Universiteit Leiden
The Netherlands

Opleiding Informatica

Hyper-minimisation of

Probabilistic Deterministic Finite Automata

Thomas Vroegop

Supervisors:
Marcello M. Bonsangue & Tobias Kappé

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

26/08/2025

Abstract

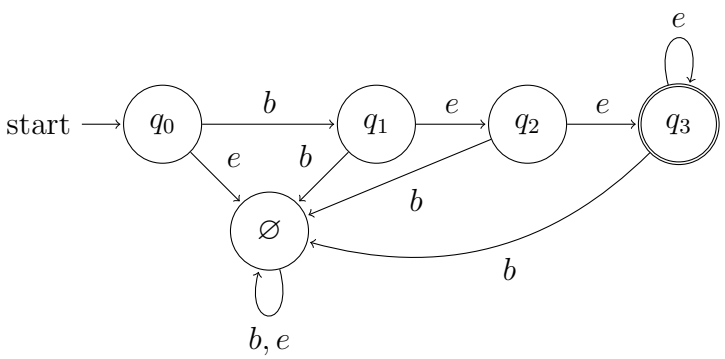
Hyper-minimisation of deterministic finite automata (DFAs) and weighted deterministic finite automata (WDFA) is a lossy compression technique that allows for finite error in the recognised language. The algorithm for hyper-minimisation of DFAs and WDFAs is first described. Transferring the almost-equivalence characterisations for the weighted setting of WDFAs to the probabilistic setting of PDFAs shows that the structure of the known hyper-minimisation algorithm for WDFAs can largely be applied to PDFAs. Analysis on the application of hyper-minimisation on PDFAs shows that the stochastic constraint on states is a delimiting factor and forms illegal state merges. These illegal merges are then partially alleviated and characterised through a probability redistribution. Lastly, a merging algorithm in $O(n^2)$ is described with the inclusion of the necessary probability redistributions.

Contents

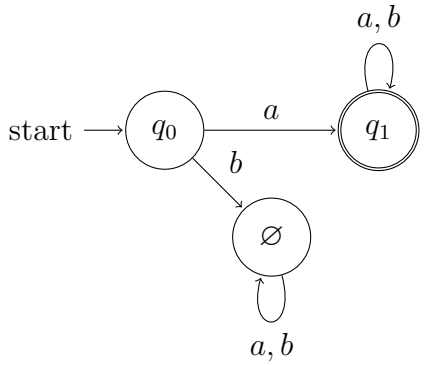
1 Introduction

The state-space explosion problem [GKO15] of finite automata (FA) [HMU07, p. 1] is a critical obstacle in their practical application and verification, calling for effective compression algorithms to reduce this state-space. State-reduction through minimisation is a well-known lossless compression technique for various types of automata [BBCF24, KW14], and is proven to produce the smallest, equivalent finite automaton [BGS09]. Greater potential reduction in state-space comes in lossy compression, like hyper-minimisation [BGS09, Mal11, MQ12], which permits a finite number of errors in the recognised language between the original and compressed automaton.

An FA is used to describe a generalised idea of languages [Mar11, p. 45] using states and transitions. It sequentially reads an input string symbol by symbol until it has no more symbols to read or until the automaton 'crashes'. Once it is finished with reading the input string it will give an output. For a simple FA it will return whether the input string is or is not in the language that the automaton describes. Other types of automata may exhibit different behaviour for its output and/or for reading a string. These automata are considered to be generalisations of the deterministic finite automata (DFA). Examples of generalisations are non-deterministic finite automata, weighted finite automata, tree automata [Bra68] and probabilistic finite automata [Rab63, KW14]. This paper will build onto research into deterministic finite automata and weighted deterministic finite automata (WDFA) by appending prior research to probabilistic deterministic finite automata (PDFA).



(a) DFA M_1 for the language where 'bee' can end with any extra e's.



(b) DFA M_2 for the language where all strings start with 'a'.

Figure 1: Examples of deterministic finite automata for Section Preliminaries.

The interest in PDFAs comes in their many practical purposes such as speech recognition [RPB96, PR96], image compression [CIK93, KW14] and natural language processing [dlHO13]. Furthermore, the probabilistic finite automaton is another notation for Markov chains [PBW10], which sees use in fields including statistics [DMPS18, p. 5] and machine learning within bioinformatics (hidden Markov models) [MVV+18].

In practical use, automata (including PDFAs) can have millions of states [MQ12]. In order to make these computational models more efficient it is of interest to have an automaton describe its language with the least amount of states possible. A reduction in the amount of states leads to less storage being required to store the automaton [KW14]. It may also improve the run-time of its computations due by requiring less computing memory. Algorithms that are performed on the automaton itself, such as verification algorithms [PZ93] may also see a reduced run-time.

