MONASH University

Formal Explainability for Artificial Intelligence in Dynamic Environments

Jime Cuartas Granada

Supervisors:
Alexey Ignatiev
Peter J. Stuckey
Julian Gutierrez

A Progress Review report at
Monash University in 2026
School of Information Technology

Abstract

In dynamic environments, a goal of Artificial Intelligence (AI) is to build intelligent agents capable of addressing sequential decision-making settings. In this context, there are two important challenges for humans to understand decisions made by agents: (1) the sequential decisions are connected, (2) the environment can play a role in the outcome and (3) the agents may use opaque black-box models for each decision.

Despite the success of AI in sequential decision-making (e.g. Reinforcement Learning), the lack of transparency in understanding their decisions can make the agents hard to validate. To address the need for transparency, there are efforts to develop Explainable Artificial Intelligence (XAI). XAI is a set of methods designed to make AI models easier to comprehend. Despite the importance of Explainable Reinforcement Learning in developing trustworthy intelligent agents, there are gaps in current research to make sequential decision-making explainable.

This project proposes to explain sequential decision-making using formal reasoning. To achieve this goal, the proposal focuses on (1) Formal Explainability for Finite Automata, to address sequential actions in deterministic environments, (2) Formal Explainability for Context-Free Grammar, to address sequential actions with stack-based memory, and (3) Formal Explainability for stochastic models, where outcomes are subject to environmental uncertainty.

Contents

Chapter 1

Introduction

The deployment of Artificial Intelligence (AI) algorithms has necessitated the need for eXplainability AI (XAI) methods to ensure transparency, trust, and accountability. While much of the field has focused on heuristic explanations for opaque models, there is an interest in formal approaches that provide rigorous guarantees about the explanations generated [1, 2].

A fundamental challenge in dynamic environments is explaining sequential decision-making. To address this, we model these processes using Automata, which provide a symbolic and tractable representation of sequential decision functions. This approach allows us to generate formal explanations, why a specific sequence of actions leads to a particular outcome. Automata are widely used in software verification [3], design of communication protocols [4], and syntax parsing in compiler [5]. When a computational model, such as a Finite Automaton (FA) or a Pushdown Automaton (PDA), accepts or rejects an input string, the reasoning behind that decision can be non-trivial. Understanding why a specific input was accepted or rejected is crucial for debugging, and refinement purposes.

This research project investigates the formalisation of explanations for sequential decision-making. Having developed an approach to deliver formal explanations for Finite Automata (FA) during the first stage of this research, submitted to ICALP 2026. I now move to address explanations for Context-Free Grammar (CFG) decisions.

Current literature focuses on the *performance* of parsing rather than the *interpretability* of the decision. Modern parsing algorithms, such as Tree-sitter [6] and ANTLR 4 [7], utilise sophisticated incremental parsing and adaptive LL(*) algorithms to ensure low-latency feedback for integrated development environments (IDEs). While these methods are efficient at error recovery [5, 8], often scaling linearly with input size, they treat the decision process as the main objective. In this work, we prioritise interpretability by formally defining and computing the minimal reasons for a parsing error with minimal sets of corrections required to resolve it.

1.1 Formal Explainability

In this chapter I present some important concepts of Formal Explainability that are foundational for this work, two types of explanations widely used in classification problems, and a brief illustration of our work developed during my first year to explain Finite Automata Decisions.

1.1.1 Classification Problems and Formal Explanations

Following XAI literature for classification problems [2, 9], we consider a classifier $\kappa : \mathbb{F} \to \mathcal{K}$ over a feature space $\mathbb{F} = \prod_{i=1}^{m} \mathbb{D}_i$ and a set of classes $\mathcal{K}$.

For an instance $\mathbf{v} \in \mathbb{F}$ with prediction $\kappa(\mathbf{v}) = c \in \mathcal{K}$, an *abductive explanation* is a minimal subset of features $\mathcal{X} \subseteq \{1, \ldots, m\}$ *sufficient* for the prediction. Formally, $\mathcal{X}$ is defined as:

$$\forall(\mathbf{x} \in \mathbb{F}). \left[\bigwedge\nolimits_{i \in \mathcal{X}} (x_i = v_i)\right] \rightarrow (\kappa(\mathbf{x}) = c) \tag{1.1}$$

Similarly, a *contrastive explanation* (CXp) is a minimal subset of features $\mathcal{Y} \subseteq \{1, \ldots, m\}$ that, if allowed to change, enables the prediction's alteration. Formally, a contrastive explanation $\mathcal{Y}$ is defined as follows:

$$\exists(\mathbf{x} \in \mathbb{F}). \left[\bigwedge\nolimits_{j \notin \mathcal{Y}} (x_j = v_j)\right] \wedge (\kappa(\mathbf{x}) \neq c) \tag{1.2}$$

Observe that abductive explanations are used to explain *why* a prediction is made by the classifier κ for a given instance, while contrastive explanations can be seen to answer *why not* another prediction is made by κ. Alternatively, CXps can be seen as answering *how* the prediction can be changed.

Importantly, abductive and contrastive explanations are known to enjoy a minimal hitting set duality relationship [10]. Given $\kappa(\mathbf{v}) = c$, let $\mathbb{A}_{\mathbf{v}}$ be the complete set of AXps and $\mathbb{C}_{\mathbf{v}}$ be the complete set of CXps for this prediction. Then each AXp $\mathcal{X} \in \mathbb{A}_{\mathbf{v}}$ is a minimal hitting set of $\mathbb{C}_{\mathbf{v}}$ and, vice versa, each CXp $\mathcal{Y} \in \mathbb{C}_{\mathbf{v}}$ is a minimal hitting set of $\mathbb{A}_{\mathbf{v}}$.[1] This fact is the basis for the algorithms used for formal explanation *enumeration* [9, 11].

1.1.2 Explaining Finite Automata Decisions

Here, we adopt standard definitions and notations for *finite automata* and *regular expressions* [12–14]. $\emptyset$ denotes the empty language, $c \in \Sigma$ denotes the language $\{c\}$, and Σ denotes the language containing the wildcard character (representing any single character), the set of all strings of length 1. In general, given a regular expression R, the language it defines is denoted by $L(R)$. The concatenation of two regular expressions $R_1 R_2$ denotes the regular language $\{r_1 r_2 \mid r_1 \in L(R_1), r_2 \in L(R_2)\}$.

[1] Given a collection of sets $\mathbb{S}$, a *hitting set* of $\mathbb{S}$ is a set H such that for each $S \in \mathbb{S}$, $H \cap S \neq \emptyset$. A hitting set is *minimal* if no proper subset of it is a hitting set.

In this work, we aim to explain the behaviour of a finite automaton $\mathcal{A}$ on an input $w \in \Sigma^*$, which can be viewed as a classifier mapping input w to a class in $\mathcal{K} = \{\text{accept}, \text{reject}\}$. Similar to classification problems, we propose two types of explanations for FA: abductive explanations (AXps) and contrastive explanations (CXps). Informally, an AXp answers "Why does $\mathcal{A}$ accept/reject w?" while a CXp answers "How can w be modified to alter the response of $\mathcal{A}$?".

In that work, we introduced a class of explanation languages based on regular expressions. They are formed by replacing selected characters in w with Σ (representing any character).

Example 1.1. *Consider the deterministic FA $\mathcal{A}$ shown in Figure 1.1. Observe that it accepts the input word* `bbbbb`*. How should we explain this? One obvious way is to trace the execution of the automaton on the input word. Clearly, reading the first two* `b`*'s leads to an accepting state, so an explanation could be* `bb`$\Sigma\Sigma\Sigma$ *($L(\texttt{bb}\Sigma\Sigma\Sigma) \subseteq L(\mathcal{A})$), where the underlined characters are those that explain the acceptance. However, other (shorter) explanations exist, e.g.* `bb`Σ`bb` *($L(\Sigma\Sigma\texttt{b}\Sigma\Sigma) \subseteq L(\mathcal{A})$) is a correct explanation, as any word of length 5 with a* `b` *in position 3 will be accepted. Once we have one type of explanation, we can easily extract the other using Minimal Hitting sets, as illustrated in Figure 1.2. AXps fix indices while CXps free them, revealing a minimal set where it is possible to flip the prediction (obtain a rejected word).* □

Formal definitions and more details about this and other explanation languages can be found in the paper at the end of this report.

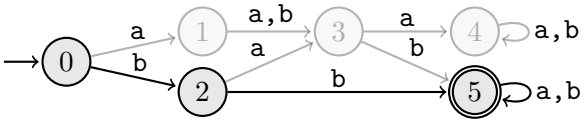


FIGURE 1.1: Input `bbbbb` is accepted by this automaton. Opaque states indicate transitions that are not traversed for the input `bbbbb`. Two valid explanations are `bb`$\Sigma\Sigma\Sigma$ and (a shorter one) `bb`Σ`bb`.

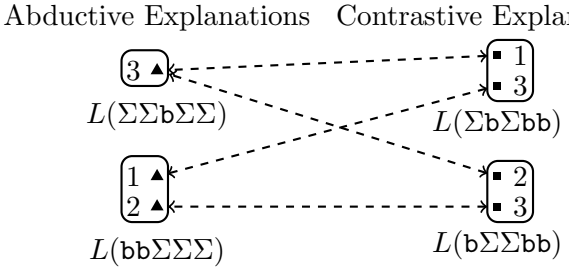


FIGURE 1.2: Duality between AXps and CXps in Σ. AXps fix the characters at given indices; CXps free them.

Chapter 2

Refined scope

While standard XAI focuses on feature attribution in classifiers, the "features" in formal languages are sequential and structural. Since the confirmation report, the research scope has been refined to address three primary gaps:

- **Research Problem 1 (Completed): Explaining Finite Automata.**
  How can we provide Formal Explanations for Finite Automata decisions?
  - To Define Formal explanations for understanding the acceptance/rejection of inputs in Finite Automata
  - To develop a method to compute Formal Explanations for Finite Automata
- **Research Problem 2: Explaining Context-Free Grammar (CFG).**
  How can we provide Formal Explanations for CFG decisions?
  - To Define Formal explanations for understanding the recognition of inputs by CFGs.
  - To develop a method to compute Formal Explanations for CFGs.
  - To propose a method to identify the most significant or "most likely" explanations.
- **Research Problem 3: Explaining Decisions of Stochastic models.**
  How can we provide formal explanations for decisions made by stochastic models?
  - To Define Formal explanations for understanding probable outcomes in Stochastic Models.
  - To develop a method to compute Formal Explanations for Stochastic Models (e.g., Probabilistic Finite Automata or Markov Models).
  - To propose a method to identify the most significant or "most likely" explanations among probabilistic paths.

2.1 Contributions - achieved and projected

This research provides both theoretical and practical contributions:

Achieved Contributions:

- Development of a theoretical and practical approach to explain Finite Automata.

- A paper submitted to ICALP 2026 titled "A Formal Framework for the Explanation of Finite Automata Decisions"

Projected Contributions:

- Explaining Context-Free Grammar: (In Progress) Extending the formal explanation framework to CFGs, which recognise Context-Free Languages. This involves developing algorithms to identify the minimal contrastive explanations (CXps) and Abductive Explanations (AXps), and quantifying the contribution of specific tokens to the decision (acceptance or rejection).
- Explaining Decisions of Stochastic models: Extending the formal explanation framework to Stochastic Models. The goal is to provide verifiable explanations able to identify the environmental factors or decision points that lead to a particular outcome.

Chapter 3

Progress

3.1 Model Proposal: Context-Free Grammar (CFG) Explanations

This work evolves from the study of Finite Automata (FA) to more expressive computational models. While FAs provide a baseline for explaining sequential behaviour, many complex problems require the model to "remember" an arbitrary number of previous inputs in the sequence to determine the validity of subsequent inputs.

Consider the abstract language $L = \{a^n b^n \mid n \geq 1\}$, which represents a sequence where every 'a' must be matched by a corresponding 'b'.

- The Limitation: A standard Finite Automaton (FA) possesses no stack-based memory. Therefore, it cannot count the number of a's to ensure they match the number of b's once n exceeds the number of states in the machine [5].
- The Explanation Failure: A Context-Free Grammar (CFG) can describe the language L, standard parsing algorithms only recognise whether an input is valid. Upon encountering a mismatch (e.g., a^4b^2), the parser identifies a failure but lacks the mechanisms to generate a contrastive explanation such as: "The sequence is invalid because the third or fourth 'a' was not closed by a matching 'b'." Instead, it can only report a "transition failure" at the specific index, obscuring the root cause of the error.
- **From Regular to Context-Free Languages:** We propose the use of Context-Free Languages to model decision-making processes that require stack-based memory.
  Unlike FAs, which are limited to Regular Languages, CFGs allow the description of more complex languages. The core challenge is how to explain these CFG decisions.

3.2 Motivation: The Gap between Detection and Explanation

One well established application of Context-Free Grammars is in the design of programming languages and compilers. Compilers are highly efficient at detecting when a sequence of tokens fails to belong to a grammar. However, a fundamental question is: are they able of generating a useful *explanation* for why the failure occurred or how to fix it?

Consider the following C code, where a typo has introduced a double opening bracket {{ in the `for` loop:

```
1 int main(){
2     for(int i=0; i<10; i++){{   // <--- Error: Double bracket
3         printf("hello");
4     }
5 }
```

When compiled (e.g., using GCC or Clang), the parser consumes the input until it reaches <<EOF>>(end-of-file), finding only then that EOF was not expected, instead there is an incomplete structure, the token '}' is expected before <<EOF>>. The resulting error message is:

```
error: expected '}' at end of input
   5 | }
     |  ^
```

The Explanation: While the compiler's output is correct, the file ended while the stack still contained an open brace that was not closed yet. It is *misleading*.

- **Root Cause:** The compiler points to line 5 (the end of the file) as the location of the error. However, the root cause is located at line 2.
- **Lack of Contrastive Reasoning:** A human (or a formal explainer) would identify that the input is "almost correct". The explanation should not just report a missing symbol at the end but rather propose a *minimal correction*.

In this case, the minimal correction is not to add a brace at the end, but to remove the redundant opening brace at the loop initialisation:

```
1 int main(){
2     for(int i=0; i<10; i++){
3         printf("hello");
4     }
5 }
```

This discrepancy motivates the need for our proposed formal framework. We aim to move beyond isolated decisions to explained decisions revealing these minimal set of edits (Contrastive Explanations).

3.3 Preliminaries

This section defines the core concepts utilised throughout this ongoing work.

Definition 3.1 (Context-Free Grammar)**.** A Context-Free Grammar (CFG) is defined as a 4-tuple $G = (V, \Sigma, R, S)$, where:

- V (Variables/Non-terminals) is a finite set of variables (non-terminal symbols).
- Σ (Terminals) is a finite set of terminal symbols, disjoint from V.
- R is a finite set of production rules of the form $A \to \alpha$, where $A \in V$ describes a variable and $\alpha \in (V \cup \Sigma)^*$ is a string of variables and terminals.
- $S \in V$ is the start variable.

