{aligned}$$

  An extra term $|v|$ gets added in the inequality in lines 1 and 2, because $|u| \leq i < |uv|$ in which case the inequality in Lemma 3 gets shifted by $|v|$ factor.

- Case $r_\omega = r'_\omega + r''_\omega$: Consider the following implications:

$$\begin{aligned}
uv^\omega[i,\infty) \vDash r'_\omega + r''_\omega &\iff v^\omega[i,\infty) \vDash r'_\omega \text{ or } uv^\omega[i,\infty) \vDash r''_\omega \\
&\iff v^\omega[i,\infty) \in [\![r'_\omega]\!] \text{ or } uv^\omega[i,\infty) \in [\![r''_\omega]\!] && \text{[Using induction hypothesis]} \\
&\iff uv^\omega[i,\infty) \in [\![r'_\omega + r''_\omega]\!]
\end{aligned}$$

□

Appendix B

Proof of Claim 2:

Proof. Using the syntax DAG of the expression $R^{\mathcal{S}}_{\omega}$ which is indexed using $1 \cdots n$, we formulate a valuation v for the propositional variables in $\Phi^{\mathcal{S}}_n$. We use $R^{\mathcal{S}}_p$ to refer to the expression rooted at the p^{th} node.

- We set $v(x_{p,\lambda}) = 1$ iff the p^{th} node is labelled by λ.
- We set $v(l_{p,q}) = 1$ iff q^{th} node is the left child of the p^{th} node and similarly, $v(r_{p,q}) = 1$ iff q^{th} node is the right child of the p^{th} node.
- We set $v(y^{u,v}_{i,p}) = 1$ iff $uv^{\omega}[i,j) \vDash R^{\mathcal{S}}_p$, when label at p is an infinite operator.
- We set $v(z^{uv^b}_{i,j,p}) = 1$ iff $uv^b[i,j) \vDash R^{\mathcal{S}}_p$, when label at p is a finite operator or an alphabet.
- We set $v(\bar{z}^{uv^b}_{i,j,p}) = 1$ iff $uv^b[i,j) \vDash (R^{\mathcal{S}}_q)^+$, when label at p is an ω operator and q^{th} node is the left child of the p^{th} node.

Firstly, it can be seen that $v \models \Phi^{\text{structure}}_n$, since the formulated valuation ensures the uniqueness of the labels of the nodes as well as that of the left and right children. Moreover, $v \models \Phi^{w}_n$ for $w \in \mathcal{P} \cup \mathcal{N}$, since, the values of the variables $z^{uv^b}_{i,j,p}$ correspond exactly to the valuation of each subexpression $R^{\mathcal{S}}_p$ on the subword $w[i,j)$ for finite operators and alphabets; and the values of the variables $y^{u,v}_{i,p}$ correspond exactly to the valuation of each subexpression $R^{\mathcal{S}}_p$ on the subword $w[i,\infty)$ for infinite operators. Finally, the fact that $R^{\mathcal{S}}_{\omega}$ is consistent with $\mathcal{S}$ implies $v \models y^{u,v}_{0,n}$ for each $w = uv^{\omega} \in \mathcal{P}$ and $v \models \neg y^{u,v}_{0,n}$ for each $w = uv^{\omega} \in \mathcal{N}$. Thus, $v \models \Phi^{\mathcal{S}}_n$, which proves that $\Phi^{\mathcal{S}}_n$ is satisfiable.

For the second statement, since, $v \models \Phi^{S}_n$, we have $v \models \Phi^{\text{structure}}_n$ as well. Hence, the valuation of the variables $x_{p,\lambda}$, $l_{p,q}$, and $r_{p,q}$ encode a syntax DAG from which we obtain the $\omega-$regular expression R^v. Again here, we consider R^v_p to be the subexpression of R^v rooted at the p^{th} node. . Now, it needs to be shown that R^v is indeed consistent with the sample $\mathcal{S}$. For that we show, $v(y^{u,v}_{i,p}) = 1 \iff uv^{\omega}[i,\infty) \vdash R^v_p$ for any $i \in \{0, \cdots, |uv| - 1\}$ for the p^{th} node which is labelled by an infinite operators. For this proof, we consider only those nodes which are labelled by infinite operators. For the finite operators, we need to show $v(z^{uv^b}_{i,j,p}) = 1 \iff uv^b[i,j) \vdash R^v_p$. But, this is exactly done in proof of Claim 1. This can be done using induction on the structure of R^v.

