Polynomial-Time Conformance Checking for Process Trees

Eduardo Goulart Rocha[1,2(✉)][0009−0000−1184−1188] and
Wil M.P. van der Aalst[2,1][0000−0002−0955−6940]

[1] Celonis Labs GmbH, Munich, Germany
[2] Process and Data Science (PADS) Chair, RWTH Aachen University, Germany
e.goulartrocha@celonis.com wvdaalst@pads.rwth-aachen.de

Abstract. Conformance-checking is the field of process mining relating modeled and observed behavior. State-of-the-art conformance checking techniques do not scale for large process models and event logs, which hampers its broader adoption.
In this paper, we present a polynomial-time method to compute the markovian-based fitness and precision metrics for process trees. For that, we first show that this is equivalent to the problem of computing the set of substrings of length at most k of the model's language. Then, we show how to exploit the tree structure to compute this set in a compositional way. The experimental evaluation shows that the proposed method outperforms state-of-the-art conformance-checking techniques by orders of magnitude, while still providing quality guarantees.

Keywords: Process Mining · Conformance Checking · Process Trees.

1 Introduction

Conformance checking is the field of process mining relating desired and observed behavior. Given an event log and a process model, conformance checking aims at identifying and quantifying differences between the event log and the process model. An important use-case for conformance checking is to assess the quality of automatically discovered process models in the form of a single number evaluation metric. For that, multiple conformance metrics with distinct runtime and quality characteristics have been proposed in the literature [1, 8, 11]. Unfortunately, most state-of-the-art methods still require a runtime that is exponential on the number of activities or do not satisfy all the desired axioms for a conformance metric [13].

A notable exception are the Projected Conformance Checking (PCC) fitness and precision metrics [7], which provide strong runtime and quality guarantees for certain classes of models (process trees with unique activities and no invisible labels). Nevertheless, the PCC metrics require multiple passes over the event log, which makes them expensive to compute for large datasets.

In this work, we focus on the problem of efficiently computing conformance metrics for process trees. Process trees are a well-established modeling formalism

in process mining because of its soundness guarantees and simple structure. For instance, many state-of-the-art process discovery algorithms return process trees. We provide two important contributions: First, we present a simplified, yet more expressive, definition of the k-th order markovian abstraction first presented in [2]. Next, we show how to compute the k-th order markovian abstraction of a process tree in polynomial time by exploiting the tree structure. The method achieves an improvement of orders of magnitude in computation time for models with a high degree of parallelism. Furthermore, the method scales linearly with the size of the event log, making it suitable for very large event-logs.

The remainder of the paper is organized as follows: Section 2 presents basic notations and concepts from automata theory, which are the backbone of the presented technique, Section 3 presents the general framework for computing the k-th order markovian abstraction of a process tree, Section 4 compares the approach to other state-of-the-art methods, Section 5 presents related work in the field. Finally, Section 6 concludes the paper with directions for future work.

2 Preliminaries

This section presents the basic concepts upon which the method is based. For a given finite alphabet Σ, Σ^k is the set of all finite words of length k formed with this alphabet and $\Sigma^* = \bigcup_{k \geqslant 0} \Sigma^k$. The projection of a word $w \in \Sigma^*$ in a set of symbols $S \subseteq \Sigma$ is written w_S. The concatenation of two words u, v is written uv. Similarly, the concatenation of two languages $U, V \subseteq \Sigma^*$ is written as $UV = \bigcup_{u \in U, v \in V} uv$. Given a word $w = w_1 w_2 \cdots w_n$ and $1 \leqslant i \leqslant j \leqslant n$, $w^{i \to j} = w_i w_{i+1} \cdots w_j$ denotes a substring of w (written $\gamma \sqsubseteq w$). We further write $\mathit{pref}^k(w)$, $\mathit{suff}^k(w)$, and $\mathit{sub}^k(w)$ to denote the set of non-empty prefixes, suffixes, and substrings of w with length less than or equal to k. The definitions of pref^k, suff^k, and sub^k are extended to languages too. Finally, the paper assumes familiarity with basic algorithms of automata theory [5]. We provide common notations for finite automata below:

Definition 1. ***(Labeled Directed Graph)*** *A Labeled Directed Graph is a triple $G = (V, \Sigma, E)$ where V is the set of vertices, Σ is the set of labels and $E \subseteq V \times \Sigma \times V$ is the set of edges. Given an edge $e = (v, l, v\prime)$, functions $\pi_{src}(e) = v$, $\pi_{tgt}(e) = v\prime$, and $\pi_l(e) = l$ return its source and target vertices and its label respectively*

For this paper, all considered graphs are labeled directed graphs. A *path* p in the a graph is a sequence of edges $p = e_1 e_2 \cdots e_n$ such that $\pi_{tgt}(e_i) = \pi_{src}(e_{i+1})\ \forall 1 \leqslant i < n$. We define $\pi_l(p) = \pi_l(e_1)\pi_l(e_2)\cdots\pi_l(e_n)$ as the path's *labeling* ($= \epsilon$ if the path is empty). And $\pi_v(p, i) = \begin{cases} \pi_{src}(e_1) & i = 0 \\ \pi_{tgt}(e_i) & i > 0 \end{cases}$ as the i-th vertex visited by the path, where $\pi_v(p, 0)$ is the path's *start vertex*.

Definition 2. ***(Nondeterministic Finite Automaton)*** *Let ϵ be the empty string. A Nondeterministic Finite Automaton (NFA) is a 5-tuple $N = (Q, \Sigma, \delta, q_0,$*

F), where Q is the set of states, Σ is the alphabet, $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ is the transition function, q_0 is the initial state and $F \subseteq Q$ is the set of final states.

Any NFA $N = (Q, \Sigma, \delta, q_0, F)$ is associated to a graph $G = (Q, \Sigma \cup \{\epsilon\}, E)$ where $E = \{(q, l, q\prime) \in Q \times (\Sigma \cup \{\epsilon\}) \times Q \mid q\prime \in \delta(q, l)\}$ (called the *NFA's graph*). Given $q, q\prime \in Q$, we define $q[w\rangle q\prime = \{p \in E^* \mid \pi_l(p) = w \wedge \pi_v(p, 0) = q \wedge \pi_v(p, |p|) = q\prime\}$ as all the paths from q to $q\prime$ labeled by w. If $q = q_0$, $q\prime \in F$, and $q[w\rangle q\prime \neq \varnothing$, then N *accepts* w and p is an *accepting path*. The accepted language of N is defined as $\mathcal{L}(N) = \{w \in \Sigma^* \mid \exists f \in F \text{ s.t. } q_0[w\rangle f \neq \varnothing\}$. Similarly, we denote $q[w_1\rangle q\prime[w_2\rangle q\prime\prime = \{p_1 p_2 \mid p_1 \in q[w_1\rangle q\prime \wedge p_2 \in q\prime[w_2\rangle q\prime\prime\}$. Last, we define $q[w\rangle = \{q\prime \in Q \mid q[w\rangle q\prime \neq \varnothing\}$ as the set of states reachable from q by replaying w.

In an NFA N, a state is *dead* if it is not reachable from the start state and it is a *trap* if there is no path q leading from the state to a final state. We say that an NFA is *trimmed* if it has no dead or trap states. For any trimmed NFA, any path p in its graph is such that $\pi_l(p)$ is a substring of $\mathcal{L}(N)$.

Definition 3. ***(Deterministic Finite Automaton (DFA))*** *A Deterministic Finite Automaton (DFA) is an NFA where $\delta(q, \epsilon) = \varnothing \ \forall q \in Q$ and $|\delta(q, l)| \leqslant 1 \ \forall q \in Q, l \in \Sigma$.*

Every DFA has the property that two paths in its graph starting from the same node are equal if and only if their labelings are the same, i.e. $|q[w\rangle| \leqslant 1$. We will abuse notation and write $q[w\rangle q\prime$ to refer to the single element of this set (when it exists). Given an NFA N, there exists an unique (up to isomorphism) DFA D with a minimal number of states such that $\mathcal{L}(N) = \mathcal{L}(D)$ that can be obtained via *determinization*. This can be achieved via the *powerset construction* followed by a *minimization* step [5]. In the worst case, D has exponentially many more states than N. If the DFA's graph is acyclic, we call it a *Deterministic Acyclic Finite State Automaton* (DAFSA). Given a finite language $L \subseteq \Sigma^*$, it is possible to construct a minimal DAFSA accepting L in linearithmic time [4].