The following CFG recognises the language of balanced parentheses (a subset of the Dyck language) [1] and is used throughout the document to illustrate the proposed ideas.

Example 3.1. *Let $G = (\{Balanced\}, \{(,)\}, R, Balanced)$ be the grammar defined by the following production rules R:*

$$\begin{aligned} &Balanced \to (\ Balanced\ )\ Balanced && \textit{(Rule 1)} \\ &Balanced \to (\ Balanced\ ) && \textit{(Rule 2)} \\ &Balanced \to (\ )\ Balanced && \textit{(Rule 3)} \\ &Balanced \to (\ ) && \textit{(Rule 4)} \end{aligned} \tag{3.1}$$

This grammar generates the language of properly nested parentheses. For instance, the string ()() can be derived as follows.

$$\begin{aligned} Balanced &\Rightarrow (\ )\ \textbf{Balanced} && \textit{(Rule 3: parentheses and Balanced)} \\ &\Rightarrow (\ )\ (\ ) && \textit{(Rule 4: Reduce 'Balanced' to '()')} \end{aligned} \tag{3.2}$$

To analyse a CFG, we must first standardise the grammar structure (e.g. Chomsky Normal Form). This enables the use of efficient parsing algorithms like CYK (Cocke-Younger-Kasami) [16, 17].

3.3.1 Chomsky Normal Form

Parsing algorithms often require the grammar to be in a canonical form to ensure predictable execution complexity.

Definition 3.2 (Chomsky Normal Form)**.** A Context-Free Grammar $G = (V, \Sigma, R, S)$ is in *Chomsky Normal Form* (CNF) [18] if every production rule in R is of one of the following two forms:

- $A \to BC$ (where $A, B, C \in V$)
- $A \to a$ (where $a \in \Sigma$ and $a \neq \epsilon$)

[1] The Dyck language describes a set of strings with balanced and properly nested brackets (e.g., (), [], {}) [15]. The example focuses solely on non-empty sequences of balanced (and).

For every CFG G whose language contains at least one string other than ϵ, then there is a grammar G_1 in Chomsky Normal Form, such that $L(G_1) = L(G) \setminus \{\epsilon\}$.

Example 3.2. *Consider the grammar G from Example 3.1. We transform G into an equivalent grammar $G' = (V', \Sigma, R', S)$ in CNF.*

Step 1: Terminals to Non-terminals. *Introduce variables L and R for terminals '(' and ')'.*

$$L \to (\quad \textit{and} \quad R \to)$$

Step 2: Binary decomposition. *Rewrite the original rules using new variables and break down productions into binary steps.*

The resulting production rules R' are:

$$\begin{aligned}
\textit{Balanced} &\to \textit{Nested Balanced} && \textit{(Represents '(Balanced) Balanced')}\\
&\mid \textit{Unclosed } R && \textit{(Represents '(Balanced)')}\\
&\mid \textit{Pair Balanced} && \textit{(Represents '( ) Balanced')}\\
&\mid L\,R && \textit{(Represents '( )')}\\
\textit{Nested} &\to \textit{Unclosed } R\\
\textit{Unclosed} &\to L\,\textit{Balanced}\\
\textit{Pair} &\to L\,R\\
L &\to (\\
R &\to)
\end{aligned}$$

3.3.2 The CYK Algorithm

The Cocke-Younger-Kasami (CYK) [16, 17] algorithm is a bottom-up parsing method that given a CFG G in CNF determines whether a string w belongs to a language $L(G)$. It operates via dynamic programming, constructing a triangular table where each cell $T[i, j]$ contains the set of non-terminals that can generate the substring of w starting at i and ending at j.

Definition 3.3 (CYK Table Construction)**.** For an input string $w = w_1 w_2 \dots w_n$:

1. **Base Case:** For each $i \in \{1, \dots, n\}$, $T[i, i] = \{A \in V \mid A \to w_i\}$.
2. **Recursive Step ($j > i$):** $T[i, j]$ contains A if there exists a rule $A \to BC$ and a split point k ($i \le k < j$) such that $B \in T[i, k]$ and $C \in T[k + 1, j]$.

The string is accepted if starting symbol $S \in T[1, n]$.

Example 3.3. *Table 3.1 illustrates the CYK Table construction to verify the acceptance of the string $w =$ ()() using the CNF grammar derived in Example 3.2. The top-right cell $T[1, 4]$ represents the entire string. It contains the Start symbol B (Balanced) because there is a rule 'Balanced $\to$ Pair Balanced' for a split point $k = 2$ where $Pair \in T[1, 2]$ and $Balanced \in T[3, 4]$.*

Similarly, the cell $T[1, 2]$ contains P (Pair) because there is a rule $Pair \to L\,R$ for a split point $k = 1$ ($L \in T[1, 1]$ and $R \in T[2, 2]$).

$w_1 = \text{`('} \{L\}$	$\{B, P\}$	$\emptyset$	$\{\mathbf{B}\}$
	$w_2 = \text{`)'} \{\text{R}\}$	$\emptyset$	$\emptyset$
		$w_3 = \text{`('} \{\text{L}\}$	{B,P}
			$w_4 = \text{`)'} \{\text{R}\}$

TABLE 3.1: CYK Table for $w = ()()$. The cells $T[i, j]$ correspond to the substring starting at i and ending at j. The diagonal elements contain the input terminals and unary rules able to generate them. **Abbreviations:** B = Balanced, P = Pair

3.4 Explaining Context-Free Languages

This report describes ongoing work to explain decisions regarding membership in a Context-Free Grammar (CFG) G for an input $w \in \Sigma^*$. Similar to explaining Finite Automata decisions [19], this problem can be viewed as a classifier mapping input w to a class in $\mathcal{K} = \{\text{accept}, \text{reject}\}$. We propose two types of explanations: **Abductive Explanations** (AXps), which answer "Why does G accept/reject w?"; and **Contrastive Explanations** (CXps), which answer "How can w be modified to alter the response of G?".

In this work, the class of explanations languages is based on regular expressions formed by replacing some characters in w by Σ.

An **abductive explanation** relative to a word, is a set of indices such that, if they are fixed, the prediction is going to remain the same regardless of the other tokens. A **contrastive explanation** is a set of indices such that, if they were free, it would be possible to flip the prediction.

Typically, accepted words in Context-Free Languages are easy to modify to flip the prediction to **rejected**. Conversely, rejected words are often robustly wrong, requiring multiple coordinated changes (or single specific one) to repair the word.

The following example illustrates this behaviour.

Example 3.4 (Fragility of Acceptance)**.** *Consider the accepted word $w =$ `()()`. The validity relies on every token. Modifying any single index is sufficient to flip the prediction to **rejected**.*

Every single index constitutes a CXp:

- ***Index 1:** Changing `(` → `)` makes `))()` (Rejected: starts with closing).*
- ***Index 2:** Changing `)` → `(` makes `(((` (Rejected: unmatched open).*
- ***Index 3:** Changing `(` → `)` makes `())` (Rejected: unmatched close).*
- ***Index 4:** Changing `)` → `(` makes `()((` (Rejected: unmatched open).*

Since every feature is critical, the AXp (the subset of features required to guarantee acceptance) is the entire word.

The problem becomes significantly more interesting for rejected words.

Algorithm 1 ExtractCXp – a Single CXp Extraction

Input: Context-Free Grammar G, Candidate set $\mathcal{Y}$ (initially $\{1..\ldots|w|\}$), word w
Output: Minimal CXp $\mathcal{Y}$

```
if not IsCXp(G, Y, w) then
    return ⊥
for all i ∈ Y do
    if IsCXp(G, Y \ {i}, w) then
        Y ← Y \ {i}
return Y
```

Example 3.5 (Robustness of Rejection)**.** *Consider the rejected word* $w =$ `))))`*. Changing a single index is insufficient to flip the prediction to* ***accepted****. There are only two Minimal CXps for this word:*

- $CXp_1 = \{1, 2\}$*: Generates* `(())`*.*
- $CXp_2 = \{1, 3\}$*: Generates* `()()`*.*

From this, we can extract meaningful AXps that are sufficient to guarantee rejection:

- $AXp_1 = \{1\}$*: The first symbol '*`)`*' guarantees rejection regardless of the remaining suffix.*
- $AXp_2 = \{2, 3\}$*: The substring '*`))`*' at indices 2 and 3 guarantees rejection for any word of length 4.*

3.4.1 Extracting One Formal Explanation

To extract a CXp for a rejected word, we must first establish an efficient method to verify whether a candidate set of indices allows the word to be accepted. We achieve this by adapting the CYK parsing algorithm as described in Definition 3.4.

Definition 3.4 (CYK-based Verification of CXp)**.** For an input string $w = w_1 w_2 \ldots w_n$ and a grammar in Chomsky Normal Form, the CYK algorithm is modified to verify if an index set $S \subseteq \{1, \ldots, n\}$ is a CXp for a rejected word as follows:

1. **Base Case:** For each $i \in \{1, \ldots, n\}$:
   - If $i \in S$: $T[i, i] = \{A \in V \mid \exists \alpha \in \Sigma, A \to \alpha\}$ (all non-terminal symbols with at least one unary rule).
   - If $i \notin S$ (Fixed): $T[i, i] = \{A \in V \mid A \to w_i\}$ (as Standard CYK).
2. **Recursive Step ($j > i$):** Standard CYK update

The set S is a valid CXp if the start symbol belongs to $T[1, n]$.

$w_1 = \Sigma$ {L,R}	{B,P}	{L,U}	{**B**}
	$w_2 = \Sigma$ {L,R}	{B,P}	$\emptyset$
		$w_3 =$ `)` {R}	$\emptyset$
			$w_4 =$ `)` {R}

Table 3.2: Modified CYK Table verifying `))))`. Indices 1 and 2 are treated as wildcards (Σ). **Abbreviations:** B = Balanced, P = Pair, U = Unclosed

Algorithm 1 extracts a minimal CXp using a greedy deletion strategy. Initially, $\mathcal{Y}$ includes all indices (effectively treating the whole word as wildcards). The algorithm iterates through the candidates, attempting to "recover" the original token w_i at each position. If the function IsCXp (Definition 3.4) confirms that the word can still be corrected *without* changing index i, then i is redundant and removed from the explanation, otherwise, i is part of the explanation. The result is a minimal CXp: removing any index from $\mathcal{Y}$ would find a pattern containing no valid strings in $L(G)$.

Abductive Explanations (AXps) are extracted similarly. Since AXps represent sufficient conditions for rejection, a set $\mathcal{X}$ is a valid AXp if treating indices in $\mathcal{X}$ as fixed (and all others as wildcards) as $\text{IsCXp}(G, \overline{\mathcal{X}}, w)$ and checking if the start symbol does not belong in $T[1, n]$.

3.4.2 Prioritising Explanations

While the extraction of Minimal CXps provides a set of valid corrections, it often produces multiple candidates. For the rejected word $w =$ `))))`, we identified two minimal sets: $\text{CXp}_1 = \{1, 2\}$ (suggesting `(())`) and $\text{CXp}_2 = \{1, 3\}$ (suggesting `()()`).

A ranking approach is *Feature Attribution*, which counts the frequency of an index across all minimal explanations. In our case:

- **Index 1:** Frequency 1.0.
- **Index 2:** Frequency 0.5.
- **Indix 3:** Frequency 0.5.

However, symbolic frequency ignores how likely is resulting correction. In domains like programming, certain structures (e.g., deeply nested brackets vs. sequential pairs) have different probabilities. To refine our suggestions, we use **Probabilistic Context-Free Grammars (PCFGs)**[20].

Example 3.6 (Rule Counting)**.** *Consider a training dataset* $D = \{$ `()()()`, `(())()`, `()(())`$\}$. *Figure 3.2 shows the derivation trees, and Table 3.3 the rule counting to estimate the probability of each production rule (B denotes the non-terminal Balanced).*

Rule	Count	Probability (P)
$B \to L\ R$	4	$4/9 = 44.\overline{4}$
$B \to$ Pair B	3	$3/9 = 33.\overline{3}$
$B \to$ Nested B	1	$1/9 = 11.\overline{1}$
$B \to$ Unclosed R	1	$1/9 = 11.\overline{1}$

TABLE 3.3: Probability distribution derived from D.

3.4.2.1 Ranking Explanations via PCFGs

To select the best correction, we calculate the probability of generating suggested correction given a fixed length constraint. For our running example to explain the rejected

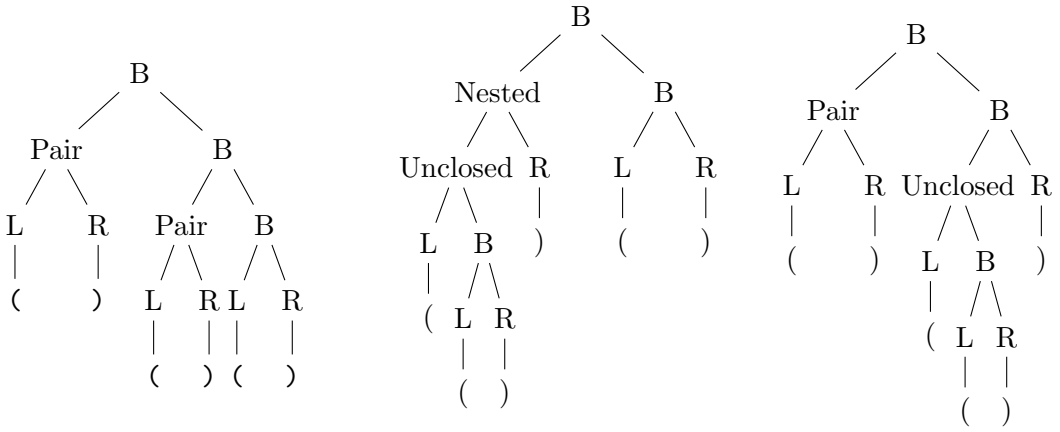


FIGURE 3.1: Parse trees for words `()()()`, `(())()`, `()(())`.

