



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 Pushdown Automata, to address sequential actions with memory, and (3) Formal Explainability for stochastic models, where the outcomes are uncertain by the environment.

Contents

| | |
|-----------------------------------------------------------------------|-----|
| Abstract | i |
| List of Figures | iii |
| List of Tables | iv |
| 1 Introduction | 1 |
| 1.1 Refined scope - problem statement | 1 |
| 1.2 Contributions - achieved and projected | 2 |
| 2 Progress | 3 |
| 2.1 Model Proposal: Context-Free Grammar (CFG) Explanations | 3 |
| 2.2 Motivation: The Gap between Detection and Explanation | 3 |
| 2.3 Preliminaries | 5 |
| 2.3.1 Pushdown Automata and Context-Free Grammars | 5 |
| 2.3.2 Chomsky Normal Form | 7 |
| 2.3.3 The CYK Algorithm | 8 |
| 2.3.4 Classification Problems and Formal Explanations | 9 |
| 2.4 Explaining Pushdown Automata decisions | 10 |
| A An Appendix | 11 |
| Bibliography | 13 |

List of Figures

- 2.1 A Pushdown Automaton \mathcal{A} accepting the language of balanced parentheses. The symbol $\$$ is used as the bottom-of-stack marker. 6

List of Tables

| | |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|
| 2.1 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 | 8 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|

Chapter 1

Introduction

The deployment of Artificial Intelligence (AI) algorithms has necessitated the need for eXplainability AI (XAI) methods in order 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 formalization of explanations for sequential decision-making. Having addressed an approach to deliver formal explanations for Finite Automata (FA) in the first stage of this research, and submitting it to a ICALP 2026. I now move to address explanations for Context-Free Languages (CFG) / Pushdown Automata (PDA) decisions.

1.1 Refined scope - problem statement

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.** Finite Automata are often assumed to be interpretable. However, large FA are cognitively inaccessible to humans. I have developed a framework to compute formal

explanations for the acceptance and rejection of inputs in FA, providing a rigorous foundation for automaton-based explainability.

- **Research Problem 2: Explaining Pushdown Automata (PDA).** Context-free languages, recognized by PDAs, introduce a stack-based memory that allows to represent more complex behaviors. My objective is the generation of Minimal Contrastive Explanations (CXPs) the minimal sets of modifications required to turn a rejected word into an accepted one; and Minimal Abductive Explanations (AXps) the minimal sets of tokens such that the word is going to be rejected. Finally, there is a lack of quantitative metrics that assign a Feature Attribution Score (“degree of responsibility”) to specific indices in a rejected string.
- **Research Problem 3: Explaining Decisions of Sthochastic models.** How can the Feature Attribution Score be extended to explain outcomes for policies in stochastic environments? This problem explores sequential decision-making, where actions influence future states under uncertain contexts.

1.2 Contributions - achieved and projected

This research provides both theoretical and practical contributions to the field of Computer Science:

Achieved Contributions:

- Development of an theoretical and practical approach to explain Finite Automata decisions.
- A paper submitted to ICALP 2026 titled “A Formal Framework for the Explanation of Finite Automata Decisions”

Projected Contributions:

- Explaining Pushdown Automata decisions: (In Progress) Extending the formal explanation framework to PDAs, which recognize 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 Sthochastic models: Extending the formal explanation framework to Markov Decision Processes (MDPs). The goal is to provide verifiable explanations able to identify the environmental factors or decision points that lead to a particular outcomes.

Chapter 2

Progress

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

The research has evolved from the study of Finite Automata (FA) to more expressive computational models. While FA provided a baseline for explaining sequential behaviors, 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 auxiliary 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.
- **The Explanation Failure:** If an FA were used to validate such a sequence, it would process inputs locally. Upon encountering a mismatch (e.g., $a^4 b^2$), it might reject the sequence, but it lacks the structural context 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 FA to PDA:** We propose the use of Pushdown Automata (PDAs) to model decision-making processes with memory.
Unlike FA, the addition of a stack allows the description of more complex languages. Here the challenge is how to explain PDA decisions.

