Learning Interpretable Models in the Property Specification Language

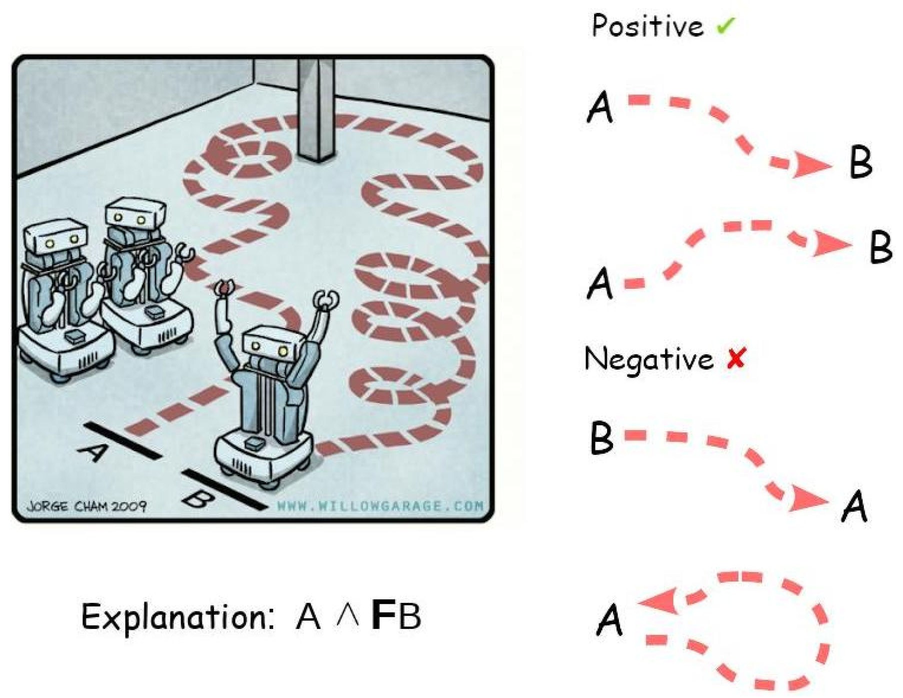
Max Planck Institute for Software Systems [1]

Rajarshi Roy [1], Dana Fisman [2], Daniel Neider [1]

Ben-Gurion University of the Negev [2]

Abstract

- Our goal is to provide explanations of black-box systems using human-interpretable models.
- We provide explanations of black-box system by observing their behavior and providing models in IEEE standard temporal logic: **Property Specification Language** (PSL) to describe them.



Property Specification Language (PSL)*

PSL is an extension of **Linear Temporal Logic** (LTL) with the **triggers** operator, one of whose operands is a **Regular expression**.

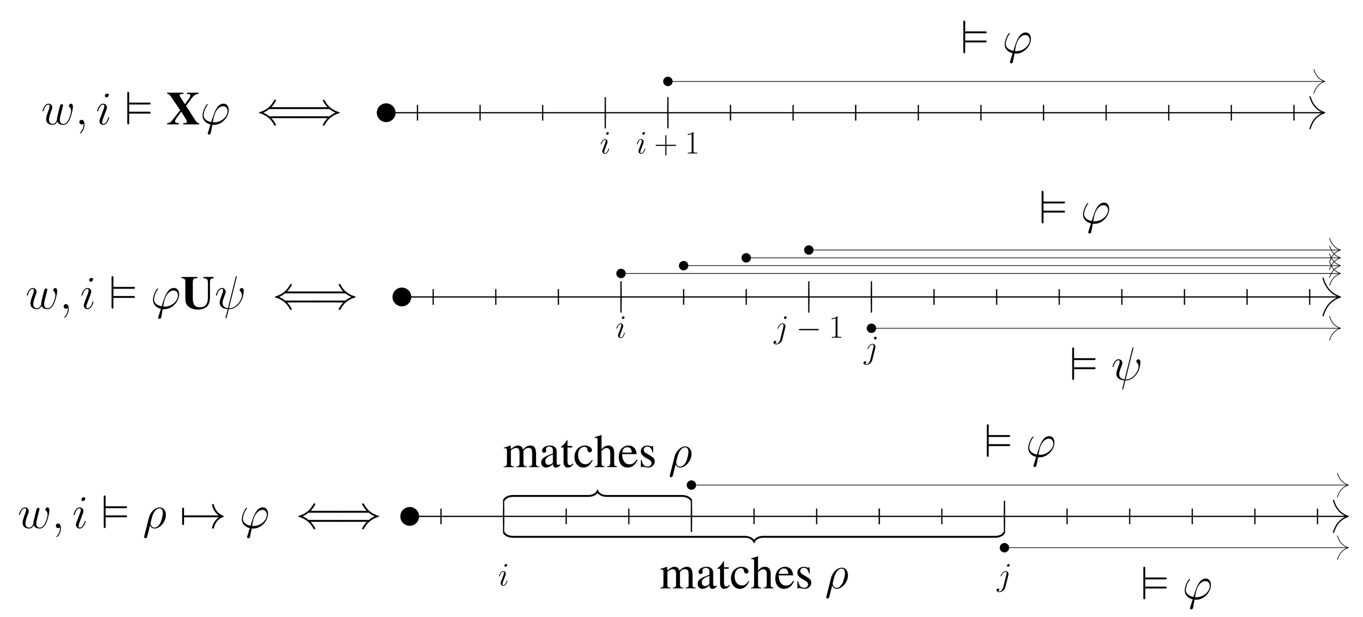
Syntax:

$$\varphi ::= p \in \mathcal{P} \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi_1 \mathbf{U} \varphi_2 \mid \rho \mapsto \varphi,$$

where **X** and **U** are standard temporal operators and ρ denotes a regular expression.

Semantics:

- PSL formulas are interpreted over infinite words which represent system traces.
- The semantics of boolean operators are defined in a usual manner.
- The semantics of temporal operators **X**, **U**, $\mapsto$ is defined as follows:



* We only consider the core fragment of PSL, whose expressive power is equivalent to that of the entire class.

Why PSL?

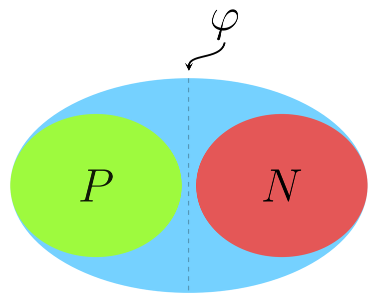
- PSL integrates **easy-to-understand** regular expressions in its syntax
- The expressive power of PSL is that of the full class of **regular omega-languages**;
- One can provide **concise descriptions** of system behavior using models in PSL. (PSL can often be more succinct than LTL while describing similar a given system behavior)

Problem Statement

Input: $\mathcal{S} = (P, N)$, where P and N consist of positive and negative words resp. All words are *ultimately periodic*, that is of the form uv^ω.

Problem: Find a minimal PSL formula φ *consistent* with $\mathcal{S}$ in that:

- All positive words $w \in P$ satisfy φ; and
- None of the negative words $w \in N$ satisfy φ.

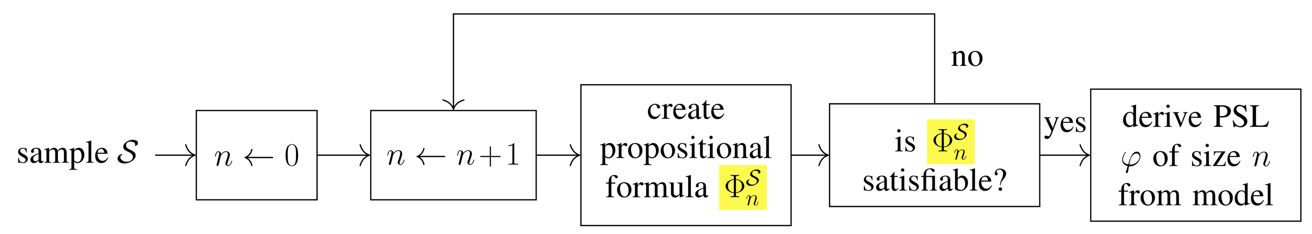


Solution approach

Given $\mathcal{S}$, we encode the problem in SAT using a series of propositional formula $(\Phi_n^{\mathcal{S}})_{n=1,2,\cdots}$, such that

1. $\Phi_n^{\mathcal{S}}$ is satisfiable $\iff$ $\exists$ a PSL formula φ of size n that is consistent with $\mathcal{S}$
2. A model of $\Phi_n^{\mathcal{S}}$ contains enough information to extract a consistent PSL formula of size n