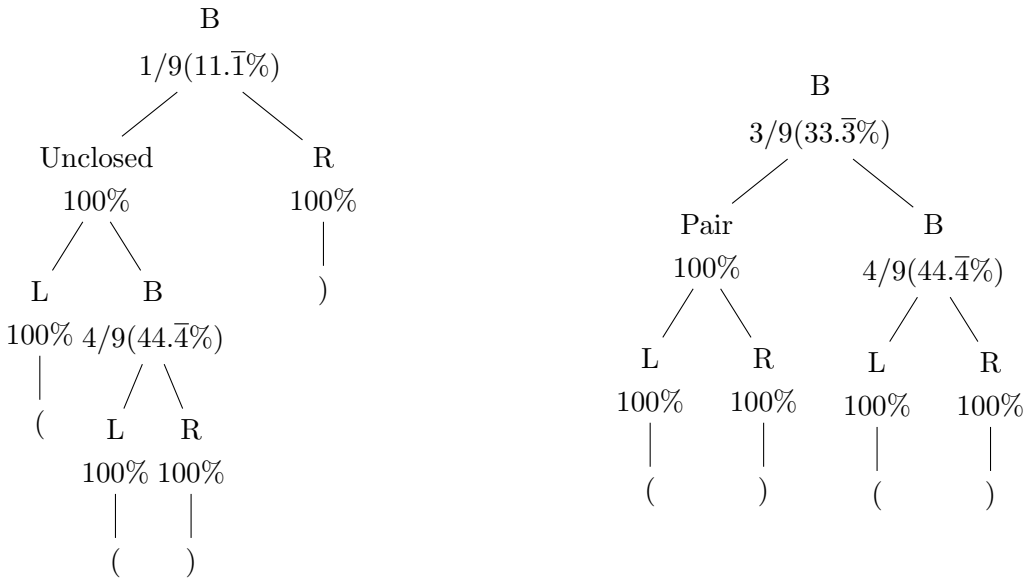


FIGURE 3.2: Parse trees for words `(())` and `()()`.

word w = `))))` we consider the set of all valid words in the language of length 4 $L_4 = \{$`()()`, `(())`$\}$.

The total probability for valid words of length 4 is:

$$P(L_4) = P(\texttt{()()}) + P(\texttt{(())}) \tag{3.3}$$

Using the estimated rule probabilities, we compute the likelihood of the corrections suggested by our CXps (shown in Figure 3.2):

- $\text{CXp}_1 \rightarrow$ `(())`: $P(\texttt{(())}) = \frac{1}{9} \times \frac{4}{9} \approx 0.049$
- $\text{CXp}_2 \rightarrow$ `()()`: $P(\texttt{()()}) = \frac{3}{9} \times \frac{4}{9} \approx 0.148$

We define the *Relative Likelihood Score* for a CXp suggesting correction w' as:

$$Score(w') = P(w')/P(L_{|w|}) \tag{3.4}$$

$$Score(\text{CXp}_1) = \frac{0.049}{0.148 + 0.049} \approx \mathbf{0.25} \quad \text{vs} \quad Score(\text{CXp}_2) \approx \mathbf{0.75}$$

Under this PCFG, the correction `()()` (suggested by CXp_2) is more likely than `(())` (suggested by CXp_1). This allows the explainer to prioritise the most statistically probable fix.

3.5 Conclusion

We present a polynomial-time algorithm for extracting Abductive and Contrastive Explanations relying on linear calls to a CYK based algorithm. Additionally, incorporating training data, this approach can prioritise the most relevant Contrastive Explanations. Future work will focus on benchmarks to evaluate the scalability and a proposal to understand how can we prioritise Abductive Explanations.

Chapter 4

Future Plan

In my third year, I plan to finalise the validation and theoretical details for Formal Explanations for Context-Free Grammars. Then, I will address Research Problem 3, investigating formal explanations for models with stochastic behaviour.

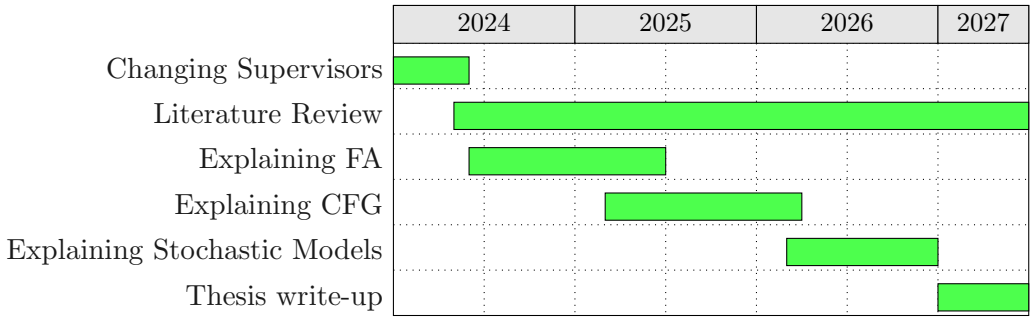


FIGURE 4.1: Timeline to completion of the PhD project

Bibliography

[1] João Marques-Silva. Logic-based explainability in machine learning. In Leopoldo E. Bertossi and Guohui Xiao, editors, *Reasoning Web. Causality, Explanations and Declarative Knowledge - 18th International Summer School 2022, Berlin, Germany, September 27-30, 2022, Tutorial Lectures*, volume 13759 of *Lecture Notes in Computer Science*, pages 24–104. Springer, 2022. doi: 10.1007/978-3-031-31414-8_2. URL https://doi.org/10.1007/978-3-031-31414-8_2.

[2] Adnan Darwiche. Logic for explainable AI. In *LICS*, pages 1–11. IEEE, 2023.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. ISBN 978-0-262-02649-9.

[4] Gerard J. Holzmann. The model checker SPIN. *IEEE Trans. Software Eng.*, 23 (5):279–295, 1997. doi: 10.1109/32.588521. URL https://doi.org/10.1109/32.588521.

[5] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley series in computer science / World student series edition. Addison-Wesley, 1986. ISBN 0-201-10088-6. URL https://www.worldcat.org/oclc/12285707.

[6] Max Brunsfeld. Tree-sitter: A new parsing system for programming tools. Presentation at Strange Loop Conference, 2018. Available at https://tree-sitter.github.io/.

[7] Terence Parr, Sam Harwell, and Kathleen Fisher. Adaptive ll(*) parsing: the power of dynamic analysis. In Andrew P. Black and Todd D. Millstein, editors, *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2014, part of SPLASH 2014, Portland, OR, USA, October 20-24, 2014*, pages 579–598. ACM, 2014. doi: 10.1145/2660193.2660202. URL https://doi.org/10.1145/2660193.2660202.

[8] Alfred V. Aho and Thomas G. Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM J. Comput.*, 1(4):305–312, 1972. doi: 10.1137/0201022. URL https://doi.org/10.1137/0201022.

[9] João Marques-Silva and Alexey Ignatiev. Delivering trustworthy AI through formal XAI. In *AAAI*, pages 12342–12350. AAAI Press, 2022.

[10] Alexey Ignatiev, Nina Narodytska, Nicholas Asher, and João Marques-Silva. From contrastive to abductive explanations and back again. In *AI*IA*, volume 12414 of *Lecture Notes in Computer Science*, pages 335–355. Springer, 2020.

[11] Jinqiang Yu, Alexey Ignatiev, and Peter J. Stuckey. On formal feature attribution and its approximation. *CoRR*, abs/2307.03380, 2023.

[12] Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2):114–125, 1959.

[13] John L. Carroll and Darrell D. E. Long. *Theory of finite automata with an introduction to formal languages*. Prentice Hall, 1989.

[14] Dominique Perrin. Finite automata. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 1–57. Elsevier and MIT Press, 1990.

[15] Jean Berstel and Luc Boasson. Context-free languages. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 59–102. Elsevier and MIT Press, 1990. doi: 10.1016/B978-0-444-88074-1.50007-X. URL https://doi.org/10.1016/b978-0-444-88074-1.50007-x.

[16] Daniel H. Younger. Recognition and parsing of context-free languages in time n^3. *Inf. Control.*, 10(2):189–208, 1967. doi: 10.1016/S0019-9958(67)80007-X. URL https://doi.org/10.1016/S0019-9958(67)80007-X.

[17] Dick Grune and Ceriel J. H. Jacobs. *Parsing Techniques - A Practical Guide*. Monographs in Computer Science. Springer, 2008. ISBN 978-0-387-20248-8. doi: 10.1007/978-0-387-68954-8. URL https://doi.org/10.1007/978-0-387-68954-8.

[18] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007. ISBN 978-0-321-47617-3.

[19] Jaime Cuartas Granada, Alexey Ignatiev, and Peter J. Stuckey. A formal framework for the explanation of finite automata decisions, 2026. URL https://arxiv.org/abs/2602.13351.

[20] Taylor L. Booth and Richard A. Thompson. Applying probability measures to abstract languages. *IEEE Trans. Computers*, 22(5):442–450, 1973. doi: 10.1109/T-C.1973.223746. URL https://doi.org/10.1109/T-C.1973.223746.

A Formal Framework for the Explanation of Finite Automata Decisions

Jaime Cuartas Granada **Alexey Ignatiev** **Peter J. Stuckey**
Department of Data Science and AI, Faculty of IT
Monash University, Melbourne, Victoria, Australia
{jaime.cuartasgranada, alexey.ignatiev, peter.stuckey}@monash.edu

Abstract

Finite automata (FA) are a fundamental computational abstraction that is widely used in practice for various tasks in computer science, linguistics, biology, electrical engineering, and artificial intelligence. Given an input word, an FA maps the word to a result, in the simple case "accept" or "reject", but in general to one of a finite set of results. A question that then arises is: why? Another question is: how can we modify the input word so that it is no longer accepted? One may think that the automaton itself is an adequate explanation of its behaviour, but automata can be very complex and difficult to make sense of directly. In this work, we investigate how to explain the behaviour of an FA on an input word in terms of the word's characters. In particular, we are interested in *minimal* explanations: what is the minimal set of input characters that explains the result, and what are the minimal changes needed to alter the result? Note that multiple minimal explanations can arise in both cases. In this paper, we propose an efficient method to determine all minimal explanations for the behaviour of an FA on a particular word. This allows us to give unbiased explanations about which input features are responsible for the result. Experiments show that our approach scales well, even when the underlying problem is challenging.

1 Introduction

With the rise of Artificial Intelligence, we have seen increasing use of black-box systems to make decisions. This in turn has led to the demand for eXplainable Artificial Intelligence (XAI) [23, 2, 22] where the decisions of these systems must be explained to a person either affected by or implementing the decisions. While much of the work on XAI has a very fuzzy or adhoc definition of an *explanation* for a decision, Formal Explainable Artificial Intelligence (FXAI) [7, 20], demands rigorous explanations that satisfy formal properties, including usually some form of minimality.

The demand for explanations of behaviour is not only a question of interest for black-box systems, but in fact any system that makes a decision, including those that are deemed inherently interpretable [21]. Motivated by the widespread use of finite automata (FA), this paper examines how to *formally* explain the result of their operation. While FA are sometimes seen as one of the simplest models of computation, their inner workings are often far from obvious. Understanding their behavior is beneficial in domains where FA are used widely as components of more complex tasks, e.g. string searching, pattern matching, lexical analysis, or deep packet inspection [30]. Furthermore, agent policies in reinforcement learning, once trained, are often represented as finite automata; the need to understand why a particular action was taken by the agent in a given situation and whether or not a different outcome could be reached by taking another action further underscores the need for FA explanations.

In light of the above, given an automaton $\mathcal{A}$ and an input string w, where $\mathcal{A}$ applied to w returns $r \in \{\text{accept}, \text{reject}\}$; this paper builds on the apparatus of [20] to answer the following questions *formally*:

- *Abductive explanation*: why does $\mathcal{A}$ return r for w, i.e. what features of w are necessary to guarantee the result r.
- *Contrastive explanation*: how can I modify w minimally obtaining w' s.t. $\mathcal{A}$ returns a different result $r' \neq r$ on w'.

Preprint.

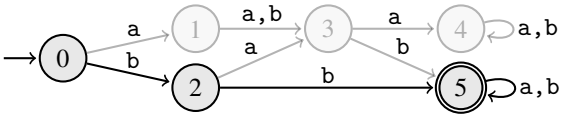


Figure 1: Input bbbbb is accepted by this automata. Opaque states indicate transitions that are not traversed for the input bbbbb. Two valid explanations are bbbbb and (a shorter one) bbbbb.

Interestingly, while the latter question can be related to the well-known problem of (and algorithms for) computing the *minimum edit distance* [29], no similar mechanisms exist for answering the former 'why' question. Moreover, as formal abductive and contrastive explanations enjoy the minimal hitting set duality relationship [20], it is often instrumental in devising efficient algorithms for their computation and enumeration, which further underscores the advantage of formal explanations studied in this paper over the minimal edit distance. Also, note that many approaches to XAI actually consider finite automata as explanations themselves [5, 12], using them as (approximate) explanations for black box models. But while FA are a simple computational mechanism, their behaviour when viewed from outside may not be obvious, and indeed in our view of explanations, we are trying to find features of the *input* which minimally explain the result of the automaton.

Example 1. *Consider the deterministic FA shown in Figure 1. Observe that it accepts the input word* bbbbb*. How should we explain this? One obvious way is to trace the execution of the automaton on the input word. Clearly, reading the first two* b*'s leads to an accepting state, so an explanation could be* bbbbb *where the underlined characters are those that explain the acceptance. But other (shorter) explanations exist, e.g.* bbbbb *is a correct explanation, as any word of length 5 with a* b *in position 3 will be accepted.* □

2 Preliminaries

2.1 Finite Automata and Regular Expressions

Here, we adopt standard definitions and notations for *finite automata* [26, 3, 24]. Let Σ be a finite alphabet of symbols. A *word* may be the *empty word* ϵ or a finite sequence of symbols over the alphabet Σ written as $w = w_1 w_2 \dots w_n$ s.t. $w_i \in \Sigma$ for all $1 \leq i \leq n$. Integer n here is referred to as the *length* of the word w, i.e. $|w| = n$. We use array notation to lookup symbol in a word, i.e. $w[i]$ is the i^{th} symbol in the string, with $1 \leq i \leq n$. Given a finite alphabet Σ, the set of *all* words over Σ is denoted as Σ^*. A *langugage* L is a subset of Σ^*. For any $k \in \mathbb{N}$, Σ^k is defined as: $\Sigma^k = \{x \mid x \in \Sigma^* \text{ and } |x| = k\}$, where $\Sigma^0 = \{\epsilon\}$.

A finite automaton (FA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where Σ is a finite alphabet; Q is a finite non-empty set of *states* including the *initial state* q_0 and a set F of *accepting states*;[1] and $\delta \subseteq Q \times \Sigma \times Q$ is a set of *transitions*. If there is $(q, c, q') \in \delta$ such that $q, q' \in Q$ and $c \in \Sigma$, then we say that there is a *transition* from state q to state q' on symbol c written as $q \xrightarrow{c} q'$. A *computation* for string w of length $n \triangleq |w|$ in an automaton $\mathcal{A}$ is a sequence of transitions $q_0 \xrightarrow{w[1]} q_1 \xrightarrow{w[2]} q_2 \xrightarrow{w[3]} \cdots \xrightarrow{w[n]} q_n$ where $(q_{i-1}, w[i], q_i) \in \delta$ and $1 \leq i \leq n$. An *accepting computation* for w in $\mathcal{A}$ from state q_0 is a computation for w in the automaton $\mathcal{A}$ where $q_n \in F$. The *regular language* of automaton $\mathcal{A}$ (often said to be *recognised* by $\mathcal{A}$) from state $q \in Q$, $L(q, \mathcal{A})$, is the set of strings w, which have an accepting computation from state q. The *language* of automaton $\mathcal{A}$ is defined as $L(\mathcal{A}) = L(q_0, \mathcal{A})$. We define the size of automaton $\mathcal{A}$ denoted $|\mathcal{A}|$ as its number $|Q|$ of states.