Early research into FAs produced minimisation algorithms that can effectively reduce the amount of states the automaton uses, with the language they describe being *equivalent*. These algorithms take a given automaton and return a new automaton with the *minimal* amount of states. Such state-reduction algorithms exist for DFAs and non-deterministic FAs [BDN12, BBCF24]. A study for the minimisation of (non-deterministic) probabilistic finite automata adopts Brzozowski's algorithm [KW14]. It has been proven that such algorithms obtain the *minimal* automaton. Meaning, it is the smallest the automaton can get that describes exactly the original language [BGS09]. The minimal automaton is unique [BBCF24]. Because the minimal automaton cannot be compressed further, researchers had to look further to reduce an automaton's size. A development with greater state-reduction than minimisation with equivalence is that of hyper-minimisation [BGS09]. Another development specific to probabilistic finite automata is that of minimisation with an accepted error [KW14]. We can compare standard minimisation and these two developments in minimisation as the difference of lossless compression and lossy compression, respectively.

The lossless compression of an automaton requires the language it recognises to be equivalent to the initial language, as seen in classic minimisation. In contrast, lossy state-reduction techniques consider a certain deviation in its recognised language permissible, so long the practical impact on its recognised language is considered negligible. The benefit of not being held by strict equivalence is that the automaton can be further reduced in size, thus optimising the computational and storage efficiency. In particular, hyper-minimisation permits a deviation for a finite amount of accepted strings [BGS09]. For this the *almost-equivalence* relation was introduced. Two automata are almost-equivalent if the language they recognise differ for finite strings [BGS09]. Hyper-minimisation uses this relation as its state-reduction constraint such that the resulting automaton largely preserves the language it recognises, with a finite deviation in the strings it accepts. In the probabilistic setting of PDFAs, hyper-minimisation would largely preserve the accuracy of prediction and correctness of acceptance, whereas this can not be guaranteed for a finite amount of strings. This deviation can be acceptable in the probabilistic setting, where predictions do not necessitate exact precision. Then, hyper-minimisation may be an effective technique to further compress PDFAs without significantly degrading their predictive accuracy. This is particularly interesting when working with noisy data models, where a finite set of noisy data may be removed through hyper-minimisation.

The concept of hyper-minimisation originates from a paper which gave a state-reduction algorithm for DFAs. Further research by A. Malleti and D. Quernheim expanded hyper-minimisation to WDFAs [MQ12]. This algorithm was shown to work for WDFAs, but not for the unique weighted setting of PDFAs. This bachelor thesis for LIACS, supervised by M. Bonsangue and T. Kappé, builds onto the algorithm for the weighted setting of WDFAs by translating it to the probabilistic setting of PDFAs. We have the following research question:

What is a hyper-minimisation algorithm for probabilistic deterministic finite automata, such that the language described by the automata it almost-equivalent to the original?

For this study we limit ourselves to deterministic variant of probabilistic finite automata as it has easy to work with algorithmic properties that previous research into hyper-minimisation took advantage of.

1.1 Thesis overview

The research question is approached by first giving the reader the necessary background information. This is followed by an explanation of the necessary components from prior work in hyper-minimisation, and an analyse in how that is applicable to PDFAs. From this we find that the properties of PDFAs cause issues that restrict their hyper-minimisation. The paper will make an attempt at alleviating these issues with an updated algorithm.

The thesis is divided into the following sections: Section Preliminaries explains important concepts, gives formal definitions and fixes notation; Section Hyper-minimisation discusses related research into hyper-minimisation algorithms and applies this to PDFAs; Section Examples gives examples of hyper-minimisation for PDFAs and gives a counter-example where hyper-minimisation fails; Section Merging algorithm makes an attempt at a merging algorithm that works for PDFAs; Section Conclusions and further research concludes the paper and discusses further research.

2 Preliminaries

Before we speak on the new theory related to hyper-minimisation we first look to explain and define background theory related to languages, automata and equivalence relations, and we fix the notation for this theory. We start with simple definitions for sets.

A **set** S is a finite or infinite collection of elements. The set of **natural numbers** is $\mathbb{N} = \{0, 1, 2, ...\}$. The set of real numbers between 0 and 1 (inclusive) is $[0, 1] \in \{k \in \mathbb{R} \mid 0 \leq k \leq 1\}$. The set of positive real numbers is $\mathbb{R}_{>0}$. $|S|$ is the notation for the **cardinality** of S. The **difference** in elements between two sets S_1, S_2 is the operator Δ with notation $S_1 \ \Delta \ S_2$.

A **language** L is a set of strings over an alphabet. An **alphabet** Σ is a finite set of symbols. A **string** s of length l is a concatenation of symbols over the alphabet: $\sigma_0...\sigma_l$ where $l \in \mathbb{N}$ and $\sigma_i \in \Sigma$ for $0 \leq i \leq l$. The string of length zero is the **empty string**: ε. The **empty language** contains zero strings. The set of all strings over the alphabet is noted as $s \in \Sigma^*$. This notation uses a **Kleene star** $*$. This is an operator on a set such that we obtain an infinite set comprising all strings obtained by concatenating elements of the set onto the empty string zero or more times, in any order [Mar11, p. 19].

