On the Tractability of SHAP Explanations under Markovian Distributions

Reda Marzouk [1] **Colin de la Higuera** [1]

Abstract

Thanks to its solid theoretical foundation, the SHAP framework is arguably one the most widely utilized frameworks for local explainability of ML models. Despite its popularity, its exact computation is known to be very challenging, proven to be NP-Hard in various configurations. Recent works have unveiled positive complexity results regarding the computation of the SHAP score for specific model families, encompassing decision trees, random forests, and some classes of boolean circuits. Yet, all these positive results hinge on the assumption of feature independence, often simplistic in real-world scenarios. In this article, we investigate the computational complexity of the SHAP score by relaxing this assumption and introducing a Markovian perspective. We show that, under the Markovian assumption, computing the SHAP score for the class of Weighted automata, Disjoint DNFs and Decision Trees can be performed in polynomial time, offering a first positive complexity result for the problem of SHAP score computation that transcends the limitations of the feature independence assumption.

1. Introduction

Since its introduction in the seminal paper (Lundberg & Lee, 2017), the local explanatory (**SH**apley **A**dditive ex**P**lanations) SHAP method gained increasing popularity in the field of interpretable ML. Nevertheless, one of its main limitations pertains to its computational intractabilty: In general, computing the SHAP score is NP-Hard (Bertossi et al., 2020; den Broeck et al., 2021). Recent studies have shown positive results regarding the tractability of computing the SHAP score under specific configurations. In particular, (Lundberg et al., 2020) proposed a polynomial-time algorithm, known as TreeSHAP, that purported to compute exactly the SHAP score for tree-based models. However, subsequent research (den Broeck et al., 2021; Arenas et al., 2023) has identified flaws in the main claim of TreeSHAP. Indeed, other works have demonstrated that the TreeSHAP algorithm is an implementation of *interventional* SHAP, another variant of the SHAP score (Janzing et al., 2020). A more rigorous proof for the tractability of the original SHAP score for various families of boolean functions has been provided in (den Broeck et al., 2021) when features are assumed to be independent. (Arenas et al., 2023) extended these positive results to cover the family of Decomposable Deterministic circuits which includes the family of decision trees among other classes of boolean circuits.

All the tractability results reported in the literature are, however, derived under the feature independence assumption. Although practical for its simplicity, this assumption is often irrealistic in real-case scenarios. A slight relaxation of this assumption has been examined in (den Broeck et al., 2021) through the lens of complexity theory by considering the family of naïve bayes models and empirical distributions. Computing the SHAP score for the family of decision trees under this relaxed assumption has been proven to be #P-Hard in both these settings.

Between independent distributions and latent variable models, an intermediate class of distributions that hasn't been explored yet is the class of Markovian distributions. Markovian distributions constitute an interesting class of distributions that incorporate a degree of feature correlation often considered sufficient to model various stochastic phenomena (Bassler et al., 2006; Kampen, 2007; Goutsias & Jenkinson, 2013).

Previous works examining the complexity of computing the SHAP score were mostly directed towards families of boolean functions. In this article, we shift our focus to sequential models, in particular the family of weighted automata (WAs). WAs offer a powerful formalism for modeling sequential tasks and encompass a large family of classical models, including Deterministic and Non deterministic finite automata, Hidden Markov Models, and has been shown to be equivalent to second-order linear RNNs (Rabusseau et al., 2019). They have been employed in various applications, such as NLP (Knight & May, 2009), speech processing (Pereira & Riley, 1996; Mohri et al.,

[1]LS2N, Université de Nantes, France. Correspondence to: Reda Marzouk <mohamed-reda.marzouk@univ-nantes.fr>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

2008) and image processing (Culik & Kari, 1993).

Recently, a line of works proposed WAs as *proxy* interpretation models for neural models (Okudono et al., 2020; Eyraud & Ayache, 2021; Weiss et al., 2019; Lacroce et al., 2021). All these works are motivated by the implicit assumption that WAs enjoy better transparency than their neural counterparts. However, the existing litterature lacks a formal argument to substantiate this claim. One of the primary motivations of this work is to shed some light on this issue.

The work presented in this article will primarily address the original formulation of the SHAP score, as introduced by (Lundberg & Lee, 2017) in their seminal paper. This specific variant of the SHAP score has proven to be particularly challenging from a computational viewpoint, and extends, up to a distributional shift, other variants such as the baseline SHAP (Sundararajan & Najmi, 2020). It's worth noting, however, that the axiomatic basis of the original SHAP score has been recurrently disputed in the academic discourse with several works discussing its limitation to capture elementary desirable properties of local models' explanations (Janzing et al., 2020; Sundararajan & Najmi, 2020; Huang & Marques-Silva, 2023).

The main results presented in this article are given as follows:

1. A constructive proof showing that the computation of the SHAP score for the class of WAs is tractable under the assumption that the background data generating distribution is Markovian (section 3).
2. Under the same assumption, a constructive proof of the tractability of computing the SHAP score for the class of disjoint DNFs and the family of decision trees (section 4).

2. Background

For a given integer $n > 0$, we denote by $[n]$ the set of all integers from 1 to n. The indicator function of a set X shall be denoted as I_X. Recall that an indicator function of a subset X in $\mathcal{X}$ is a binary-valued function that assigns the value 1 to $x \in X$, 0 otherwise.

A computational function problem $f : \mathcal{I} \to \mathbb{R}$, where $\mathcal{I}$ is referred to as the set of instances, is in FP if it can be computed exactly using an algorithm that runs in time polynomial in the size of the instance.

• **Languages and seq2seq languages.** Let Σ be a finite alphabet. The elements of Σ will be referred to as symbols. Σ^* (resp. Σ^∞) denotes the set of all finite (resp. infinite) sequences formed by Σ. For a given sequence $w \in \Sigma^*$, we denote by $|w|$ its length, $w_{i:j}$ the subsequence of w that spans from the i-th symbol to the j-th symbol in w, and w_i to refer to its i-th symbol. A language f is a mapping from Σ^* to $\mathbb{R}$. When the image of a language f is binary, then it will be called *unweighted*, in which case the language represents a subset of Σ^* equal to $L_f = f^{-1}(\{1\})$. We extend the definition of languages to cover unweighted languages over Σ^*, by allowing the notation: $f(L) \stackrel{\text{def}}{=} \sum_{w \in L} f(w)$ (if it exists) for an unweighted language L. An analogous concept of a language is the concept of a *seq2seq language*. For two finite alphabets Σ and Δ, a seq2seq language is a mapping from $\Sigma^* \times \Delta^*$ to $\mathbb{R}$.

When a language (or, a seq2seq language) f is computed by a model M, such as a weighted automaton (WA) or a weighted transducer (WT), we shall use the notation f_M to designate the language (or, seq2seq language) computed by M.

• **Operators over languages/seq2seq languages.** In this article, three operators over languages will be useful in our analysis. We shall briefly define them in the following:

1. **The product operator:** The product operator, also known as *the hadamard product (Droste & Gastin, 2009; Mohri, 2004)*, takes two languages $f,\ g$ over Σ^* and outputs the product language $f \cdot g$. We shall employ the notation $f \otimes g$ to refer to the product language of f and g.
2. **The partition constant operator:** The partition constant operator takes a language f over Σ^*, an integer n, and outputs the quantity $f(\Sigma^n) = \sum_{w \in \Sigma^n} f(w)$. The partition constant operation of a language f at the support $n > 0$ will be denoted as $|f|_n$.
3. **The projection operator:** The projection operator takes as input a language f over Σ^* and a seq2seq language g over $\Sigma^* \times \Delta^*$ and outputs a language h over Δ^* given as

$$h(u) = \sum_{w \in \Sigma^{|u|}} f(w) \cdot g(w, u)$$

In the sequel, we shall use the notation $\Pi(f, g)$ to refer to the projection operator.

• **Patterns.** For an alphabet Σ, a pattern p is a regular expression that takes the form: $\Sigma^{i_1} w_1 \ldots \Sigma^{i_n} w_n \Sigma^{i_{n+1}}$, where $\{i_k\}_{k \in [n+1]}$ is a set of integers, and $\{w_k\}_{k \in [n+1]}$ is a collection of sequences over Σ^*. The language accepted by a pattern p shall be denoted L_p. Analogous to sequences, the symbol $|p|$ will refer to its length. In addition, $|p|_\#$ will denote the number of occurrences of the symbol Σ in p.

In this article, the pattern formalism will be employed to represent coalitions of features in the SHAP score formula. Often, they shall be treated as sequences formed by an extended alphabet $\Sigma_\# = \Sigma \cup \{\#\}$, where $\#$ is a special

symbol that replaces the symbol Σ present in the regular expression associated to a pattern p. For example, the pattern $p = \Sigma 00 \Sigma^2$ over the binary alphabet $\Sigma = \{0, 1\}$ is represented by the sequence $p = \#00\#\#$ over $\Sigma_\#$.

By treating patterns as sequences over $\Sigma_\#^*$, we can describe languages over patterns in the usual way. In particular, the following languages over patterns will be used in the remainder of this article. Given a sequence $w \in \Sigma^*$ and an integer $k \in [|w|]$, define the (unweighted) language over $\Sigma_\#^*$ as:

$$\mathcal{L}_k^w \stackrel{\text{def}}{=} \{p \in \Sigma_\#^{|w|} : \ w \in L_p \wedge |p|_\# = k\}$$

The uniform distribution over the set $\mathcal{L}_k^w$ will be referred to as $\mathcal{P}_k^w$, and the language $\bigcup_{k \in [|w|]} \mathcal{L}_k^w$ as $\mathcal{L}^w$.

A final operation over patterns that will appear in the reformulation of the SHAP score formula introduced later in this section is the $\texttt{swap}$ operation. Given a pattern $p \in \Sigma_\#^*$, an integer $i \in [|w|]$, $\texttt{swap}(p, i)$ refers to the (perturbed) pattern p' generated by replacing the i-th element of p with $\#$. For example, $\texttt{swap}(\#\#00\#1, 3) = \#\#\#0\#1$.

• **Markovian distributions.** Formally, a Markovian probability distribution P over Σ^∞ is parametrized by $< P_{init}, \{P_n\}_{n>0} >$, where P_{init} is a probability distribution over Σ, and for any integer $n > 0$, P_n is a stochastic matrix [1] in $|\Sigma| \times |\Sigma|$. For an integer $n > 0$, P induces a probability distribution over the support Σ^n, denoted $P^{(n)}$, such that for any sequence $w \in \Sigma^n$:

$$P^{(n)}(w) \stackrel{\text{def}}{=} P(w\Sigma^\infty) = P_{init}(w_1) \cdot \prod_{i=1}^{n-1} P_i[w_i, w_{i+1}]$$

We shall abuse notation and use $P_i(\sigma'|\sigma)$ instead of $P_i[\sigma, \sigma']$ interpreted as the probability of generating the symbol σ' at position $i + 1$ conditioned on the generation of the symbol σ at position i. When there is no confusion of the support of the distribution, we shall omit the subscript from the notation $P^{(n)}$.

For computational considerations, we constrain the family of Markovian distributions to those whose set of parameters can be efficiently queried:

Definition 2.1. A Markovian distribution over Σ^∞ is polynomial-time computable if there exists an algorithmic procedure that takes as input an integer $n > 0$, runs in $O(\texttt{poly}(n, |\Sigma|))$ and outputs the transition matrix P_n.

As a notable example, the probability distribution generated by the class of 1-gram models is trivially polynomial-time computable. The set of polynomial-time computable Markovian distributions will be denoted $\texttt{MARKOV}$. When $P \in \texttt{MARKOV}$ is given as an input instance to a computational function problem, it refers to a machine that implements the algorithmic procedure defined implicitly in definition 2.1.

An additional technical assumption on Markovian distributions considered in this article is that all elements of their stochastic matrices and P_{init} are greater than 0.

[1] A stochastic matrix is a positive matrix such that the sum of its row elements is equal to 1

2.1. Weighted Automata/Transducers

• **Weighted Automata.** Weighted Automata (WAs) extend the classical family of finite automata accepting unweighted languages by allowing transitions to be endowed with weights, construed as probabilities, costs, or scores depending on the application at hand. A linear representation of WAs is formally defined as follows:

Definition 2.2. ((Denis & Esposito, 2008)) Let Σ be an alphabet and $n > 0$ be an integer. A WA A over Σ^* is represented by a tuple $< \alpha, \{A_\sigma\}_{\sigma \in \Sigma}, \beta >$ where $A_\sigma \in \mathbb{R}^{n \times n}$ is the transition matrix associated to a symbol σ in Σ, and α (resp. β) are vectors in $\mathbb{R}^n$ that represent the initial (resp. final) vectors. The integer n is called the size of A, denoted $\texttt{size}(A)$.

A WA $A =< \alpha, \{A_\sigma\}_{\sigma \in \Sigma}, \beta >$ over Σ^* computes the language

$$f_A(w) = \alpha^T \cdot A_w \cdot \beta$$

where $A_w \stackrel{\text{def}}{=} \prod_{i=1}^{|w|} A_{w_i}$.

• **Weighted Transducers.** Weighted Transducers (WTs) represent the analogous version of WAs adapted to model seq2seq languages. It has been employed in applications including speech processing (Mohri et al., 2008; Lehr & Shafran, 2010), machine translation (Kumar et al., 2006) and image processing (Culik & Friš, 1995)

Analogous to WAs, WTs admit a linear representation given as follows:

Definition 2.3. Let Σ, Δ be two finite alphabets and $n > 0$ be an integer. A WT T over $\Sigma^* \times \Delta^*$ is represented by the tuple $< \alpha, \{A_\sigma^{\sigma'}\}_{(\sigma,\sigma') \in \Sigma \times \Delta}, \beta >$, where $\alpha \in \mathbb{R}^n$, $A_\sigma^{\sigma'} \in \mathbb{R}^{n \times n}$, $\beta \in \mathbb{R}$. The integer n is called the size of T, denoted $\texttt{size}(T)$.