A finite automaton is *deterministic* (DFA) if for each pair $(q, c) \in Q \times \Sigma$ there is at most one state $q' \in Q$ such that $(q, c, q') \in \delta$. Finite automata not satisfying the above condition are called *non-deterministic* (NFA). For any NFA, there exists a DFA recognising the same regular language [24]. Due to this fact, this work focuses on DFAs.

Given a DFA $\mathcal{A}$ over the alphabet Σ, its *complement* is another FA, denoted as $\overline{\mathcal{A}}$, over the same alphabet Σ, and it recognises the language $L(\overline{\mathcal{A}}) = \overline{L(\mathcal{A})}$, and $\overline{L(\mathcal{A})} = \Sigma^* \setminus L(\mathcal{A})$. It is well understood how to construct a DFA for $\overline{\mathcal{A}}$ from $\mathcal{A}$ with at most one more state. Clearly, for any $\mathcal{A}$, $L(\mathcal{A}) \cap L(\overline{\mathcal{A}}) = \emptyset$, and $L(\mathcal{A}) \cup L(\overline{\mathcal{A}}) = \Sigma^*$. As a consequence, for any word $w \in \Sigma^*$, $w \in L(\mathcal{A}) \iff w \notin L(\overline{\mathcal{A}})$.

We make use of standard regular expression notation. We denote by $\emptyset$ the empty language, by $c \in \Sigma$ the regular expression defining the language $\{c\}$, and by Σ the language $\{\Sigma\}$, viewed as the set of all strings of length

[1] While the approach we consider can be applied to finite classifiers which return a result $r \in R$ from a set, the explanations we consider either aim to guarantee the same result r or lead to *any* different result $r' \in R \setminus \{r\}$, thus treating all results $R \setminus \{r\}$ equivalently, so only two output results are ever required. Hence we restrict to classic automata.

1. In general, given a regular expression R, the language it defines is denoted by $L(R)$. The concatenation of two regular expressions R_1R_2 denotes the regular language $\{w_1w_2 \mid w_1 \in L(R_1), w_2 \in L(R_2)\}$. The union of two regular expressions $R_1|R_2$ denotes the language $L(R_1) \cup L(R_2)$. The Kleene star R^* expression is the language defined as $\{w_1 \cdots w_n \mid n \geq 0, w_i \in L(R)\}$. Given a *range* $l..u$ to denote the set of non-negative integers $\{i \in \mathbb{N} \cup \{0\} \mid l \leq i \leq u\}$, we define extended regular expression notation Σ_l^u defining all strings of length at least l and at most u, defined as $\Sigma_l^u \equiv \overbrace{\Sigma \cdots \Sigma}^{l}\overbrace{(\Sigma|\emptyset)\cdots(\Sigma|\emptyset)}^{u-l}$, $u < \infty$ and $\Sigma_l^u \equiv \overbrace{\Sigma \cdots \Sigma}^{l}\Sigma^*$, $u = \infty$. Observe that $\Sigma^* = \Sigma_0^\infty$ and $\Sigma^+ = \Sigma_1^\infty = \Sigma^* \setminus \Sigma^0$.

We denote by $\mathcal{A}$ a Deterministic Finite Automaton (DFA) and by R a regular expression. Their recognised languages, or sets of words, are written as $L(\mathcal{A})$ and $L(R)$, respectively.

2.2 Classification Problems and Formal Explanations

Following [20, 6], we assume classification problems to be defined on a set $\mathcal{F}$ of m features and a set $\mathcal{K}$ of k classes. Each feature $i \in \mathcal{F}$ is in some domain $\mathbb{D}_i$ while the feature space is $\mathbb{F} = \prod_{i=1}^{m} \mathbb{D}_i$. A classifier is assumed to compute a total function $\kappa : \mathbb{F} \to \mathcal{K}$.

Next, we assume an instance $\mathbf{v} \in \mathbb{F}$ such that $\kappa(\mathbf{v}) = c \in \mathcal{K}$. We are interested in explaining why $\mathbf{v}$ is classified as class c. Given an instance $\mathbf{v} \in \mathbb{F}$ such that $\kappa(\mathbf{v}) = c \in \mathcal{K}$, an *abductive explanation* (AXp) $\mathcal{X} \subseteq \mathcal{F}$ is a minimal subset of features *sufficient* for the prediction. Formally, $\mathcal{X}$ is defined as:

$$\forall(\mathbf{x} \in \mathbb{F}).\ \Big[\bigwedge\nolimits_{i \in \mathcal{X}}(x_i = v_i)\Big] \rightarrow (\kappa(\mathbf{x}) = c) \tag{1}$$

Similarly, one can define another kind of explanation. Namely, given an instance $\mathbf{v} \in \mathbb{F}$ such that $\kappa(\mathbf{v}) = c$, a *contrastive explanation* (CXp) is a minimal subset of features $\mathcal{Y} \subseteq \mathcal{F}$ that, if allowed to change, enables the prediction's alteration. Formally, a contrastive explanation $\mathcal{Y}$ is defined as follows:

$$\exists(\mathbf{x} \in \mathbb{F}).\ \Big[\bigwedge\nolimits_{j \notin \mathcal{Y}}(x_j = v_j)\Big] \wedge (\kappa(\mathbf{x}) \neq c) \tag{2}$$

Observe that abductive explanations are used to explain *why* a prediction is made by the classifier κ for a given instance while contrastive explanations can be seen to answer *why not* another prediction is made by κ. Alternatively, CXps can be seen as answering *how* the predication can be changed.

Importantly, abductive and contrastive explanations are known to enjoy a minimal hitting set duality relationship [16]. Given $\kappa(\mathbf{v}) = c$, let $\mathbb{A}_\mathbf{v}$ be the complete set of AXps and $\mathbb{C}_\mathbf{v}$ be the complete set of CXps for this prediction. Then each AXp $\mathcal{X} \in \mathbb{A}_\mathbf{v}$ is a minimal hitting set of $\mathbb{C}_\mathbf{v}$ and, vice versa, each CXp $\mathcal{Y} \in \mathbb{C}_\mathbf{v}$ is a minimal hitting set of $\mathbb{A}_\mathbf{v}$.[2] This fact is the basis for the algorithms used for formal explanation *enumeration* [20, 33].

A challenge arising in formal XAI is given there are many AXps or CXps, which is the *best* explanation to present to a user. An obvious answer is to use one of minimal size, but an alternate answer is to generate a result that records the influence of all explanations. A *formal feature attribution* (FFA) weighs the importance of each feature [33, 32] in determining the result that $\kappa(\mathbf{v}) = c$. We define the formal feature attribution of feature i as

$$\mathsf{FFA}_\mathbf{v}(i) = \frac{|\{\mathcal{X} \mid \mathcal{X} \in \mathbb{A}_\mathbf{v}, i \in \mathcal{X}\}|}{|\mathbb{A}_\mathbf{v}|} \tag{3}$$

The proportion of all AXps shows the (non-zero) contribution of each feature to the decision. A *formal feature attribution* for $\kappa(\mathbf{v}) = c$ is the set $\{(i, \mathsf{FFA}_\mathbf{v}(i)) \mid i \in \mathcal{F}, \mathsf{FFA}_\mathbf{v}(i)) > 0\}$. Critically, formal feature attribution gives an *unbiased* measure of the influence of input features on the decision.

3 Explainable Finite Automata

In this work, we aim to explain the behaviour of a finite automaton $\mathcal{A}$ on an input $w \in \Sigma^*$, which can be viewed as a classifier mapping input w to a class in $\mathcal{K} = \{\text{accept}, \text{reject}\}$. Similar to classification problems, we propose two types of explanations for FA: abductive explanations (AXps) and contrastive explanations (CXps). Informally, an AXp answers "why does $\mathcal{A}$ accept/reject w?" a CXp answers "How can w be modified to alter the response of $\mathcal{A}$?".

[2] Given a collection of sets $\mathbb{S}$, a *hitting set* of $\mathbb{S}$ is a set H such that for each $S \in \mathbb{S}$, $H \cap S \neq \emptyset$. A hitting set is *minimal* if no proper subset of it is a hitting set.

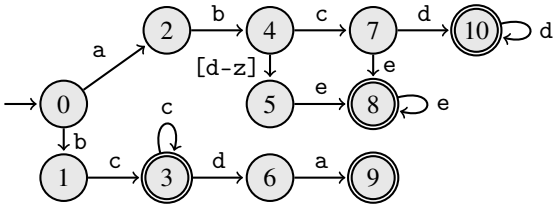


Figure 2: DFA recognising the language `(abcd+)|(ab[c-z]e+)|(bc+da)|(bc+)`.

We first define explanations abstractly, then consider concrete instantiations. An explanation is a regular language L from a set $\mathcal{L}(w, \mathcal{A})$ of candidate explanations, defined over the alphabet Σ and depending on both w and $\mathcal{A}$. We refer to $\mathcal{L}(w, \mathcal{A})$ as the *language of explanations*.[3]

Definition 1 (Abductive Explanation - AXp)**.** *A* weak AXp *for $w \in L(\mathcal{A})$ is a language $\mathcal{X} \in \mathcal{L}(w, \mathcal{A})$ such that $w \in \mathcal{X}$ and $\mathcal{X} \subseteq L(\mathcal{A})$. A (minimal)* AXp *for $w \in L(\mathcal{A})$ is a language $\mathcal{X} \in \mathcal{L}(w, \mathcal{A})$ where there is no weak AXp $\mathcal{X}' \in \mathcal{L}(w, \mathcal{A})$ of $w \in L(\mathcal{A})$ such that $\mathcal{X}' \supset \mathcal{X}$.*

AXps capture maximality under set inclusion over languages in $\mathcal{L}(w, \mathcal{A})$. This property allows an AXp to define the *most general* explanation for why $w \in L(\mathcal{A})$.

Definition 2 (Contrastive Explanation - CXp)**.** *A* weak CXp *for $w \in L(\mathcal{A})$ is a language $\mathcal{Y} \in \mathcal{L}(w, \mathcal{A})$ such that $w \in \mathcal{Y}$ and $\mathcal{Y} \not\subseteq L(\mathcal{A})$. A (minimal)* CXp *for $w \in L(\mathcal{A})$ is a language $\mathcal{Y} \in \mathcal{L}(w, \mathcal{A})$ where there is no weak CXp $\mathcal{Y}' \in \mathcal{L}(w, \mathcal{A})$ of $w \in L(\mathcal{A})$ such that $\mathcal{Y}' \subset \mathcal{Y}$.*

In other words, a CXp is a *minimal language* including w and a word *not accepted* by $\mathcal{A}$.

These abstract definitions hinge upon the definitions of the language of explanations $\mathcal{L}(w, \mathcal{A})$. Clearly, if $\mathcal{L}(w, \mathcal{A})$ includes all regular languages then w itself defines its own AXp $L(w)$, and any $w|w'$ with $w' \notin L(\mathcal{A})$ defines a CXp $L(w|w')$. Neither of these offers a *meaningful* explanation.

Let us define a class of explanation languages $\mathbb{W}_l^u \triangleq \mathcal{L}(w, \mathcal{A})$ where $0 \leq l \leq 1 \leq u$, defined by regular expressions formed by replacing some characters in w by Σ_l^u, i.e. replacing a character in w by *any* string of characters of length between l and u (inclusive). Note that since $l \leq 1 \leq u$, for all $\mathcal{Z} \in \mathbb{W}_l^u$ it holds that $w \in \mathcal{Z}$. We focus on the case where $l = u = 1$, i.e. when a character in w may be replaced by *any* single character.

The motivation for considering $\mathbb{W}_1^1$ is that the *retained* characters in w, i.e. those not replaced by Σ, give us an understanding of what is essential in w for its acceptance, while the *modified* parts of the word are *irrelevant* to the automaton's decision. Note that in $\mathbb{W}_1^1$, all strings are of length equal to $|w|$, and their deterministic automaton can be built trivially with $|w| + 1$ states.

Example 2 (AXp and CXp)**.** *Consider the automaton $\mathcal{A}$ in Figure 2 accepting the language `(abcd+)|(ab[c-z]e+)|(bc+da)|(bc+)` over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$,* [4] *and the word $w = \mathtt{accc}$, $w \notin L(\mathcal{A})$. Using $\mathbb{W}_1^1$, a trivial* weak *AXp is $L(\mathtt{accc})$, retaining all symbols. More informative explanations are minimal AXps, such as $L(\mathtt{ac}\Sigma\Sigma)$ and $L(\mathtt{a}\Sigma\Sigma\mathtt{c})$, each revealing a different reason why $w \notin L(\mathcal{A})$. These AXps represent subset-minimal sets of symbols of w that are sufficient to explain the rejection of w. A minimal CXp is $L(\Sigma\mathtt{ccc}) \not\subset L(\overline{\mathcal{A}})$, e.g. a possible correction of rejection is $w' = \underline{\mathtt{b}}\mathtt{ccc} \in L(\mathcal{A})$. Another minimal CXp is $L(a\Sigma\mathtt{c}\Sigma)$ since $w'' = \mathtt{a}\underline{\mathtt{b}}\mathtt{c}\underline{\mathtt{d}} \in L(\mathcal{A})$.* □

Similarly to FXAI [20], we define a shorthand notation for explanations in $\mathbb{W}_l^u$ using subsets of positions in $w \in L(\mathcal{A})$:

$$\mathsf{AXp}_l^u(S, w) = L(s_1 \cdots s_n) \mid s_i = w[i] \text{ if } i \in S \text{ else } \Sigma_l^u$$
$$\mathsf{CXp}_l^u(S, w) = L(s_1 \cdots s_n) \mid s_i = w[i] \text{ if } i \notin S \text{ else } \Sigma_l^u$$

Note that Σ_1^1, Σ_1^∞, and Σ_0^∞ correspond to Σ, Σ^+, and Σ^*, allowing replacements at position i by any symbol, any non-empty string, or any string (including empty), respectively.

Example 3. *Consider the setup of Example 2. Given the word $w = \mathtt{accc}$, $L(\mathtt{ac}\Sigma\Sigma) = \mathsf{AXp}_1^1(\{1, 2\}, w)$ and $L(\mathtt{a}\Sigma\Sigma\mathtt{c}) = \mathsf{AXp}_1^1(\{1, 4\}, w)$. Similarly, $L(\Sigma\mathtt{ccc}) = \mathsf{CXp}_1^1(\{1\}, w)$ and $L(\mathtt{a}\Sigma\mathtt{c}\Sigma) = \mathsf{CXp}_1^1(\{2, 4\}, w)$.* □

[3] We restrict ourselves to explanations for $w \in L(\mathcal{A})$. Observe that we can explain $w \notin L(\mathcal{A})$ by explaining $w \in L(\overline{\mathcal{A}})$.