An example of a language is the following: $L = \{x \in \{a, b\}^* \mid \text{x starts with a}\}$ (see DFA in Figure 1b). Examples of strings in this language are: $a, aa, ab, aab, aba, etc.$ We notate some element x being in a set S as $x \in S$. The notation for our language L reads as follows: L is the set of all strings over the alphabet consisting of only a's and b's, where all strings x must start with a.

Regular languages (a subset of all languages) can be described by DFAs [Kim19]. A DFA consists of 5 key components: the states, alphabet, transitions between states, initial state and accepting states.

Formally, a **deterministic finite automaton** M is a tuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is non-empty finite set of states, Σ a finite alphabet of input symbols, δ a transition map $Q \times \Sigma \to Q$, $q_0 \in Q$ is the initial state and $F \subseteq Q$ the set of accepting states.

The **initial state** is where the DFA starts its execution. All states, including the initial state, can be **accepting states**. All other states are said to be not accepting. If the DFA's current state is an accepting state after being finished with reading an input string, then we *accept* the input

string as part of the language. If the current state is not accepting, then we do not *accept* the input string. The **garbage state** is a special non-accepting state in Q which can never be exited, no matter which input symbol we read.

A **transition map** is a function that receives the current state and current input symbol, and returns the next state. E.g., $\delta(q, \sigma) = p$ where $\sigma \in \Sigma$ and $q, p \in Q$.

The **transition mapping** δ can be extended to work for strings: $Q \times \Sigma^* \to Q$. Define $\delta(q, \varepsilon) = q$ and $\delta(q, \sigma s) = \delta(\delta(q, \sigma), s)$ for all $q \in Q$, $\sigma \in \Sigma$ and $s \in \Sigma^*$. In other words, for an empty input the DFA will stay in the same state, and for a string of symbols the DFA repeatedly reads the first symbol of the string and follows its corresponding transition to the next state. The route of states that an input string takes is called the **path**. A DFA M recognises the language $L(M) = \{s \in \Sigma^* \mid \delta(s, q_0) \in F\}$. We say a string s is **accepted** by M if $s \in L(M)$.

An FA is called **deterministic** if for each distinct pair $Q \times \Sigma$ the mapping δ has a transition to precisely one state. A DFA has exactly one initial state. The result of this is that for each input string we know exactly which state we end up, because there is only one path the string may follow.

In Figure 1 we see two DFAs M_1 and M_2. The initial state and garbage state for both are q_0 and $\varnothing$, respectively. The accepting states for M_1 is q_3 and for M_2 it is q_1. For better readability of state diagrams, some DFAs in this paper do not have their garbage state (if it exists) and its incoming transitions drawn, but these are implied.

For DFAs we can define equivalence relations that are used for the definitions of minimality and hyper-minimality. For that we need some definitions for states.

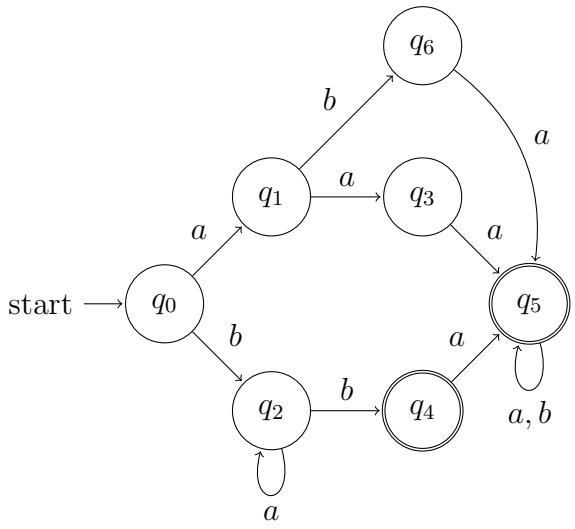
A state p is called **reachable** from some state q if there exists a path from q to p. That is, there exists a string $s \in \Sigma^*$ such that $\delta(q, s) = p$. A state is called **accessible** if it is reachable from an initial state.

The **left language** of a state $q \in Q$ is the set $\overleftarrow{q}$ of all strings that reach q from the initial state: $\overleftarrow{q} = \{s \in \Sigma^* \mid \delta(q_0, s) = q\}$ [BBCF24, p. 4] where q_0 is the initial state. The **right language** of a state $q \in Q$ is the set $\overrightarrow{q}$ of all strings that reach an accepting state starting from state q: $\overrightarrow{q} = \{s \in \Sigma^* \mid \delta(q, s) \in F\}$ [BBCF24, p. 4].