A WT $T =< \alpha, \{A_\sigma^{\sigma'}\}_{\sigma \in \Sigma, \sigma' \in \Delta}, \beta >$ over $\Sigma^* \times \Delta^*$ computes the seq2seq language

$$f_T(w, u) = \alpha^T \cdot \prod_{i=1}^{|w|} A_{w_i}^{u_i} \cdot \beta$$

where $(w, u) \in \Sigma^* \times \Delta^*$ such that $|w| = |u|$.

Earlier in this section, we introduced three operators over languages/seq2seq languages, namely the product operator, the partition constant and the projection operator. The algorithmic construction we shall furnish in later sections to compute the SHAP score will involve performing a sequence of these operations over languages/seq2seq languages described by WAs/WTs whose parametrization will depend on the input instance of the problem.

The following provides a technical lemma proving the computational efficiency of implementing these operators over languages/seq2seq languages represented by WAs/WTs.

Lemma 2.4. *Fix two finite alphabets Σ, Δ.*

1. ***The product operator.*** *There exists an algorithm that takes as input two WAs A, B, runs in $O(\texttt{poly}(\texttt{size}(A), \texttt{size}(B), |\Sigma|))$ and outputs a WA $A \otimes B$ that computes the product language $f_A \otimes f_B$.*

2. ***The partition constant operator.*** *There exists an algorithm that takes as input a WA A and an integer $n > 0$, runs in $O(\texttt{poly}(\texttt{size}(A), n, |\Sigma|))$ and outputs $|f_A|_n$.*

3. ***The projection operator.*** *There exists an algorithm that takes as input a WA A, a WT T, runs in $O(\texttt{poly}(\texttt{size}(A), \texttt{size}(T), |\Sigma|)$ and outputs the language $\Pi(f_A, f_T)$.*

The proof of lemma 2.4 can be found in appendix A.

2.2. The SHAP score.

Stemming its root from the field of cooperative game theory (Deng & Papadimitriou, 1994), the SHAP framework is built on top of an analogy between cooperative games and the local explainability problem of ML models. A cooperative game is described by a set of players N and a value function v that assigns a *generated wealth* for each subset of players, referred to as a *coalition*, cooperating in the game. By analogy, in the context of explainable ML, the players are the input features of a ML model subject to explanatory analysis. And, the value assigned to a coalition is equal to the the expected model's output conditioned on the event that the features forming the coalition possess a value equal to the instance to explain.

Similar to Shapley's original cooperative game theory (Shapley, 1953), the SHAP explainability method offers at its core a formal characterization of a fair distribution mechanism across input features that reflects their respective degree of contribution to the generated model's output for a given instance to explain, culminating in what's commonly known as the SHAP score.

Formally, let M be a model that computes a function f_M from a discrete set $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ to $\mathbb{R}$, and P be a probability distribution over $\mathcal{X}$. For an input $x \in \mathcal{X}$, and an integer $i \in [n]$, the original SHAP score assigned to the i-th feature for the instance x is given as (Lundberg & Lee, 2017):

$$\text{SHAP}(M, x, i, P) \stackrel{\text{def}}{=} \sum_{S \subseteq [n]} \frac{|S|!(n - |S| - 1)!}{n!} \cdot [v(S; M, x, P) - v(S \setminus \{i\}; M, x, P)] \tag{1}$$

where for a subset $S \subseteq [n]$, the value function v is defined as

$$v(S; M, x, P) \stackrel{\text{def}}{=} \mathbb{E}_{X \sim P}[f_M(X) | X_S = x_S] \tag{2}$$

We propose an alternative formulation of the SHAP score formula tailored to better suit sequential models computing languages. For an alphabet Σ, a model M that computes a language over Σ^*, a probability distribution P over Σ^∞, a string $w \in \Sigma^*$ and an integer $i \in [|w|]$. The SHAP value assigned to the symbol w_i in w is given as:

$$\text{SHAP}(M, w, i, P) = \sum_{k=1}^{|w|-1} \frac{1}{|w| - k} \mathbb{E}_{p \sim \mathcal{P}^w_{|w|-k}}[V(p; M, w, P) - V(\texttt{swap}(p, i); M, w, P)] \tag{3}$$

where

$$V(p; M, w, P) \stackrel{\text{def}}{=} \mathbb{E}_{w' \sim P^{|w|}}[f_M(w') | w' \in L_p] \tag{4}$$

The main idea behind this reformulation consists at modeling coalitions as patterns. For example, for a sequence $w = abbaa$ over the alphabet $\Sigma = \{a, b\}$, and $k = 2$, the pattern $p = \#b\#a\#$ in $\mathcal{L}^w_3$ coincides with the coalition of size 2 formed by the second and the forth symbol of w.

The faithfulness of the SHAP value formula given in (3) to the one in (1) (for the case of sequential models) can be checked by decomposing the summation in the original formulation of the SHAP score (equation (1)) over coalitions of the same size, and by noting that for $k \in [|w| - 1]$, and a pattern $p \in \mathcal{L}^w_{|w|-k}$, we have

$$\mathcal{P}^w_{|w|-k}(p) = \frac{1}{|L^w_{|w|-k}|} = \frac{k! \cdot (|w| - k)!}{|w|!}$$

In the sequel, whenever the SHAP score formula is mentioned, it shall refer to the one tailored for sequential models using the pattern formalism (equation (3)). To avoid confusion between models computing languages and boolean functions treated in section 4.4, we shall use the notation $\overrightarrow{\text{SHAP}}$ for this latter case.

The formal definition of the *meta*-computational problem associated to SHAP score is given as follows:

Fix an alphabet Σ. Let $\mathcal{M}$ be a class of sequential models that compute languages over Σ^*, and $\mathcal{P}$ is a class of

probability distributions over Σ^∞. The computational *meta*-problem associated to the SHAP score is given formally as follows:

• **Problem:** $\texttt{SHAP}(\mathcal{M}, \mathcal{P})$
Instance: $M \in \mathcal{M}$, a sequence $w \in \Sigma^*$, an integer $i \in [|w|]$, and $P \in \mathcal{P}$
Output: Compute $\text{SHAP}(M, w, i, P)$

The next section is dedicated to the examination of the computational complexity of the particular instance of this problem where $\mathcal{M} = \texttt{WA}$ and $\mathcal{P} = \texttt{MARKOV}$, namely $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$.

3. The problem $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$ is in FP.

The main result of the article is stated in the following theorem:

Theorem 3.1. *The computational problem* $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$ *is in* FP.

In essence, Theorem 3.1 states the existence of an algorithm that computes exactly the SHAP score for the class of WAs under Markovian distributions in $O(\texttt{poly}(\texttt{size}(A), |\Sigma|, |w|))$ time where A is the WA given in input instance.

The remainder of this section is dedicated to provide the high-level steps of the proof of theorem 3.1. Technically engaged proofs of intermediary results are delegated to the appendix. At a high-level, the structure of the proof follows two steps, where the second step is decomposed in two sub-steps:

1. *A decomposition of the problem* $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$*:* The first step involves a decomposition of the SHAP score formula into a sum of functions, denoted SHAP_1, SHAP_2, which will be defined later in this section. By means of a reduction argument, we shall prove that if the computational problems associated to SHAP_1, SHAP_2 are in FP, then $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$ is also in *FP* (lemma 3.2).
2. *The problems* $\textit{SHAP}_1$, $\textit{SHAP}_2$ *are in* FP*:* In the second step, we shall show that the computational problems associated to SHAP_1, SHAP_2 are in FP. (lemma 3.3). The proof of this statement will follow two sub-steps:
   (a) In the first sub-step, we shall prove that the computation of SHAP_1 and SHAP_2 is reduced to performing a finite sequence of operations over languages/seq2seq languages whose parametrization will depend on the input instance of the problem (lemma 3.4).
   (b) In the second sub-step, we show that WAs/WTs that compute languages/seq2seq languages over which operations are performed in the previous step can be constructed.

The proof is essentially constructive, and can be translated to a practical implementation. The organisation of the remainder of this section will follow the structure of the proof given above.

3.1. Step 1: A decomposition of the problem $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$.

For a model M computing a language over Σ^*, a sequence $w \in \Sigma^*$, an integer $(i, k) \in [|w|] \times [|w| - 1]$, and P a probability distribution over Σ^∞. Define the following two functions:

$$\text{SHAP}_1(M, w, k, P) \stackrel{\text{def}}{=} \mathbb{E}_{p \sim \mathcal{P}_k^w} V(p; M, w, P) \tag{5}$$

and,

$$\text{SHAP}_2(M, w, i, k, P) \stackrel{\text{def}}{=} \mathbb{E}_{p \sim \mathcal{P}_k^w} V(\texttt{swap}(p, i); M, w, P) \tag{6}$$

By a simple manipulation of the SHAP score formula in (3), we obtain

$$\text{SHAP}(M, w, i, P) = \sum_{k=1}^{|w|-1} \frac{1}{k} [\text{SHAP}_1(M, w, k, P) - \text{SHAP}_2(M, w, i, k, P)] \tag{7}$$

The formal definition of the computational problems associated with the computation of $\texttt{SHAP}_1$, $\texttt{SHAP}_2$ for the class of WAs under the family of Markovian distributions is given as follows:

• **Problem:** $\texttt{SHAP}_1(\texttt{WA}, \texttt{MARKOV})$
Instance: A WA A, a sequence w in Σ^*, an integer $k \in [|w|]$, $P \in \texttt{MARKOV}$
Output: Compute $\text{SHAP}_1(A, w, k, P)$

• **Problem:** $\texttt{SHAP}_2(\texttt{WA}, \texttt{MARKOV})$
Instance: A WA A, a sequence w in Σ^*, two integers $(k, i) \in [|w|]^2$, $P \in \texttt{MARKOV}$
Output: Compute $\text{SHAP}_2(A, w, i, k, P)$

The polynomial-time reduction of the problem $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$ to $\texttt{SHAP}_1(\texttt{WA}, \texttt{MARKOV})$ and $\texttt{SHAP}_2(\texttt{WA}, \texttt{MARKOV})$ is straightforward in light of equation (7). The following lemma formally states this fact:

Lemma 3.2. *If* $\texttt{SHAP}_1(\texttt{WA}, \texttt{MARKOV})$ *and* $\texttt{SHAP}_2(\texttt{WA}, \texttt{MARKOV})$ *are in* FP*, then* $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$ *is in* FP.

Proof. The proof is straightforwardly obtained from equation (7). Assume $\texttt{SHAP}_1(\texttt{WA}, \texttt{MARKOV})$ and

$\texttt{SHAP}_2(\texttt{WA}, \texttt{MARKOV})$ are in FP. Then, there exists two algorithms, say $\mathcal{A}_1$ and $\mathcal{A}_2$, that solve the problems $\texttt{SHAP}_1$ and $\texttt{SHAP}_2$ respectively in $O(\texttt{poly}(\texttt{size}(A), |w|, |\Sigma|))$ time.

Fix an input of instance $< M, w, i, P >$ of $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$. To compute $\texttt{SHAP}(M, w, i, P)$ using $\mathcal{A}_1$, $\mathcal{A}_2$ as oracles, run the following schema:

1. Call $\mathcal{A}_1$ on the set of input instances $\{< M, w, k, p >\}_{k\in[|w|]}$ yielding $\{y_k\}_{k\in[|w|]}$
2. Call $\mathcal{A}_2$ on the set of input instances $\{< M, w, i, k, P >\}_{k\in[|w|]}$ yielding $\{y'_k\}_{k\in[|w|]}$
3. Output: $\sum_{k=1}^{|w|-1} \frac{1}{k}(y_k - y'_k)$

The correctness of this schema to solve $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$ is guaranteed by equation (7). In addition, by assumptions on $\mathcal{A}_1$, $\mathcal{A}_2$, this schema runs also in $O(\texttt{poly}(\texttt{size}(A), |\Sigma|, |w|)$ time. □

3.2. Step 2: $\texttt{SHAP}_1(\texttt{WA}, \texttt{MARKOV})$ and $\texttt{SHAP}_2(\texttt{WA}, \texttt{MARKOV})$ are in FP.

This segment is dedicated to provide the outline of the proof of the following lemma:

Lemma 3.3. *The problems $\texttt{SHAP}_1(\texttt{WA}, \texttt{MARKOV})$ and $\texttt{SHAP}_2(\texttt{WA}, \texttt{MARKOV})$ are in* FP.

The result of the main theorem 3.1 is an immediate corollary of lemma 3.2 and lemma 3.3 presented in the previous segment of this section.

The proof of lemma 3.3 will follow two steps. In the first step, the formulas of SHAP_1 and SHAP_2 will be reformulated in terms of operations over languages/seq2seq languages defined in section 2. The parametrization of these languages depends on the input instance of the problem. In the second step, we will show that WAs and WTs can be constructed in polynomial time that compute these languages/seq2seq languages. Combining the results of the two steps and the efficiency of implementing these operators for the case of WAs/WTs (lemma 2.4), the proof of lemma 3.3 can be easily obtained.