[4] Similar *union* languages are often an object of study in Deep Packet Inspection [30].

This notation shows that AXps are minimal subsets of retained character positions, and CXps are minimal subsets of freed character positions. This makes it easy to see the hitting set duality between AXps and CXps for FA.

Proposition 1. *(Proofs for this and all other propositions are provided in the Appendix) Given $w \in L(\mathcal{A})$, assume that the sets of all AXps and CXps are denoted as $\mathbb{A} = \{X \subseteq \{1,\ldots,|w|\} \mid \mathsf{AXp}^{u}_{l}(X,w) \in \mathbb{W}\}$ and $\mathbb{C} = \{Y \subseteq \{1,\ldots,|w|\} \mid \mathsf{CXp}^{u}_{l}(Y,w) \in \mathbb{W}\}$, respectively. It holds that each $X \in \mathbb{A}$ is a minimal hitting set of $\mathbb{C}$ and, vice versa, each $Y \in \mathbb{C}$ is minimal hitting set of $\mathbb{A}$.*

Various explanation languages can be related as follows:

Proposition 2. *Let $w \in L(\mathcal{A})$ and $l_1..u_1 \supseteq l_2..u_2$. If S is such that $\mathsf{AXp}^{u_1}_{l_1}(S,w)$ is a weak AXp for $w \in L(\mathcal{A})$ using $\mathbb{W}^{u_1}_{l_1}$ then $\mathsf{AXp}^{u_2}_{l_2}(S,w)$ is a weak AXp using $\mathbb{W}^{u_2}_{l_2}$.*

Example 4. *Consider language $L(R)$ defined by the regular expression $R =$ `(abcd+)|(ab[c-z]e+)|(bc+da)|(bc+)` depicted in in Figure 2 and $w =$ `ceccd`. Observe that $w \notin L(R)$, i.e. $w \in \overline{L(R)}$. Using $\mathbb{W}^1_1$, an AXp is $\mathsf{AXp}^1_1(\{2\},w) = L(\Sigma\mathtt{e}\Sigma\Sigma\Sigma)$ since $L(\Sigma\mathtt{e}\Sigma\Sigma\Sigma) \subseteq \overline{L(R)}$. However, a more general language $L(\Sigma^+\mathtt{e}\Sigma^+\Sigma^+\Sigma^+)$ (using $\mathbb{W}^\infty_1$) is not an AXp as it contains a word `abeeee` $\notin \overline{L(R)}$. $\mathsf{AXp}^\infty_1(\{2,3\},w)$ is an AXp, i.e., $L(\Sigma^+\mathtt{ec}\Sigma^+\Sigma^+) \subseteq \overline{L(R)}$ as it reveals substring `ec` to be a reason for rejection.* □

Although a richer explanation language may lead to larger AXps in terms of character positions, these may sometimes offer better expressivity to a user if that is of their concern.

Proposition 3. *Let $w \in L(\mathcal{A})$, and $l_1..u_1 \subseteq l_2..u_2$. If S is such that $\mathsf{CXp}^{u_1}_{l_1}(S,w)$ is a weak CXp for $w \in L(\mathcal{A})$ using $\mathbb{W}^{u_1}_{l_1}$ then $\mathsf{CXp}^{u_2}_{l_2}(S,w)$ is a weak CXp using $\mathbb{W}^{u_2}_{l_2}$.*

As a corollary of Propositions 2–3, for $l_1..u_1 \supseteq l_2..u_2$ and each CXp S using $\mathbb{W}^{u_1}_{l_1}$ there is a subset of S which is a CXp using $\mathbb{W}^{u_2}_{l_2}$, similarly for each AXp S using $\mathbb{W}^{u_1}_{l_1}$ there is a superset of S which is an AXp using $\mathbb{W}^{u_2}_{l_2}$.

4 Why Formal Explanations?

Since DFAs are often considered as one of the simplest models of computation, one might wonder why we need formal explanations for their behavior in the first place. Indeed, it may be assumed that DFAs are inherently interpretable as one can easily trace the path through the states of the DFA given a particular word and claim the path to be the reason for acceptance or rejection (see Example 1). Here, we argue that this is not always the case because there are languages whose (smallest size) DFAs are exponentially larger than the regular expressions generating these languages. Importantly, some of these languages admit short formal explanations.

Proposition 4. *Given an alphabet Σ and a parameter $n \in \mathbb{N}$, there exists a family of regular languages L_n over Σ such that any DFA $\mathcal{A}_n$ for L_n has least 2^n states while $\forall w \in \Sigma^*$ for any $\mathsf{AXp}^1_1(S,w)$ and any $\mathsf{CXp}^1_1(S,w)$ it holds that $|S| \leq k$, for some constant $k \in \mathbb{N}$, $k < n$.*

One might assume that a *computation* (the sequence of states visited) can reveal the explanation for the acceptance/rejection of a word. However, a *computation* provides a causal chain without isolating the necessary symbols for an abductive explanation. To illustrate the deficiency of *computation* inspection, we revisit the automaton $\mathcal{A}$ from Example 1. Figure 1 shows the DFA for $\mathcal{A}$ while processing the word $w = bbbbb$. The computation requires the user to follow the path $0 \xrightarrow{b} 2 \xrightarrow{b} 5 \xrightarrow{b} 5 \xrightarrow{b} 5 \xrightarrow{b} 5$, one might incorrectly identify the first two symbols "b" as the reason for reaching the acceptance state 5. In contrast, our approach also identifies a more succinct explanation with smaller cardinality: the symbol at position 3, which is an AXp, $\mathsf{AXp}^1_1(\{3\},w) = L(\Sigma\Sigma\mathtt{b}\Sigma\Sigma)$.

A plausible alternative to formal explanations for DFAs builds on another well-studied problem in the context of languages, namely, the computation of a *minimum edit distance* [29]. Given a language L and a word $w \notin L$, the minimum edit distance from w to L is the smallest number of symbols to replace in w in order to obtain a word w' such that $w' \in L$. While minimum edit distance can seemingly be related with smallest size CXps, it is often not informative enough to explain the behavior of a DFA, thus warranting the needs for AXps:

Proposition 5. *Given an alphabet Σ and a parameter $n \in \mathbb{N}$, there exists a family of regular languages L_n over Σ and a word $w \notin L_n$ with the minimum edit distance of at least n such that for any AXp $\mathsf{AXp}^1_1(S,w)$ it holds that $|S| \leq k$, for some constant $k \in \mathbb{N}$, $k < n$.*

As a result, understanding the behavior of a DFA either by directly inspecting it or by relying on the minimum edit distance may be quite misleading in practice.

Algorithm 1 EXTRACTAXP – a Single AXp Extraction

Input: Candidate set X, automaton $\mathcal{A}$, word w, bounds l, u
Output: Minimal X with $\mathsf{AXp}_l^u(X, w) \subseteq L(\mathcal{A})$

1: **for all** $i \in X$ **do**
2: **if** $\mathsf{AXp}_l^u(X \setminus \{i\}, w) \subseteq L(\mathcal{A})$ **then**
3: $X \leftarrow X \setminus \{i\}$
4: **return** X

The apparatus of formal explainability discussed hereinafter offers a potent alternative enabling a user to opt for succinct abductive or contrastive explanations, or (importantly) both. Crucially and following [20], besides minimality, formal abductive (resp., contrastive) explanations for DFAs guarantee logical correctness as they can be seen to satisfy the property (1) (resp., (2)). The following sections describe the algorithms for computing formal explanations and demonstrate their practical scalability.

5 Computing One Formal Explanation

While computing an AXp is generally hard in the context of classification problems [20], this section shows it to be straightforward for DFAs.

Given a word $w \in L(\mathcal{A})$, Algorithm 1 extracts a minimal AXp by refining a *weak* AXp, initially $L(w)$. The algorithm "drops" characters from w replacing them one by one with Σ_l^u, iterating only over indices in the candidate set X. If the resulting language is still included in $L(\mathcal{A})$, the character is considered irrelevant and permanently "dropped"; otherwise, it is part of the resulting AXp.

The output is minimal, as "dropping" any remaining character would necessarily introduce a language with at least one counterexample.

Proposition 6. *Given a DFA $\mathcal{A}$ with m states and a word w s.t. $|w| = n$, computing an AXp for $w \in L(\mathcal{A})$ in languages $\mathbb{W}_1^1$, $\mathbb{W}_1^\infty$, or $\mathbb{W}_0^\infty$ can be done in $O(|\Sigma|mn^2)$ time.*

But if the language is given as a regular expression instead of a DFA, extracting just one AXp may be *intractable*.

Proposition 7. *Computing a single AXp using $\mathbb{W}_0^\infty$ for $w \in L(R)$ given a regular expression R is PSPACE-hard.*

Note for DFAs, CXp extraction for $\mathbb{W}_1^1$, $\mathbb{W}_1^\infty$, and $\mathbb{W}_0^\infty$ can be done similarly in $O(|\Sigma|mn^2)$ time.

6 Enumerating All Explanations

While a single AXp/CXp extraction is quite efficient, for unbiased explanations we want to extract all AXps. Explanation enumeration poses other challenges, since the number of AXps can be exponential both in the size of the DFA $\mathcal{A}$ and in the number of CXps. That is a consequence of Proposition 1. Whenever one type of explanation (either AXp or CXp) is a collection of disjoint sets, the collection of minimal hitting sets, i.e. dual explanations (either CXp or AXp), is exponential (see appendix for a concrete example).

Exhaustive enumeration of all AXps and CXps is achievable by exploiting the minimal hitting set duality between these explanations, similar to the MARCO algorithm [25, 18, 19] used in the context of over-constrained systems. The process results in collecting all target and dual explanations.

Algorithm 2 enumerates all explanations, both AXps and CXps. It can run in two modes: with $b = true$ it *targets* AXps with dual explanations being CXps, and if $b = false$ it targets CXps with dual AXps. Given an automaton $\mathcal{A}$ and word w to explain, it initialises the sets of target and dual explanations $\mathbb{E}_t$ and $\mathbb{E}_d$ as $\emptyset$. It generates a new candidate target explanation μ as a minimal hitting set of the dual sets $\mathbb{E}_d$ not already in or a superset of something in $\mathbb{E}_t$. This candidate generation can be made efficient using a Boolean satisfiability (SAT) solver set to enumerate *minimal* or *maximal models* [10, 18]. Initially, μ will be the empty set. It then checks whether this candidate μ is a target explanation using ISTARGETXP. Depending on the target type, it builds the regular language W defined by μ, by fixing or freeing μ's symbols, and checks the *language inclusion*. The candidate μ is a target explanation if the check agrees with b. If this is the case, we simply add μ to $\mathbb{E}_t$. Otherwise, we extract a minimal subset ν of the complement of μ using EXTRACTDUALXP, which uses b to call the correct minimization procedure (either EXTRACTCXP or EXTRACTAXP). We then add counterexample ν to $\mathbb{E}_d$. The loop continues until no new candidate hitting sets remain for $\mathbb{E}_d$, in which case we have enumerated all target and dual explanations.

Algorithm 2 XPENUM – Explanation Enumeration

Input: Automaton $\mathcal{A}$, word w, bounds l, u, AXp/CXp flag b
Output: Explanation sets $\mathbb{E}_t$ (target) and $\mathbb{E}_d$ (dual)

$(\mathbb{E}_t, \mathbb{E}_d) \leftarrow (\emptyset, \emptyset)$
while true **do**
  $\mu \leftarrow \text{MINIMALHS}(\mathbb{E}_d, \mathbb{E}_t)$
  if $\mu = \bot$ **then**
    break
  if $\text{ISTARGETXP}(\mu, \mathcal{A}, w, l, u, b)$ **then**
    $\mathbb{E}_t \leftarrow \mathbb{E}_t \cup \{\mu\}$ *// collect target explanation*
  else
    $\nu \leftarrow \text{EXTRACTDUALXP}(\overline{\mu}, \mathcal{A}, w, l, u, b)$
    $\mathbb{E}_d \leftarrow \mathbb{E}_d \cup \{\nu\}$ *// collect dual explanation*
return $(\mathbb{E}_t, \mathbb{E}_d)$

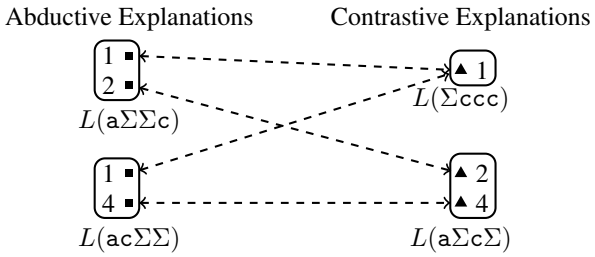


Figure 3: Duality between AXps and CXps in $\mathbb{W}_1^1$. AXps fix the characters at given indices; CXps free them.

Example 5. *Table 1 traces the execution of Algorithm 2 targeting CXps ($b = false$) for the DFA in Figure 2 and word $w =$ accc from Example 2. Figure 3 illustrates the duality relationship between the AXps and CXps.*

In the context of explanation enumeration, we can define a formal feature attribution for each position in the string, explaining how involved it is in generating the acceptance.

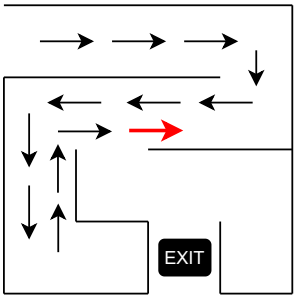
Example 6 (FFA)**.** *Consider the DFA and rejected word $w =$ accc illustrated in Figure 2, with its AXps shown in Example 4. We can generate a feature attribution for w's rejection, from the AXps $\mathsf{AXp}_1^1(\{1,2\}, w)$ and $\mathsf{AXp}_1^1(\{1,4\}, w)$. The FFA is $\{(1,1),(2,0.5),(4,0.5)\}$, indicating that position 1 is the most important position for the rejection. The next most important positions are 2 and 4. And the position 3 does not play any role.* □

While previous example shows the calculation of FFAs, the following example illustrates how FFA can provide an intuition easily representable graphically on why some of the decisions are critical in rejecting a word by a DFA.

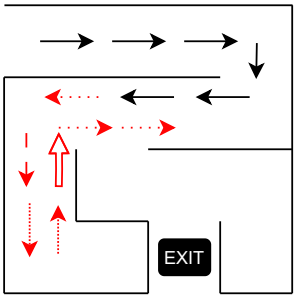
Example 7. *Consider a language where words represent paths through a maze. A word is accepted if the path reaches the "EXIT" cell without hitting walls. Figure 4 exemplifies the computation of formal feature attribution (FFA) for the decisions made a in grid maze. In particular, Figure 4a shows one AXp of minimal cardinality (size one) corresponding to the final right arrow, which indicates that any path of length 13 ending with this symbol is rejected by the DFA using the explanation language $\mathbb{W}_1^1$. Table 4b then shows all AXps and computes the FFA for each input symbol / decision. Finally, Figure 4c visually depicts the resulting attributions, where arrows represent the relative importance of specific decisions. The most important symbol leading to rejection occurs at position 11, the symbol at this step has the highest occurrence in the AXps, and thus the highest attribution. Considering*

Table 1: Step-by-step enumerating all explanations targeting CXp for word accc in the DFA illustrated in Figure 2 using $\mathbb{W}_1^1$. Here, μ represents the Minimal Hitting Set (MHS) calculated at each iteration.