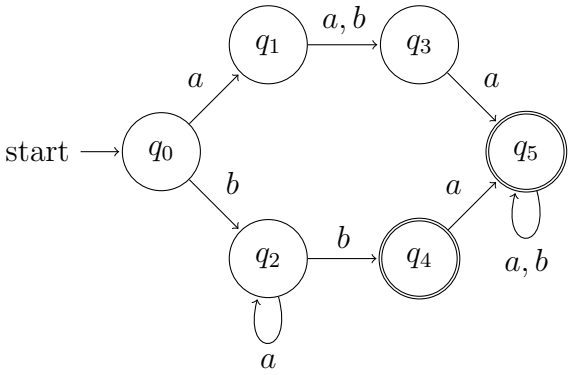
For the notion of minimality we need certain definitions **equivalence** relations. Two languages $L_1, L_2 \in \Sigma^*$ are equal if their elements coincide: $L_1 = L_2$. Two DFAs M_1 and M_2 are equivalent if the languages they recognise are equal: $M_1 \equiv M_2$ if and only if $L(M_1) = L(M_2)$ [Mal11]. Two states $q, p \in Q$ are equivalent if their right languages are equal: $p \equiv q$ if and only if $\overrightarrow{q} = \overrightarrow{p}$. A DFA is **minimal** if no other DFA with fewer states exists that is equivalent. Furthermore, we state that a DFA is minimal if it has no equivalent states [BBCF24, p. 4]. All states of a minimal automaton must be accessible.

For an example for minimisation of DFAs, see Figure 2. Here was can see that states q_3 and q_6 are equivalent because their right-languages are equal. These are $\overrightarrow{q_3} = \{a, aa, ab, aaa, aab, ...\} = \overrightarrow{q_6}$. We can derive a minimal automaton by merging these two states into one state.

Before we get to the definition of PDFAs we need some definitions for vectors and probabilities. For $n \in \mathbb{N}$ a **vector** $\alpha \in \mathbb{R}^n$ is of n entries over the set of real numbers. For some set S we write $\alpha \in \mathbb{R}^S$ a vector for elements of S over the set of real numbers. We denote vector entries by $\alpha[i]$ where $\alpha[x]$ for $1 \leq i \leq n$ and $x \in S$, respectively. A **probability** ρ is real number between 0 and 1: $\rho \in [0, 1]$, though in practical applications we may also define it for rational numbers: $\rho \in (\mathbb{Q} \cap [0, 1])$.



(a) DFA M_1 that is not minimal. q_3 and q_6 are equivalent states.



(b) DFA M_2 is the minimal automaton for $L(M_1)$. q_3 and q_4 are almost-equivalent states.

Figure 2: Example of a DFA that can be minimised.

A **probabilistic language** is a generalisation of weighted languages. A **weighted language** φ is a mapping that assigns to each word a **weight** in some set $\mathbb{K}$: $\varphi : \Sigma^* \to \mathbb{K}$. A probabilistic language sets this weight to the probabilistic setting, such that $\varphi : \Sigma^* \to [0, 1]$ and such that φ is a (discrete) probability distribution $\sum_{s \in \Sigma^*} \varphi(s) = 1$ [Chu24, p. 14]. A probabilistic language allows for the representation of uncertainty or to describe stochastic processes [Chu24, p. 14].

Probabilistic languages can be described by probabilistic finite automata. This is accomplished by calculating the probability of an input string over its path. The generalisation to *deterministic* probabilistic finite automata is less powerful in that it can recognises a subset of languages recognised by probabilistic finite automata [Chu24, p. 19].

Formally, a **probabilistic deterministic finite automaton** M is a tuple $M = (Q, \Sigma, \delta, P, q_0, F)$ [Chu24] where Q, Σ, δ and q_0 follow the definition for DFAs. P is a distribution $Q \times \Sigma \to (\mathbb{Q} \cap [0, 1])$ that coincides with δ such that each transition for a pair $(q \in Q, \sigma \in \Sigma)$ is attributed a probability and a next state. F is a vector $(\mathbb{Q} \cap [0, 1])^Q$ of final probabilities for each state. For all states $q \in Q$ the **reduced stochastic property** must hold: $\sum_{\sigma \in \Sigma} P(q, \sigma) + F[q] \leq 1$.

The **transition mapping** δ and **probability distribution** P can be extended to strings. For δ this follows the definition as for DFAs. For P we define: $P(q, \varepsilon) = 1$ and $P(\sigma s) = P(q, \sigma) \cdot P(\delta(q, \sigma), s)$ for all $q \in Q, \sigma \in \Sigma$ and $s \in \Sigma^*$.

The **probabilistic language** accepted by a PDFA M is a distribution $\varphi_M : \Sigma^* \to [0, 1]$ where $\varphi_M(s) = P(q_0, s) \cdot F[\delta(q_0, s)]$ for some $s \in \Sigma^*$. A string s is said to be accepted by M if $\varphi_M(s) > 0$.

A state $q \in Q$ is called **accepting** if $F[q] > 0$. Else, q is not accepting. A **garbage state** $\varnothing \in Q$ is a state with $F[\varnothing] = 0$ and $\delta(\varnothing, \sigma) = \varnothing$ for all $\sigma \in \Sigma$ such that no strings can ever be accepted once entering $\varnothing$.