3.2.1. Step 2.a: Computation SHAP_1, SHAP_2 in terms of language operators.

The following lemma provides a reformulation of the functions SHAP_1 and SHAP_2 in the form of operations over languages whose properties depend on the input instance of their respective problems:

Lemma 3.4. *Let A be a WA over Σ^*, a sequence $w \in \Sigma^*$, two integers $(i, k) \in [|w|] \times [|w|-1]$, and P be an arbitrary probability distribution over Σ^∞. We have*

$$\textit{SHAP}_1(A, w, k, P) = |f_{w,k} \otimes \Pi(f_A, g^{(1)}_{w,P})|_{|w|} \quad (8)$$

and,

$$\textit{SHAP}_2(A, w, i, k, P) = |f_{w,k} \otimes \Pi(f_A, g^{(2)}_{w,i,P})|_{|w|} \quad (9)$$

where

- $f_{w,k} = \mathcal{P}^w_k$,
- $g^{(1)}_{w,P}$ *is a seq2seq language over $\Sigma^* \times \Sigma^*_\#$ that satisfies the following constraint:*

$$\forall (w', p) \in \Sigma^{|w|} \times \Sigma^{|w|}_\# : \; g^{(1)}_{w,P}(w', p) = P(w' | w' \in L_p) \quad (10)$$

- $g^{(2)}_{w,i,p}$ *is a seq2seq language over $\Sigma^* \times \Sigma^*_\#$ that satisfies the following constraint:*

$$\forall (w', p) \in \Sigma^{|w|} \times \Sigma^{|w|}_\# : \; g^{(2)}_{w,i,P}(w', p) = P(w' | w' \in L_{p'}) \quad (11)$$

where $p' = \texttt{swap}(p, i)$

The proof is given in appendix B.

Expressions (8) and (9) reduce the problem of computing SHAP_1 and SHAP_2 to that of performing operations over a language $f_{w,k}$ and two seq2seq languages, $g^{(1)}_{w,P}$, $g^{(2)}_{w,i,P}$ whose properties are given by equations (10) and (11) , respectively. The missing link to complete the proof of lemma 3.3 is to prove that a WA that implements the language $f_{w,k}$, and WTs that compute seq2seq languages $g^{(1)}_{w,P}$, $g^{(2)}_{w,i,P}$ whose properties are given in lemma 3.4 can be constructed in polynomial time.

3.2.2. Step 2.b: Construction of WAs/WTs that compute $f_{w,k}$, $g^{(1)}_{w,P}$, $g^{(2)}_{w,i,P}$

The key insight of the article is the following:

If $P \in \texttt{MARKOV}$, two seq2seq languages $g^{(1)}_{w,P}$ and $g^{(2)}_{w,i,P}$ that satisfy the constraints (10) and (11), respectively, admit a representation using the WA/WT formalism. In addition, the construction of WAs and WTs that compute these languages/seq2seq languages can be performed in time polynomial in the size of the input instance.

The next lemma provides a formal statement of this fact while also covering the language $f_{w,k}$.

Lemma 3.5. *1. **The language** $f_{w,k}$**:** There exists an algorithm $\mathcal{A}_1$ that takes as input, a sequence $w \in \Sigma^*$, an integer $k \in [|w| - 1]$, runs in $O(\texttt{poly}(|w|))$, and outputs a WA $A_{k,w}$ over $\Sigma^*_\#$ that computes the language $f_{w,k} = \mathcal{P}^w_k$.*

2. ***The seq2seq language*** $g^{(1)}_{w,P}$***:*** *There exists an algorithm* $\mathcal{A}_2$ *that takes a sequence* $w \in \Sigma^*$*, and* $P \in$ *MARKOV, runs in* $O(\texttt{poly}(|w|, |\Sigma|))$*, and outputs a WT* $T_{w,P}$ *that computes a seq2seq language that satisfies the constraint* (10).

3. ***The seq2seq language*** $g^{(2)}_{w,i,P}$***:*** *There exists an algorithm* $\mathcal{A}_3$ *that takes as input a sequence* $w \in \Sigma^*$*, an integer* $i \in [|w|]$*, and* $P \in$ *MARKOV, runs in* $O(\texttt{poly}(|w|, |\Sigma|))$*, and outputs a WT that implements a seq2seq language over that satisfies the constraint* (11)

In the sequel, we shall refer to algorithms that compute $f_{w,k}$, $g^{(1)}_{w,P}$ and $g^{(2)}_{w,i,P}$ by $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, respectively.

The proof of lemma 3.5 is constructive, and can be found in appendix C.

The construction of $\mathcal{A}_1$ is relatively easy. As for $\mathcal{A}_2$ and $\mathcal{A}_3$, the key observation stems from the Bayes' formula:

$$P(w|w \in L_p) = \frac{P(w) \cdot I_{L_p}(w)}{P(L_p)} \tag{12}$$

In light of the equation (12), the construction of $\mathcal{A}_2$ and $\mathcal{A}_3$ will follow the same spirit of the main algorithm for solving $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$. In other words, it will involve the construction of a WT over $\Sigma^* \times \Sigma^*_{\#}$ that implements the language $I_{L_p}(w)$, and two WAs over Σ^* and $\Sigma^*_{\#}$ that implement the languages $P(w)$ and $\frac{1}{P(L_p)}$, respectively. Since WAs/WTs are not closed under the division operation, the major difficulty in the construction lies in the design of a WA that implements the language $\frac{1}{P(L_p)}$ involving a division operation.

We note that since $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ run in time polynomial in their respective input instances implies that the size of their output machines is also polynomial in the size of their input instance[2]. This fact will appear explicitly in the constructive proof of lemma 3.5.

In light of lemma 3.4 and 3.5, we are ready to prove the main lemma of this subsection:

Proof. (lemma 3.3) We shall prove that $\texttt{SHAP}_1$ is in FP. A similar argument can be applied to derive the same result for SHAP_2.

Define the following algorithmic schema that takes as input an instance $< A, w, k, P >$ where A is a WA, $w \in \Sigma^*$, $i \in [|w|]$ and $P \in \texttt{MARKOV}$:

1. $A_{w,k} \leftarrow \mathcal{A}_1(w, k)$
2. $T_{w,P} \leftarrow \mathcal{A}_2(w, P)$
3. **Output:** $|f_{A_{w,k}} \otimes \Pi(f_A, f_{T_{w,P}})|_{|w|}$

By lemma 3.4 (equation (8)), and the properties of $\mathcal{A}_1$, $\mathcal{A}_2$ (lemma 3.5), this schema solves exactly the problem $\texttt{SHAP}_1(\texttt{WA}, \texttt{MARKOV})$.

In addition, this schema also runs in $O(\texttt{poly}(\texttt{size}(A), |w|, |\Sigma|))$. Indeed, by lemma 3.5, steps 1 and 2 run in $O(\texttt{poly}(|w|)$ and $O(\texttt{poly}(|w|, |\Sigma|))$, respectively. Consequently, by *FP* $\subset$ *FP*SPACE, the size of their outputs $A_{w,k}$, and $T_{w,k,P}$ is also polynomial in $|w|$ and $|\Sigma|$.

On the other hand, given that the operators $\otimes, |.|_n$, Π over languages represented by WAs/WTs can be computed in polynomial time with respective to the size of their input instances (lemma 2.4), this proves that the third step of the schema also runs in $O(\texttt{poly}(\texttt{size}(A), |w|, |\Sigma|))$ time.

□

4. SHAP(D-DNF, MARKOV) and SHAP(DT, MARKOV) are in FP

In this section, we switch our focus to boolean functions, in particular the class of disjoint-DNFs (d-DNF) [3]. The choice of this family of models is mainly motivated by the fact that it encompasses the family of decision trees, a central class of *glass-box* models capturing substantial attention within the explainable AI community. Recent works have been dedicated to exploring the computation of SHAP scores for Tree-based models across diverse configurations (Lundberg et al., 2020; Yang, 2021; Arenas et al., 2023; Yu et al., 2022). Later in this section, we shall prove that computing the SHAP score for the family of decision trees under Markovian distributions is reducible in polynomial time to $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$, offering a polynomial-time algorithmic construction to compute the original SHAP score for the family of decision trees under the Markovian assumption.

The class of disjoint-DNFs (d-DNFs) is formally defined as follows:

Definition 4.1 (Disjoint DNF)**.** A d-DNF is a logical expression $\Phi(X_1, X_2, \ldots, X_n)$ where $\{X_1, X_2, \ldots, X_n\}$ is a set of input boolean variables, such that:

- Φ is expressed as a disjunction (logical OR) of clauses, where each clause is expressed as one or more conjunctions (logical AND) of literals.

[2] FP $\subset$ FPSPACE

[3] For the general case of arbitrary DNFs, it has been shown that computing the SHAP score for this class of models when features are assumed to be independent is intractable under widely believed complexity assumptions (Arenas et al., 2023).

- Each clause in the expression is mutually exclusive from the others, ensuring that for any input combination $(X_1, X_2, \ldots, X_n)$, only one clause evaluates to true.

Example. Let $X = \{X_1, X_2, X_3, X_4\}$ be a set of binary variables. The formula

$$\Phi = (X_1 \wedge X_3 \wedge X_4) \vee (\bar{X}_1 \wedge X_2 \wedge X_3) \vee (X_2 \wedge \bar{X}_3) \quad (13)$$

is a d-DNF over the variables $\{X_i\}_{i\in[4]}$ comprising 3 clauses. Indeed, for any two distinct clauses (C_i, C_j) for $(i, j) \in [3]^2$, the intersection of the set of satisfying variable assignments for C_i and C_j is empty.

A Markovian distribution P over a boolean random vector of dimension N is given as:

$$P(X_1, \ldots, X_N) = P_{init}(X_1) \prod_{i=1}^{N-1} P_i(X_{i+1}|X_i)$$

To avoid confusion with the sequential case, the set of Markovian distributions over boolean vectors shall be denoted $\overrightarrow{\texttt{MARKOV}}$. For an integer $N > 0$, $\overrightarrow{\texttt{MARKOV}}_N$ will refer to the set of Markovian distributions over boolean vectors of dimension N.

The formal definition of the computational problem associated to compute the SHAP score of the class of d-DNFs under Markovian distributions is given as follows:

• **Problem:** $\texttt{SHAP}(\texttt{d-DNF}, \overrightarrow{\texttt{MARKOV}})$
Instance: A d-DNF Φ over N boolean variables, an instance $\overrightarrow{x} \in \{0,1\}^N$, an integer $i \in [N]$, $P \in \overrightarrow{\texttt{MARKOV}}_N$
Output: Compute $\overrightarrow{\text{SHAP}}(\Phi, x, i, P)$

The complexity size of the input instance of this problem is given by the number of variables of Φ, denoted $|\Phi|$, and the number of clauses in the d-DNF denoted $|\Phi|_{\#}$.

The claim of this section is given in the following theorem:

Theorem 4.2. *$\texttt{SHAP}(\texttt{d-DNF}, \overrightarrow{\texttt{MARKOV}})$ is in* FP.

The proof of theorem 4.2 will proceed by reduction to the problem $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$.

Before providing the details of the reduction strategy, we shall present an interesting corollary of theorem 4.2, stating that the SHAP score computational problem for the family of decision trees under Markovian distributions is in FP.

Corollary 4.3. *Denote by $\texttt{DT}$ the set of decision trees computing boolean functions. The problem $\texttt{SHAP}(\texttt{DT}, \overrightarrow{\texttt{MARKOV}})$ is in FP.*

Proof. This result follows immediately from theorem 4.2, and the fact that given an arbitrary decision tree in $\texttt{DT}$, an equivalent d-DNF can be constructed in polynomial time with respect to the size of the decision tree (Property 1, (Aizenstein & Pitt, 1992)). □

4.1. Proof of theorem 4.2: Reduction strategy

Unlike WAs, d-DNFs compute boolean functions instead of languages. For the sake of the reduction, a first step consists at performing a *sequentialization* operation of the input instance of the problem $\texttt{SHAP}(\texttt{d-DNF}, \overrightarrow{\texttt{MARKOV}})$. We give next details of the construction.

• **Sequentialization of Markovian distributions:** For an integer $N > 0$, the sequentialization of a Markovian distribution in $\overrightarrow{\texttt{MARKOV}}_N$ to one in $\texttt{MARKOV}$ must ensure that both distributions are equal in the support $\{0,1\}^N$. Indeed, since the SHAP score of boolean functions over N variables considers only the support $[N]$, the choice of the transition probability matrices for integers larger than $i > N$ can be set arbitrary, provided the resulting Markovian distribution remains polynomial-time computable. A possible sequentialization strategy of a distribution of P in $\overrightarrow{\texttt{MARKOV}}_N$ is given by a $\tilde{P} \in \texttt{MARKOV}$ (which depends on P) such that:

$$\tilde{P}_{init} = P_{init},\ \tilde{P}_i(X_{i+1}|X_i) = \begin{cases} P_i(X_{i+1}|X_i) & \text{if } i \in [N] \\ P_{unif}(X_{i+1}) & \text{elsewhere} \end{cases}$$

where $P_{unif}(X_{i+1})$ is the uniform distribution over $\{0,1\}$.

• **Sequentialization of d-DNFs.** For any integer $N > 0$, and any boolean vector $\overrightarrow{X}$ over $\{0,1\}^N$, $\texttt{SEQ}(\overrightarrow{X})$ refers to the sequence $X_1 \ldots X_N$ formed by the binary alphabet.

For a given d-DNF Φ over N variables. Its sequential version is represented by the unweighted language L_Φ over Σ^* such that:

$$L_\Phi \stackrel{\text{def}}{=} \{w \in \{0,1\}^{|\Phi|} : \overrightarrow{X} = \texttt{SEQ}^{-1}(w) \text{ satisfies } \Phi\}$$

Basically, L_Φ comprises the set of all satisfied assignments by the formula Φ arranged in a sequence. The following lemma is key to prove theorem 4.2. It establishes the existence of an algorithm that constructs in polynomial time a WA that computes the language I_Φ.