$\mathbb{E}_{\text{CXp}}$	$\mathbb{E}_{\text{AXp}}$	μ (MHS)	$\mathsf{CXp}_1^1(\mu, w)$	ISCXP?	Result
$\emptyset$	$\emptyset$	$\emptyset$	accc	false	ExtractAXp: aΣΣc $\{1,4\}$
$\emptyset$	$\{1,4\}$	$\{1\}$	Σccc	true	$\mathbb{E}_{\text{CXp}} \leftarrow \mathbb{E}_{\text{CXp}} \cup \mu$
$\{1\}$	$\{1,4\}$	$\{4\}$	accΣ	false	ExtractAXp: acΣΣ $\{1,2\}$
$\{1\}$	$\{1,4\}, \{1,2\}$	$\{2,4\}$	aΣcΣ	true	$\mathbb{E}_{\text{CXp}} \leftarrow \mathbb{E}_{\text{CXp}} \cup \mu$
$\{1\}, \{2,4\}$	$\{1,4\}, \{1,2\}$	$\bot$	**break**	—	—



AXps		
$\{13\}$, $\{8,9\}$, $\{8,10\}$, $\{10,11\}$, $\{9,11\}$, $\{7,11\}$, $\{11,12\}$, $\{8,11\}$		
Index	**Freq.**	**FFA** ($\frac{Freq.}{\lvert AXps\rvert}$)
11	**5**	0.625
8	3	0.375
9, 10	2	0.250
7, 12, 13	1	0.125



(a) Example maze with a rejected path. An AXp is highlighted in red, showing that every word ending in $\rightarrow$ is rejected.

(b) All AXps in language $\mathbb{W}_1^1$ and FFA. Each FFA value is represented with an arrow of different shape.

(c) Illustration of FFAs in the maze. The arrows indicate the importance of each position in the path for rejection.

Figure 4: FFA for maze using language $\mathbb{W}_1^1$.

the language $\mathbb{W}_1^1$, which fixes the word length, making the symbol "up" a bad decision when there are only two remaining symbols. □

6.1 Extra Heuristics.

In practice, changing a single symbol in $w \in L(\mathcal{A})$ often yields a counterexample, i.e. many CXps are singletons. We can determine all singleton CXps for $w \in L(\mathcal{A})$ in $\mathbb{W}_l^u$ by checking for each $i \in \{1, \ldots, |w|\}$ whether $\mathsf{CXp}_l^u(\{i\}, w) \not\subseteq L(\mathcal{A})$. If this holds then $\mathsf{CXp}_l^u(\{i\}, w)$ is a CXp. If we run Algorithm 2 targeting AXps ($b = \mathit{true}$) then we can *warm-start* the algorithm by replacing $\mathbb{E}_d$ by the set of all singletons $\{i\}$ where $\mathsf{CXp}_l^u(\{i\}, w)$ is a CXp. This approach reduces the number of iterations of Algorithm 2 and we will use it in the experiments to speed-up AXp computation. Another modification we can make to Algorithm 2 is to replace MINIMALHS, with MINIMUMHS, which finds a *minimum size* hitting set not already explored. This can be done by using, e.g., an efficient integer linear programming (ILP) solver [11] or a maximum satisfiability (MaxSAT) solver [15]. This will guarantee that the first target explanation we find is the smallest possible.

7 Experiments

Given that the proposed explanations build on the definitions grounded in formal logic, the purpose of our experiments is to determine how performant they are in practice rather than to test how meaningful they are to humans. Also, as argued in Section 2, it is practically challenging to select a *best* feature selection explanation (either AXp or CXp). This can be tackled by determining how *important* individual symbols in the input are for a given decision of a DFA. A way to do that, formal feature attribution (FFA), requires one to enumerate explanations. Therefore, the primary objective of our experimental evaluation is to assess the scalability of exhaustive explanation enumeration. Note that while computing a single minimum edit distance explanation requires polynomial time and so can be done efficiently, it struggles with complete explanation enumeration. FFA computation is illustrated with the Maze benchmarks in Example 7, where the parts of inputs with high FFA represent the most critical decision point in a path.

Overall, we test Algorithm 2 on three benchmark families: (1) Deep Packet Inspection (DPI) rules, (2) a generated corpus of random FA, (3) DNA sequences containing a known motif, and (4) Mazes represented as DFA. All the experiments were run on Ubuntu 20.04 LTS with an Intel Xeon 8260 CPU and 16 GB RAM. The implementation builds on the PySAT toolkit [14, 17] to enumerate minimal hitting sets and the library Mata [4] for language inclusion checks. Each run had a 600 seconds timeout. Hereinafter, we focus on $\mathbb{W}_1^1$. We compare three different approaches for computing all explanations: with the flag set to *true*, targeting AXps; with the flag set to *true*, initialized with all singleton CXps; and with the flag set to *false* targeting CXps. All the plots are shown as *cactus plots* sorting results by runtime to compare overall effectiveness of the approaches.

Deep Packet Inspection. Deep Packet Inspection identifies malicious traffic by matching packet contents to known signature patterns, often represented as regular expressions.

We use a dataset of 98 File Transfer Protocol (FTP) signatures from [31].[5] The task is to check whether a given word matches any of these regular expressions.

We created an FA with the union of the 98 regular expressions, resulting in an (non-deterministic) FA with 30,590 states and 2,752,238 transitions.[6]

This large FA is used to test scalability of Algorithm 2. The length of the input words range from 4 to 2557, and the average length is 156.

Figure 5 depicts the performance of the three modes of the Algorithm 2. Observe that the algorithm scales with large automata, where the warm-starting makes a huge difference, while targeting AXps alone is worse than targeting CXps.

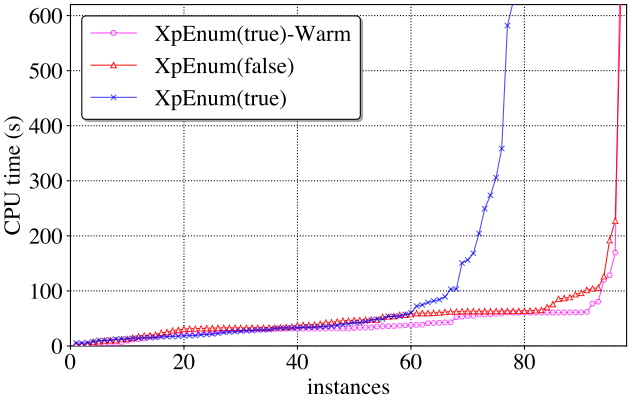


Figure 5: Deep Packet Inspection performance.

Generated corpus of FA. We generated a corpus of FAs to test explanation enumeration for long words matching a set of possible words. This corpus of FAs is similar to the corpus used in the work [9] in a different context. First, we generate m random words with length l over the alphabet $\{1, \ldots, d\}$ for $l \in \{5, 10, 15, 20\}$, $m \in \{1, 3, 5, 10\}$, $d \in \{2, 3, 5, 10\}$. Then, we build an FA for each configuration such that it accepts the language $\{\Sigma^* w \Sigma^* \mid w \in M\}$. For each of the 64 resulting FAs, we generated 10 accepted and 10 rejected words of length $\{i \times 100 \mid i \in \{1, \ldots, 10\}\}$.

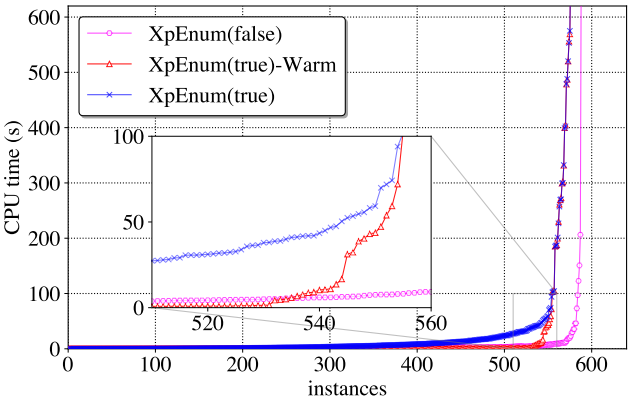


Figure 6: Generated FA corpus, with *accepted* words.

Figure 6 shows how the three alternatives perform on accepted words. The warm-started enumeration focusing on AXps is usually the best method, but for the hardest instances, targeting CXps is more effective due to the large number of CXps.

DNA Sequence. DNA is a molecule conformed by two complementary chains of nucleotides. These nucleotides contain four types: adenine (A), thymine (T), cytosine (C), and guanine (G). DNA sequences contain motifs, which are recurring patterns that are deemed to have a biological function [8].

We used a benchmark of 25 datasets of real DNA sequences with a known motif [27].[7] Each dataset has multiple pairs of (DNA sequence, motif); in total, there are 810 sequences. This case is used to test the ability to find AXps for words with different lengths. The shortest sequence has 62 nucleotides, the longest 2000 and the average length is 1246.

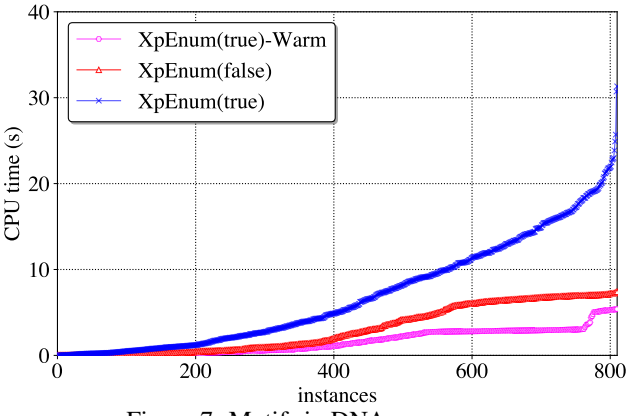


Figure 7: Motifs in DNA sequences.

Figure 7 shows once more that targeting AXps with warm-start is the superior approach, and without warm-start, targeting CXps is better than targeting AXps.

[5] `https://pages.cs.wisc.edu/~vg/papers/raid2010/data/ftp-98.txt`
[6] The automata package we used cannot convert it to a DFA
[7] Available at `https://tare.medisin.ntnu.no/pages/tools.php`.

Maze. This benchmark set is based on the Example 7, we generate 861 mazes with dimensions ranging from 10×10 to 50×50. For each height $h \in \{10, \ldots, 50\}$, we produce mazes of size $h \times w$ for all $w \in \{h, \ldots, 50\}$. This results in a total of $\sum_{h=10}^{5} 0(51 - h) = 861$ distinct maze sizes.

Walls were randomly placed with a 1/3 probability per cell. To generate a rejected path, we randomly select a path from the start to the end of the maze and introduce between 1 and 5 random changes to the path, ensuring that the modified path is still rejected by the maze. If the maze does not have a solution, we repeat the process until we find a solvable maze for that dimension.

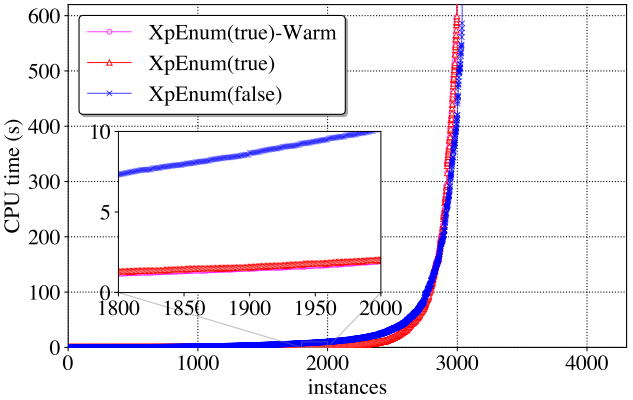


Figure 8: Rejected paths in mazes.

Figure 8 evaluates complete explanation enumeration, necessary for defining the FFA. Given that this is a complex task, ≈ 1000 problem instances reach the 600s time out. Targeting AXps is preferable except for the hardest instances. The warm start does not improve performance here.

Summary. Overall, our results indicate that our approach to enumerate AXps and CXps for finite automata is scalable. In particular, we are able to solve the problem for the automata with the size varying from 10 to 2,500, excluding the Deep Packet Inspection scenario, which has many more states. The average number of AXps and CXps is 50 and 255, respectively. The average number of unit-size CXps is 8. The average size of an AXp is 11 while the average size of a CXp is 15. Based on the instances where the enumeration process was completed, the average completion times were 23.97, 21.59, and 19.36 seconds for the Algorithm 2 with flags *false*, *true*, and *true* initialized with all singleton CXps, respectively. This shows better performance when targeting AXps, and warm-starting with all singleton CXps. However, harder cases favor targeting CXps directly.

Finally, the general observations made here for $\mathbb{W}_1^1$ also apply to $\mathbb{W}_1^\infty$ and $\mathbb{W}_0^\infty$ (results can be found in the appendix). Importantly, as suggested by Propositions 3–2, the structure and thus the number of explanations for $\mathbb{W}_1^\infty$ and $\mathbb{W}_0^\infty$ affect the performance of the enumeration and highlight an advantage of targeting CXps.

8 Conclusions and Related Work

We introduce a first approach to computing formal explanations for finite automata (FA) decisions $w \in L(\mathcal{A})$. We show we can usually efficiently compute a single AXp, even for large automata and long words. And for many problems we can enumerate all explanations, which enables us to determine formal feature attribution (FFA) for all the input symbols.

The results of our work are applicable to problems that often rely on finite automata, including lexical analysis, string searching, and pattern matching. While AXps can be employed to understand why certain acceptance or rejection decisions have been made, CXps can be used to suggest how those decisions can be altered. Furthermore, a significant advantage of our approach is that these formal explanations are verifiable. The proposed AXps and CXps provide logical guarantees that can be validated through formal methods, such as model checking. Finally, FFA can be applied to reveal the importance of certain features of the input words that are otherwise hidden.

References

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley series in computer science / World student series edition. Addison-Wesley, 1986.

[2] Alejandro Barredo Arrieta, Natalia Díaz Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. *Inf. Fusion*, 58:82–115, 2020.

[3] John L. Carroll and Darrell D. E. Long. *Theory of finite automata with an introduction to formal languages*. Prentice Hall, 1989.

[4] David Chocholatý, Tomás Fiedor, Vojtech Havlena, Lukás Holík, Martin Hruska, Ondrej Lengál, and Juraj Síc. Mata: A fast and simple finite automata library. In *TACAS (2)*, volume 14571 of *Lecture Notes in Computer Science*, pages 130–151. Springer, 2024.

[5] Mohamad H. Danesh, Anurag Koul, Alan Fern, and Saeed Khorram. Re-understanding finite-state representations of recurrent policy networks. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 2388–2397. PMLR, 2021.