While finite automata can be used to represent any regular language, process analysts need a compact and understandable modeling formalism. Among which, process trees [3] stand out for their soundness guarantees and block structure. Process trees are graphs with a tree structure. In a process tree, the leaf nodes represent activities in Σ or skips (τ) and the internal nodes represent one of four possible operators: *exclusive* ($\times$), *sequence* ($\rightarrow$), *loop* ($\circlearrowleft$), and *parallel* ($\wedge$). The tree's accepted language is defined recursively as follows:

Definition 4. ***(Process Trees Semantics)*** *Let $\sqcup\!\sqcup$ be the shuffle product of two words, defined as:* $\begin{cases} w \sqcup\!\sqcup \epsilon = \epsilon \sqcup\!\sqcup w = \{w\} & w \in \Sigma^* \\ xu \sqcup\!\sqcup yv = \{x\}(u \sqcup\!\sqcup yv) \cup \{y\}(xu \sqcup\!\sqcup v) & x, y \in \Sigma \wedge u, v \in \Sigma^* \end{cases}$

For languages A, B, define $A \sqcup\!\sqcup B = \bigcup_{w_a \in A, w_b \in B} w_a \sqcup\!\sqcup w_b$. Then, the accepted language of a process tree is recursively defined as:

- $\mathcal{L}(\tau) = \{\epsilon\}$

- $\mathcal{L}(a) = \{a\}$
- $\mathcal{L}(\times(T_1, \cdots, T_n)) = \bigcup_{i=1}^{n} \mathcal{L}(T_i)$
- $\mathcal{L}(\rightarrow(T_1, \cdots, T_n)) = \mathcal{L}(T_1)\mathcal{L}(T_2)\cdots\mathcal{L}(T_n)$
- $\mathcal{L}(\circlearrowleft(T_1, \cdots, T_n)) = \mathcal{L}(T_1)(\mathcal{L}(\times(T_2, \cdots, T_n))\mathcal{L}(T_1))^*$
- $\mathcal{L}(\wedge(T_1, \cdots, T_n)) = \mathcal{L}(T_1) \sqcup\!\sqcup \mathcal{L}(T_2) \sqcup\!\sqcup \cdots \sqcup\!\sqcup \mathcal{L}(T_n)$

Given a process tree T, there exists a unique minimal DFA D such that $\mathcal{L}(T) = \mathcal{L}(D)$ [3]. However, the size of D might be exponential with the size of T. This exponential blow-up is the bottleneck for most conformance checking techniques, including the metrics based on the k-th order markovian abstraction presented in [2]. This paper focuses on improving the runtime for computing the k-th order markovian abstraction. For this, we use a slightly different definition than the one originally introduced in [2], based on the set of k-trimmed substrings of a language:

Definition 5. ***(K-Trimmed Substrings)*** *Let Σ be an alphabet and $k \geqslant 2$. Given a word $w \in \Sigma^*$, the set of k-trimmed substrings $s^k(w)$ is defined as:*

$$s^k(w) = \begin{cases} \{w\} & \text{if } |w| \leqslant k \\ \{w^{i \rightarrow i+k-1} \mid 1 \leqslant i \leqslant |w| - k + 1\} & \text{otherwise} \end{cases}$$

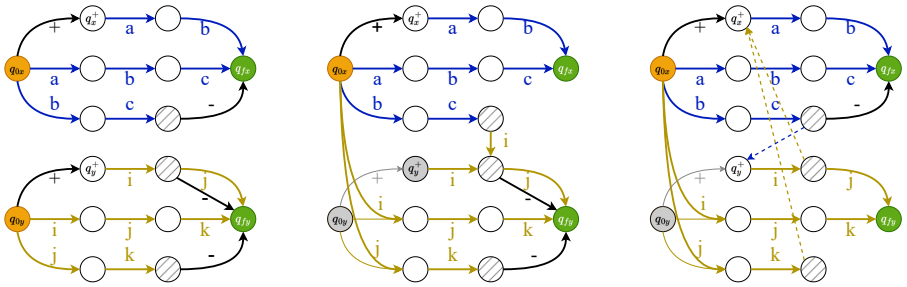
We extend the definition of s^k to languages as $s^k(L) = \bigcup_{w \in L} s^k(w)$. Consider languages $X = \{abc\}$ and $Y = \{i, ijk\}$ (these will serve as a running example for the remainder of the paper), then $s^1(X) = \{a, b, c\}$, $s^2(X) = \{ab, bc\}$, and $s^k(X) = \{abc\}$ for $k \geqslant 3$ and $s^1(Y) = \{i, j, k\}$, $s^2(Y) = \{i, ij, jk\}$, and $s^k(Y) = \{i, ijk\}$ for $k \geqslant 3$. The k-th order markovian abstraction (defined below) is similar to the set of k-trimmed substrings, but with special start/end markers ($+/-$) to track the language's prefixes/suffixes.

Definition 6. ***(The Modified k-th Order Markovian Abstraction)*** *Let Σ be an alphabet, $+/- \notin \Sigma$ be special start/end markers, and $k \geqslant 2$. Given a word $w \in \Sigma^*$, the k-order markovian abstraction of w is defined as follows:*

$$m^k(w) = s^k(+w-)$$

Similarly, m^k of a language $L \subseteq \Sigma^*$ is defined as $\bigcup_{w \in L} m^k(w)$. In principle, m^k is defined for arbitrary languages, but throughout the rest of this work we focus on computing m^k for regular languages. We will always assume that $+, - \notin \Sigma$ and write $\Sigma^{\pm} = \Sigma \cup \{+, -\}$ and $+L- = \{+\}L\{-\}$. For any language $L \subseteq \Sigma^*$, $m^k(L)$ represents a finite set of finite words and thus can be associated to a unique minimal DAFSA (written M_L^k). Figure 1a shows the minimal DAFSAs M_X^3 and M_Y^3 accepting $m^3(X) = \{+ab, abc, bc-\}$ and $m^3(Y) = \{+i-, +ij, ijk, jk-\}$ respectively. In general, M_L^k has a very specific structure, detailed below:

Proposition 1. ***(Basic Properties of $\mathbf{M_L^k}$)*** *Let Σ be an alphabet, $L \subseteq \Sigma^*$ be an arbitrary language, $k \geqslant 2$, and $M_L^k = (Q, \Sigma, \delta, q_0, F)$ the minimal DAFSA accepting $m^k(L)$, then:*



(a) M_X^3 and M_Y^3 (b) $M_X^3 \odot^{\pm} M_Y^3$ (c) $M_X^3 \circlearrowleft^{\pm} M_Y^3$

Fig. 1: M_X^3 and M_Y^3 for $X = \{abc\}$ and $Y = \{ijk, i\}$ and their sequence and loop concatenations. Start states are colored orange, final states are colored green, dead states are gray, states in Q^- are hatched, and ϵ-transitions are represented as dashed lines.

1. $m^k(L) \subseteq m^k(\Sigma^*) = \left(\bigcup_{0 \leqslant i < k-1} +\Sigma^i-\right) \cup (+\Sigma^{k-1}) \cup (\Sigma^{k-1}-) \cup \Sigma^k$
2. M_L^k has only one final state, i.e. $F = \{q_f\}$
3. M_L^k has exactly one edge labeled $+$. This edge has q_0 as its source and we write q^+ to represent its target, i.e. $\delta(q_0, +) = \{q^+\}$
4. All $-$-labeled edges in M_L^k lead to its unique final state. We define $Q^- = \{q \in Q \mid q_f \in \delta(q, -)\}$
5. For every $\gamma \in sub^k(+L-)$, there exists a path p in M_L^k such that $\pi_l(p) = \gamma$, and for every path p in M_L^k, there exists $\gamma \in sub^k(+L-)$ such that $\gamma = \pi_l(p)$.
6. For every path p in M_L^k, $+ \in \pi_l(b) \iff \pi_l(p) \in pref^k(+L-)$. Similarly, $- \in \pi_l(p) \iff \pi_l(p) \in suff^k(+L-)$
7. $|Q| \leqslant |\Sigma|^{k-1} + 2$

Finally, Definition 7 presents the (modified) markovian-based fitness and precision metrics. The metrics return almost the same (but not the same) values as the ones presented in [2], because the original definition counts words of length smaller than k twice. However, monotonicity still holds for our definition of m^k, i.e. if $A \subseteq B \Rightarrow m^k(A) \subseteq m^k(B)$, such that the proofs of the axioms presented in [2] are still valid.