Lemma 4.4. *There exists an algorithm that takes as input a d-DNF Φ, runs in time polynomial in $|\Phi|$ and $|\Phi|_{\#}$, and outputs a WA that computes the language I_{L_Φ}.*

The proof of lemma 4.4 can be found in appendix D.

Next, we provide the proof of theorem 4.2.

Proof. (Theorem 4.2) For an input instance $< \Phi, \overrightarrow{x}, i, P >$ of the problem $\texttt{SHAP}(\texttt{d-DNF}, \texttt{MARKOV})$. One can observe that:

$$\overrightarrow{\text{SHAP}}(\Phi, x, i, P) = \text{SHAP}(I_{L_\Phi}, \texttt{SEQ}(\overrightarrow{x}), i, \tilde{P}) \quad (14)$$

Equation (4.2) suggests the following polynomial-time reduction strategy from $\texttt{SHAP}(\texttt{d-DNF}, \overrightarrow{\texttt{MARKOV}})$ to $\texttt{SHAP}(\texttt{d-DNF}, \texttt{MARKOV})$:

1. Construct a WA that computes the language I_{L_Φ}. By lemma 4.4, this can be performed in $O(\texttt{poly}(|\Phi|, |\Phi|_\#))$ time.

2. Apply the $\texttt{SEQ}(.)$ operation on $\overrightarrow{x}$.

3. Wrap the parameters of P in a machine implementing $\tilde{P}$. For an input integer $i > 0$, it tests whether $i > N$. If the answer is yes, it returns the uniform distribution. Otherwise, it returns P_i. The construction of this machine runs in $O(|\Phi|)$ time. In addition, the resulting Markovian distribution is polynomial-time computable.

□

5. Conclusion

In this article, we established the tractability of the SHAP score computational problem under the Markovian assumption for the family of weighted automata and the family of disjoint-DNFs which encompasses, up to a polynomial-time reduction, the family of decision trees. The proof is constructive and is readily amenable to a translation into a practical algorithm that extends TreeSHAP to handle the Markovian case.

In conclusion, we note that, by revisiting algorithms designed to generate WTs that compute the seq2seq languages $g^{(1)}_{w,P}$, $g^{(2)}_{w,i,P}$ (lemma 3.5), the algorithmic construction described in this article can be easily extended to adapt to higher-order markovian distributions, e.g.n-gram models (Fink, 2014), provided the order of the distribution is of reasonably small size.

In feature research, we aim at exploring the possibility to extend the tractability of SHAP explanations for other families of models under the Markovian assumption. An interesting family to be considered as a natural extension is the class of Deterministic Decomposable Circuits whose SHAP score computation under the feature independence assumption has been proven to be in FP (Arenas et al., 2023).

Acknowledgements

We would like to express our gratitude to the anonymous reviewers for their invaluable feedback and constructive comments, which greatly contributed to enhancing the quality and clarity of this paper. In particular, we are thankful for their contributions in directing our attention to works addressing the limitations of the SHAP score.

Impact Statement

Overall, our research contributes to the burgeoning field of explainable artificial intelligence (XAI) by focusing on the SHAP score, a widely employed method for interpreting ML models. We believe that the newly introduced theoretical tools used in our work to substantiate our findings, particularly those that establish connections between boolean circuits and finite state automata, hold the potential to inspire the development of novel algorithms within the realm of formal XAI, thus contributing to advance the field as a whole. Ultimately, our work contributes to the ongoing efforts to promote transparency, accountability, and trustworthiness in AI systems, paving the way for their responsible deployment across various domains.

References

Aizenstein, H. and Pitt, L. Exact learning of read-k disjoint dnf and not-so-disjoint dnf. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pp. 71–76, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X. doi: 10.1145/130385.130393.

Arenas, M., Barcelo, P., Bertossi, L., and Monet, M. On the complexity of shap-score-based explanations: Tractability via knowledge compilation and non-approximability results. *Journal of Machine Learning Research*, 24(63): 1–58, 2023.

Bassler, K. E., Gunaratne, G. H., and McCauley, J. L. Markov processes, hurst exponents, and nonlinear diffusion equations: With application to finance. *Physica A: Statistical Mechanics and its Applications*, 369(2): 343–353, 2006.

Bertossi, L., Li, J., Schleich, M., and Vagena, D. S. Z. Causality-based explanation of classification outcomes. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning (DEEM'20)*, New York, NY, USA, 2020. Association for Computing Machinery.

Culik, K. and Friš, I. Weighted finite transducers in image processing. *Discrete Applied Mathematics*, 58(3):223–237, 1995.

Culik, K. and Kari, J. Image compression using weighted finite automata. *Computers & Graphics*, 17(3):305–313, 1993. ISSN 0097-8493. doi: https://doi.org/10.1016/0097-8493(93)90079-O.

den Broeck, G. V., Lykov, A., Schleich, M., and Suciu, D. On the tractability of shap explanations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(7): 6505–6513, May 2021. doi: 10.1609/aaai.v35i7.16806.

Deng, X. and Papadimitriou, C. H. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994.

Denis, F. and Esposito, Y. On rational stochastic languages. *Fundam. Inf.*, 86(1,2):41–77, apr 2008. ISSN 0169-2968.

Droste, M. and Gastin, P. *Weighted Automata and Weighted Logics*, pp. 175–211. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-01492-5. doi: 10.1007/978-3-642-01492-5_5.

Eyraud, R. and Ayache, S. Distillation of weighted automata from recurrent neural networks using a spectral approach. *Machine Learning*, 2021. doi: 10.1007/s10994-021-05948-1.

Fink, G. A. *n-Gram Models*, pp. 107–127. Springer London, London, 2014. ISBN 978-1-4471-6308-4. doi: 10.1007/978-1-4471-6308-4_6.

Goutsias, J. and Jenkinson, G. Markovian dynamics on complex reaction networks. *Physics Reports*, 529(2): 199–264, 2013. ISSN 0370-1573. doi: https://doi.org/10.1016/j.physrep.2013.03.004. Markovian Dynamics on Complex Reaction Networks.

Huang, X. and Marques-Silva, J. The inadequacy of shapley values for explainability. *ArXiv*, abs/2302.08160, 2023. URL https://api.semanticscholar.org/CorpusID:256900674.

Janzing, D., Minorics, L., and Bloebaum, P. Feature relevance quantification in explainable ai: A causal problem. In Chiappa, S. and Calandra, R. (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 2907–2916. PMLR, 26–28 Aug 2020. URL https://proceedings.mlr.press/v108/janzing20a.html.

Kampen, N. G. V. *Stochastic Processes in Physics and Chemistry*. North-Holland, Amsterdam, The Netherlands, 2007.

Kiefer, S., Andrzej, S. M., Ouaknine, J., Wachter, B., and Worrell, J. On the complexity of equivalence and minimisation for q-weighted automata. *Log. Methods Comput. Sci.*, 9, 2013.

Knight, K. and May, J. Applications of weighted automata in natural language processing. In *Handbook of Weighted Automata*, pp. 571–596. Springer Berlin Heidelberg, 2009.

Koller, D. and Friedman, N. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. ISBN 0262013193.

Kumar, S., Deng, Y., and Byrne, W. A weighted finite state transducer translation template model for statistical machine translation. *Natural Language Engineering*, 12 (1):35–75, 2006.

Lacroce, C., Panangaden, P., and Rabusseau, G. Extracting weighted automata for approximate minimization in language modelling. In Chandlee, J., Eyraud, R., Heinz, J., Jardine, A., and van Zaanen, M. (eds.), *Proceedings of the Fifteenth International Conference on Grammatical Inference*, volume 153 of *Proceedings of Machine Learning Research*, pp. 92–112. PMLR, 23–27 Aug 2021.

Lehr, M. and Shafran, I. Learning a discriminative weighted finite-state transducer for speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 19 (5):1360–1367, 2010.

Lundberg, S. M. and Lee, S.-I. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, 2017.

Lundberg, S. M., Gabriel, E., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., and Lee, S.-I. From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, 2:56–67, 2020. doi: 10.1038/s42256-019-0138-9.

Mohri, M. *Weighted Finite-State Transducer Algorithms. An Overview*, pp. 551–563. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-39886-8. doi: 10.1007/978-3-540-39886-8_29.

Mohri, M., Pereira, F., and Riley, M. Speech recognition with weighted finite-state transducers. In *Springer Handbook of Speech Processing*, pp. 559–584. 2008.

Okudono, T., Waga, M., Sekiyama, T., and Hasuo, I. Weighted automata extraction from recurrent neural networks via regression on state spaces. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04): 5306–5314, Apr. 2020. doi: 10.1609/aaai.v34i04.5977.

Pereira, F. C. and Riley, M. D. Speech recognition by composition of weighted finite automata. *arXiv preprint cmp-lg/9603001*, 1996.

Rabusseau, G., Li, T., and Precup, D. Connecting weighted automata and recurrent neural networks through spectral learning. In Chaudhuri, K. and Sugiyama, M. (eds.), *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pp. 1630–1639. PMLR, 16–18 Apr 2019.

Schützenberger, M. On the definition of a family of automata. *Information and Control*, 4(2):245–270, 1961. ISSN 0019-9958. doi: https://doi.org/10.1016/S0019-9958(61)80020-X.

Shapley, L. S. A value for n-person games. *Contributions to the Theory of Games*, 2:307–317, 1953.

Sundararajan, M. and Najmi, A. The many shapley values for model explanation. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9269–9278. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/sundararajan20b.html.

Tacettin, M. and Ünlüyurt, T. An alternative proof that exact inference problem in bayesian belief networks is np-hard. In *Proceedings of the 20th International Conference on Computer and Information Sciences*, ISCIS'05, pp. 947–955, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3540294147. doi: 10.1007/11569596_96.

Weiss, G., Goldberg, Y., and Yahav, E. *Learning deterministic weighted automata with queries and counterexamples*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Yang, J. Fast treeshap: Accelerating shap value computation for trees. *ArXiv*, abs/2109.09847, 2021.

Yu, P., Bifet, A., Read, J., and Xu, C. Linear tree shap. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 25818–25828. Curran Associates, Inc., 2022.

A. Proof lemma 2.4

Lemma 2.4 establishes the existence of efficient procedures to compute the product, partition constant and projection operators over languages/seq2seq languages computed by means of WAs/WTs. In the same spirit of all results provided in this article, we shall provide a constructive proof of this lemma. In the sequel, we fix two finite alphabets $\Sigma,\ \Delta$.

The proof will rely on the notion of *Kronecker product* between matrices. A brief recall of this latter is given in the following.

• **The Kronecker product:** The Kronecker product between $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{k\times l}$, denoted $A \otimes B$, is a matrix in $\mathbb{R}^{(n\cdot k)\times(m\cdot l)}$ constructed as follows

$$A \otimes B = \begin{bmatrix} a_{1,1}\cdot B & a_{1,2}\cdot B & \dots & a_{1,m}\cdot B] \\ a_{2,1}\cdot B & a_{2,2}\cdot B & \dots & a_{2,m}\cdot B] \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,1}\cdot B & a_{n,2}\cdot B & \dots & a_{n,m}\cdot B \end{bmatrix}$$

where, for $(i,j) \in [n]\times[m]$ $a_{i,j}$ corresponds to element in the i-th row and j-th column of A.

A useful property of the Kronecker product in our context is the mixed-product property:

Proposition A.1. *(Proposition 2.1, (Kiefer et al., 2013))* *Given $A,\ B,\ C,\ D$ matrices with judicious dimensions, we have $(A\cdot B)\otimes(C\cdot D) = (A\otimes C)\cdot(B\otimes D)$*

Next, we prove the result of the three points mentioned in the lemma:

• **The product language of WAs:** An important property of WAs is their closure under the product operation. This fact is classical in the theory of rational languages and has been proven in Schützenberger's seminal paper (Schützenberger, 1961) where WAs have been first introduced.

For the sake of completeness, the following proposition provides the details of the construction of a WA that computes the product of two languages represented by their WAs:

Proposition A.2. *Let $A =< \alpha, \{A_\sigma\}_{\sigma\in\Sigma}, \beta >$ and $A' =< \alpha', \{A'_\sigma\}_{\sigma\in\Sigma}, \beta' >$ be two WAs over Σ^*. The WA $A\otimes A' =< \alpha\otimes\alpha', \{A_\sigma\otimes A'_\sigma\}_{\sigma\in\Sigma}, \beta\otimes\beta' >$ over Σ^* computes the language*

$$f_{A\otimes A'}(w) = f_A(w)\cdot f_{A'}(w)$$

for any $w \in \Sigma^*$.

Proof. Let $A =< \alpha, \{A_\sigma\}_{\sigma\in\Sigma}, \beta >$, and $A' =< \alpha', \{A'_\sigma\}_{\sigma\in\Sigma}, \beta' >$ be two WAs. Denote by $A\otimes A'$ the WA $< \alpha\otimes\alpha', \{A_\sigma\otimes A'_\sigma\}_{\sigma\in\Sigma}, \beta >$.