2.2 Motivation: The Gap between Detection and Explanation

One well established applications 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 initialization:

```

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).

2.3 Preliminaries

2.3.1 Pushdown Automata and Context-Free Grammars

To provide a foundation for the proposed explanation extraction methods, standard definitions and notations for Pushdown Automata and Context-Free Grammar are adopted [6, 7].

Definition 2.1 (Pushdown Automaton). A Pushdown Automaton (PDA) extends the capabilities of a Finite Automaton by incorporating an infinite memory stack. A PDA is formally defined as a 7-tuple $\mathcal{A} = (Q, \Sigma, V, \delta, q^0, v^0, F)$, where:

- Q is a finite set of states.
- Σ is the input alphabet, a finite terminal alphabet.
- V is the stack alphabet, a finite nonterminal alphabet that can be pushed onto or popped from the stack.
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times V \rightarrow \mathcal{P}(Q \times V^*)$ ¹ is the transition function. It dictates how the machine transitions between states and modifies the stack based on the current state, input symbol, and the top symbol of the stack.
- $q^0 \in Q$ is the initial state.
- $v^0 \in V$ is the initial pushdownstore symbol.
- $F \subseteq Q$ is the set of accepting states.

A configuration of \mathcal{A} is a triple $c = (q, \gamma, x)$ in $Q \times V^* \times \Sigma^*$. And the automaton moves from c into configuration $c' = (q', \gamma', x')$, denoted as $c \vdash c'$ if:

- $\gamma = v\gamma_1 (v \in V)$ ², $x = ax' (a \in \Sigma)$, $\gamma' = m\gamma_1 (m \in V^*)$ and $(q', m) \in \delta(q, a, v)$, namely “a-move”;
- or $\gamma = v\gamma_1 (v \in V)$, $x = x'$, $\gamma' = m\gamma_1 (m \in V^*)$ and $(q', m) \in \delta(q, \epsilon, v)$, namely “ ϵ -move”.

A *word* is accepted by a pushdown automaton if, starting with an empty stack, there is a path through the automaton such that the automaton stops in an accepting state after the entire string has been read. The *language* recognized by a PDA \mathcal{A} is the set of all accepted words, denoted as $L(\mathcal{A})$.

The following PDA recognizes the language of balanced parentheses (a subset of the Dyck language)³ and is used throughout the document to illustrate the proposed ideas.

Example 2.1. Let \mathcal{A} be the PDA that accepts the language generated by G in [Example 2.2](#). The PDA uses a stack to track the depth of nesting, pushing a symbol for every open parenthesis and popping for every closed one. B represents Balanced in G .

- States: $Q = \{q_0, q_1, q_f\}$, $q^0 = q_0$, $F = \{q_f\}$
- Input Alphabet: $\Sigma = \{(,)\}$

¹ \mathcal{P} denotes the power set (the set of all its subsets). V^* and Σ^* denote the Kleene closures of the stack and input alphabets, respectively, representing the sets of all finite strings formed by those alphabets.

² $\gamma_1 \in V^*$ represents the remaining symbols on the stack below the top symbol v . Thus, γ represents the full current stack, formed by the top symbol v (to be processed) and the rest of the stack γ_1 .

³The Dyck language describes a set of strings with balanced and properly nested brackets (e.g., $(,)$, $[]$, $\{\}\}$) [7]. The example focuses solely on non-empty sequences of balanced $($ and $)$.

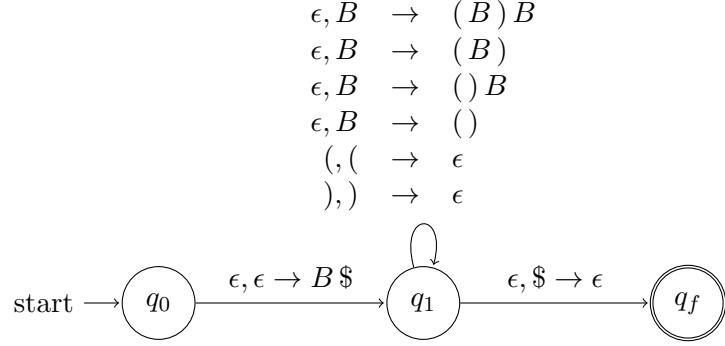


FIGURE 2.1: A Pushdown Automaton \mathcal{A} accepting the language of balanced parentheses. The symbol $\$$ is used as the bottom-of-stack marker.