Just like for a DFA, the **deterministic** property of the PDFA means that for each pair of $Q \times \Sigma$ there is at most one transition in δ (with coinciding weight), and that there is at most one initial state.

This paper considers the reduced stochastic property for states, as opposed to the **stochastic property**: $\sum_{\sigma \in \Sigma} P(q, \sigma) + F[q] = 1$. The choice for the reduced property is that it will be easier to work with for the merging algorithm of Section 4. Unlike for the classic stochastic property, a language recognised by a PDFA with the reduced stochastic property is not necessarily a probability distribution, even if all states are accessible and can reach an accepting state. However, with a simple construction a PDFA can be made to adhere to the stochastic property, without it changing the language it recognises. A construction for turning a PDFA $M = (Q, \Sigma, \delta, P, q_0, F)$ into a PDFA $M' = (Q', \Sigma', \delta', P', F')$ that adheres to the stochastic property is as follows:

1. Copy the states, alphabet and initial state. We make sure Q' has a garbage state and we add a new symbol x to the alphabet: Let $Q' = Q \cup \{\varnothing\}$ where $\varnothing$ is a garbage state; $\Sigma' = \Sigma \cup \{x\}$ for some $x \notin \Sigma$; and $q'_0 = q_0$.

2. Copy the transitions, probability distribution and final probabilities: For all $q \in Q$ and $\sigma \in \Sigma$ let $\delta'(q, \sigma) = \delta(q, \sigma)$; $P'(q, \sigma) = P(q, \sigma)$; and $F'[q] = F[q]$.

3. Define the transitions and probability distribution for the new symbol x. The probability for transitions with x is the probability needed for the sum of probabilities from state q to be equal to 1: $\delta'(q, x) = \varnothing$ and $P'(q, x) = 1 - (\sum_{\sigma \in \Sigma} P(q, \sigma) + F[q])$;

4. Lastly, define the garbage state: Let $F[\varnothing] = 0$; $\sum_{\sigma \in \Sigma} P(\varnothing, \sigma) = 0$ and $P(\varnothing, x) = 1$. For all $\sigma' \in \Sigma'$ let $\delta(\varnothing, \sigma') = \varnothing$.

Step three ensures that M' adheres to the stochastic property. We note that a string over Σ^* that contains x is never accepted by M' because it never exits the garbage state. The added garbage state is accessible, and reaching an accepting state from it is never possible, which is why the language accepted by M' is not a probability distribution. The path for all strings over Σ' not containing x is unchanged. Then, $L(M') = L(M)$.

For better readability of state diagrams for PDFAs we choose to leave out some information with associated implications. The garbage state and its incoming transitions are not drawn. Whenever a transition or a final probability is not drawn, it is assumed to have a probability of zero.

An example of a PDFA is seen in Figure 3. We see q_0 as initial state and q_2, q_4 and q_5 as accepting states with final probabilities 0.4, 0.2 and 0.5, respectively. The garbage state and its incoming transitions are implied.

For PDFAs we can also define equivalence relations that are used for the definitions of minimality and hyper-minimality. These will look similar to that of DFAs.

A state $p \in Q$ is **reachable** from state $q \in Q$ if there exists a string $s \in \Sigma$ such that $\delta(q, s) = p$ where $P(q, s) > 0$. The definitions for left and right languages of states in PDFAs are changed to work with distributions. The **left language** of a state $q \in Q$ is $\overleftarrow{q} = \{P(q_0, s) \mid s \in \Sigma^* \wedge \delta(q_0, s) = q \wedge P(q_0, s) > 0\}$ where q_0 is the initial state. The **right language** of a state $q \in Q$ is $\overrightarrow{q} = \{P(q, s) \cdot F[\delta(q, s)] \mid s \in \Sigma^* \wedge P(q, s) \cdot F[\delta(q, s)] > 0\}$. We denote for some $s \in \Sigma^*$ the mapping $\overrightarrow{q}(s) = P(q, s) \cdot F[\delta(q, s)]$. We define **left support** of the left language and **right support** of right languages. The supports is the set of all the strings from which their respective languages are obtained: $supp(\overleftarrow{q}) = \{s \in \Sigma^* \mid \delta(q_0, s) = q \wedge P(q_0, s) > 0\}$ where q_0 is the initial state; and $supp(\overrightarrow{q}) = \{s \in \Sigma^* \mid P(q, s) \cdot F[\delta(q, s)] > 0\}$.