[6] Adnan Darwiche. Logic for explainable AI. In *LICS*, pages 1–11. IEEE, 2023.

[7] Adnan Darwiche and Auguste Hirth. On the (complete) reasons behind decisions. *J. Log. Lang. Inf.*, 32(1):63–88, 2023.

[8] Patrik D'haeseleer. What are dna sequence motifs? *Nature Biotechnology*, 24(4):423–425, 2006.

[9] Graeme Gange, Pierre Ganty, and Peter J. Stuckey. Fixing the state budget: Approximation of regular languages with small dfas. In *ATVA*, volume 10482 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2017.

[10] Enrico Giunchiglia and Marco Maratea. Solving optimization problems with DLL. In *ECAI*, pages 377–381, 2006.

[11] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2024. URL: `http://www.gurobi.com`.

[12] Mohammadhosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham, and Daniel Kroening. Deepsynth: Automata synthesis for automatic task segmentation in deep reinforcement learning. In *AAAI*, pages 7647–7656. AAAI Press, 2021.

[13] Harry B. III Hunt. On the time and tape complexity of languages. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, pages 10–19. ACM Press, 1973. `doi:10.1145/800125.804030`.

[14] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018. `doi:10.1007/978-3-319-94144-8_26`.

[15] Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019.

[16] Alexey Ignatiev, Nina Narodytska, Nicholas Asher, and João Marques-Silva. From contrastive to abductive explanations and back again. In *AI*IA*, volume 12414 of *Lecture Notes in Computer Science*, pages 335–355. Springer, 2020.

[17] Alexey Ignatiev, Zi Li Tan, and Christos Karamanos. Towards universally accessible SAT technology. In *SAT*, pages 4:1–4:11, 2024. URL: `https://doi.org/10.4230/LIPIcs.SAT.2024.16`, `doi:10.4230/LIPICS.SAT.2024.16`.

[18] Mark H. Liffiton and Ammar Malik. Enumerating infeasibility: Finding multiple muses quickly. In *CPAIOR*, volume 7874 of *Lecture Notes in Computer Science*, pages 160–175. Springer, 2013.

[19] Mark H. Liffiton, Alessandro Previti, Ammar Malik, and João Marques-Silva. Fast, flexible MUS enumeration. *Constraints An Int. J.*, 21(2):223–250, 2016.

[20] João Marques-Silva and Alexey Ignatiev. Delivering trustworthy AI through formal XAI. In *AAAI*, pages 12342–12350. AAAI Press, 2022.

[21] Joao Marques-Silva and Alexey Ignatiev. No silver bullet: interpretable ML models must be explained. *Frontiers Artif. Intell.*, 6, 2023. URL: `https://doi.org/10.3389/frai.2023.1128212`, `doi:10.3389/FRAI.2023.1128212`.

[22] Stephanie Milani, Nicholay Topin, Manuela Veloso, and Fei Fang. Explainable reinforcement learning: A survey and comparative review. *ACM Comput. Surv.*, 56(7):168:1–168:36, 2024.

[23] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.*, 267:1–38, 2019.

[24] Dominique Perrin. Finite automata. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 1–57. Elsevier and MIT Press, 1990.

[25] Alessandro Previti and João Marques-Silva. Partial MUS enumeration. In Marie desJardins and Michael L. Littman, editors, *AAAI*, pages 818–825, 2013.

[26] Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2):114–125, 1959.

[27] Geir Kjetil Sandve, Osman Abul, Vegard Walseng, and Finn Drabløs. Improved benchmarks for computational motif discovery. *BMC Bioinform.*, 8, 2007.

[28] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[29] Robert A. Wagner. Order-n correction for regular languages. *Commun. ACM*, 17(5):265–268, 1974.

[30] Chengcheng Xu, Shuhui Chen, Jinshu Su, Siu-Ming Yiu, and Lucas Chi Kwong Hui. A survey on regular expression matching for deep packet inspection: Applications, algorithms, and hardware platforms. *IEEE Commun. Surv. Tutorials*, 18(4):2991–3029, 2016.

[31] Liu Yang, Rezwana Karim, Vinod Ganapathy, and Randy Smith. Improving nfa-based signature matching using ordered binary decision diagrams. In *RAID*, volume 6307 of *Lecture Notes in Computer Science*, pages 58–78. Springer, 2010.

[32] Jinqiang Yu, Graham Farr, Alexey Ignatiev, and Peter J. Stuckey. Anytime approximate formal feature attribution. In *SAT*, pages 30:1–30:23, 2024.

[33] Jinqiang Yu, Alexey Ignatiev, and Peter J. Stuckey. On formal feature attribution and its approximation. *CoRR*, abs/2307.03380, 2023.

A Supplementary Material

A.1 Propositions and proofs

This section provides the complete set of formal propositions and their corresponding proofs.

Proposition 1. *Given $w \in L(\mathcal{A})$, assume that the sets of all AXps and CXps are denoted as $\mathbb{A} = \{X \subseteq \{1,\ldots,|w|\} \mid \mathsf{AXp}^u_l(X,w) \in \mathbb{W}\}$ and $\mathbb{C} = \{Y \subseteq \{1,\ldots,|w|\} \mid \mathsf{CXp}^u_l(Y,w) \in \mathbb{W}\}$, respectively. It holds that each $X \in \mathbb{A}$ is a minimal hitting set of $\mathbb{C}$ and, vice versa, each $Y \in \mathbb{C}$ is minimal hitting set of $\mathbb{A}$.*

Proof. Using the explanation language $\mathbb{W}$, we can treat the symbols of input w as *features*, and $\mathsf{AXp}^u_l(S,w)$ is an AXp iff S is an AXp of the classification problem $\kappa(w) =$ "accept" s.t. κ is the DFA applied to w. Similarly $\mathsf{CXp}^u_l(S,w)$ is a CXp iff S is a CXp of the classification problem $\kappa(w) =$ "accept". Hence, we directly have the hitting set duality between AXps and CXps of $w \in L(\mathcal{A})$. □

Proposition 2. *Let $w \in L(\mathcal{A})$ and $l_1..u_1 \supseteq l_2..u_2$. If S is such that $\mathsf{AXp}^{u_1}_{l_1}(S,w)$ is a weak AXp for $w \in L(\mathcal{A})$ using $\mathbb{W}^{u_1}_{l_1}$ then $\mathsf{AXp}^{u_2}_{l_2}(S,w)$ is a weak AXp using $\mathbb{W}^{u_2}_{l_2}$.*

Proof. The relation between the intervals implies that $\Sigma^{u_1}_{l_1} \supseteq \Sigma^{u_2}_{l_1}$. Note that a larger set of replacements yields a larger set of languages for any S subset of $\{1,\ldots|w|\}$, and we have that $\mathsf{AXp}^{u_1}_{l_1}(S,w) \supseteq \mathsf{AXp}^{u_2}_{l_2}(S,w)$.

$\mathsf{AXp}^{u_1}_{l_1}(S,w)$ is a weak AXp, so $\mathsf{AXp}^{u_1}_{l_1}(S,w) \subseteq L(\mathcal{A})$ and hence $\mathsf{AXp}^{u_2}_{l_2}(S,w) \subseteq L(\mathcal{A})$. Thus, $\mathsf{AXp}^{u_2}_{l_2}(S,w)$ is a weak AXp for $w \in L(\mathcal{A})$ using $\mathbb{W}^{u_2}_{l_2}$. □

Proposition 3. *Let $w \in L(\mathcal{A})$, and $l_1..u_1 \subseteq l_2..u_2$. If S is such that $\mathsf{CXp}^{u_1}_{l_1}(S,w)$ is a weak CXp for $w \in L(\mathcal{A})$ using $\mathbb{W}^{u_1}_{l_1}$ then $\mathsf{CXp}^{u_2}_{l_2}(S,w)$ is a weak CXp using $\mathbb{W}^{u_2}_{l_2}$.*

Proof. By the definition of a CXp, $\mathsf{CXp}^{u_1}_{l_1}(S,w) \not\subseteq L(\mathcal{A})$, there exists a word $w' \in \mathsf{CXp}^{u_1}_{l_1}(S,w)$, such that $w' \notin L(\mathcal{A})$. Since $w' \in \mathsf{CXp}^{u_1}_{l_1}(S,w)$, each position i in S is replaced in w' by a string of length between l_1 and u_1. As $(l_1,u_1) \subseteq (l_2,u_2)$, the replacementsare also valid under bound (l_2,u_2), to have $\mathsf{CXp}^{u_2}_{l_2}(S,w)$. Thus, $\mathsf{CXp}^{u_2}_{l_2}(S,w)$ is a CXp, since it contains w and a counterexample $w' \notin L(\mathcal{A})$. □

Proposition 4. *Given an alphabet Σ and a parameter $n \in \mathbb{N}$, there exists a family of regular languages L_n over Σ such that any DFA $\mathcal{A}_n$ for L_n has least 2^n states while $\forall w \in \Sigma^*$ for any $\mathsf{AXp}^1_1(S,w)$ and any $\mathsf{CXp}^1_1(S,w)$ it holds that $|S| \leq k$, for some constant $k \in \mathbb{N}$, $k < n$.*

Proof. Let $\Sigma = \{\mathtt{a},\mathtt{b}\}$ and $R_n = \mathtt{(a|b)^*a(a|b)}^{n-1}$ defining the language of strings over Σ whose n^{th} symbol from the end is `a`. It is well-known that any DFA $\mathcal{A}_n$ for L_n has at least 2^n states [1, 28]. Now, consider any string $w \in \Sigma^*$. Observe that $w \in L_n$ if and only if the n^{th} character from the end of w is `a`. Hence, a sole AXp using $\mathbb{W}^1_1$ for this fact has size 1; namely, it is $\mathsf{AXp}^1_1(\{|w|-n+1\},w)$. (In other words, as long as we keep the symbol at position $|w|-n+1$ unchanged, the outcome of the DFA $\mathcal{A}_n$ will remain the same.) Similarly, changing this particular character is the only way to change the result of the DFA, i.e., $\mathsf{CXp}^1_1(\{|w|-n+1\},w)$ is the only CXp for w. □

Proposition 5. *Given an alphabet Σ and a parameter $n \in \mathbb{N}$, there exists a family of regular languages L_n over Σ and a word $w \notin L_n$ with the minimum edit distance of at least n such that for any AXp $\mathsf{AXp}^1_1(S,w)$ it holds that $|S| \leq k$, for some constant $k \in \mathbb{N}$, $k < n$.*

Proof. Let $\Sigma = \{\mathtt{a},\mathtt{b}\}$ and $R_n = \mathtt{(a|b)^*a}^n\mathtt{(a|b)^*}$ defining the language of strings containing n consecutive symbols `a`'s. Consider a string $w = \mathtt{bb\ldots b}$ of size $m = 2n$. Clearly, $w \notin L_n$, and the minimum edit distance from w to L_n is n, since at least n consecutive characters in w must be changed to `a`'s (starting from any suitable position) to obtain a word $w' \in L_n$. Observe that a possible AXp for w using $\mathbb{W}^1_1$ is $\mathsf{AXp}^1_1(\{1,n\},w)$. Indeed, as long as the two symbols at the beginning and the middle of w are both `b`, there is *no way* to obtain a word w' with n consecutive `a`'s. In fact, the complete set of AXps for w is $\mathsf{AXp}^1_1(\{i,j\},w)$ for any $i,j \in \mathbb{N}$ such that $i < n$ and $n \leq j < i+n$. Each such AXp has size 2. Similar reasoning applies more generally: when $w = \mathtt{bb\ldots b}$ has length $m = kn$, all the AXps of w are of size k. □

Proposition 6. *Given a DFA $\mathcal{A}$ with m states and a word w s.t. $|w| = n$, computing an AXp for $w \in L(\mathcal{A})$ in languages $\mathbb{W}^1_1$, $\mathbb{W}^\infty_1$, or $\mathbb{W}^\infty_0$ can be done in $O(|\Sigma|mn^2)$ time.*

Proof. The call EXTRACTAXP($\{1,\ldots,|w|\}, \mathcal{A}, w, l, u$) always generates a minimal AXp of $w \in L(\mathcal{A})$ in language $\mathbb{W}_l^u$. This follows since $\mathsf{AXp}_l^u(\{1,\ldots,|w|\}, w)$ is just a weak AXp $L(w)$, and the resulting set X is minimal by construction, since adding back any removed position results in a weak AXp, which is strictly larger than the AXp without the position. To extract a minimal AXp for $w \in L(\mathcal{A})$, the EXTRACTAXP procedure examines exactly n possible AXp candidates to find a minimal one. A candidate $\mathsf{AXp}_l^u(X, w)$ is kept as a DFA $\mathcal{A}_w$ recognising the corresponding language, which is initially set to recognise $L(w)$ (and so $\mathcal{A}_w$ initially has $n+1$ states) and it is updated *incrementally* at each iteration. Assuming the language $\mathbb{W}_1^1$, each iteration of Algorithm 1 involves the following operations:

1. replacing a $w[i]$-transition in $\mathcal{A}_w$ with a Σ-transition, in $O(1)$ time;
2. replacing a Σ-transition in $\mathcal{A}_w$ with a $w[i]$-transition, in $O(1)$ time (if the check determines that position i must be kept);
3. checking whether $\mathsf{AXp}_l^u(X \setminus \{i\}, w) \triangleq L(\mathcal{A}_w) \subseteq L(\mathcal{A})$, in $O(|\Sigma|mn)$ time because $|\mathcal{A}| = m$ and $|\mathcal{A}_w| = n+1$ (the above operations *do not add* new states in $\mathcal{A}_w$).

Hence, the overall complexity of the algorithm is $O(|\Sigma|mn^2)$ for the language $\mathbb{W}_1^1$.[8] □

Proposition 7. *Computing a single AXp using $\mathbb{W}_0^\infty$ for $w \in L(R)$ given a regular expression R is PSPACE-hard.*

Proof. Note that $\emptyset$ defines an $\mathsf{AXp}_0^\infty(\emptyset, w)$ for $w \in L(R)$ iff R is a *universal language*. If it is the case then $\mathsf{AXp}_0^\infty(\emptyset, w)$ is *unique*, due to minimality. An algorithm for determining an AXp for $w \in L(R)$ in $\mathbb{W}_0^\infty$ thus gives a check that $L(R)$ is universal, which is PSPACE-complete [13]. □

B Exponential Size of Explanations

Each type of explanation AXp/CXp may be exponential in the other type of explanation. Whenever one type of explanation AXp/CXp is a collection of disjoint sets, the collection of minimal hitting sets (dual explanation) is exponential.