- *Stack Alphabet:* $V = \{((), B, \$\}, v^0 = B$
- *Transitions:*
 1. $\delta(q_0, \epsilon, \epsilon) = \{(q_1, B \$)\}$ (*Initialize stack with Start Symbol*)
 2. $\delta(q_1, \epsilon, B) = \{(q_1, (B)B), (q_1, (B)), (q_1, ()B), (q_1, ())\}$ (*Expand B*)
 3. $\delta(q_1, (, ()) = \{(q_1, \epsilon)\}$ (*Match input ‘(’ with stack ‘(’*)
 4. $\delta(q_1,),)) = \{(q_1, \epsilon)\}$ (*Match input ‘)’ with stack ‘)’*)
 5. $\delta(q_1, \epsilon, \$) = \{(q_f, \epsilon)\}$ (*Accept if bottom marker is reached*)

Figure 2.1 illustrates this PDA. The configuration history of \mathcal{A} for the input “ $(())$ ” is:

$$\begin{aligned}
 & (q_0, ((), \epsilon) \\
 \vdash & (q_1, ((), B\$) \quad (\text{Initialize}) \\
 \vdash & (q_1, ((), ()B\$) \quad (\text{Expand } B \rightarrow ()B) \\
 \vdash & (q_1, ((),)B\$) \quad (\text{Match ‘(’}) \\
 \vdash & (q_1, ((), B\$) \quad (\text{Match ‘)’}) \\
 \vdash & (q_1, ((), ())\$) \quad (\text{Expand } B \rightarrow ()) \\
 \vdash & (q_1, ((),)\$) \quad (\text{Match ‘(’}) \\
 \vdash & (q_1, \epsilon, \$) \quad (\text{Match ‘)’}) \\
 \vdash & (q_f, \epsilon, \epsilon) \quad (\text{Accept})
 \end{aligned} \tag{2.1}$$

Definition 2.2 (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 $A \rightarrow \alpha$, 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.

A fundamental equivalence between CFGs and PDAs: a language L is context-free iff there exists a PDA \mathcal{A} such that $L(\mathcal{A}) = L$ [6, 7]. This equivalence allows us to use grammar-based parsing algorithms, such as CYK, to analyze the behavior of PDAs.

Example 2.2. Let $G = (\{Balanced\}, \{(,)\}, R, Balanced)$ be the grammar defined by the following production rules R :

$$\begin{aligned} Balanced &\rightarrow (Balanced) Balanced & (Rule 1) \\ Balanced &\rightarrow (Balanced) & (Rule 2) \\ Balanced &\rightarrow () Balanced & (Rule 3) \\ Balanced &\rightarrow () & (Rule 4) \end{aligned} \tag{2.2}$$

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

$$\begin{aligned} Balanced &\Rightarrow () \text{Balanced} & (Rule 3: \text{parentheses and Balanced}) \\ &\Rightarrow ()() & (Rule 4: \text{Reduce 'Balanced' to '()'}) \end{aligned} \tag{2.3}$$

To analyze the behavior of the PDA using grammar-based approaches, we must first standardize the grammar structure (e.g. Chomsky Normal Form). This enables the use of efficient parsing algorithms like CYK (Cocke-Younger-Kasami) [8, 9].

2.3.2 Chomsky Normal Form

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

Definition 2.3 (Chomsky Normal Form). A Context-Free Grammar $G = (V, \Sigma, R, S)$ is in *Chomsky Normal Form* (CNF) [6] if every production rule in R is of one of the following two forms:

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

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.

Example 2.3. Consider the grammar G from [Example 2.2](#). We transform G into an equivalent grammar $G' = (V', \Sigma, R', S)$ in CNF.

Step 1: Terminals to Non-terminals. To introduce variables L and R for terminals ‘(’ and ‘)’.

$$L \rightarrow (\quad \text{and} \quad R \rightarrow)$$

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

The resulting production rules R' are:

| | | | |
|-----------------|---------------|------------------------|---------------------------------------|
| <i>Balanced</i> | \rightarrow | <i>Nested Balanced</i> | (Represents '(Balanced) Balanced') |
| | | <i>Unclosed R</i> | (Represents '(Balanced)') |
| | | <i>Pair Balanced</i> | (Represents '() Balanced') |