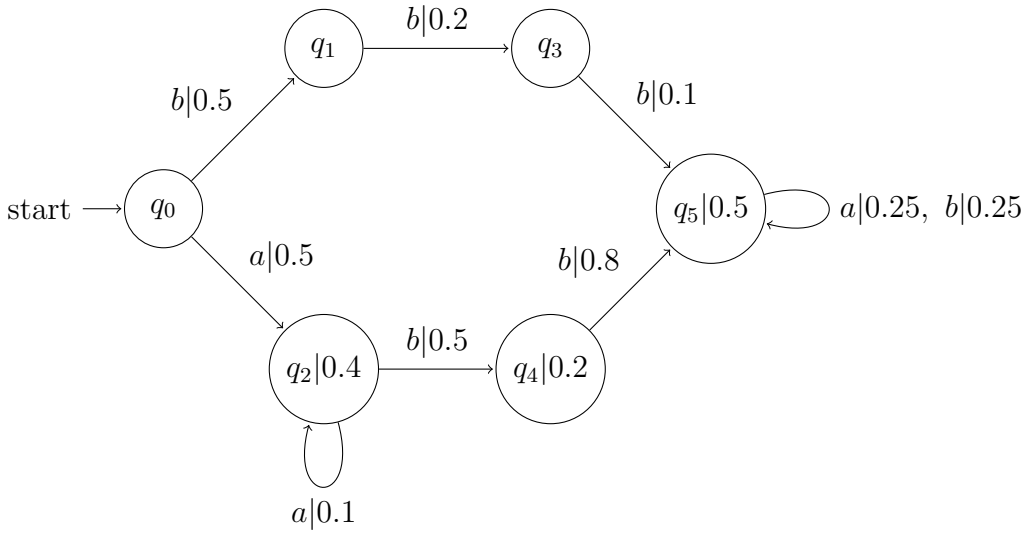


Figure 3: An example of a PDFA M_1 with the reduced stochastic property.

The definitions of equivalence relations for PDFAs and their languages, automata and states are the same as for DFAs. Likewise, the definition of a minimal automaton is unchanged. Note that for the equivalence of states we use the definition of right language for states in PDFAs.

Lastly, we quickly go over the definitions for WDFAs. Due to the similarities to PDFAs We only broach that what is necessary for Section 3.1. A **weighted deterministic finite automaton** recognises a *weighted language*, and is defined as $M = (Q, \Sigma, \delta, wt, q_0, k_0, F)$ [MQ12] where Q, Σ, δ, q_0 and F follow the definitions for DFAs. wt is a weight mapping $wt : Q \times \Sigma \to \mathbb{K}$ that coincides with δ such that each transition for a pair $(q \in Q, \sigma \in \Sigma)$ is attributed a weight and a next state. $k_0 \in \mathbb{K}$ is the initial weight. δ and wt can be extended the same as for δ and P of PDFAs, respectively.

A state $p \in Q$ is reachable from state $q \in Q$ if there exists a $s \in \Sigma$ such that $\delta(q, s) = 0$ where $wt(q, s) \neq 0$. The **left language** of state $q \in Q$ is $\overleftarrow{q} = \{wt(q_0, s) \mid s \in \Sigma^* \wedge \delta(q_0, s) = q)\}$. The **right language** of state $q \in Q$ is $\overrightarrow{q} = \{wt(q, s) \mid s \in \Sigma^* \wedge \delta(q, s) \in F\}$. We denote for some $s \in \Sigma^*$ the mapping $\overrightarrow{q}(s) = wt(q, s)$ if $\delta(q, s) \in F$, otherwise $\overrightarrow{q}(s) = 0$. The **left support** is defined the same as for PDFAs, and the **right support** is $supp(\overrightarrow{q}) = \{s \in \Sigma^* \mid \delta(q, s) \in F\}$. The weighted language accepted by M is $\varphi_M = k_0 \cdot \overrightarrow{q_0}$.

3 Hyper-minimisation

Hyper-minimality works on the idea that automata can describe languages with an infinite amount of words. Then, if we remove a finite amount of words from the language we are still left with an infinite amount of words. This follows from the ideal of lossy compression where certain information loss is seen as acceptable in order the compress something even more. In this case, the acceptable loss is that of a finite amount of words. For this the term *almost-equivalent* is introduced.

For hyper-minimality we need definitions for **almost-equivalence** relations, which slightly differ from the equivalence relations for minimality.

Definition 1. (Almost-equivalence) [Mal11, def. 1] for regular languages and DFAs:

- Two languages are almost-equal if there are finite amount of strings that make the difference between the two sets: $L_1 \approx L_2$ if and only if $L_1 \triangle L_2$ is finite.
- Two DFAs M_1 and M_2 are almost-equivalent if the languages they recognise are almost-equivalent: $M_1 \approx M_2$ if and only if $L(M_1) \triangle L(M_2)$ is finite.
- Two states are almost-equivalent if their right languages are almost-equal: $p \approx q$ if and only if $\overrightarrow{q} \triangle \overrightarrow{p}$ is finite.

It should be noted that a language with a finite amount of strings is almost-equivalent to the empty language, because $L \triangle \{\} = L$. This is not particularly interesting for hyper-minimisation, which is why hyper-minimisation is focussed on infinite languages.

Known algorithms for hyper-minimisation make use of state comparisons, and choosing the right states to merge together [Mal11, p. 40]. For this the algorithms use the notion of preamble and kernel states.