For an arbitrary string $w \in \Sigma^*$, we have:

$$\begin{aligned} f_A(w)\cdot f_{A'}(w) &= (\alpha^T\cdot\prod_{i=0}^{|w|} A_{w_i}\cdot\beta)\cdot(\alpha'^T\cdot\prod_{i=0}^{|w|} A'_{w_i}\cdot\beta') \\ &= (\alpha^T\otimes\alpha'^T)\cdot\prod_{i=1}^{|w|}(A_{w_i}\otimes A_{w'_i})\cdot(\beta\otimes\beta') \\ &= (\alpha\otimes\alpha')^T\cdot\prod_{i=1}^{|w|}(A_{w_i}\otimes A_{w'_i})\cdot(\beta\otimes\beta') \\ &= f_{A\otimes A'}(w) \end{aligned}$$

where the second equality results from the mixed-product property of the Kronecker product (proposition A.1). □

The construction of the product WA runs in $O(|\Sigma|\cdot \texttt{size}^2(A)\cdot \texttt{size}^2(A'))$.

• **The partition constant operator of WAs:** The following proposition provides an implicit polynomial-time procedure that computes the quantity $|f_A|_n$ for a WA A.

Proposition A.3. *Let $A =< \alpha, \{A_\sigma\}_{\sigma\in\Sigma}, \beta >$ be a WA over Σ^* and an integer $n > 0$. We have:*

$$|f_A|_n = \alpha^T \cdot (\sum_{\sigma\in\Sigma} A_\sigma)^n \cdot \beta$$

Proof. Let $A =< \alpha, \{A_\sigma\}_{\sigma\in\Sigma}, \beta >$ be a WA over Σ^* and an integer $n > 0$. We first prove by induction that for any $n > 0$, we have

$$\sum_{w\in\Sigma^n} A_w = (\sum_{\sigma\in\Sigma} A_\sigma)^n \tag{15}$$

The case $n = 1$ is trivial.

Assume the expression (15) is true for an integer $n > 0$. Let's prove it is also the case for $n + 1$. We have:

$$\sum_{w\in\Sigma^{n+1}} A_w = \sum_{w\in\Sigma^n}\sum_{\sigma\in\Sigma} A_{w\sigma} = \sum_{w\in\Sigma^n} A_w \cdot \sum_{\sigma\in\Sigma} A_\sigma = (\sum_{\sigma\in\Sigma} A_\sigma)^{n+1}$$

which proves the equality (15).

Let $A =< \alpha, \{A_\sigma\}_{\sigma\in\Sigma}, \beta >$. For an integer $n > 0$, we have:

$$|f_A|_n = \alpha^T \cdot \sum_{w\in\Sigma^n} A_w \cdot \beta = \alpha^T \cdot (\sum_{\sigma\in\Sigma} A_\sigma)^n \cdot \beta$$

where the second result is obtained from (15). □

The complexity of implementing this operation is given as: $O(\texttt{size}(A)^2(|\Sigma| + \texttt{size}(A))$.

• **The projection operator:** The following proposition provides a proof of the third point of lemma 2.4:

Proposition A.4. *Let Σ, Δ be two finite alphabets. Let $A =< \alpha, \{A_\sigma\}_{\sigma\in\Sigma}, \beta >$ be a WA over Σ^* ,and $T =< \alpha', \{\bar{A}_\sigma^{\sigma'}\}_{\sigma\in\Sigma,\ \sigma'\in\Sigma'}, \beta' >$ a WT over $\Sigma^* \times \Delta^*$. The WA $\Pi(A, T) =< \alpha \otimes \alpha', \{\sum_{\sigma\in\Sigma} A_\sigma \otimes \bar{A}_\sigma^{\sigma'}\}_{\sigma'\in\Delta}, \beta \otimes \beta' >$ over Σ^* computes the language $\Pi(f_A \otimes f_T)$.*

Proof. Let A be a WA over Σ^*, and T be a WT over $\Sigma^* \times \Delta^*$.

Define the WA $\Pi(A, T) =< \alpha \otimes \alpha', \{\sum_{\sigma\in\Sigma} A_\sigma \otimes \bar{A}_\sigma^{\sigma'}\}_{\sigma'\in\Delta}, \beta \otimes \beta' >$ constructed from A and T.

For an arbitrary $u \in \Delta^*$, we have

$$\begin{aligned}
\sum_{w\in\Sigma^{|u|}} f_A(w) \cdot f_T(w, u) &= \sum_{w\in\Sigma^{|u|}} (\alpha^T \cdot \prod_{i=1}^{|w|} A_{w_i} \cdot \beta) \\
&\quad \cdot (\alpha'^T \cdot \prod_{i=1}^{|u|} \bar{A}_{w_i}^{u_i} \cdot \beta') \\
&= \sum_{w\in\Sigma^{|u|}} (\alpha \otimes \alpha')^T \cdot (\prod_{i=1}^{|u|} A_{w_i} \otimes \bar{A}_{w_i}^{u_i}) \cdot (\beta \otimes \beta') \\
&= (\alpha \otimes \alpha')^T \cdot (\prod_{i=1}^{|u|} \sum_{\sigma\in\Sigma} A_\sigma \otimes \bar{A}_\sigma^{u_i}) \cdot (\beta \otimes \beta') \\
&= f_{\Pi(A,T)}(u)
\end{aligned}$$

□

The complexity of the construction implicitly outlined in proposition A.4 is $O(\texttt{size}(A)^2 \times \texttt{size}(T)^2 \times |\Sigma|)$.

B. Proof lemma 3.4

We'll show the expression (8). The expression (9) can be obtained by mimicking the proof herein.
Let A be a WA, a sequence $w \in \Sigma^*$, an integer $k \in [|w|-1]$, and P be an arbitrary distribution over Σ^∞. Let $f_{w,k}$ (resp. $g^{(1)}_{w,P}$) a language (resp. seq2seq language) whose properties are given in the statement of the lemma. We have:

$$\begin{aligned}
\text{SHAP}_1(A, w, i, k, P) &= \mathbb{E}_{p\sim\mathcal{P}^w_k}\mathbb{E}_{w'\sim P^{(|w|)}}[f_A(w')|w' \in L_p] \\
&= \sum_{p\in\Sigma^{|w|}_\#} \mathcal{P}^w_k(p) \sum_{w'\in\Sigma^{|w|}} f_A(w') \cdot P(w'|w' \in L_p) \\
&= \sum_{p\in\Sigma^{|w|}_\#} f_{w,k}(p) \cdot \sum_{w'\in\Sigma^{|w|}} f_A(w') \cdot g^{(1)}_{w,P}(w', p) \\
&= \sum_{p\in\Sigma^{|w|}_\#} f_{w,k}(p) \cdot \Pi(f_A, g^{(1)}_{w,P})(p) \\
&= \sum_{p\in\Sigma^{|w|}_\#} f_{w,k}(p) \cdot \Pi(f_A, g^{(1)}_{w,P})(p) \\
&= |f_{w,k} \otimes \Pi(f_A, g^{(1)}_{w,P})|_{|w|}
\end{aligned}$$

C. Proof lemma 3.5

The core statement of lemma 3.5 encompasses three results stating the existence of three efficient algorithmic procedures, namely $\mathcal{A}_1,\ \mathcal{A}_2$ and $\mathcal{A}_3$, that construct a collection of WAs/WTs whose characteristics are given in the lemma statement.

This appendix will be split into two segments. The first segment furnishes the algorithmic construction of $\mathcal{A}_1$. Due to the close similarities of algorithms $\mathcal{A}_2,\ \mathcal{A}_3$, they shall be treated simultaneously in the second segment.

Before outlining these constructions, we furnish a brief recall of some sub-families of WAs and WTs serving as a technical background on top of which the proof will be built. In particular, three sub-families will be introduced: Determinstic Finite Automata, deterministic WAs, and Deterministic Finite Transducers.

In the sequel, we fix an alphabet $\Sigma,\ \Delta$.

• **Deterministic finite Automata.** The class of deterministic finite automata (DFAs) is a popular sub-family of WAs adapted to model unweighted languages. A DFA is formally represented by a tuple $< Q, q_{init}, \delta, F >$, where:

- Q is a finite set of states,
- $q_{init} \in Q$ is called the initial state,
- $\delta : Q \times \Sigma \to Q$ is a partial function [4] called the transition function,
- $F \subseteq Q$ is called the the set of final states,

For a DFA $A =< Q, q_{init}, \delta, F >$, a valid path over A labeled by a sequence $w \in \Sigma^*$ is a sequence of state-symbol pairs taking the form: $q_0 w_1 q_1 \dots w_{|w|} q_{|w|}$, such that for any $i \in \{0, \dots |w|-1\} : \delta(q_i, w_{i+1}) = q_{i+1}$. A valid path labeled by w is said to be accepting if $q_0 = q_{init}$ and $q_{|w|} \in F$.

An important property of DFAs lies in that the cardinality of the set of its valid paths labeled by an arbitrary sequence $w \in \Sigma^*$ is at most equal to 1. The unweighted language accepted by a DFA corresponds to the set of sequences that label a valid accepting path over the DFA.

• **Deterministic Finite Transducers.** Deterministic Finite Transducers (DFTs) represent the analogous counterpart of DFAs adapted to seq2seq languages, and constitutes a sub-family of WTs that compute unweighted seq2seq languages. A DFT over $\Sigma \times \Delta$ is formally represented by a tuple $< Q, q_{init}, \delta, F >$

[4] A partial function f from a set X to Y is a function whose input domain is a subset of X (i.e. it doesn't necessarily assign an output to every element of x)

- Q is a finite set of states.
- q_{init} is the initial state.
- $\delta : Q \times \Sigma \to \Delta \times Q$ is a partial function called the transition function.
- F is called the set of final states.

The formal description of a DFT resembles to that of DFAs, and operates in a closely similar manner.

For a DFT $T =< Q, q_{init}, \delta, F >$, a valid path over T labeled by a pair of sequences $(u, v) \in \Sigma^* \times \Delta^*$ such that $|u| = |v| = n$ is a sequence of elements in $Q \times \Sigma \times \Delta$ taking the form: $q_0 u_1 v_1 q_1 \dots u_n v_n q_n$ where for any $i \in \{0, \dots n-1\} : \delta(q_i, u_{i+1}) = (v_{i+1}, q_{i+1})$. A valid path over $(u, v) \in \Sigma^* \times \Delta^*$ is said to be accepting if $q_0 = q_{init}$ and $q_n \in F$.

DFTs enjoy a similar property than DFAs in that for any pair of sequences over $\Sigma^* \times \Delta^*$ with the same length, there exists at most one valid path labeled by this pair. The unweighted seq2seq language accepted by a DFT is equal to the set of sequence pairs over $\Sigma^* \times \Delta^*$ that label a valid accepting path.

• **Deterministic Weighted Automata.** DWAs is the weighted variant of DFAs. It aligns with the structure of DFA while augmenting its transitions with real-valued weights. Formally, a DWA is defined as follows:

- Q is a finite set of states,
- $q_{init} \in Q$ is called the initial state,
- $W : Q \times \Sigma \to Q \times \mathbb{R}$ is a partial function [5] called the weight function,
- $F \subseteq Q$ is called the the set of final states,

Similar to DFAs, any sequence $w \in \Sigma^*$ labels at most a valid path, where the notion of a valid path is equivalent to that of DFAs. However, unlike DFAs, valid paths are assigned real-valued weighted instead of the boolean notion of acceptability. The weight assigned to a path starting from the initial state $q_{init} w_1 \dots q_{n-1} w_{n-1} q_n$ is equal to:

$$\cdot \prod_{i=1}^{n-1} W(q_i, w_i)[2] \cdot I_F(q_n)$$

where $W(q, \sigma)[2]$ refers to the weight associated to the transition $\delta(q, \sigma)$.

This weight coincides with the value assigned to the sequence $w_1 \dots w_n$ by the seq2seq language computed by the WT. Sequences that label no valid path are assigned the weight 0 by default.

After presenting this brief technical background, we are now ready to prove the core statement of the lemma:

C.1. Construction of $\mathcal{A}_1$.

Recall that $\mathcal{A}_1$ refers to an algorithm that takes as input a string $w \in \Sigma^*$, an integer $k \in [|w|]^2$, runs in $O(\texttt{poly}(|w|))$, and outputs a WA over $\Sigma^*_\#$ that computes the language $\mathcal{P}^w_k$. The probability distribution $\mathcal{P}^w_k$ refers to the uniform distribution over the set of patterns:

$$\mathcal{L}^w_k \stackrel{\text{def}}{=} \{p \in \Sigma^{|w|}_\# : \ |p|_\# = k \wedge w \in L_p\}$$

The algorithmic construction of $\mathcal{A}_1$ aligns with two sequential steps:

1. Create a DFA over $\Sigma^*_\#$ that accepts the language $\mathcal{L}^w_k$,

[5] A partial function f from a set X to Y is a function whose input domain is a subset of X (i.e. it doesn't necessarily assign an output to all elements of x)

2. Normalize the resulting DFA by the quantity $\frac{1}{|\mathcal{L}_k^w|}$ to obtain the output WA. Note that $|\mathcal{L}_k^w|$ is equal to $\frac{|w|!}{(k)!\cdot(|w|-k)!}$ and can be computed in $O(\texttt{poly}(|w|))$ time.

The second step of the algorithmic construction, i.e. the normalization step, is straightforward. Indeed, given a WA $A =< \alpha, \{A_\sigma\}_{\sigma\in\Sigma}, \beta >$ and a normalizing constant $C \in \mathbb{R}$, the WA $A' =< C \cdot \alpha, \{A_\sigma\}_{\sigma\in\Sigma}, \beta >$ computes the (normalized) language $f_{A'} = C \cdot f_A$. In addition, it's easy to observe that this operation can be performed in polynomial time with respect to the size of A. Our claim, that we shall prove next in this subsection, is that the size of the DFA A is $O(\texttt{poly}(|w|))$. Assuming this claim holds, the normalization operation runs in $(\texttt{poly}(|w|)$.