Definition 7. ***(Markovian-Based Fitness and Precision with the Binary Cost Function)*** *Let $\mathbb{L}$ be an event log with language $L \subseteq \Sigma^*$, $\mathbb{P}$ be a process model with language $P \subseteq \Sigma^*$, $k \geqslant 2$, and $\#_{\mathbb{L}}(\gamma)$ the number of occurrences of substring γ in $\mathbb{L}$. Then $MAF^k(L, P) = 1 - \frac{\sum_{\gamma \in (m^k(L) \setminus m^k(P))} \#_{\mathbb{L}}(\gamma)}{\sum_{\gamma \in m^k(L)} \#_{\mathbb{L}}(\gamma)}$ and $MAP^k(L, P) = 1 - \frac{|m^k(P) \setminus m^k(L)|}{|m^k(P)|}$ are the markovian-based fitness and precision metrics respectively.*

The metrics are the set difference of the languages' substrings. The fitness metric is normalized by the substring frequency. Since that the current setting

does not consider any notion of trace frequency for process models, the precision metric is normalized by $1/|m^k(P)|$. It is possible to obtain a variation of the metric by changing the cost function (see [2]). The markovian-based fitness and precision metrics were empirically shown to agree with other state-of-the-art conformance metrics such as escaping edges and PCC. However, the original method for computing m^k requires the computation of the process model's DFA and thus does not scale for larger models. In the next section, we show how to compute m^k for process trees without computing its state space, hence improving scalability.

3 General Framework

This section shows how to efficiently compute m^k for arbitrary process trees. First, we present a method to compute s^k for arbitrary regular languages. Next, we show a compositional approach to compute m^k for binary and uniquely-labeled process trees which works by recursively computing m^k for each tree node from the m^k of its child nodes. Last, we show how to generalize it for arbitrary process trees.

3.1 Computing s^k of a Regular Language

This section presents a method to compute $s^k(L)$ from a DFA accepting L.

Definition 8. ***(All-Substrings NFA)*** *Given a trimmed DFA $D = (Q, \Sigma, \delta, q_0, F)$, its all-substrings NFA is defined as $Sub_D = (Q \cup \{\hat{q}\}, \Sigma, \hat{\delta}, \hat{q}, Q \cup \{\hat{q}\})$, where $\hat{\delta}$ is defined as follows:*

$$\hat{\delta}(q, l) = \begin{cases} \delta(q, l) & \text{if } l \in \Sigma \wedge q \in Q \\ Q & \text{if } l = \epsilon \wedge q = \hat{q} \\ \varnothing & \text{otherwise} \end{cases}$$

Notice that Sub_D accepts all substrings of DFA D. This can be used to efficiently compute s^k as follows:

Lemma 1. ***(Computing s^k)*** *Let $L \subseteq \Sigma^*$ be a regular language and $D = (Q, \Sigma, \delta, q_0, F)$ a DFA accepting L. Then $s^k(L)$ can be computed in $\mathcal{O}(|Q||\Sigma|^k)$.*

Proof. We assume D to be trimmed, otherwise D can be trimmed in $\mathcal{O}(|Q|)$. We define $s^{=k}(L) = \{w \in s^k(L) \mid |w| = k\}$ and $s^{<k}(L) = s^k(L) \backslash s^{=k}(L)$. Notice that $s^{<k}(L)$ can be computed in $\mathcal{O}(|\Sigma|^{k-1})$ time by running BFS from the start node with a maximum depth of $k-1$. We prove that $\mathcal{L}(Sub_D) \cap \Sigma^k = s^{=k}(L)$:

($\subseteq$) Any $w \in \mathcal{L}(Sub_D) \cap \Sigma^k$ is such that there exists an accepting path $p \in \hat{q}[\epsilon\rangle q_i[w\rangle q_{i+k}$ in Sub_D. And since D is trimmed, there exists $u, v \subseteq \Sigma^*$ such that $q_0[u\rangle q_i \neq \varnothing$ and $q_{i+k}[v\rangle f \neq \varnothing$, $f \in F$. Hence, $q_0[u\rangle q_i[w\rangle q_{i+k}[v\rangle f \neq \varnothing \Rightarrow uwv \in L \Rightarrow w \in s^{=k}(L)$.

($\supseteq$) For any $w \in s^k(L)$, $\exists t \in L \mid t = uwv$, so $p = q_0[u\rangle q_i[w\rangle q_{i+k}[v\rangle q_n$ is a path in D. But this directly implies that there exists a unique path $p \in \hat{q}[\epsilon\rangle q_i[w\rangle q_{i+k}$ and that p is an accepting path of Sub_D. And since that $w \in \Sigma^k$, then $w \in (\mathcal{L}(Sub_D) \cap \Sigma^k)$.

The runtime bound is achieved by computing $\mathcal{L}(Sub_D) \cap \Sigma^k$ without determinizing Sub_D. The product construction builds a DAFSA. It expands at most $|\Sigma|^k$ nodes, where each node expansion has cost bound by $|Q|$. □

3.2 Leaf And Exclusive Nodes

This section shows how to compute m^k for leaf and exclusive nodes, which do not require any special constructs:

Lemma 2. ***(Leaf Nodes)*** *For* $a \in \Sigma \cup \{\epsilon\}$, $m^k(a) = \begin{cases} \{+-\} & k \geqslant 2,\ a = \epsilon \\ \{+a, a-\} & k = 2,\ a \in \Sigma \\ \{+a-\} & k > 2,\ a \in \Sigma \end{cases}$

Proof. Follows directly from Definition 6. □

Lemma 3. ***(Exclusive Node)*** *Let* $A, B \subseteq \Sigma^*$ *be arbitrary languages. Then:*

$$m^k(A \cup B) = m^k(A) \cup m^k(B)$$

Proof. From Definition 6, $m^k(A \cup B) = s^k(+(A \cup B)-) = s^k(+A- \cup +B-) = s^k(+A-) \cup s^k(+B-) = m^k(A) \cup m^k(B)$. □

3.3 Sequence Node

For the sequence node, m^k is computed based on automata operations. For that, we first define the markovian sequence concatenation $\odot^{\pm}$ as follows:

Definition 9. ***(Markovian Sequence Concatenation)*** *Let* A, B *be arbitrary languages with disjoint alphabets* Σ_A, Σ_B, *and* $M_A^k = (Q_A, \Sigma_A^{\pm}, \delta_A, q_{0a}, \{q_{fa}\})$ *and* $M_B^k = (Q_B, \Sigma_B^{\pm}, \delta_B, q_{0b}, \{q_{fb}\})$ *be the minimal DAFSAs accepting* $m^k(A)$ *and* $m^k(B)$ *respectively. The markovian sequence concatenation* $M_A^k \odot^{\pm} M_B^k$ *builds the DFA* $(Q_A \cup Q_B, (\Sigma_A \cup \Sigma_B)^{\pm}, \hat{\delta}, q_{0a}, \{q_{fa}, q_{fb}\})$ *where* $\hat{\delta}$ *is defined as follows:*

$$\hat{\delta}(q,l) = \begin{cases} \delta_A(q,l) & q \in Q_A, l \in \Sigma_A \cup \{+\} \\ \delta_B(q_{0b},l) & q = q_{0a}, l \in \Sigma_B \\ \delta_B(q_b^+,l) & q \in Q_A^-, l \in \Sigma_B \cup \{-\} \\ \delta_B(q,l) & q \in Q_B, l \in \Sigma_B^{\pm} \\ \varnothing & \textit{otherwise} \end{cases}$$

Figure 1b shows $M_X^3 \odot^{\pm} M_Y^3$, which accepts $\{+ab, abc, bcij, bci-, ijk, jk-\}$. Intuitively, the markovian sequence concatenation is merging the transition function of state q_b^+ into the transition functions of states in Q_A^-. Notice that $s^3(\mathcal{L}(M_X^3 \odot^{\pm} M_Y^3)) = \{+ab, abc, bci, cij, ci-, ijk, jk-\} = m^3(XY)$. Lemma 4 below formalizes this fact, which can be used to compute m^k for the sequence node:

Lemma 4. ***($\mathbf{m^k}$ of Language Concatenation)*** *Let A, B be arbitrary languages with disjoint alphabets Σ_A, Σ_B, and $M_A^k = (Q_A, \Sigma_A^{\pm}, \delta_A, q_{0a}, \{q_{fa}\})$ and $M_B^k = (Q_B, \Sigma_B^{\pm}, \delta_B, q_{0b}, \{q_{fb}\})$ the minimal DAFSAs accepting $m^k(A)$ and $m^k(B)$ respectively. Then:*

$$m^k(AB) = s^k(\mathcal{L}(M_A^k \odot^{\pm} M_B^k))$$

Proof. ($\subseteq$) For any $w \in +AB-$, there exists $\hat{w}_a \in A, \hat{w}_b \in B$ such that $w = +\hat{w}_a\hat{w}_b-$. Then for any $\gamma \in s^k(+\hat{w}_a\hat{w}_b-)$ one of the following holds:

$$\gamma = \begin{cases} \gamma_a & \gamma_a \sqsubseteq +\hat{w}_a \\ \gamma_b & \gamma_b \sqsubseteq \hat{w}_b- \\ \gamma_a\gamma_b & \gamma_a \in \mathit{suff}^k(+\hat{w}_a), \gamma_b \in \mathit{pref}^k(\hat{w}_b-) \end{cases}$$

For Case 1, the condition implies $|\gamma_a| = k$ and so $\gamma_a \in m^k(A)$. Let p be the path in M_A^k accepting γ_a. Since that $- \notin \gamma_a$, and that only $-$-labeled edges were removed from M_A^k, then p is also an accepting path in $M_A^k \odot^{\pm} M_B^k$.

Similarly for Case 2, $|\gamma_b| = k$ and $\gamma_b \in m^k(B)$. Let $p = q_{0b}[\gamma_b\rangle q_{fb}$ be the path in M_B^k accepting γ_b. Then $\hat{p} = q_{0a}[\gamma_b\rangle q_{fb}$ is an accepting path in $M_A^k \odot^{\pm} M_B^k$.

For Case 3, the condition implies $|\gamma_a|, |\gamma_b| < k$, $\gamma_a- \sqsubseteq +\hat{w}_a-$ and $+\gamma_b \sqsubseteq +\hat{w}_b-$. And since $|\gamma_a - |, | + \gamma_b| \leqslant k$ then there exists α_a, β_b such that $\alpha_a\gamma_a- \in s^k(+\hat{w}_a-) \subseteq m^k(A)$ and $+\gamma_b\beta_b \in s^k(+\hat{w}_b-) \subseteq m^k(B)$ (Proposition 1-5), and so $p_a = q_{0a}[\alpha_a\rangle\hat{q}_a[\gamma_a\rangle q_a^-[-\rangle q_{fa}$ and $p_b = q_{0b}[+\rangle q_b^+[\gamma_b\rangle\hat{q}_b[\beta_b\rangle q_{fb}$ are accepting paths in M_A^k and M_B^k. Then $p = q_{0a}[\alpha_a\rangle\hat{q}_a[\gamma_a\rangle q_a^-[\gamma_b\rangle\hat{q}_b[\beta_b\rangle$ is a path in $M_A^k \odot^{\pm} M_B^k$ such that $\pi_l(p) = \alpha_a\gamma\beta_b$, and since $|\gamma| = k$, then $\gamma \in s^k(\alpha_a\gamma\beta_b) \subseteq s^k(\mathcal{L}(M_A^k \odot^{\pm} M_B^k))$.

($\supseteq$) Notice that $M_A^k \odot^{\pm} M_B^k$ is acyclic. Therefore, for every accepting path p in $M_A^k \odot^{\pm} M_B^k$ accepting $\pi_l(p) = w \in \mathcal{L}(M_A^k \odot^{\pm} M_B^k)$, there exists $0 \leqslant j \leqslant n$ such that $\pi_s(p, i) \in Q_A \ \forall i \leqslant j$ and $\pi_s(p, i) \in Q_B \ \forall i > j$. If $j = 0$, then $w \in s^k(+B-)$ and $+ \notin w \Rightarrow w \in \mathit{sub}^k(B-)$. If $j = n$, then $w \in s^k(+A-)$ and $- \notin w \Rightarrow w \in \mathit{sub}^k(+A)$. In both cases, $|w| = k$, thus $w \in s^k(+AB-)$.

If $0 < j < n$, it holds that $e_j = (q_{j-1}, w_j, q_j)$ where $q_{j-1} \in Q_A^-$, $w_j \in \Sigma_B$, and $q_j \in \delta_B(q_b^+, w_j)$. So $w^{1\to j-1}- \in s^k(+A-)$ and $+w^{j\to n} \in s^k(+B-)$ (Proposition 1-6). Which implies that w is a substring of $+AB-$. Now if $|w| \leqslant k$, then $+, - \in w$, which implies that $w \in +AB-$ and thus $\{w\} = s^k(w) \subseteq s^k(+AB-)$. If $|w| > k$, then it follows directly that $s^k(w) \subseteq s^k(+AB-)$. □

$M_A^k \odot^{\pm} M_B^k$ is a DAFSA with $|Q_A| + |Q_B| \in \mathcal{O}(|\Sigma_A \cup \Sigma_B|^k)$ states. From Lemma 1, it follows that $m^k(AB)$ can be computed in $\mathcal{O}(|\Sigma_A \cup \Sigma_B|^{2k})$.

3.4 Loop Node

In Section 3.3, we have seen how to concatenate two DAFSAs to compute m^k for language concatenation. Similarly, we define the loop concatenation as a construct to compute m^k for the loop node's language. We first define an NFA constructed from both markovians' DAFSAs and show how this relates to the markovian of the loop node. Then, we show that determinizing this construct is polynomial-time due to its specific structure, thus still being efficient.

Definition 10. ***(Markovian Loop Concatenation)*** *Let A, B be arbitrary languages with disjoint alphabets Σ_A and Σ_B and $M_A^k = (Q_A, \Sigma_A^{\pm}, \delta_A, q_{0a}, \{q_{fa}\})$ and $M_B^k = (Q_B, \Sigma_B^{\pm}, \delta_B, q_{0b}, \{q_{fb}\})$ the minimal DAFSAs accepting $m^k(A)$ and $m^k(B)$ respectively. The markovian loop concatentation $M_A^k \circlearrowleft^{\pm} M_B^k$ builds the NFA $N = (Q_A \cup Q_B, (\Sigma_A \cup \Sigma_B)^{\pm}, \hat{\delta}, q_{0a}, \{q_{fa}, q_{fb}\})$ where $\hat{\delta}$ is defined as follows:*

$$\hat{\delta}(q,l) = \begin{cases} \delta_A(q,l) & q \in Q_A,\ l \in \Sigma_A^{\pm} \\ \delta_B(q_{0b},+) & q \in Q_A^-,\ l = \epsilon \\ \delta_A(q_{0a},+) & q \in Q_B^-,\ l = \epsilon \\ \delta_B(q,l) & q \in Q_B,\ l \in \Sigma_B \cup \{+\} \\ \delta_B(q_{0b},l) & q = q_{0a},\ l \in \Sigma_B \\ \varnothing & \textit{otherwise} \end{cases}$$

Figure 1c shows $M_X^3 \circlearrowleft^{\pm} M_Y^3$ accepting $\{+ab, abc, bc-, bcij, bciab, ijk, jkab\}$. Notice that $s^3(\mathcal{L}(M_X^3 \circlearrowleft^{\pm} M_Y^3)) = \{+ab, abc, bc-, bci, cij, cia, iab, ijk, jka, kab\} = m^3(A(BA)^*)$. Lemma 5 below formalizes this fact, which can be used to compute m^k for loop nodes:

Lemma 5. ***($\mathbf{m^k}$ of the Loop Node)*** *Let A, B be arbitrary languages with disjoint alphabets Σ_A and Σ_B, and $M_A^k = (Q_A, \Sigma_A^{\pm}, \delta_A, q_{0a}, \{q_{fa}\})$ and $M_B^k = (Q_B, \Sigma_B^{\pm}, \delta_B, q_{0b}, \{q_{fb}\})$ the minimal DAFSAs accepting $m^k(A)$ and $m^k(B)$ respectively. Then:*

$$m^k(A(BA)^*) = s^k(\mathcal{L}(M_A^k \circlearrowleft^{\pm} M_B^k))$$

Proof. We define sets $\hat{A} = \{w_a \in A \mid |w_a| \leqslant k-2\}$ and $\hat{B} = \{w_b \in B \mid |w_b| \leqslant k-2\}$. The graph of $M_A^k \circlearrowleft^{\pm} M_B^k$ is such that, for every $\hat{w}_a \in \hat{A}$, there exists $q_a^- \in Q_A^-$ such that $q_a^+[\hat{w}_a\rangle q_a^- \neq \varnothing$ (analogous for $\hat{B}$).