- *Case* $R^v_p = (R^v_q)^{\omega}$: In this case, $v(x_{p,\omega})$ and $v(l_{p,q})$ are set to 1. There could be two cases depending on value of i. In, both the cases we consider that $k \equiv j \mod |v|$.

– When $i < |u|$

$$\begin{aligned} v(y^{u,v}_{i,p}) = 1 &\iff \exists j,k,\ |u| \le j \le k \le \left|uv^b\right|,\ v(\bar{z}^{uv^b}_{i,j,q}) = 1 \text{ and } v(\bar{z}^{uv^b}_{j,k,q}) = 1 && \text{[Using Subform. 4.6]} \\ &\iff \exists j,k,\ |u| \le j \le k \le \left|uv^b\right|,\ uv^b[i,j) \vdash (R^v_q)^+ \text{ and } uv^b[i,j) \vdash (R^v_q)^+ && \text{[Using ind. hypo.]} \\ &\iff (R^v_q)^\omega && \text{[Using Theorem 2]} \end{aligned}$$

– When $i > |u|$

$$\begin{aligned} v(y^{u,v}_{i,p}) = 1 &\iff \exists j,k,\ i \le j \le k \le \left|uv^b\right|,\ v(\bar{z}^{uv^b}_{i,j,q}) = 1 \text{ and } v(\bar{z}^{uv^b}_{j,k,q}) = 1 && \text{[Using Subform. 4.6]} \\ &\iff \exists j,k,\ i \le j \le k \le \left|uv^b\right|,\ uv^b[i,j) \vdash (R^v_q)^+ \text{ and } uv^b[i,j) \vdash (R^v_q)^+ && \text{[Using ind. hypo.]} \\ &\iff (R^v_q)^\omega && \text{[Using Theorem 1]} \end{aligned}$$

In both the cases above, we assumed that $v(\bar{z}^{u,v}_{i,j,q}) = 1$ iff $w[i,j) \vdash (R^v_q)^+$ in the second step, which cannot be deduced from the induction on the structure of R^v_p. This is obtained in the following manner:

$$\begin{aligned} v(\bar{z}^{uv^b}_{i,j,p}) = 1 &\iff \exists k \in \mathbb{N},\ i \le k \le j,\ v(z^{uv^b}_{i,k,q}) = 1 \text{ and } v(\bar{z}^{uv^b}_{k,j,p}) = 1 && \text{[Using Subformula (4.6)]} \\ &\iff \exists k \in \mathbb{N},\ i \le k \le m,\ w[i,k) \vdash R^v_q \text{ and } w[k,j) \vdash (R^v_q)^+ && \text{[Using ind. hypothesis]} \\ &\iff w[i,j) \vdash (R^v_q)^+ && \text{[Using Def. 1]} \end{aligned}$$

Here, $v(\bar{z}^{uv^b}_{k,j,p}) = 1$ iff $w[k,j) \vdash (R^v_q)^+$ requires another level of induction, similar to the last case of the proof of Claim 1.

– *Case* $R^v_p = R^v_q \circ_\omega R^v_{q'}$: In this case, $v(x_{p,\circ_\omega})$, $v(l_{p,q})$, $v(r_{p,q'})$ are all set to 1 and this leads to the following implications:

$$\begin{aligned} v(y^{u,v}_{i,p}) = 1 &\iff \exists k \in \mathbb{N},\ 1 \le j \le |uv^c|, v(z^{u,v}_{i,k,q}) = 1 \text{ and } v(y^{u,v}_{M(j),q'}) = 1 && \text{[Using Subformula 4.7]} \\ &\iff \exists k \in \mathbb{N},\ 1 \le k \le m, w[i,j) \vdash R^v_q \text{ and } w[M(j),\infty) \vdash R^v_{q'} && \text{[Using ind. hypothesis]} \\ &\iff \exists k \in \mathbb{N},\ 1 \le k \le m, w[i,j) \vdash R^v_q \text{ and } w[j,\infty) \vdash R^v_{q'} && \text{[Using Observation 4.1]} \\ &\iff w[i,j) \vdash R^v_q \circ_\omega R^v_{q'} && \text{[Using Def. 2]} \end{aligned}$$