The rest of this subsection will focus on the first step of the algorithmic construction:

• *Creation of a DFA that accepts the language $\mathcal{L}_k^w$:*
Fix an input instance $w \in \Sigma^*$, $k \in [|w|]$. A key observation for the DFA construction consists at noting that, during a forward processing run over an input pattern to check its membership in $\mathcal{L}_k^w$, a sufficient information to keep of the run's history is summarized in the following:

- *The position of the next symbol:* This information is useful to ensure that the input pattern satisfies the constraint $w \in L_p$ imposed by definition of $\mathcal{L}_k^w$. Additionally, this information will enable rejecting the patterns whose length is greater than $|w|$. In our case, this information lies in the interval $\{0, 1, \ldots, |w|\}$,
- *The number of occurrences of the symbol # in the processed prefix of the input pattern:* This information enables to ensure that only patterns that satisfies the constraint $|p|_\# = k$ will be accepted. In our case, this information lies in the range $\{0, 1, \ldots, k\}$.

In light of this discussion, the construction of the DFA that accepts the language $\mathcal{L}_k^w$:

- **The state space:** $Q = \{0, 1, \ldots, |w|] \times \{0, 1, \ldots, k\}$
- **The initial state:** $q_{init} = (0, 0)$. The first element of the pair signifies that the forward run is at position 0 (i.e. no symbol in the input pattern has been processed so far). The second element signifies that 0 occurrences of the symbol # has been encountered in the processed input pattern so far.
- **The transition function:** For a state $(l, l') \in \{0, \ldots, |w| - 1\} \times \{0, \ldots, k - 1\}$

  Case 1 ($p_{l+1} = \#$). We increment both the number of occurences of # in the input pattern and the position of the sequence by 1 which entails a transition to $(l + 1, l' + 1)$:

  $$\delta((l, l'), \#) = (l + 1, l' + 1)$$

  Case 2 ($p_{l+1} = w_{l+1}$). we increment the position of the input pattern to $l + 1$ without incrementing the number of occurrences of #.

  $$\delta((l, l'), w_{l+1})) = (l + 1, l')$$

  No other transitions are added to the transition map for all the other cases.
- **The final set of states:** $F = \{(|w|, k)\}$.

One can check that the complexity of this algorithmic construction runs in $O(|w|^2)$ time.

C.2. Constructions of $\mathcal{A}_2$ and $\mathcal{A}_3$.

Due to the close similarities in the construction of algorithms $\mathcal{A}_2$ and $\mathcal{A}_3$, we dedicate this segment to treat both algorithms simultaneously. The presence of the $\texttt{swap}(.)$ operation in $\mathcal{A}_3$ brings an additional difficulty to this latter, when compared to $\mathcal{A}_2$. Consequently, we choose to treat $\mathcal{A}_3$ as a main case. The subtle differences between $\mathcal{A}_2$ and $\mathcal{A}_3$ will take the form of notes where these differences will be highlighted.

In lemma 3.4, $\mathcal{A}_3$ designates an algorithm that takes as input a string $w \in \Sigma^*$, an integer $i \in [|w|]$, a probability distribution $P \in \texttt{MARKOV}$, and outputs a WT $T_{w,P}$ over $\Sigma^* \times \Sigma^*_{\#}$ that computes a seq2seq language that satisfies the following constraint:

$$\forall (w', p) \in \Sigma^{|w|} \times \Sigma_{\#}^{|w|} : f(w', p) = P(w' | w' \in L_{\texttt{swap}(p,i)}) \tag{16}$$

Instead of this formulation, we'll exploit an equivalent re-expression of the constraint in the algorithmic design obtained using Bayes' rule:

$$\forall (p, w') \in \Sigma^{|w|} \times \Sigma_{\#}^{|w|} : f(w', p) = \frac{P(w') \cdot I_{L_{\texttt{swap}(p,i)}}(w')}{P(L_{\texttt{swap}(p,i)})} \tag{17}$$

The algorithms $\mathcal{A}_2$, and $\mathcal{A}_3$ will be designed following the same paradigm employed to construct the main algorithm for solving $\texttt{SHAP}(\texttt{WA}, \texttt{MARKOV})$. Specifically, it will involve the construction of WAs/WTs that compute languages dependent on the input instance of the problem. Then, the application of efficiently computable operators over these constructed WAs/WTs will yield a WT that satisfies the constraint (17).

Besides operators introduced in section 2, namely the product operator, the partition constant operator and the projection operator, we shall introduce two additional operators over seq2seq languages which will be useful in this context. An emphasis will be put on the computational efficiency of implementing these operators for the case of seq2seq languages represented by WTs.

Fix two finite alphabets Σ and Δ.

• **The inverse operator:** The inverse operator takes as input a language a seq2seq language $\Sigma^* \times \Delta^*$ f, and returns the seq2seq language denoted $\texttt{inv}(f)$ such that:

$$\texttt{inv}(f)(u, s) \stackrel{\text{def}}{=} f(s, u)$$

for $(u, s) \in \Sigma^* \times \Delta^*$ such that $|u| = |s|$.

This operator settles for performing a swap operation of the arguments given to compute the seq2seq language for a given pair of sequences.

When a seq2seq language over $\Sigma^* \times \Delta^*$ is computed by a WT $T =< \alpha, \{A_\sigma^{\sigma'}\}_{(\sigma,\sigma') \in \Sigma \times \Delta}, \beta >$, the WT that computes the seq2seq language $\texttt{inv}(f_T)$ can be trivially obtained as $< \alpha, \{A_{\sigma'}^{\sigma}\}_{(\sigma',\sigma) \in \Delta \times \Sigma}, \beta >$.

• **The multiplicative operator:** This operator, which we'll refer to as *the multiplicative operator*, takes as input a language f over Σ^* and a seq2seq language g over $\Sigma^* \times \Delta^*$, and outputs a seq2seq language over $\Sigma^* \times \Delta^*$, denoted $f \times g$ such that

$$(f \times g)(u, s) = f(u) \cdot g(u, s) \tag{18}$$

for any $(u, s) \in \Sigma^* \times \Delta^*$ such that $|u| = |s|$.

When the language f and the seq2seq language g given as arguments to this operator are represented by a WA A and a WT T, respectively, then $f \times g$ can be computed by a WT. Moreover, the construction of this WT can be performed in time polynomial in the size of A and T. The followin proposition provides a proof of this fact:

Lemma C.1. *Let $A =< \alpha, \{A_\sigma\}_{\sigma \in \Sigma}, \beta >$ be a WA over Σ^*, $T =< \alpha', \{B_\sigma\}_{\sigma \in \Sigma}^{\sigma' \in \Delta}, \beta' >$ a WT over $\Sigma^* \times \Delta^*$.*

The WT $A \times T =< \alpha \otimes \alpha', \{A_\sigma \otimes B_\sigma^{\sigma'}\}_{\sigma \in \Sigma}^{\sigma' \in \Delta}, \beta \otimes \beta' >$ over $\Sigma^* \times \Delta^*$ computes the seq2seq language $f_{A \times T}$.

Proof. Let $A =< \alpha, \{A_\sigma\}_{\sigma \in \Sigma}, \beta >$ be a WA over Σ^*, $B =< \alpha', \{B_\sigma\}_{\sigma \in \Sigma}^{\sigma' \in \Delta}, \beta' >$ a WT over $\Sigma_* \times \Delta^*$. Let $A \times B =< \alpha \otimes \alpha', \{A_\sigma \otimes B_\sigma^{\sigma'}\}_{\sigma \in \Sigma}^{\sigma' \in \Delta}, \beta \otimes \beta' >$ be the constructed WT from A and B.

Fix a pair $(u, s) \in \Sigma^* \times \Delta^*$ such that $|u| = |s|$. We have

$$\begin{aligned} f_A(u) \cdot f_T(u, s) &= (\alpha^T \cdot \prod_{i=1}^{|u|} A_{u_i} \cdot \beta) \cdot (\alpha'^T \cdot \prod_{i=1}^{|w|} B_{w_i}^{s_i} \cdot \beta') \\ &= (\alpha \otimes \alpha')^T \cdot \prod_{i=1}^{|u|} (A_{u_i} \otimes B_{w_i}^{s_i}) \cdot (\beta \otimes \beta') \\ &= f_{A \times B}(u, s) \end{aligned}$$

where the second equality is an application of the mixed product property of the Kronecker product (proposition A.1). □

After introducing the inverse and the multiplicative operator, we are now ready to provide the overall structure of algorithms $\mathcal{A}_2$ and $\mathcal{A}_3$.

Fix an input instance $w \in \Sigma^*$, a pair of integers $i \in [|w|]$, and $P \in \texttt{MARKOV}$.

The algorithm $\mathcal{A}_3$ will follow three steps:

- *Step 1:* Construct a DWA, denoted $A_{w,P}$, over Σ^* that computes the language

$$f_{A_{w,P}}(w') = \begin{cases} P(w') & \text{if } w' \in \Sigma^{|w|} \\ 0 & \text{elsewhere} \end{cases} \tag{19}$$

- *Step 2:* Construct a DFT, denoted T_i, over $\Sigma^* \times \Sigma_\#^*$ that computes the (unweighted) seq2seq language:

$$f_{T_{w,i}}(w', p) = I_{L_{\texttt{swap}(p,i)}}(w') \tag{20}$$

  for any pair $(w', p) \in \Sigma^* \times \Sigma_\#^*$.

- *Step 3:* Construct a DWT over $\Sigma_\#^*$, denoted $A_{w,i,P}$ that computes a language over $\Sigma_\#^*$ such that:

$$f_{A_{w,i,P}}(p) = \frac{1}{P(L_{\texttt{swap}(p,i)})} \tag{21}$$

  for any $p \in \Sigma_\#^{|w|}$.

Assume we have the WAs $A_{w,P}$, $A_{w,i,P}$ and the WT $T_{w,i}$ that compute languages/seq2seq languages described in steps 1, 2 and 3, respectively. In light of the equation (17), the seq2seq language computed by the WT

$$\texttt{inv}(A_{w,i,P} \times \texttt{inv}(A_{w,P} \times T_i))$$

satisfies the constraint of the seq2seq language $g_{w,i,P}^{(1)}$. This resulting WT represents the output of $\mathcal{A}_3$.

• **Note.** *At this stage, a slight difference between $\mathcal{A}_2$ and $\mathcal{A}_3$ lies in steps 2 and 3. For the case of $\mathcal{A}_2$, the pattern $\texttt{swap}(p, i)$ should be replaced by p in equations (20) and (21), in which case a different DFT and DWT have to be designed to compute these set of languages/seq2seq languages. Later in this segment, we shall highlight their construction.*

It's left to show how to construct these three machines in polynomial time with respect to the size of the input instance. The constructions of $A_{w,P}$ and $T_{w,i}$ are relatively easy. The construction $A_{w,i,P}$ is more challenging.

The remainder of this section will be split in three segments, each of which is dedicated to provide the implementation details of one of the steps of the algorithmic structure outlined above.

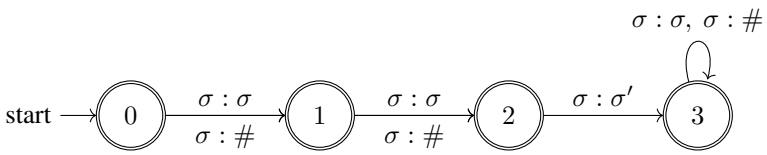

Figure 1. A DFT T_i that computes the seq2seq language $g(w', p) = I_{L_{\texttt{swap}(p,i)}}(w')$ for $i = 3$. σ (resp. σ') refers to any symbol in Σ (resp. $\Sigma_{\#}$.

C.2.1. Step 1: Construction of $A_{w,P}$.

$A_{w,P}$ refers to the WA that computes the language expressed in 19.

Given a string $w \in \Sigma^*$ and $P \in \texttt{MARKOV}$. A Markovian distribution over the finite support $\Sigma^{|w|}$ can be easily simulated by a DWA. The construction consists at maintaining in the state memory of the DWA the position reached so far in the sequence and the last generated symbol. These two pieces of information are sufficient to simulate a Markovian distribution.

For the sake of the construction, we add a new symbol, denoted $< BOS >$, that refers to the beginning of a sequence.

The outline of the construction is given as follows:

- *The state space:* $Q = \{0, 1, .., |w|\} \times (\Sigma \cup < \text{BOS} >)$,
- *The initial state:* $q_{init} = (0, < \text{BOS} >)$
- *The weight function:* Let $q = (i, \sigma)$ be a state in Q. We denote by σ' an arbitrary symbol in Σ. We distinguish between two cases:
  - *Case 1* $((i, \sigma) = (0, < BOS >))$*:*
$$W((0, < BOS >), \sigma') = ((1, \sigma'), P_{init}(\sigma'))$$
  - *Case 2* $(i < |w|)$*:*
$$W((i, \sigma), \sigma') = ((i + 1, \sigma') = P_i(\sigma'|\sigma))$$
- *The final weight vector:* $F = \{(|w|, \sigma) :\ \sigma \in \Sigma\}$

A valid path labeled by a sequence $w' \in \Sigma^{|w|}$ over the constructed DWA is given as:

$$(0, < BOS >)w'_1(1, w'_1) \dots (|w| - 1, w'_{|w|-1})w'_{|w|}(|w|, w'_{|w|})$$

The weight of this path is equal to $P_{init}(w'_1) \cdot \prod_{i=1}^{|w|-1} P_i(w'_{i+1}|w'_i) \cdot I_F\left((|w|, w'_{|w|})\right) = P(w')$.