Proposition 8. *Given an finite automaton $\mathcal{A}$, the number of AXps/CXps for a word $w \in L(\mathcal{A})$, computed using $\mathbb{W}_1^1$, may be exponential in both the size of the $\mathcal{A}$ and the number of its dual explanations.*

Proof. Let $\mathcal{A}$ be an FA over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\} \cup \{\mathtt{A}, \mathtt{B}, \ldots, \mathtt{Z}\}$, where $|\Sigma| = n$. The language $L(\mathcal{A})$ recognised by $\mathcal{A}$ is defined as:

$$L(\mathcal{A}) = \mathtt{ab}\Sigma^{\frac{n}{2}-2} \cup \Sigma^2\mathtt{cd}\Sigma^{\frac{n}{2}-4} \cup \Sigma^4\mathtt{ef}\Sigma^{\frac{n}{2}-6} \cup \ldots \cup \Sigma^{\frac{n}{2}-2}\mathtt{yz}$$

Each term in the union represents a language where a fixed adjacent pair of symbols are lowercase and appears at specific positions while the remaining positions can be any symbol from Σ.

If we consider the word w with all symbols in uppercase defined as: $w = \mathtt{ABCD...YZ}$, with $|w| = \lfloor \frac{n}{2} \rfloor$ then $w \notin L(\mathcal{A})$ because the considered word does not match any fixed adjacent pair of lowercase symbols in $L(\mathcal{A})$. The set of minimal CXps using $\mathbb{W}_1^1$ for w is: $\mathbb{C} = \{\mathsf{CXp}_1^1(\{1,2\}, w), \mathsf{CXp}_1^1(\{3,4\}, w), \ldots, \mathsf{CXp}_1^1(\{\lfloor \frac{n}{2} \rfloor - 1, \lfloor \frac{n}{2} \rfloor\}, w)\}$.

- $\{1,2\}$ is a CXp because $w' = abCD\ldots YZ$ is in $L(\mathcal{A})$ and $w \notin L(\mathcal{A})$.
- Similarly $\{3,4\}$ is a CXp because $w'' = ABcd\ldots YZ$ is in $L(\mathcal{A})$ and $w \notin L(\mathcal{A})$.
- The same reasoning holds for each pair in $\mathbb{C}$.

The number of CXps for w is $m = \lfloor \frac{n}{2} \rfloor \times \frac{1}{2}$ and all CXps are disjoint sets. The set of AXps is defined by the all minimal hitting sets of $\mathbb{C}$ due to the minimal hitting set duality, (see Theorem 1), it is of size 2^m, exponentially larger than the number of CXps (m) and the size of $\mathcal{A}$ (as an NFA) $O(m^2)$.

Similarly, if we consider the word w_2 with all symbols in lowercase defined as: $w_2 = \mathtt{abcd...yz}$, then $w_2 \in L(\mathcal{A})$ because the considered word matches at least a fixed adjacent pair of lowercase symbols in $L(\mathcal{A})$ (matches all fixed adjacent pairs). The set of minimal AXps using $\mathbb{W}_1^1$ for w_2 is:

[8] If we consider a more general language $\mathbb{W}_1^\infty$ ($\mathbb{W}_0^\infty$, resp.) then replacing a $w[i]$-transition in the corresponding DFA A_w with a Σ_1^∞-transition (Σ_0^∞-transition, resp.) adds a single new state to A_w in the worst case. This way, A_w still has $O(n)$ states at each iteration of the algorithm. Therefore, the complexity result holds.

Algorithm 3 EXTRACTCXP – a Single CXp Extraction

Input: Candidate set $\mathcal{Y}$, automaton $\mathcal{A}$, word w, bounds l, u
Output: Minimal $\mathcal{Y}$ with $\mathsf{CXp}_l^u(\mathcal{Y}, w) \not\subseteq L(\mathcal{A})$

if $\mathsf{CXp}_l^u(\mathcal{Y}, w) \subseteq L(\mathcal{A})$ **then**
  return $\bot$
for all $i \in \mathcal{Y}$ **do**
  if $\mathsf{CXp}_l^u(\mathcal{Y} \setminus \{i\}, w) \not\subseteq L(\mathcal{A})$ **then**
    $\mathcal{Y} \leftarrow \mathcal{Y} \setminus \{i\}$
return $\mathcal{Y}$

$\mathbb{A} = \{\mathsf{AXp}_1^1(\{1,2\}, w_2), \mathsf{AXp}_1^1(\{3,4\}, w_2), \ldots, \mathsf{AXp}_1^1(\{\lfloor \frac{n}{2} \rfloor - 1, \lfloor \frac{n}{2} \rfloor\}, w_2)\}$. Using the same reasoning as above, we can see that: the number of AXps for w_2 is $m = \lfloor \frac{n}{2} \rfloor \times \frac{1}{2}$ and all AXps are disjoint sets, for this case the number of CXps (2^m) is exponentially larger than the number of AXps (m) and the size of $\mathcal{A}$ (as an NFA) $O(m^2)$. □

C ExtractCXp

The extraction of a minimal CXp is outlined in Algorithm 3. The input is a candidate explanation $\mathcal{Y}$ where $\mathsf{CXp}(\mathcal{Y})$ is a weak CXp (includes at least one counterexample string to $L(\mathcal{A})$), an FA $\mathcal{A}$, and an accepted word w. EXTRACTCXP builds a candidate CXp automaton W from $\mathcal{Y}$ by trying to replace Σ characters in W by the corresponding character in w. It then checks whether $L(W) \subseteq L(\mathcal{A})$ which will only hold if the $L(W)$ includes no counterexamples. If $\mathcal{Y} = \{1, \ldots, |w|\}$ this means that $\Sigma^n \subseteq L(\mathcal{A})$ and there are *no CXps* for $w \in L(\mathcal{A})$. In this case, the function returns $\bot$ indicating there is no CXp. Otherwise, it tries replacing the Σ character at each position i with $w[i]$ and checks if $L(W) \subseteq L(\mathcal{A})$. If this is the case, we must leave position i free, and we record i in $\mathcal{Y}_{min}$. At the end we have constructed a minimal CXp since replacing any Σ character left over will leave no counterexample.

Similarly to EXTRACTAXP, procedure EXTRACTCXP has the complexity of $O(|\Sigma| mn^2)$.

Corollary 1. *Given a DFA $\mathcal{A}$ with m states and a word w s.t. $|w| = n$, finding a CXp for $w \in L(\mathcal{A})$ using $\mathbb{W}_1^1$, $\mathbb{W}_1^\infty$ or $\mathbb{W}_0^\infty$ can be done in $O(|\Sigma| mn^2)$ time.*

Proof. (Sketch) We can extract a CXp for $w \in L(\mathcal{A})$ using $\mathbb{W}_1^1$ using the call EXTRACTCXP$(1..|w|, \mathcal{A}, w, l, u)$. The correctness follows by construction, removing a position from $\mathcal{Y}$ only occurs when this maintians a CXp. The complexity argument is essentially the same. □

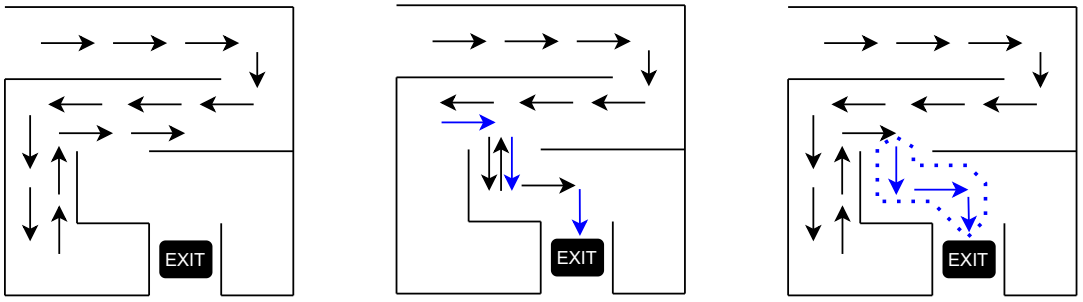
D Additional Experiments

As in the main paper, we compare the performance of different approaches for computing all AXps: XPENUM($true$) uses the XPENUM algorithm to find all AXps targeting AXps; XPENUM($true$)-WARM as above but warm starting with all singleton CXps; and XPENUM($false$) uses the XPENUM algorithm to find all AXps targeting CXps.

Maze using the languages $\mathbb{W}_0^\infty$ and $\mathbb{W}_1^\infty$. Similar to the experiments with mazes in the main paper, here we consider the same mazes but we use the languages $\mathbb{W}_0^\infty$ and $\mathbb{W}_1^\infty$. Note that explanations are different for different languages. Figure 9 shows an example of a maze with a rejected path and its minimal cardinality CXps using $\mathbb{W}_1^1$ and $\mathbb{W}_1^\infty$. Using language $\mathbb{W}_1^1$ the CXp with minimal cardinality is $\{8, 11, 13\}$ while using $\mathbb{W}_1^\infty$ the CXp with minimal cardinality is $\{13\}$ because one replacement can introduce many symbols.

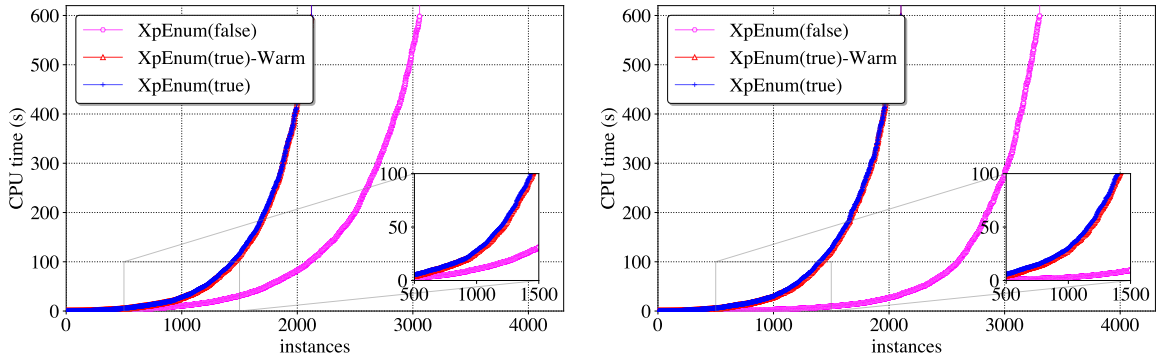
As experiments in the main paper, to evaluate this case, we generated mazes of different sizes, starting with grids of size 10×10 and increasing the dimension by 1 up to 50×50, getting 861 different maze sizes. We built mazes where each wall has 1 in 3 chance of being present. After the construction of each maze, we get a possible solution path and introduce $\{1, 2, \ldots, 5\}$ random changes in the path, getting five different rejected paths for each maze. If the random maze did not have any solution, we repeated the process until we found a solvable maze for that dimension.

In the enumeration of all AXps shown in Figure 10 we can see that the enumeration of all AXps using $\mathbb{W}_1^\infty$ and $\mathbb{W}_0^\infty$ is more challenging than using $\mathbb{W}_1^1$ depicted in the main paper not because because extracting a single AXp



(a) Example maze with a rejected path. (b) Minimal cardinality CXp using $\mathbb{W}_1^1$. (c) Minimal cardinality CXp using $\mathbb{W}_1^\infty$.

Figure 9: Maze with a rejected path and its minimal cardinality CXps using $\mathbb{W}_1^1$ and $\mathbb{W}_1^\infty$.



(a) Enumerating all AXps using $\mathbb{W}_1^\infty$ for maze. (b) Enumerating all AXps using $\mathbb{W}_0^\infty$ for maze.

Figure 10: Maze with a rejected path and its minimal cardinality CXps using $\mathbb{W}_1^1$ and $\mathbb{W}_1^\infty$.

is harder, but because there are more explanations to enumerate. Clearly for these examples targeting CXps is the best approach.

Table 2: Average number of explanations computed where enumeration process was completed using languages $\mathbb{W}_1^\infty$ and $\mathbb{W}_0^\infty$.

	$\mathbb{W}_1^\infty$		$\mathbb{W}_0^\infty$	
	AXp	CXp	AXp	CXp
XpEnum($false$)	6.3	3339.6	6.6	4328.5
XpEnum($true$)	5.0	585.8	5.0	573.4
XpEnum($true$)-Warm	5.0	581.3	5.0	570.1

Table 2 shows the average number of explanations computed for instances where the enumeration process was completed using languages $\mathbb{W}_1^\infty$ and $\mathbb{W}_0^\infty$. In most cases, the number of CXps is higher than the number of AXps. This is expected because using a languages $\mathbb{W}_1^\infty$ or $\mathbb{W}_0^\infty$ allows for greater flexibility in replacements, resulting in more possible ways to fix an input word. This data reveals the reasons why targeting CXps is the best approach for the enumeration of all explanations for these scenarios. As studied in the main paper, usually targeting CXps is the preferable when the problems get harder. Tipically, matching a word against a pattern is restricted leading to a limited number of AXps, but a large number of CXps.

Generated corpus. The generated corpus of FAs is the same as the one presented in the main paper, and in this section, we analyse the behaviour for rejected words using language $\mathbb{W}_1^1$. For the generated corpus, we generated m random words M with length l over the alphabet $\{1, \ldots, d\}$ for $l \in \{5, 10, 15, 20\}$, $m \in \{1, 3, 5, 10\}$,

Table 3: Average number of explanations computed if reaching the timeout.

	Corpus (rejected)	
	AXp	CXp
XpEnum($false$)	254.0	1.0
XpEnum($true$)	51976.7	540.2
XpEnum($true$)-Warm	52964.1	543.0

$d \in \{2, 3, 5, 10\}$. Figure 11 shows the results for the enumeration of all AXps for the rejected words. Then, we build an FA for each configuration such that it accepts the language $\{\Sigma^* w \Sigma^* \mid w \in M\}$.

Many patterns in this corpus are easy to match with long words. For example, with $l = 5$, $m = 1$, $d = 2$ a possible pattern is `(a|b)*abbba(a|b)*`. A long word that does not match this pattern may have a large number of didsjoint CXps, as many positions can be modified to match the pattern. Impliying that the number of AXps is exponential in the number of CXps (Proposition 8).

Table 3 shows, for the cases that did not finish within the time limit, the reason for the timeout is the large number of CXPs, and even more AXps, often growing exponentially.

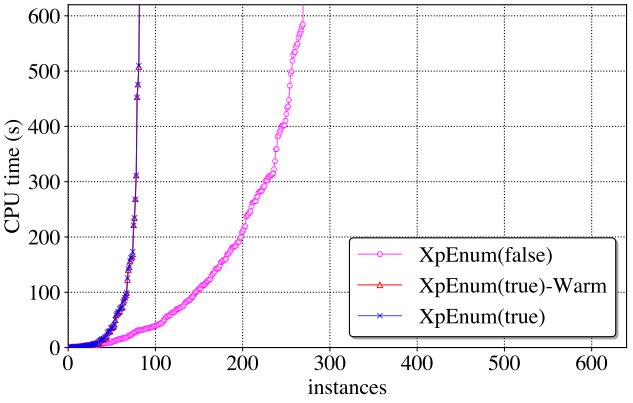


Figure 11: Generated corpus with **rejected** words