– *Case* $R^v_p = R^v_q +_\omega R^v_{q'}$: In this case, $v(x_{p,+_\omega})$, $v(l_{p,q})$, $v(r_{p,q'})$ are all set to 1, and this leads to the following implications:

$$\begin{aligned} v(y^{u,v}_{i,p}) = 1 &\iff v(y^{u,v}_{i,q}) = 1 \text{ or } v(y^{u,v}_{i,q'}) = 1 && \text{[Using Subformula 4.8]} \\ &\iff w[i,\infty) \vdash R^v_q \text{ or } w[i,\infty) \vdash R^v_{q'} && \text{[Using ind. hypothesis]} \\ &\iff w[i,j) \vdash R^v_q +_\omega R^v_{q'} && \text{[Using Def. 2]} \end{aligned}$$

□

Proof of Claim 3

Proof. Using the syntax DAG of the expression $\varphi^{\mathcal{S}}$ which is indexed using $1 \cdots n$, we formulate a valuation v for the propositional variables in $\Phi^{\mathcal{S}}_n$. We use $\varphi^{\mathcal{S}}_p$ to refer to the formula rooted at the p^{th} node.

– We set $v(x_{p,\lambda}) = 1$ iff the p^{th} node is labelled by λ.

– We set $v(l_{p,q}) = 1$ iff q^{th} node is the left child of the p^{th} node and similarly, $v(r_{p,q}) = 1$ iff q^{th} node is the right child of the p^{th} node.

– We set $v(y^{u,v}_{i,p}) = 1$ iff $uv^\omega[i,\infty) \vdash \varphi^{\mathcal{S}}_p$, when label at p is an LTL operator or a triggers($\mid\!\!\rightarrow$) operator or a proposition in AP.

– We set $v(z^{uv^b}_{i,j,p}) = 1$ iff $uv^b[i,j) \vdash \varphi^{\mathcal{S}}_p$, when label at p is a operator from regular expressions or a letter of the alphabet 2^{AP}.

Firstly, it can be seen that $v \models \Phi^{\text{structure}}_n$, since the formulated valuation ensures the uniqueness of the labels of the nodes as well as that of the left and right children. Moreover, $v \models \Phi^w_n$ for $w \in \mathcal{P} \cup \mathcal{N}$, since, the values of the variables $z^{uv^b}_{i,j,p}$ correspond exactly to the valuation of each subformula $\varphi^{\mathcal{S}}_p$ on the subword $uv^b[i,j)$ for $\mid\!\!\rightarrow$, LTL operators and letters of alphabet 2^{AP}; and the values of the variables $y^{u,v}_{i,p}$ correspond exactly to the valuation of each subformula $\varphi^{\mathcal{S}}_p$ on the subword $uv^\omega[i,\infty)$ for $\mid\!\!\rightarrow$, LTL operators or a proposition in AP. Finally, the fact that $\varphi^{\mathcal{S}}$ is consistent with $\mathcal{S}$ implies $v \models y^{u,v}_{0,n}$ for each $w = uv^\omega \in \mathcal{P}$ and $v \models \neg y^{u,v}_{0,n}$ for each $w = uv^\omega \in \mathcal{N}$. Thus, $v \models \Phi^{\mathcal{S}}_n$, which proves that $\Phi^{\mathcal{S}}_n$ is satisfiable.