($\subseteq$) For every $w \in +A(BA)^*-$, then $w = +w_{a,1}w_{b,1}w_{a,2}\cdots w_{a,n}-$ s.t. $w_{a,i} \in \hat{A},\ \forall 1 \leqslant i \leqslant n$ and $w_{b,i} \in \hat{B},\ \forall 1 \leqslant i < n$. We distinguish between two cases:

Case 1: ($|w| \leqslant k$) Then $|+w_{a,i}-| \leqslant k$ for every $i \leqslant n$, which implies that $+w_{a,i}- \in \mathcal{L}(M_A^k)$. Therefore, $p_{a,i} = q_{0a}[+\rangle q_a^+[w_{a,i}\rangle q_{a,i}^-[-\rangle q_{fa}$ in M_A^k is such that $q_{a,i}^- \in Q_A^-$. Similarly for B, for every $1 \leqslant i \leqslant n-1$, $p_{b,i} = q_{0b}[+\rangle q_b^+[w_{b,i}\rangle q_{b,i}^-[-\rangle q_{fb}$ is such that $q_{b,i}^- \in Q_B^-$. Thus, the set $q_{0a}[+\rangle q_a^+[w_{a,i}\rangle q_{a,i}^-[\epsilon\rangle q_b^+[w_{b,i}\rangle q_{b,i}^-[-\rangle q_{fb}$ is not empty and contains paths $\hat{p}$ in $M_A^k \circlearrowleft^{\pm} M_B^k$ such that $\pi_l(\hat{p}) = +w_{a,i}w_{b,i}-$. This can be continued to find a path p in $M_A^k \circlearrowleft^{\pm} M_B^k$ such that $\pi_l(p) = w$. And since $|w| \leqslant k$, then $w \in s^k(\mathcal{L}(M_A^k \circlearrowleft^{\pm} M_B^k))$.

Case 2: ($|w| > k$) Then for every $\gamma \in s^k(w)$, it holds:

$$\gamma \in \begin{cases} \mathit{suff}^k(+A)\hat{B}(\hat{A}\hat{B})^*\mathit{pref}^k(A-) \\ \mathit{suff}^k(+A)(\hat{B}\hat{A})^*\mathit{pref}^k(B) \\ \mathit{suff}^k(B)(\hat{A}\hat{B})^*\mathit{pref}^k(A-) \\ \mathit{suff}^k(B)\hat{A}(\hat{B}\hat{A})^*\mathit{pref}^k(B) \\ s^k(+A-) \\ s^k(B) \end{cases} \tag{1}$$

For the first 4 cases, it is possible to apply an argument similar to Case 1, observing that prefixes/suffixes of A and B lead to q_a^+/Q_A^- and q_b^+/Q_B^- (Proposition 1-6). For the fifth case, since that M_A^k is fully contained in $M_A^k \circlearrowleft^{\pm} M_B^k$, then $\gamma \in m^k(A) \subseteq m^k(\mathcal{L}(M_A^k \circlearrowleft^{\pm} M_B^k))$. For the sixth case, $+,- \notin \gamma$ which implies that there exist path $q_{0b}[\gamma^{1\to 1}\rangle \hat{q}_1[\gamma^{2\to k}\rangle q_{fb}$ in M_B^k. And so $q_{0a}[\gamma^{1\to 1}\rangle \hat{q}_1[\gamma^{2\to k}\rangle q_{fb} \neq \varnothing$ in the graph of $M_A^k \circlearrowleft^{\pm} M_B^k$.

($\supseteq$) We first show that every $w = w_1 w_2 \cdots w_n \in \mathcal{L}(M_A^k \circlearrowleft^{\pm} M_B^k)$ is a substring of $+A(BA)^*-$. For that, consider all accepting paths p in $M_A^k \circlearrowleft^{\pm} M_B^k$. If p only passes through edges in M_A^k, then $\pi_l(p) \in m^k(A) \subseteq m^k(A(BA)^*)$. Else, if $w_1 \in \Sigma_B$ and p does not pass through an ϵ edge, then $p \in q_{0a}[w_1\rangle \hat{q}[w^{2\to n}\rangle q_{fb}$ in $M_A^k \circlearrowleft^{\pm} M_B^k$ and so $q_{0b}[w_1\rangle \hat{q}[w^{2\to n}\rangle q_{fb}$ is a path in $M_B^k \Rightarrow w \in m^k(B)$. And since that p does not pass through an $+/-$-labeled edge (they were removed from M_B^k), then $|w| = k$, which implies $w \in m^k(A(BA)*)$. Else, if p passes through an ϵ edge, then one of the following holds:

$$p \in \begin{cases} q_{0a}[\gamma_{a_1}\rangle q_a^-[\epsilon\rangle q_b^+[\gamma_{b_1}\rangle q_b^-[\epsilon\rangle q_a^+[\gamma_{a_2}\rangle \cdots q_a^+[\gamma_{a_i}\rangle q_{fa} \\ q_{0a}[\gamma_{a_1}\rangle q_a^-[\epsilon\rangle q_b^+[\gamma_{b_1}\rangle q_b^-[\epsilon\rangle q_a^+[\gamma_{a_2}\rangle \cdots q_b^+[\gamma_{b_i}\rangle q_{fb} \\ q_{0a}[\gamma_{b_1}\rangle q_b^-[\epsilon\rangle q_a^+[\gamma_{a_1}\rangle q_a^-[\epsilon\rangle q_b^+[\gamma_{b_2}\rangle \cdots q_a^+[\gamma_{a_i}\rangle q_{fa} \\ q_{0a}[\gamma_{b_1}\rangle q_b^-[\epsilon\rangle q_a^+[\gamma_{a_1}\rangle q_a^-[\epsilon\rangle q_b^+[\gamma_{b_2}\rangle \cdots q_b^+[\gamma_{b_i}\rangle q_{fb} \end{cases} \tag{2}$$

We only prove the first case (the other cases are analogous). For this case, it holds that $\gamma_{a_1} \in \mathit{suff}^k(+\hat{A})$, $\gamma_{a_i} \in \mathit{pref}^k(\hat{A}-)$ and $\gamma_{a_j} \in \hat{A}\ \forall 1 < j < i$ and $\gamma_{b_j} \in \hat{B}\ \forall 1 \leqslant j < i$. This all implies that $w \in \mathit{suff}^k(+A)\hat{B}(\hat{A}\hat{B})^*\mathit{pref}^k(A-)$ (notice the correspondence to the first case of (1)) and that w is a substring of $+\hat{A}\hat{B}(\hat{A}\hat{B})^*\hat{A}- \subseteq +A(BA)^*-$. Now if $|w| = k$, then $w \in m^k(+A(BA)^*-)$. Else if $|w| < k$, then $|\gamma_{a_1}| < k-1 \Rightarrow |\gamma_{a_1} -| < k$ and since that $\gamma_{a_1}- \in m^k(A)$, then $+ \in \gamma_{a_1}$. Similarly, we derive that $- \in \gamma_{a_i}$ and thus $w \in +\hat{A}(\hat{B}\hat{A})^*- \Rightarrow w \in m^k(+A(BA)^*-)$. □

Lemma 5 shows that $s^k(\mathcal{L}(M_A^k \circlearrowleft^{\pm} M_B^k) = m^k(A(BA)*)$. But $M_A^k \odot^{\pm} M_B^k$ is an NFA and the algorithm from Lemma 1 requires a DFA as input. NFA determinization is worst-case exponential in its size. The following lemma shows that this does not happen for $M_A^k \circlearrowleft^{\pm} M_B^k$ due to its specific structure. The basic idea is that the ϵ transitions are the only source of non-determinism and that tokens of non-determinism "die" after at most k steps.