| | | <i>L R</i> | (Represents '()') |
| <i>Nested</i> | \rightarrow | <i>Unclosed R</i> | |
| <i>Unclosed</i> | \rightarrow | <i>L Balanced</i> | |
| <i>Pair</i> | \rightarrow | <i>L R</i> | |
| <i>L</i> | \rightarrow | (| |
| <i>R</i> | \rightarrow |) | |

2.3.3 The CYK Algorithm

The Cocke-Younger-Kasami (CYK) [8, 9] 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 2.4 (CYK Table Construction). For an input string $w = w_1 w_2 \dots w_n$:

1. **Base Case:** For each $i \in \{1, \dots, n\}$, $T[i, i]$ contains A if there is a rule $A \rightarrow w_i$.
2. **Recursive Step ($j > i$):** $T[i, j]$ contains A if there exists a rule $A \rightarrow BC$ and a split point k ($i \leq k < j$) 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 2.4. The [Table 2.1](#) illustrates a CYK Table construction to verify the acceptance of the string $w = "()()$ using the CNF grammar derived in [Example 2.3](#). The top-right cell $T[1, 4]$ represents the entire string. It contains the Start symbol B (Balanced) because there is a rule 'Balanced \rightarrow Pair Balanced' for a split point $k = 2$ where $\text{Pair} \in T[1, 2]$ and $\text{Balanced} \in T[3, 4]$.

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

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

TABLE 2.1: 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:** $B = \text{Balanced}$, $P = \text{Pair}$

2.3.4 Classification Problems and Formal Explanations

Following [2, 10], 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 $\kappa : \mathbb{F} \rightarrow \mathcal{K}$.

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:

$$\forall (\mathbf{x} \in \mathbb{F}). \left[\bigwedge_{i \in \mathcal{X}} (x_i = v_i) \right] \rightarrow (\kappa(\mathbf{x}) = c) \quad (2.4)$$

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) $\mathcal{Y} \subseteq \mathcal{F}$ that, if allowed to change, enables the prediction's alteration. Formally, a contrastive explanation \mathcal{Y} is defined as follows:

$$\exists (\mathbf{x} \in \mathbb{F}). \left[\bigwedge_{j \notin \mathcal{Y}} (x_j = v_j) \right] \wedge (\kappa(\mathbf{x}) \neq c) \quad (2.5)$$

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 [11]. 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 $\mathbb{A}_{\mathbf{v}}$.⁴ This fact is the basis for the algorithms used for formal explanation *enumeration* [10, 12].

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 [12, 13] in determining the result that $\kappa(\mathbf{v}) = c$. We define the formal feature attribution of feature i as

$$\text{FFAv}(i) = \frac{|\{\mathcal{X} \mid \mathcal{X} \in \mathbb{A}_{\mathbf{v}}, i \in \mathcal{X}\}|}{|\mathbb{A}_{\mathbf{v}}|} \quad (2.6)$$

The proportion of all AXps shows the contribution of each feature to the decision. A *formal feature attribution* for $\kappa(\mathbf{v}) = c$ is the set $\{(i, \text{FFAv}(i)) \mid i \in \mathcal{F}, \text{FFAv}(i) > 0\}$.

Critically, formal feature attribution gives an *unbiased* measure of the influence of input features on the decision.

⁴Given 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.

2.4 Explaining Pushdown Automata decisions

Appendix A

An Appendix