Provided P is polynomial-time computable, this construction runs in $O(\texttt{poly}(|w|, |\Sigma|))$ time.

C.2.2. Step 2: Construction of T_i.

Given an integer $i > 0$, the goal is to construct a DFT T_i over $\Sigma^* \times \Sigma^*_{\#}$ that computes the seq2seq language whose expression is given in (20).

The construction is relatively easy. The state of the DFT will keep in its memory the current position of the pair of sequences being parsed up to position i. At a position $j < i$, the DFT will enable a transition from a state j to a state $j + 1$ if and only if the current pair of symbols to parse (w'_{j+1}, p_{j+1}) satisfies the constraint $(w'_{j+1} = p_{j+1}) \vee p_{j+1} = \#)$,. For the particular case, $j = i - 1$, where the swap operation needs to be taken into account, a transition is allowed to $j + 1$ regardless of the pair of symbols (p_i, w'_i) fed to the DFT.

The formal description of a DFT T_i is given in the following. An illustrative example of this construction is given in figure 1.

- *The state space:* $Q = \{0, 1, \dots, i\}$

- *The initial state:* $q_{init} = 0$,
- *The transition function:* Let j be a state in Q. We distinguish between three cases:
  1. *Case 1 ($j < i-1$):*
  $$\delta(j, (\sigma, \sigma')) = j + 1$$
  for $(\sigma, \sigma') \in \Sigma \times \Sigma_{\#}$ such that $(\sigma = \sigma' \vee \sigma' = \#)$
  2. *Case 2 ($j = i-1$):*
  $$\delta(j, (\sigma, \sigma')) = j + 1$$
  for any pair of symbols $(\sigma, \sigma') \in \Sigma \times \Sigma_{\#}$
  3. *Case 3 ($j = i$):*
  $$w(i, (\sigma, \sigma')) = i$$
  for $(\sigma, \sigma') \in \Sigma \times \Sigma_{\#}$ such that $(\sigma = \sigma' \vee \sigma' = \#)$
- *The set of final states:* Q.

• **Note.** *For the case of the algorithm $\mathcal{A}_2$, a DFT that computes the seq2seq language $f(p, w') = I_{L_p}(w')$ is a trivial single-state DFT that settles for testing at each step during the forward run whether the pair of input symbols $(\sigma, \sigma') \in \Sigma \times \Sigma_{\#}$ satisfies the constraint: $\sigma' = \sigma \vee \sigma' = \#$.*

C.2.3. Step 3: Construction of $A_{w,i,P}$.

In the remainder of this segment, we fix a string $w \in \Sigma^*$, and an integer $i \in [|w|]$, and $P \in \texttt{MARKOV}$.

Recall that the DWA $A_{w,i,P}$ over $\Sigma_{\#}^*$ is required to compute a language that satisfies the constraint (21). The construction of the DWA $A_{w,i,P}$ is more challenging than the construction pf $A_{w,P}$ and T_i detailed in previous segments. The difficulty lies in the fact that, unlike the product operation, the set of WAs is not closed under the division operation.

By means of Bayes' rule, the constraint (21) is explicitly given as

$$\forall p \in \Sigma_{\#}^{|w|} : \quad f_{A_{w,i,P}}(p) = \frac{1}{P_{init}(w'_1 \in L_{\texttt{swap}(p,i)_1})} \cdot \frac{1}{\prod_{j=1}^{|w|-1} P(w'_{j+1} \in L_{\texttt{swap}(p,i)_{j+1}} | w'_{1:j} \in L_{\texttt{swap}(p,i)_{1:j}})} \tag{22}$$

When trying to construct a DWA that satisfies the formula (22), a difficulty arises by noting that the product terms forming the right-side of the equation requires maintaining the full history of the input pattern. A construction of a DWA that naïvely simulates the equation (22) would have a state space whose size is $O(|\Sigma|^{|w|})$.

To circumvent this issue, an intermediary question to raise is concerned with the size of the minimal sufficient information to hold about a running pattern $p_{1,j}$ to compute the quantity $P(w'_{j+1} \in L_{p_{j+1}} | w'_{1:j} \in L_{p_{1:j}})$. Under the assumption that $P \in \texttt{MARKOV}$, one can observe that the minimal sufficient information to retain about the past of a pattern during a forward run is:

1. The current position in the processed sequence.
2. The last position where a symbol $\sigma \in \Sigma$ has been encountered during the processing run.
3. The symbol that holds the position described in the previous point.

To gain some intuition on the points discussed above, we provide an illustrative example:

• **Example:** Let $\Sigma = \{a, b\}$ be an alphabet, and $P \in$ MARKOV. Let $p = a\#a\#b$ be a pattern (the support is equal to 5). Let's fix as a goal the computation of the quantity $P(w \in L_{a\#a\#b})$. Using Bayes' rule, we have

$$P(w \in L_{a\#a\#b}) = P(w_5 = b | w_1 = a \wedge w_3 = a) \cdot P(w_3 = a | w_1 = a) \cdot P(w_1 = a)$$

Since $P \in$ MARKOV, w_5 is independent of w_1 given w_3. Thus,

$$P(w \in L_{a\#a\#b}) = P(w_5 = b|w_3 = a) \cdot P(w_3 = a|w_1 = a) \cdot P(w_1 = a)$$

Note that each product term in the right side of the equation depends only the current position, the last position where a symbol different than # has been encountered and the symbol found in this position.

The points 2 and 3 are formalized by introducing the following two functions:

- The $\texttt{pos}(.)$ function:

$$\begin{aligned} \texttt{pos}\colon \Sigma_{\#}^{*} &\longrightarrow \mathbb{N} \\ p &\longrightarrow \max_{i \in \{0,1,\ldots,|p|\}} \{i \in \mathbb{N} : \ p_i \neq \#\} \end{aligned} \tag{23}$$

- The $\texttt{sym}(.)$ function:

$$\begin{aligned} \texttt{sym}\colon \Sigma_{\#}^{*} &\longrightarrow \Sigma \cup < BOS > \\ p &\longrightarrow \begin{cases} < BOS > & \text{if } \texttt{pos}(p) = 0 \\ p_{\texttt{pos}(p)} & \text{elsewhere} \end{cases} \end{aligned} \tag{24}$$

• **Example:** For the alphabet $\Sigma = \{a, b\}$ and the pattern $p = a\#a\#$. The last position held by a symbol in Σ in p is the position 3. It is held by the symbol a. Consequently, for this example, we have $\texttt{pos}(p) = 3$, and $\texttt{sym}(p) =' a'$.
For patterns that contain only the symbol $'\#'$, e.g. $p' = \#\#\#\#$, we have $\texttt{pos}(p') = 0$ and $\texttt{sym}(p') =$ BOS.

Next, we shall see how to reformulate the equation (22) using the functions $\texttt{pos}(.)$, and $\texttt{sym}(.)$.

For a given pattern p in $\Sigma_{\#}^{*}$, define the language $\tilde{L}_p$ over Σ^* described as follows:

$$\tilde{L_p} \stackrel{\text{def}}{=} \{w \in \Sigma^{|p|} : \ w_{\texttt{pos}(p)} = \texttt{sym}(p)\} \tag{25}$$

By convention, if $\texttt{pos}(p) = 0$, $\tilde{L}_p$ is equal to $\#^{|p|}$.

Given that $P \in \texttt{MARKOV}$, we have

$$P(w'_{j+1} \in L_{p_{j+1}}|w'_{i:j} \in L_{p_{1:j}}) = P(w'_{j+1} \in L_{p_{j+1}}|w'_{1:j} \in \tilde{L}_{p_{1:j}}) \tag{26}$$

At this stage, a key observation is that the quantity present in the right-hand side of the equation (26) depends only on P, p_{j+1}, $\texttt{pos}(p_{1:j})$, $\texttt{sym}(p_{1:j})$, and j. Indeed, by definition of the language $\tilde{L}_p$ (equation (25)), the language $\tilde{L}_{p_{1:j}}$ depends only on these last three parameters.

To make this dependency appearing explicitly, we shall introduce a definition of a new function $\mathbf{G}$ given as follows:

$$\mathbf{G}\left(p_{j+1}, \texttt{pos}(p_{i,j}), \texttt{sym}(p_{1:j}), j, P\right) \stackrel{\text{def}}{=} P(w'_{j+1} \in L_{p_{j+1}}|w' \in \tilde{L}_{p_{1:j}}) \tag{27}$$

Using the equality (26), we can rewrite the constraint (22) with this newly introduced notation as:

$$\forall p \in \Sigma_{\#}^{|w|} : \ f_{A_{w,i,P}}(p) = \frac{1}{P_{init}(w_1 \in L_{\texttt{swap}(p,i)_1})} \cdot \frac{1}{\prod_{j=1}^{|w|-1} \mathbf{G}(\texttt{swap}(p,i)_{j+1}, \texttt{pos}(\texttt{swap}(p,i)_{1:j}), \texttt{sym}(\texttt{swap}(p,i)_{1:j}), j, P)} \tag{28}$$

Toward the stated objective of constructing a deterministic WA over $\Sigma_{\#}^{*}$ that computes a language satisfying the constraint (22), the expression (28) offers a better reformulation of this equation by considering two aspects:

1. The product terms forming the right-hand side of expression (28) offers a compressed representation of the history of the processed pattern required to perform next processing operations, by maintaining only the current position in the sequence, the last symbol different that # encountered during the forward run and its position in the sequence.

2. The functions $\texttt{pos}(.)$ and $\texttt{sym}(.)$ can be easily simulated by a sequential machine that processes sequences from left-to-right, such as WAs. Specifically, for any pattern $p \in \Sigma_{\#}^{*}$ and a symbol $\sigma \in \Sigma$, we have

$$\texttt{pos}(p\sigma) = \begin{cases} \texttt{pos}(p) & \text{if } \sigma = \# \\ \texttt{pos}(p) + 1 & \text{elsewhere} \end{cases}$$

$$\texttt{sym}(p\sigma) = \begin{cases} \texttt{sym}(p) & \text{if } \sigma = \# \\ \sigma & \text{elsewhere} \end{cases}$$

We shall leverage these two insights to construct a DWA that simulates the computation of the expression (28).

Assume for now that the function $\mathbf{G}$ can be computed in polynomial time with respect to the input instance (this fact will be proved later in this segment), a polynomial-time construction of $A_{w,i,P}$ that satisfies the constraint (21):

- *The state space:* $Q = \{0, 1, \ldots, |w|\}^2 \times (\Sigma \cup < BOS >)$.
  The semantics of the elements of a state $q = (k, l, \sigma) \in Q$ correspond to the current position in the sequence, $\texttt{pos}(.)$ and $\texttt{sym}(.)$, respectively.
- *The initial state:* $q_{init} = (0, 0, < BOS >)$
- *The transition function:* Let $q = (k, l, \sigma)$ be a state in Q:
  1. *Case 1 ($k = 0$):*
     - Case 1.1. ($\sigma' = \#$)
     $$W((0, 0, < BOS >), \#) = \left((1, 0, < BOS >), \frac{1}{P_{init}(w'_1 \in L_{\#})}\right)$$
     - Case 1.2. ($\sigma' \in \Sigma$)
     $$W((0, 0, < BOS >), \sigma') = \left((1, 1, \sigma'), \frac{1}{P_{init}(w'_1 \in L_{\sigma'})}\right)$$
  2. *Case 2. ($k = i - 1$)* For any $\sigma' \in \Sigma_{\#}$
     $$W((i - 1, l, \sigma), \sigma') = \left((i, l, \sigma), \frac{1}{\mathbf{G}(\#, l, \sigma, k + 1, P)}\right)$$
  3. *Case 3 ($k \neq i - 1 \wedge k \in [|w| - 1]$):*
     - *Case 3.1. ($\sigma' = \#$)*
     $$W((k, l, \sigma), \#) = \left((k + 1, l, \sigma), \frac{1}{\mathbf{G}(\#, l, \sigma, k + 1, P)}\right)$$
     - *Case 3.2. ($\sigma' \in \Sigma$)*
     $$W((k, l, \sigma), \sigma') = \left((k + 1, k + 1, \sigma')) = \frac{1}{\mathbf{G}(\sigma', l, \sigma, k + 1, P)}\right)$$
- *The set of final states:* $F = \{|w|\} \times \{0, 1, \ldots, |w|\} \times (\Sigma \cup < \text{BOS} >)$

• **Note:** *The case $k = i - 1$ in the algorithmic construction outlined above corresponds to the case where the swap operation is taken into account. The adaption of this construction to algorithm $\mathcal{A}_2$ consists simply at omitting this case and considering only cases 1 and 3, where case 3 covers the set $k \in [|w| - 1]$.*

For illustrative purposes, we shall give next an example of the path followed by a pattern in the constructed DWA.

• **Example.** Fix the alphabet $\Sigma = \{a, b\}$ and $P \in \texttt{MARKOV}$. Let $w = aabab$ the instance to explain and the symbol for which we aim at computing the SHAP score is the third symbol, i.e. $i = 3$.