Lemma 6. ***(Determinizing the Markovian Loop Concatenation Does Not Explode)*** *Let A, B be arbitrary languages with disjoint alphabets Σ_A, Σ_B, and $M_A^k = (Q_A, \Sigma_A^{\pm}, \delta_A, q_{0a}, \{q_{fa}\})$ and $M_B^k = (Q_B, \Sigma_B^{\pm}, \delta_B, q_{0b}, \{q_{fb}\})$ the minimal DAFSAs accepting $m^k(A)$ and $m^k(B)$ respectively, $Q_{AB} = Q_A \cup Q_B$ and $\Sigma_{AB} = \Sigma_A \cup \Sigma_B$. Then runtime to determinize $M_A^k \circlearrowleft^{\pm} M_B^k$ is in $\mathcal{O}(k|Q_{AB}|^k)$.*

Proof. Start by noticing that all ϵ-edges in $M_A^k \circlearrowleft^{\pm} M_B^k$ lead to either q_a^+ or q_b^+. That means, that the ϵ-closure $\hat{S}$ of any state $S \subseteq Q_{AB}$ reached during the powerset construction is such that $\hat{S} \subseteq S \cup \{q_a^+, q_b^+\}$. Furthermore, the construction is such that for any reachable state $q \in Q_{AB}$, and $l_a \in \Sigma_A^{\pm}$,

if $|\hat{\delta}(q, l_a)| \neq \varnothing$, then $\hat{\delta}(q, l_a) \subseteq Q_A$ and similarly for every state $q \in Q_{AB}$ and $l_b \in \Sigma_B$, if $|\hat{\delta}(q, l_b)| \neq \varnothing$, then $\hat{\delta}(q, l_b) \subseteq Q_B$. This implies that every reachable state S in the powerset construction is such that either $S \subseteq Q_A$ or $S \subseteq Q_B$. Therefore, any path in the powerset construction is such that $\pi_s(p) = (S_{a_{1,1}} S_{a_{1,2}} \cdots S_{a_{1,n_1}})(S_{b_{1,1}} S_{b_{1,2}} \cdots S_{b_{1,m_1}})(S_{a_{2,1}} S_{a_{2,2}} \cdots S_{a_{2,n_2}}) \cdots$, where $S_{a_{i,j}} \subseteq \mathcal{P}(Q_A)$, $S_{b_{i,j}} \subseteq \mathcal{P}(Q_B)$, and $|S_{a_{i,1}}| = |S_{b_{i,1}}| = 1$.

Let $q_{a_{i,1}}$ and $q_{b_{i,1}}$ be the single elements of $S_{a_{i,1}}$ and $S_{b_{i,1}}$. Now consider the path $S_{a_{i,1}}[w_a\rangle S_{a_{i,n_i}}$, $w_a \in (\Sigma_A^{\pm})^*$ in the powerset construction. Then $S_{a_{i,n_i}} \subseteq q_{a_{i,1}}[w_a\rangle \cup \bigcup_{1<i\leqslant n} q_a^+[w_a^{i\to n}\rangle$ (in M_A^k). But since that M_A^k is a DAFSA with maximum word length k, then $q_a^+[w_a^{i\to n}\rangle = \varnothing$ if $|w_a^{i\to n}| \geqslant k$, which implies that $|S_{a,n}| \leqslant k$. A similar argument applies for $w_b \in \Sigma_B^*$. Therefore, the powerset construction expands at most $|Q_{AB}|^k$ nodes, with each node expansion costing at at most k, where $|Q_{AB}| \leqslant |\Sigma_A|^{k-1} + |\Sigma_A|^{k-1} + 2$ (Proposition 1-7). □

From Lemma 1, it follows that m^k of the loop node can be computed in $\mathcal{O}(k|\Sigma|^{k^2})$. This exponent seems very high at first, but in practice it does not happen. This is related to the fact that if one of the subtrees does not accept the empty word, then there is no real non-determinism in $M_A^k \circlearrowleft^{\pm} M_B^k$.

3.5 Parallel Node

Finally, we consider the parallel node. Parallel nodes largely contribute to the original method's inefficiency because they inevitably lead to an explosion in the state space's size. Before presenting the construction for the parallel node, we must define the parallel composition of two languages [5]:

Definition 11. ***(Parallel Composition)*** *Given languages $A \subseteq \Sigma_A^*, B \subseteq \Sigma_B^*$, the parallel composition $A \parallel B \subseteq (\Sigma_A \cup \Sigma_B)^*$ is such that:*

$$w \in A \parallel B \iff w_{\Sigma_A} \in A \ \wedge \ w_{\Sigma_B} \in B$$

The parallel composition is closely related to the shuffle product. In fact, if $\Sigma_A \cap \Sigma_B = \varnothing$, then $A \parallel B = A \sqcup\!\sqcup B$. Lemma 7 shows how to exploit this relation to compute m^k for parallel nodes:

Lemma 7. ***($\mathbf{m^k}$ of the Shuffle Product)*** *Let A, B be arbitrary languages such that $\Sigma_A \cap \Sigma_B = \varnothing$, and $\Sigma_{AB} = \Sigma_A \cup \Sigma_B$. Then:*

$$m^k(A \sqcup\!\sqcup B) = sub^k(m^k(A)) \parallel sub^k(m^k(B)) \parallel m^k(\Sigma_{AB})$$

which can be computed in $\mathcal{O}(|\Sigma_{AB}|^{2k})$.

Proof. Observe that $\Sigma_A \cap \Sigma_B = \varnothing \Rightarrow \ m^k(A \sqcup\!\sqcup B) = s^k(+(A \sqcup\!\sqcup B)-) = s^k(+A- \parallel +B-)$ and that $sub^k(m^k(A)) = sub^k(+A-)$ (Proposition 1-5).

($\subseteq$) Consider $w \in +A- \parallel +B-$. Then for every $\gamma \in s^k(w)$, it holds that $\gamma_{\Sigma_A^{\pm}} \sqsubseteq w_{\Sigma_A^{\pm}}$. And since $w_{\Sigma_A^{\pm}} \in +A-$ and $|\gamma| \leqslant k$, then $\gamma_{\Sigma_A^{\pm}} \in sub^k(+A-)$.

Analogously, $\gamma_{\Sigma_B^\pm} \in sub^k(+B-)$. Finally, since $\gamma = \gamma_{\Sigma_{AB}^\pm}$ and $\gamma \in s^k(+(A \sqcup\!\sqcup B)-) \subseteq m^k(\Sigma_{AB}^*)$, then $\gamma \in sub^k(+A-) \parallel sub^k(+B-) \parallel m^k(\Sigma_{AB}^*)$.

($\supseteq$) For every $\gamma \in sub^k(+A-) \parallel sub^k(+B-) \parallel m^k(\Sigma_{AB}^*)$, there exists $w_a = \alpha_a \gamma_{\Sigma_A^\pm} \beta_a \in +A-$ and $w_b = \alpha_b \gamma_{\Sigma_B^\pm} \beta_b \in +B-$. Notice that $+$ is only present at most once in γ and $\gamma \in \gamma_{\Sigma_A^\pm} \parallel \gamma_{\Sigma_B^\pm}$, therefore $+ \in \alpha_a \iff + \in \alpha_b$. Similarly, $- \in \beta_a \iff - \in \beta_b$. Also notice that $+ \notin \alpha_a, \alpha_b$ implies $\alpha_a = \alpha_b = \epsilon$ and $- \notin \beta_a, \beta_b$ implies $\beta_a = \beta_b = \epsilon$.

If $+ \in \alpha_a$, then $+ \in \alpha_b \Rightarrow \alpha_a = +\hat{\alpha}_a$, $\alpha_b = +\hat{\alpha}_b$ and we define $\alpha = +\hat{\alpha}_a\hat{\alpha}_b$. Else, if $+ \notin \alpha_a$ then $\alpha = \epsilon$. Similarly, we define $\beta = \hat{\beta}_a\hat{\beta}_b-$ or $\beta = \epsilon$. In all cases, $\alpha\gamma\beta \in +A- \parallel +B- \Rightarrow s^k(\alpha\gamma\beta) \subseteq s^k(+A- \parallel +B-)$. Notice that if $|\gamma| = k \Rightarrow \gamma \in s^k(\alpha\beta\gamma)$. And that if $|\gamma| < k$, then $\alpha = \beta = \epsilon$ (since that $\gamma \in m^k(\Sigma_{AB}^*)$), $\Rightarrow \gamma \in s^k(\alpha\gamma\beta)$. Putting it together, $\gamma \in s^k(\alpha\gamma\beta) \subseteq s^k(+A- \parallel +B-)$.