For the second statement, since, $v \models \Phi^S_n$, we have $v \models \Phi^{\text{structure}}_n$ as well. Hence, the valuation of the variables $x_{p,\lambda}$, $l_{p,q}$, and $r_{p,q}$ encode a syntax DAG from which we obtain the PSL formula φ^v. Again here, we consider φ^v_p to be the subformula of φ^v rooted at the p^{th} node. Now, it needs to be shown that φ^v is indeed consistent with the sample $\mathcal{S}$. For that we show, $v(y^{u,v}_{i,p}) = 1 \iff uv^\omega[i,\infty) \vdash \varphi^v_p$ for any $i \in \{0, \cdots, |uv|-1\}$ for the p^{th} node which is labelled by $\mid\!\!\rightarrow$, LTL operators or a proposition in AP. For the regular operators and its corresponding alphabet, we need to show $v(z^{uv^b}_{i,j,p}) = 1 \iff uv^b[i,j) \vdash \varphi^v_p$. But, this is exactly done in proof of Claim 1. For this proof, we consider only those nodes which are labelled by only the $\mid\!\!\rightarrow$ operator. This can be done using induction on the structure of φ^v.

– *Base case* $\varphi^v_p = \mathfrak{p}$: In this case, $v(x_{p,\mathfrak{p}}) = 1$, which leads to the following implications:

$$\begin{aligned} v(y^{u,v}_{i,p}) = 1 &\iff \mathfrak{p} \in uv^\omega[i,i) && \text{[Using Subformula (5.1)]} \\ &\iff uv^\omega[i,j) \vDash \mathfrak{p} && \text{[Using Def. 3]} \end{aligned}$$

– *Case* $\varphi^v_p = \varphi^v_q \vee \varphi^v_{q'}$: In this case, $v(x_{p,\vee})$, $v(l_{p,q})$, $v(r_{p,q'})$ are all set to 1, and this leads to the following implications:

$$\begin{aligned} v(y^{u,v}_{i,p}) = 1 &\iff v(y^{u,v}_{i,q}) = 1 \text{ or } v(y^{u,v}_{i,q'}) = 1 && \text{[Using Subformula (5.2)]} \\ &\iff uv^\omega[i,\infty) \vDash \varphi^v_q \text{ or } uv^\omega[i,\infty) \vDash \varphi^v_{q'} && \text{[Using ind. hypothesis]} \\ &\iff uv^\omega[i,\infty) \vDash \varphi^v_q \vee \varphi^v_{q'} && \text{[Using Def. 3]} \end{aligned}$$

– *Case* $\varphi^v_p = \boldsymbol{X}(\varphi^v_q)$: In this case, $v(x_{p,\mathbf{X}})$ and $v(l_{p,q})$ are set to 1 and this leads to the following

implications:

$$
\begin{aligned}
v(y^{u,v}_{i,p}) = 1 &\iff \begin{cases} v(y^{u,v}_{i+1,q}) = 1 \text{ for } 0 \leq i < |uv| - 1 \\ v(y^{u,v}_{|u|,q}) = 1 \text{ for } i = |uv| - 1 \end{cases} && \text{[Using Subformula (5.3)]} \\
&\iff \begin{cases} uv^\omega[i+1, \infty) \vDash \varphi^v_q \text{ for } 0 \leq i < |uv| - 1 \\ uv^\omega[\![|u|, \infty) \vDash \varphi^v_q \text{ for } i = |uv| - 1 \end{cases} && \text{[Using ind. hypothesis]} \\
&\iff uv^\omega[i, \infty) \vDash \mathbf{X}(\varphi^v_q) && \text{[Using Def. 3]}
\end{aligned}
$$

– *Case* $\varphi^v_p = \varphi^v_q \boldsymbol{U} \varphi^v_{q'}$: In this case, $v(x_{p,\mathbf{U}})$, $v(l_{p,q})$, $v(r_{p,q'})$ are all set to 1, and this leads to the following implications:

$$
\begin{aligned}
v(y^{u,v}_{i,p}) = 1 &\iff \begin{cases} \exists j,\ i \leq j \leq |uv| - 1,\ v(y^{u,v}_{j,q'}) = 1 \text{ and } \forall k, i \leq k < j,\ v(y^{u,v}_{k,q}) = 1 \text{ for } i < |u| & \text{[Using} \\ \exists j,\ |u| \leq j < |uv|,\ v(y^{u,v}_{j,q'}) = 1 \text{ and } \forall k, k \in \mathcal{I}^{u,v}(i,j),\ v(y^{u,v}_{k,q}) = 1 \text{ for } i \geq |u| & \text{Subform. (5.4)]} \end{cases} \\
&\iff \begin{cases} \exists j,\ i \leq j \leq |uv| - 1,\ uv^\omega[j, \infty) \vDash \varphi^v_{q'} \text{ and } \forall k, i \leq k < j,\ uv^\omega[k, \infty) \vDash \varphi^v_q \text{ for } i < |u| & \text{[Using} \\ \exists j,\ |u| \leq j < |uv|,\ uv^\omega[j, \infty) \vDash \varphi^v_{q'} \text{ and } \forall k, k \in \mathcal{I}^{u,v}(i,j),\ uv^\omega[k, \infty) \vDash \varphi^v_q \text{ for } i \geq |u| & \text{ind. hyp} \end{cases} \\
&\iff uv^\omega[i, j) \vDash \varphi^v_q \mathbf{U} \varphi^v_{q'}
\end{aligned}
$$

– *Case* $\varphi^v_p = \varphi^v_q \mapsto \varphi^v_{q'}$: In this case, $v(x_{p,\Rightarrow})$, $v(l_{p,q})$, $v(r_{p,q'})$ are all set to 1, and this leads to the following implications:

$$
\begin{aligned}
v(y^{u,v}_{i,p}) = 1 &\iff \forall j,\ i \leq j \leq \left|uv^b\right|,\ v(z^{uv^b}_{i,j,q}) = 1 \implies v(y^{u,v}_{M(j-1),q'}) = 1 && \text{[Using Subformula (6.} \\
&\iff \forall j,\ i \leq j \leq |u| + |v| + n|v|,\ uv^b[i, j) \vdash \varphi^v_q \implies uv^\omega[M(j-1), q') \vDash \varphi'_q && \text{[Using ind. hypothesi} \\
&\iff \forall j,\ i \leq j \leq |u| + |v| + n|v|,\ uv^b[i, j) \vdash \varphi^v_q \implies uv^\omega[j-1, q') \vDash \varphi'_q && \text{[Using Observation 4.} \\
&\iff uv^\omega[i, \infty) \vDash \varphi^v_q \mapsto \varphi^v_{q'} && \text{[Using Def. 3]}
\end{aligned}
$$

Here, we assumed that $v(y^w_{k,j,p}) = 1$ iff $w[k, j) \vDash \varphi^v_p$ in the second step, which cannot be deduced from the induction on the structure of φ^v_p. This we obtained using another level of induction, which is on the length of the subword that matches φ^v_p. More precisely, we induct on k which ranges from j to $i + 1$, where by induction hypothesis, it is assumed that

$$v(y^w_{k,j,p}) = 1 \iff w[k, j) \vDash \varphi^v_p \quad \forall k,\ i < k \leq j$$

The base case occurs when $k = j$, and then, we have $v(y^w_{k,j,p}) = 1 \iff k = j \iff w[k, j) \vDash \varphi^v_p$.

□

Bibliography

[1] Alves, T., Teresa Martins, A., and Martins Ferreira, F. On finding a first-order sentence consistent with a sample of strings. vol. 277, pp. 220–234.

[2] Armoni, R., Fix, L., Flaisher, A., Gerth, R., Ginsburg, B., Kanza, T., Landver, A., Mador-Haim, S., Singerman, E., Tiemeyer, A., Vardi, M. Y., and Zbar, Y. The forspec temporal logic: A new temporal property-specification language. In *TACAS* (2002).

[3] Baier, C., and Katoen, J.-P. *Principles of Model Checking*, vol. 26202649. 01 2008.

[4] Eisner, C., and Fisman, D. *A Practical Introduction to PSL*. 07 2006.

[5] Narayan, A., Cutulenco, G., Joshi, Y., and Fischmeister, S. Mining timed regular specifications from system traces. *ACM Trans. Embed. Comput. Syst. 17*, 2 (Jan. 2018), 46:1–46:21.

[6] Neider, D., and Gavran, I. Learning linear temporal properties. In *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018* (2018), pp. 1–10.

[7] Pnueli, A. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1977), SFCS '77, IEEE Computer Society, pp. 46–57.

[8] Wolper, P. Temporal logic can be more expressive. In *Proceedings of the 22Nd Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1981), SFCS '81, IEEE Computer Society, pp. 340–348.