Let's consider the pattern $p = \#\#aab$. By Bayes' rule, the probability of generating a sequence that follows the pattern $\texttt{swap}(p, 3) = \#\#\#ab$ is equal to:

$$\begin{aligned} P(w' \in L_{\#\#\#ab}) &= P(w'_5 \in L_b | w'_{1:4} \in L_{\#\#\#a}) \cdot P(w'_4 = L_a | w'_{1,3} \in L_{\#\#\#}) \cdot P(w'_3 \in L_\# | w'_{1,2} \in L_{\#\#}) \\ &\quad \cdot P(w'_2 \in L_\# | w'_1 \in L_\#) \cdot P(w'_1 \in L_\#) \\ &= \mathbf{G}(b, 4, a, 5, P) \cdot \mathbf{G}(a, 0, < \text{BOS} >, 4, P) \cdot \mathbf{G}(\#, 0, < \text{BOS} >, 3, P) \cdot \mathbf{G}(\#, 0, < \text{BOS} >, 2, P) \\ &\quad \cdot P_{init}(w'_1 \in L_\#) \end{aligned}$$

$\mathbf{G}(b, 4, a, 5, P)$ holds the semantics of the conditional probability of generating the symbol b at position 5 given that the symbol a is generated at position 4. Similarly, $\mathbf{G}(a, 0, < \text{BOS} >, 4, P)$ holds the semantics of the marginal probability of generating the symbol a at position 4.

The unique path followed by the pattern p on the DWA constructed above is:

$$(0, 0, < \text{B0S} >) \ \# \ (1, 0, < \text{BOS} >) \ \# \ (2, 0, < BOS >) \ a \ (3, 0, < BOS >) \ a \ (4, 4, a) \ b \ (5, 5, b)$$

The weight assigned to this path by the constructed DWA is equal to

$$\frac{1}{P_{init}(w'_1 \in L_\#)} \cdot \frac{1}{\mathbf{G}(\#, 0, < \text{BOS} >, 2, P)} \cdot \frac{1}{\mathbf{G}(\#, 0, < \text{BOS} >, 3, P)} \cdot \frac{1}{\mathbf{G}(a, 0, < \text{BOS} >, 4, P)} \cdot \frac{1}{\mathbf{G}(b, 4, a, 5, P)}$$

By noting that for any $\sigma \in \Sigma_\# : \quad P_{init}(w' \in L_\#) = \mathbf{G}(\sigma, 0, < \text{BOS} >, 1, P)$, the weight of this path is equal to $\frac{1}{P(w' \in L_{\#\#\#ab})} = \frac{1}{P(w' \in L_{\texttt{swap}(\#\#aab, 3)})}$.

• **Computation of the function G.**
In order for this constructed DWA to run in time polynomial in the size of its input instance, a necessary and sufficient condition is that the computation of the function $\mathbf{G}$, can also be performed in polynomial time. We shall prove next that this last statement is true.

Formally, the computational problem associated to the function $\mathbf{G}$ is given as follows:

• **Problem:** The computational problem $\mathbf{G}$
Instance: $\sigma' \in \Sigma_\#$, two integers $n,\ m > 0$ such that $n < m$, a symbol $\sigma \in \Sigma \cup < \text{BOS} >$ and $P \in \texttt{MARKOV}$.
Output: Compute $G(\sigma', n, \sigma, m, P)$ (equation (27)).

For an input instance $< \sigma', n, \sigma, m, P >$, the quantity $\mathbf{G}(\sigma', n, \sigma, m, P)$ refers to the conditional probability of generating a symbol in $L_{\sigma'}$ at position m given that the symbol σ has been generated at position n. In essence, the computational problem $\mathbf{G}$ is reduced to the classical problem of inference in Bayesian Networks (Koller & Friedman, 2009). In general, the exact inference in Bayesian Networks is intractable (Tacettin & Ünlüyurt, 2005). However, in our case, leveraging the Markovian structure of the probability distribution enables building a tractable solution for the problem using a dynamic programming approach.

Fix an input instance $< \sigma', n, \sigma, m, P >$ of the problem $\mathbf{G}$. Define the random vector $(X_n, \ldots, X_m)$ that takes values over the set Σ^{m-n}. Its joint probability distribution is given as follows:

$$Q(\sigma_n, \ldots, \sigma_m) = Q_{init}(\sigma_n) \cdot \prod_{i=n}^{m-1} P_i(\sigma_{i+1} | \sigma_i)$$

such that

$$Q_{init}(\sigma_n) = \begin{cases} 1 & \text{if } \sigma_n = \sigma \\ 0 & \text{elsewhere} \end{cases}$$

It's easy to observe that

$$G(\sigma', n, \sigma, m, P) = Q(X_m \in L_{\sigma'}) \tag{29}$$

If $\sigma' = \#$, the computation of $G(\sigma', n, \sigma, m, P)$ is trivial. Indeed, the fact that $L_\# = \Sigma$ and $Q(X_m \in \Sigma) = 1$ entail, by equation (29) that $G(\#, n, \sigma, m, P) = 1$.

For the general case $\sigma' \in \Sigma$, a recursive formula to compute $G(\sigma', n, \sigma, m, P)$ can be obtained, using Bayes' rule, as follows:

$$\begin{aligned}\mathbf{G}(\sigma', n, \sigma, m, P) &= Q(X_m = \sigma') \\ &= \sum_{\tilde{\sigma} \in \Sigma} Q(X_{m-1} = \tilde{\sigma} \wedge X_m = \sigma) \\ &= \sum_{\tilde{\sigma} \in \Sigma} Q(X_m = \sigma | X_{m-1} = \tilde{\sigma}) \cdot Q(X_{m-1} = \tilde{\sigma}) \\ &= \sum_{\tilde{\sigma} \in \Sigma} P_{m-1}(\sigma|\tilde{\sigma}) \cdot G(\tilde{\sigma}, n, \sigma, m-1, P)\end{aligned} \tag{30}$$

This last equation provides a recursive formula that enables the computation of $\mathbf{G}$ using a dynamic programming approach. The outline of this approach is given as follows:

- **Base case:** $m = n + 1$
  1. If $n = 0$:
  $$\mathbf{G}(\sigma', 0, \sigma, m, P) = P_{init}(\sigma')$$
  2. If $n > 0$:
  $$\mathbf{G}(\sigma', n, \sigma, m, P) = P_{n+1}(\sigma'|\sigma)$$
- **General case:** $m > n + 1$
  $$\mathbf{G}(\sigma', n, \sigma, m, P) = \sum_{\tilde{\sigma} \in \Sigma} P_{m-1}(\sigma|\tilde{\sigma}) \cdot G(\sigma', n, \sigma, m-1, P)$$

The complexity of this dynamic programming algorithm is $O(m.|\Sigma|)$.

D. Proof of lemma 4.4

Lemma 4.4 states the existence of an algorithm that takes as input a d-DNF Φ, runs in $O(\texttt{poly}(|\Phi|, |\Phi|_{\#}))$, and outputs a WA that implements the language L_Φ. Recall that L_Φ is defined as

$$L_\Phi = \{w \in \Sigma^{|\Phi|} : \texttt{SEQ}^{-1}(w) \text{ satisfies } \Phi\}$$

The unweighted language L_Φ includes the set of satisfying variable assignments of the boolean variables arranged in a sequence format.

The structure of the algorithm that performs this task follows two steps:

1. Encode every clause C in the input d-DNF in the form of a DFA. The resulting DFA accepts the language L_C.
2. Perform a union operation over all these DFAs to obtain a resulting WA. The key observation at the heart of this step is that, for the case of disjoint DNFs the union operation can be performed using a basic sum operation over DFAs constructed in the first step.

Next, we shall provide details of these two steps of the algorithmic construction.

D.1. Step 1: Encoding clauses as DFAs.

The basic intuition for performing this step is that an equivalent representation of the language accepted by a clause can be alternatively represented by a pattern of length $|p|$. On the other hand, a pattern of length $|p|$ can be implemented using a DFA of size at most $|p| + 1$.

Let $C = l_1 \wedge \ldots \wedge l_k$ be a conjunctive clause over N boolean variables. We shall denote by L_C the set of satisfying variable assignments of the clause C arranged in a sequence format.

The construction of a pattern p such that $L_p = L_C$ can be performed by scanning the literals of the clause C from left-to-right. Assume that the clause C doesn't possess a variable and its negation in its set of literals [6]. The algorithmic schema is given as follows:

1 Initialize a pattern p as $\#^N$
2 For each literal l_i in C:
- If l_i corresponds to a variable X_k, then set $p_k = 1$
- If l_i corresponds to the negation a variable $\bar{X}_k$, then set $p_k = 0$

The algorithmic schema ensures that the language of the outputted pattern accepts all and only sequences that satisfy the constraints enforced by all literals of the clause.

The pattern construction of a clause in a d-DNF Φ as well as its conversion to a DFA can be performed in $O(|\Phi|)$ time. And, the size of resulting DFA is $O(|\Phi|)$. Repeating the same operation over all clauses of Φ runs in $O(|\Phi| \cdot |\Phi|_{\#})$ time.

• **Example:** The pattern associated to $C = X_2 \wedge \bar{X}_4 \wedge \bar{X}_3$ over the set of boolean variables $\{X_1, X_2, X_3, X_4, X_5\}$ is $\#100\#$.

D.2. Step 2: The union of DFAs representing clauses.

Fix a d-DNF $\Phi = C_1 \vee \ldots \vee C_M$ over N boolean variables. Let $A_1, \ \ldots, \ A_M$ be a collection of DFAs (outputted by step 1) that accept the languages $L_{C_1}, \ \ldots, \ L_{C_M}$, respectively. The main problem of this step is how to exploit DFAs outputted in the first step to construct a WA A such that $I_{L_\Phi} = f_A$.

The main intuition at this point is to note that for a d-DNF, I_{L_Φ} can be expressed as as a sum of the indicator functions of $\{I_{L_{C_i}}\}_{i\in[M]}$. Since $f_{A_i} = I_{L_{C_i}}$ for any $i \in [M]$, then f_A can be computed as a sum over languages computed by DFAs . This observation will result into a reduction of the problem of constructing a WA A that computes I_{L_Φ} into performing a sum operation over a collection of DFAs. Fortunately, WAs are closed under the sum operation. Moreover, it can be computed in polynomial time with respect to the size.

Lemma D.1. *Let $\Phi = C_1 \vee \ldots \vee C_M$ be a disjoint DNF over N boolean variables. We have:*

$$I_{L_\Phi} = \sum_{i\in[M]} I_{L_{C_i}}$$

Proof. Let $\Phi = C_1 \vee \ldots \vee C_M$ be a disjoint DNF over N boolean variables. Let w be an arbitrary sequence in $\{0,1\}^n$. Our claim is that $I_{L_\Phi}(w) = \sum_{i\in[M]} I_{L_{C_i}}(w)$. Note that $L_\Phi = \bigcup_{i=1}^{M} L_{C_i}$ by definition of Φ. Also, $\bigcap_{i=1}^{M} C_i = \emptyset$ by the disjoint property of Φ.

- *Case 1 ($w \notin L_\Phi$):* This implies that $I_{L_\Phi}(w) = 0$. On the other hand, $w \notin L_\Phi$ and $L_\Phi = \bigcup_{i=1}^{M} L_{C_i}$ implies that

$$\forall i \in [M] : w \notin L_{C_i} \implies \forall i \in [M] : I_{L_{C_i}}(w) = 0 \implies \sum_{i=1}^{M} I_{L_{C_i}}(w) = 0$$

- *Case 2 ($w \in L_\Phi$):* In this case, $I_{L_\Phi}(w) = 1$. On the other hand, $L_\Phi = \bigcup_{i=1}^{M} L_{C_i}$ implies that there exists at least one clause C_i such that $w \in L_{C_i}$. This fact combined with the fact that $\bigcap_{i=1}^{M} C_i = \emptyset$ implies that this clause is unique. Denote by C^* this clause. We have $I_{L_{C^*}}(w) = 1$. And, $I_{L_C}(w) = 0$ for any clause $C \in \{C_i\}_{i\in[M]} \setminus C^*$. Consequently, $\sum_{i=1}^{M} I_{L_{C_i}}(w) = 1$.

[6] Clauses that exhibit this degenerate case can be checked and removed before running the algorithmic schema outlined here.

□

The result of lemma D.1 implies that a WA A that computes the language I_{L_ϕ} satisfies:

$$f_A = \sum_{i=1}^{M} I_{L_{C_i}} = \sum_{i=1}^{M} f_{A_i} \tag{31}$$

For two WAs $A_1 =< \alpha, \{A_\sigma\}_{\sigma\in\Sigma}\beta >$ and $A_2 =< \alpha', \{A'_\sigma\}_{\sigma\in\Sigma}, \beta' >$, the WA whose set of parameters is given as

$$< \begin{pmatrix} \alpha \\ \alpha' \end{pmatrix}, \{ \begin{pmatrix} A_\sigma & \mathbf{O}_{\texttt{size}(A_1)\times\texttt{size}(A_2)} \\ \mathbf{O}_{\texttt{size}(A_2)\times\texttt{size}(A_1)} & A'_\sigma \end{pmatrix} \}_{\sigma\in\Sigma}, \begin{pmatrix} \beta \\ \beta' \end{pmatrix} >$$

where $\mathbf{0}_{n\times m}$ is the zero matrix in $\mathbb{R}^{n\times m}$, computes the language $f_A + f_{A'}$.

The resulting WA runs in $O(\texttt{size}(A_1) + \texttt{size}(A_2))$ time, and has size equal to $\texttt{size}(A_1) + \texttt{size}(A_2)$.

Hence, the construction of the target WA A by performing the sum operation over $\{A_i\}_{i\in[M]}$ as outlined by equation (31) would take $O(\sum_{i=1}^{M} \texttt{size}(A_i))$ operations. Since the DFAs $\{A_i\}_{i\in[M]}$ have size equal to $O(|\Phi|)$. Then, the overall operation runs in $O(|\Phi| \cdot |\Phi|_\#)$.