For the runtime bound, notice that $sub^k(m^k(A))$ and $sub^k(m^k(B))$ can be computed in $\mathcal{O}(|\Sigma_{AB}|^{2k})$ and that the computation of the network automaton [5] associated to $sub^k(m^k(A)) \parallel sub^k(m^k(B)) \parallel m^k(\Sigma_{AB}^*)$, expands at most $|\Sigma_{AB}|^k$ states. □

Total Runtime Boundary As shown above, computing m^k for each tree node is in $\mathcal{O}(k|\Sigma|^{k^2})$. For a process tree T containing n operator nodes, the runtime to compute $m^k(T)$ is in $\mathcal{O}(kn|\Sigma|^{k^2})$. Oftentimes, n is linear with $|\Sigma|$.

3.6 Handling Arbitrary Process Trees

The previous sections have shown how to compute m^k for binary process trees with unique visible label nodes. Notice that any process tree can be transformed into a binary tree accepting the same language (hence having the same m^k). For trees with repeated labels, the results below show that it suffices to first map each visible label node in the tree T to a unique label, and then map $m^k(\mathcal{L}(T))$ back to the original labels.

Lemma 8. ***($\mathbf{m^k}$ of a Remapped Language)*** *Let $A \subseteq \Sigma_A^*$ and $B \subseteq \Sigma_B^*$ be arbitrary languages and $\lambda : \Sigma_B \to \Sigma_A$ s.t. $A = \lambda(B)$, then $s^k(A) = \lambda(s^k(B))$.*

Proof. Notice that for all w_a, w_b such that $w_a = \lambda(w_b)$, then $\forall_{i \leqslant j}\ w_a^{i \to j} \in s^k(w_a) \iff w_b^{i \to j} \in s^k(w_b)$.

($\subseteq$) For any $w_a \in A$, there exists $w_b \in B$ s.t. $w_a = \lambda(w_b)$. For all $\gamma_a \in s^k(w_a)$, $\gamma_a = w^{i \to j}$ for some $i \leqslant j$. Thus, $\gamma_a = \lambda(w_b^{i \to j})$ and since $|w_a| = |w_b|$, then $w_b^{i \to j} \in s^k(w_b) \Rightarrow s^k(w_a) \subseteq \lambda(s^k(w_b)) \subseteq \lambda(s^k(B))$.

($\supseteq$) For any $w_b \in B$, there exists $w_a \in A$ s.t. $w_a = \lambda(w_b)$. For all $\gamma_b \in s^k(w_b)$, it holds that $\gamma_b = w_b^{i \to j}$ for some $i \leqslant j$. Thus, $\lambda(\gamma_b) = w_a^{i \to j}$ and since $|w_a| = |w_b|$, then $w_a^{i \to j} \in s^k(w_a) \Rightarrow s^k(A) \supseteq s^k(w_a) \supseteq \lambda(s^k(w_b))$. □

The result above can be extended to m^k by defining $\lambda^\pm : \Sigma_B^\pm \to \Sigma_A^\pm$ such that $\lambda^\pm(l) = \begin{cases} \lambda(l) & l \in \Sigma_B \\ l & l \in \{+,-\} \end{cases}$. Then $+A- = \lambda^\pm(+B-) \Rightarrow m^k(A) = \lambda^\pm(m^k(B))$. This mapping function can always be constructed for a process tree as follows:

Lemma 9. ***(m^k of Arbitrary Process Trees)*** *For an arbitrary process tree T_A with alphabet Σ_A and visible label nodes $N = \{n_1, \cdots n_i\}$. Given an alphabet Σ_B such that $|\Sigma_B| = k$ and a bijective mapping $r : N \to \Sigma_B$ defining a map $\lambda : \Sigma_B \to \Sigma_A$ as $\lambda(b) = \mathcal{L}(r^{-1}(b))$, then the process tree $\hat{T}$ obtained by relabeling each visible node n of T with $r(n)$ is such that $\mathcal{L}(T) = \lambda(\mathcal{L}(\hat{T}))$.*

Proof. It follows directly from Definition 4 by noticing that $\lambda(AB) = \lambda(A)\lambda(B)$ and $\lambda(A \sqcup\!\sqcup B) = \lambda(A) \sqcup\!\sqcup \lambda(B)$. □

The results from this Section show that it is possible to compute m^k for arbitrary process trees in polynomial time. It is also possible to compute m^k for event logs in linear time. Thus, the markovian conformance metrics (Definition 7) can also be computed in polynomial time.

4 Experimental Evaluation

This section compares the proposed method with the previous approach described in [2] and state-of-the-art techniques in terms of runtime and the induced metrics. For a fair comparison, all techniques are implemented in pure Python[3].

4.1 Effect of Parallelism

The first experiment measures the influence of parallelism in the runtime. For that, we generate artificial process trees with a fixed number of activities (30) and varying degrees of parallelism (0.2 to 0.5). For each configuration, 50 process trees are generated. For each tree, an event log consisting of 2000 distinct variants is sampled and a small amount of noise is injected into the logs by adding, removing, and swapping activities.

We compare the runtime to compute three types of conformance artifacts: trace alignment (align), the model and log projections required by the PCC framework (PCC), and the markovian abstraction. For the latter two metrics, we vary their k parameter, indicating the projection size and substring size respectively, from 2 to 4 and break down the runtime for each method by the time taken to process the log and the model. For the markovian abstraction, we compare the method originally presented in [2] (m^k-orig) and the proposed method (m^k-opt). For each experiment run, we set a timeout of 20 minutes.

The results are summarized in Table 1. Trace alignment is by far the slowest method, with an average execution time of over five minutes and multiple timeouts. For comparison, none of the other methods timed out. PCC is arguably the second-slowest method, being the slowest in all but two scenarios. m^k-opt is the fastest method in all scenarios.

For models with little parallelism, m^k-orig and m^k-opt perform similarly well, with m^k-opt being slightly faster. This is explained by the fact that these

[3]The datasets and experiment results can be found at: github.com/EduardoGoulart1/efficient-mk/

Table 1: The effect of parallelism on the runtime required to compute conformance artifacts, broken down by log and model processing times (if applicable). The number of timeouts (if any) is indicated in parenthesis.

k	Method	Time(s) [Log | Model] par=0.2		par=0.3		par=0.4		par=0.5	
	align	343.4 (14)		368.4 (10)		454.2 (10)		439.9 (18)	
2	PCC	0.262	0.054	0.228	0.050	0.310	0.051	0.258	0.048
	m^k-orig	0.009	0.108	0.009	0.320	0.010	0.564	0.010	1.465
	m^k-opt		0.029		0.034		0.047		0.051
3	PCC	4.376	0.507	3.424	0.476	5.442	0.487	3.976	0.445
	m^k-orig	0.010	0.358	0.009	0.895	0.011	1.727	0.010	4.127
	m^k-opt		0.143		0.241		0.354		0.438
4	PCC	41.704	3.388	36.459	3.133	36.795	3.207	37.661	2.909
	m^k-orig	0.011	2.005	0.010	4.327	0.012	8.751	0.011	20.408
	m^k-opt		0.925		1.503		2.680		3.663

models have a small and linear state-space. However, increasing the amount of parallelism from 0.2 to 0.5 causes a tenfold increase in the runtime for m^k-orig, while for m^k-opt it increases by a factor of at most 4, to which we conclude that m^k-opt can better handle large models. In comparison, PCC is unaffected by the degree of parallelism. Instead, its runtime is dominated by the event log projections.

In general, the experiment shows the shortcomings of trace alignment and the PCC framework in terms of runtime, especially considering large event logs. It also shows that m^k-orig struggles to process large models. m^k-opt emerged as the clear winner in terms of performance. For event logs, m^k-opt can be up to 400 times faster than PCC. At the same time, computing m^k for process models takes roughly the same time as computing the tree projections.