Definition 2. (Preamble and kernel states) [Mal11, p. 37] for DFAs:
The set of **preamble states** P contain all states reached from the initial state by a finite amount of words. In other words, the left language of these states is finite. The set of **kernel states** K contain the states of which the left language is infinite. It holds that $Q \setminus P = K$.

A DFA is hyper-minimal if no other almost-equivalent DFA exists with fewer states. From [BGS09, Theorem 3.4] follows the characterisation of hyper-minimality in [Mal11], including the proof. Another characterisation of hyper-minimality is that the hyper-minimal DFA is not unique [BGS09, p. 70].

Theorem 1. (Hyper-minimal) [Mal11, theorem. 5] A DFA is hyper-minimal if and only if:

1. The DFA is *minimal*.
2. The DFA has no *preamble state* that is almost-equivalent to another state.

In Figure 4 we can see a hyper-minimal automaton of our earlier example from Figure 2b. We note that q_3 is a preamble state, because its left language is finite: $\overleftarrow{q_3} = \{aa, ab\}$. The preamble states of M_2 are $P = \{q_0, q_1, q_3\}$. All other states are kernel states. Looking at the right language of q_3 and q_4 we find that these are almost-equivalent. $\overrightarrow{q_3} = \{a, aa, ab, aaa, aab, ...\}$ and $\overrightarrow{q_4} = \{\varepsilon, a, aa, ab, aaa, aab, ...\}$ where $\overrightarrow{q_3} \triangle \overrightarrow{q_4} = \{\varepsilon\}$. This is a finite difference, thus these two states are almost-equivalent and can be merged into one state.

3.1 Hyper-minimisation for WDFAs and PDFAs

The prior paper by A. Malleti and D. Quernheim into the hyper-minimisation of WDFAs [MQ12] can largely be adopted to the PDFAs of this study. For that we will first describe their hyper-minimisation algorithm for WDFAs. After that we show the similarities between the properties

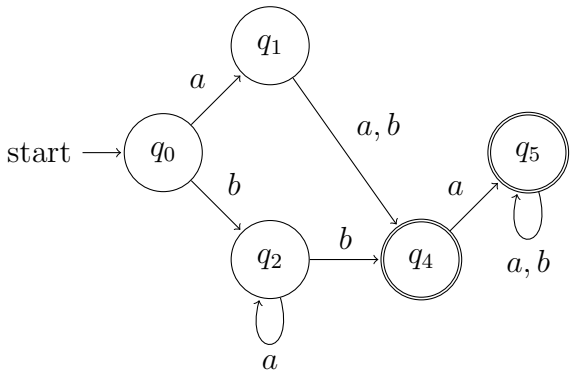


Figure 4: DFA M_3 is the hyper-minimal automaton of M_2 from Figure 2.

of WDFAs and PDFAs that allow us to apply this hyper-minimisation algorithm to PDFAs. In Section Examples we will find that there are limitations as to when a PDFA can be hyper-minimised.

A common technique used in hyper-minimisation algorithms is that of state merges [MQ12, p. 8]. For the hyper-minimisation of DFAs it was sufficient to work with preamble and kernel states [BGS09]. The proposed algorithm for WDFAs in [MQ12] additionally uses the notion of co-preamble and co-kernel states.

Definition 3. (Preamble and co-preamble states) [**MQ12, Def. 3, 8**] for the states in a WDFA:

- A state $q \in Q$ is preamble if $supp(\overleftarrow{q})$ is finite. Otherwise it is a kernel state.
- A state $q \in Q$ is co-preamble if $supp(\overrightarrow{q})$ is finite. Otherwise it is a co-kernel state.
- The set of preamble states and co-preamble states of Q are P and $\overline{P}$, respectively. The set of kernel states and co-kernel states of Q are $K = Q - P$ and $\overline{K} = Q - \overline{P}$, respectively.

Kernel states can be characterised by Lemma 2.

Lemma 2. (kernel states) [**HM10, MQ12**] A state $q \in Q$ of a minimal WDFA is kernel if it is reachable by a strongly connected component or if it contains a self loop with a weight not zero.

Proof: Two states $p, q \in Q$ are called strongly connected if p is reachable from p and q is reachable from q. Then, there exists a string s_1 and s_2 where $\delta(q, s_1) = p$, $wt(q, s_1) \neq 0$, $\delta(p, s_2)$ and $wt(p, s_2) \neq 0$. For $s \in \{s_1 s_2\}^*$ it holds that $\delta(q, s) = q$. $\{s_1 s_2\}^*$ is an infinite set and $\{z(s_1 s_2)^* \mid z \in \Sigma^* \wedge \delta(q_0, z) = q \wedge wt(q_0, z) > 0\} \subseteq supp(\overleftarrow{q})$ where q_0 is the initial state. Thus, $supp(\overleftarrow{q})$ is infinite. Any state $r \in Q$ reachable from q has for its left support $\{(supp(\overleftarrow{q})x \mid x \in \Sigma^* \wedge \delta(q, x) = r \wedge wt(q, x) > 0\} \subseteq supp(\overleftarrow{r})$. Then the left support of r is infinite and r is kernel. If some state $t \in Q$ contains a self loop for symbol $\sigma \in \Sigma$ where $wt(t, \sigma) \neq 0$ then for all $s \in \{\sigma\}^*$ it holds that $\delta(t, s) = t$. By the logic as with strongly connected states it follows that t is kernel.