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus at pulvinar nisi. Phasellus hendrerit, diam placerat interdum iaculis, mauris justo cursus risus, in viverra purus eros at ligula. Ut metus justo, consequat a tristique posuere, laoreet nec nibh. Etiam et scelerisque mauris. Phasellus vel massa magna. Ut non neque id tortor pharetra bibendum vitae sit amet nisi. Duis nec quam quam, sed euismod justo. Pellentesque eu tellus vitae ante tempus malesuada. Nunc accumsan, quam in congue consequat, lectus lectus dapibus erat, id aliquet urna neque at massa. Nulla facilisi. Morbi ullamcorper eleifend posuere. Donec libero leo, faucibus nec bibendum at, mattis et urna. Proin consectetur, nunc ut imperdiet lobortis, magna neque tincidunt lectus, id iaculis nisi justo id nibh. Pellentesque vel sem in erat vulputate faucibus molestie ut lorem.

Quisque tristique urna in lorem laoreet at laoreet quam congue. Donec dolor turpis, blandit non imperdiet aliquet, blandit et felis. In lorem nisi, pretium sit amet vestibulum sed, tempus et sem. Proin non ante turpis. Nulla imperdiet fringilla convallis. Vivamus vel bibendum nisl. Pellentesque justo lectus, molestie vel luctus sed, lobortis in libero. Nulla facilisi. Aliquam erat volutpat. Suspendisse vitae nunc nunc. Sed aliquet est suscipit sapien rhoncus non adipiscing nibh consequat. Aliquam metus urna, faucibus eu vulputate non, luctus eu justo.

Donec urna leo, vulputate vitae porta eu, vehicula blandit libero. Phasellus eget massa et leo condimentum mollis. Nullam molestie, justo at pellentesque vulputate, sapien velit ornare diam, nec gravida lacus augue non diam. Integer mattis lacus id libero ultrices sit amet mollis neque molestie. Integer ut leo eget mi volutpat congue. Vivamus sodales, turpis id venenatis placerat, tellus purus adipiscing magna, eu aliquam nibh dolor id nibh. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Sed cursus convallis quam nec vehicula. Sed vulputate neque eget odio fringilla ac sodales urna feugiat.

Phasellus nisi quam, volutpat non ullamcorper eget, congue fringilla leo. Cras et erat et nibh placerat commodo id ornare est. Nulla facilisi. Aenean pulvinar scelerisque eros eget interdum. Nunc pulvinar magna ut felis varius in hendrerit dolor accumsan. Nunc pellentesque magna quis magna bibendum non laoreet erat tincidunt. Nulla facilisi.

Duis eget massa sem, gravida interdum ipsum. Nulla nunc nisl, hendrerit sit amet commodo vel, varius id tellus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc ac dolor est. Suspendisse ultrices tincidunt metus eget accumsan. Nullam facilisis, justo vitae convallis sollicitudin, eros augue malesuada metus, nec sagittis diam nibh ut sapien. Duis blandit lectus vitae lorem aliquam nec euismod nisi volutpat. Vestibulum ornare dictum tortor, at faucibus justo tempor non. Nulla facilisi. Cras non massa nunc, eget euismod purus. Nunc metus ipsum, euismod a consectetur vel, hendrerit nec nunc.

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] 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.
- [7] 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>.
- [8] 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](https://doi.org/10.1016/S0019-9958(67)80007-X).
- [9] 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>.
- [10] João Marques-Silva and Alexey Ignatiev. Delivering trustworthy AI through formal XAI. In *AAAI*, pages 12342–12350. AAAI Press, 2022.

- [11] 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.
- [12] Jinqiang Yu, Alexey Ignatiev, and Peter J. Stuckey. On formal feature attribution and its approximation. *CoRR*, abs/2307.03380, 2023.
- [13] Jinqiang Yu, Graham Farr, Alexey Ignatiev, and Peter J. Stuckey. Anytime approximate formal feature attribution. In *SAT*, pages 30:1–30:23, 2024.