4.2 Real Datasets

Next, we evaluate the markovian-based conformance metrics on two real-world datasets: the Italian Road Fines event log, and the BPI Challenge 2015 event log (BPIC-15). We filter the BPIC-15 log for the municipality 1, subprocess 8, and remove repeated activities. This preprocessing is needed as otherwise the used process discovery methods would only return flower constructs. For each event log, we mine four process trees with the Inductive Miner infrequent variant with noise thresholds of 02 and 05 (IMf02 and IMf05 respectively), the Inductive Miner incomplete variant (IMc) and the flower miner. We use alignment-based trace fitness [1] (AL) as the ground-truth fitness measure and escaping edges precision [9] (ETC) as the ground truth precision measure. We vary the respective k parameter of PCC, MAF, and MAP from 2 to 4. The results are summarized in Table 2.

The first thing to notice is that the basic property that language inclusion implies fitness of 1.0 is fulfilled by metrics for the IMc and Flower models for both datasets. Next, for both datasets, PCC and MAF generate the same fitness rankings as the ground truth alignment-based fitness measure (AL) for all k-s. As k increases, the difference in fitness between models IMf02 and IMf05 increases.

For the Road Fines datasets, all metrics induce different precision rankings. ETC is assigning a higher precision to the flower model than to the IMc model. For $k = 2, 3$, PCC and MAP agree on their rankings, but assign IMc as being more precise than IMf02. This is counter-intuitive since that the IMc model has a lot more parallelism and self-loops. For $k = 4$, PCC even assigns IMc as the most precise model. For the BPIC 2015 datasets, all metrics agree on the model rankings. However, the PCC metric will assign a relatively high precision for models such as IMc and the Flower model.

In summary, the experiment shows that MAF and MAP induce similar fitness and precision rankings as other state-of-the-art techniques. Notice that for both datasets, as k increases, MAP tends towards zero. This is expected from the definition of MAP, which does not consider any notion of substring frequency.

Table 2: Quality evaluation of fitness and precision metrics.

	Miner	Road Fines				BPIC 2015			
		IMf02	IMf05	IMc	Flower	IMf02	IMf05	IMc	Flower
	AL	0.982	0.784	1.0	1.0	0.899	0.773	1.0	1.0
Fitness	PCC2	0.986	0.857	1.0	1.0	0.985	0.962	1.0	1.0
Fitness	PCC3	0.976	0.755	1.0	1.0	0.977	0.937	1.0	1.0
Fitness	PCC4	0.967	0.664	1.0	1.0	0.968	0.912	1.0	1.0
Fitness	MAF2	0.965	0.953	1.0	1.0	0.881	0.737	1.0	1.0
Fitness	MAF3	0.936	0.826	1.0	1.0	0.833	0.475	1.0	1.0
Fitness	MAF4	0.899	0.745	1.0	1.0	0.805	0.422	1.0	1.0
	ETC	0.895	0.653	0.318	0.325	0.497	0.817	0.261	0.127
Precision	PCC2	0.946	0.949	0.931	0.593	0.735	0.924	0.660	0.635
Precision	PCC3	0.814	0.838	0.831	0.497	0.624	0.824	0.551	0.534
Precision	PCC4	0.658	0.703	0.718	0.423	0.529	0.722	0.462	0.451
Precision	MAP2	0.735	0.949	0.830	0.542	0.549	0.729	0.316	0.225
Precision	MAP3	0.277	0.389	0.353	0.134	0.115	0.411	0.047	0.020
Precision	MAP4	0.082	0.122	0.106	0.020	0.015	0.199	0.005	0.001

5 Related Work

Conformance checking is the field of process mining focused on comparing a process' desired to its observed behavior. The process model describes the desired behavior. It is often encoded as a Petri net or any equivalent model with execution semantics (YAWL, Process Trees, etc.).

Conformance-checking is especially challenging because computing the process model's behavior has often worst-case exponential time due to the state explosion problem. Hence, most state-of-the-art methods such as token-based replay [11], alignments [1], entropia [10], or Earth mover's distance [8] have worst-case exponential time. This also includes the original method for computing markovian-based conformance metrics presented in [2].

A notable exception is the *Projected Conformance Checking* framework (PCC framework) [7] which uses projections on subsets of activities to significantly alleviate the state explosion problem. In fact, for certain classes of process trees the runtime is polynomial. Nevertheless, PCC requires multiple passes over the event log, which is impractical for large production datasets, as shown in Section 4.

The idea of exploiting the tree structure to speed up computations is not new. In [12] a method is presented to approximate alignments by constructing an equivalent optimization problem from the tree structure. In [14], a method is presented to repair alignments for iterative scenarios, for the use-case where alignments need to be computed for similar process trees. Our work differs from them in which we provide a speed up in computation time without the need to approximate. Finally, in [15] a method is presented to compute trace probabilities by transforming the tree into a probabilistic context-free grammar, this transformation is only possible because of the process tree' structure.

Last, sampling techniques [6] can be orthogonally applied to any conformance method, including our technique. However, sampling only provides a linear speed-up and previously exponential techniques will remain exponential. In production settings, where controllable runtime is important, exponential factors are rarely a good idea.

6 Conclusion

This paper provides two important contributions. First, we presented an alternative definition of the markovian abstraction that can be more easily manipulated using techniques from automata theory. Next, we showed how to exploit the tree-structure of process trees to perform polynomial-time conformance checking with guarantees. The experimental evaluation shows an improvement of multiple orders of magnitude in the runtime compared to the original approach presented in [2] and other state-of-the-art conformance checking techniques, while at the same time still generating similar fitness and precision rankings. Most importantly, the runtime of the approach is bounded by a polynomial, making it more controllable.

As future work, we plan to apply the proposed technique to optimization-based discovery techniques such as the evolutionary tree miner [3], which requires repetitive computation of conformance metrics. We also plan to explore the stochastic perspective, by computing the probability of each substring in the process tree's language, to address the problem of vanishing precision values for MAP.

Acknowledgements We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

1. Adriansyah, A., van Dongen, B., van der Aalst, W.: Conformance checking using cost-based fitness analysis. In: 2011 IEEE 15th International Enterprise Distributed Object Computing Conference. pp. 55–64 (2011)
2. Augusto, A., Armas-Cervantes, A., Conforti, R., Dumas, M., Rosa, M.L.: Measuring fitness and precision of automatically discovered process models: A principled and scalable approach. IEEE Transactions on Knowledge and Data Engineering **34**(4), 1870–1888 (2022)
3. Buijs, J.: Flexible evolutionary algorithms for mining structured process models. Technische Universiteit Eindhoven **57** (2014)
4. Daciuk, J., Mihov, S., Watson, B.W., Watson, R.E.: Incremental construction of minimal acyclic finite-state automata. Computational linguistics **26**(1), 3–16 (2000)
5. Esparza, J., Blondin, M.: Automata Theory: An Algorithmic Approach. MIT Press (2023)
6. Kabierski, M., van der Aa, H., Weidlich, M.: Estimating process conformance by trace sampling and result approximation (09 2019)
7. Leemans, S.J.J., Fahland, D., van der Aalst, W.: Scalable process discovery and conformance checking. Software and Systems Modeling **17**, 599 – 631 (2016)
8. Leemans, S.J., van der Aalst, W.W., Brockhoff, T., Polyvyanyy, A.: Stochastic process mining: Earth movers' stochastic conformance. Information Systems **102**, 101724 (2021)
9. Munoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. vol. 6336, pp. 211–226 (09 2010)
10. Polyvyanyy, A., Solti, A., Weidlich, M., Ciccio, C.D., Mendling, J.: Monotone precision and recall measures for comparing executions and specifications of dynamic systems. ACM Trans. Softw. Eng. Methodol. **29**(3) (jun 2020)
11. Rozinat, A., van der Aalst, W.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**(1), 64–95 (2008)
12. Schuster, D., van Zelst, S., van der Aalst, W.W.: Alignment Approximation for Process Trees, pp. 247–259 (03 2021)
13. Syring, A.F., Tax, N., van der Aalst, W.W.: Evaluating conformance measures in process mining using conformance propositions. Transactions on Petri Nets and Other Models of Concurrency XIV pp. 192–221 (2019)
14. Vázquez-Barreiros, B., van Zelst, S., Buijs, J., Lama, M., Mucientes, M.: Repairing alignments: Striking the right nerve. pp. 266–281 (06 2016)
15. Watanabe, A., Takahashi, Y., Ikeuchi, H., Matsuda, K.: Grammar-based process model representation for probabilistic conformance checking. In: 2022 4th International Conference on Process Mining (ICPM). pp. 88–95. IEEE (2022)