The framework of the algorithm that we follow is depicted below:



The SAT encoding

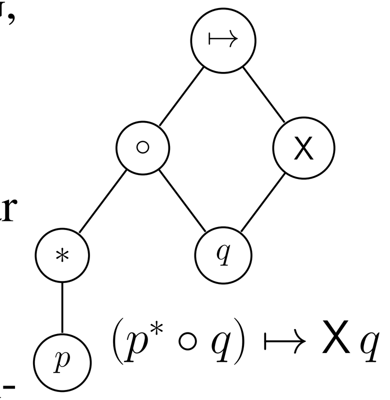
There are two parts to the SAT-encoding $\Phi_n^{\mathcal{S}}$:

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

- $\Phi_n^{structure}$: encodes structure of PSL formulas.
- $\Phi_n^{consistency}$ checks for consistency with sample

Constraints

Structural constraints: PSL formulas can be represented using tree-like structures know as syntax DAG. For example, the syntax DAG for the formula $(p \circ q) \mapsto \mathsf{X}\, q$ is as follows:



$\Phi_n^{structure}$ consists of constraints to encode such a syntax DAG, expressible in propositional logic, such as:

- each node should be labeled with one operator;
- each node should have at least one left child and similar constraints that

Constraints for consistency: $\Phi_n^{consistency}$ consists of constraints, again expressible in propositional logic, that

- track the satisfaction of PSL formula on every position of words in sample using the semantics of PSL operators (this step involves figuring out how to reduce satisfaction of PSL on infinite words to satisfaction of PSL on finite words).
- ensure that positive words are satisfied and negative words are not satisfied.

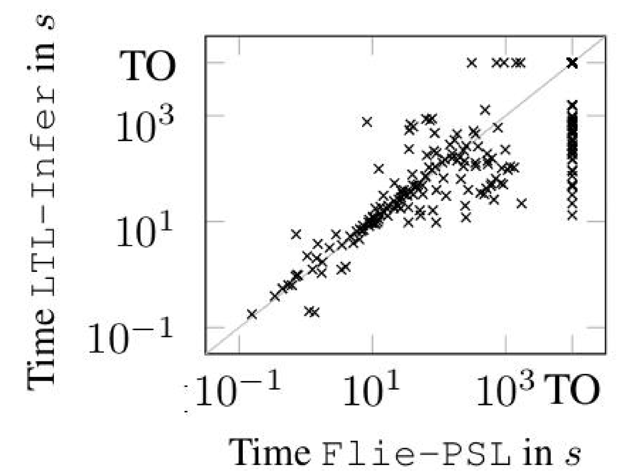
Theoritical Guarentees

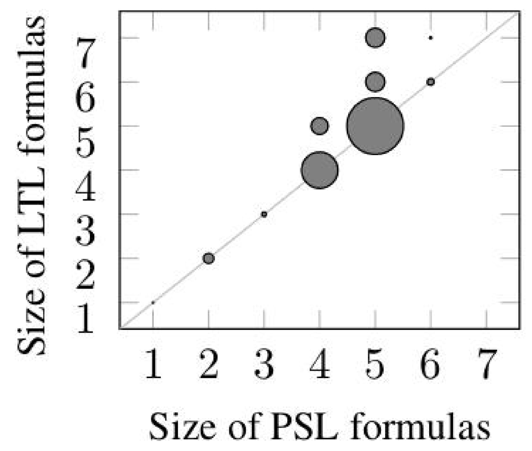
Theorem: Given sample $\mathcal{S}$, the learning algorithm terminates and outputs a minimal PSL formula that is consistent with $\mathcal{S}$.

Corollary: Since PSL subsumes regular expressions in its syntax, with simple modfication the learning algorithm infers minimal regular expression.

Empirical evaluation

- We implemented a prototype `Flie-PSL` of the learning algorithm in python using Z3 as a SAT solver.
- We have also compared it against state-of-the-art tool `LTL-Infer` for inferring LTL by Neider and Gavran.
- One of the benchmark suites used is synthetic data derived from common PSL formulas appearing in practice.
- Out of the 390 benchmarks, `Flie-PSL` ran faster than `LTL-Infer` in 25 benchmarks and inferred a smaller formula on 52 benchmarks.





Contact Info: rajarshi,neider@mpi-sws.org, dana@cs.bgu.ac.il