In the example WDFA of Figure 5a we find $P = \{q_0, q_1, q_3\}$, $\overline{P} = \{q_6, g\}$, $K = \{q_2, q_3', q_4, q_5, g\}$ and $\overline{K} = \{q_0, q_1, q_2, q_3, q_3', q_4, q_5\}$.

For WDFAs and their weighted languages we need altered definitions for the **almost-equivalence** relation. These definitions consider that languages / states are almost-equivalent as long as there is a scalar that exists such that the two weighted languages / right languages, respectively, differ finitely.

Definition 4. (almost-equivalence) [MQ12, **def. 1**] for WDFAs:

- Two weighted languages φ_1, φ_2 are almost-equivalent if for almost all $s \in \Sigma^*$ holds $\varphi_1(s) = k \cdot \varphi_2(s)$ for some factor $k \in \mathbb{K} \setminus \{0\}$.
- Two WDFAs M_1 and M_2 are almost-equivalent if φ_{M_1} is almost-equivalent to φ_{M_1} with factor $k = 1$.
- Two states $p, q \in Q$ are almost-equivalent if for a finite amount of strings $s \in \Sigma^*$ holds that $\overrightarrow{q}(s) \neq k \cdot \overrightarrow{p}(s)$ for some $k \in \mathbb{K} \setminus \{0\}$.

A WDFA is called hyper-minimal if there is no almost-equivalent WDFA with fewer states [MQ12, p. 5]. The characterisation of hyper-minimality in Theorem 1 also applies to WDFAs [MQ12, Theorem 7].

Algorithm 1 (1) shows the general structure and time complexities of hyper-minimisation algorithms as noted in [Mal11] with the additions of line 5, as necessary for the algorithm for WDFA in [MQ12]. The algorithm is given a WDFA M, and returns the hyper-minimal WDFA. First, M is minimised using a minimisation algorithm by Eisner [Eis03] or if the WDFA is also a DFA by Hopcroft [Hop71], such that we do not have any inaccessible states for the remainder of the algorithm. Lines 2 and 3 are for the computation of kernel and co-kernel states, which by Lemma 2 can be accomplished with an algorithm for strongly connected components such as Tarjan's algorithm [Tar72]. The partition with blocks for almost-equivalent states is computed in Algorithm 2 (2) with the WDFA and the set of co-kernel states as inputs. The results of line 4 are given as input to state merge algorithm, as seen in Algorithm 3 (3).

Algorithm 1 Structure of hyper-minimization algorithm		
Require: a WDFA M with n states		
Return: an almost-equivalent hyper-minimal WDFA		
	$M \leftarrow Minimise(M)$	Hopcroft's or Eisner's algorithm; $O(n\ log\ n)$
2:	$K \leftarrow ComputeKernel(M)$	Tarjan's algorithm; $O(n)$
	$\overline{K} \leftarrow ComputeCoKernel(M)$	Tarjan's algorithm; $O(n)$
4:	$(\approx, t) \leftarrow ComputeAlmostEquivalence(M, \overline{K})$	Algorithm 2; $O(n\ log\ n)$
	return $MergeStates(M, K, \approx, t)$	Algorithm 3; $O(n)$

Table 1: The general structure of hyper-minimization algorithms, as in [MQ12] for WDFAs.

The almost-equivalence computation in Algorithm 2 from [MQ12] makes use of the almost-equivalence of states by comparing the right supports. The observation in [HM10, MQ12] is that

the almost-equivalence of states can be obtained with a *signature map*. The algorithm for WDFAs needs to account for the weights of transitions. It standardises these weights by calculating the ratio between weights if and only if those transitions go to a co-kernel state. Note that the right support of co-preamble states contains a finite amount of strings, therefore the size of the weight going to such states only impacts the calculated weight for finite amount of accepted strings. For that reason the signature map only accounts for weights of transitions going to co-kernel states.

Definition 5. (Standard signature) [MQ12, def. 10] The standard signature of a state $q \in Q$ is the mapping $sig_q : \Sigma \to Q \times \mathbb{K}$ where for all $\sigma \in \Sigma$:

- If $\delta(q, \sigma) \in \overline{P}$ then $sig_q(\sigma) = \langle \bot, 1 \rangle$
- Else, let $\sigma_0 \in \Sigma$ be the smallest symbol such that $\delta(q, \sigma_0) \in \overline{K}$. Then $sig_q(\sigma) = \langle \delta(q, \sigma), \frac{wt(q,\sigma)}{wt(q,\sigma_0)} \rangle$

The almost-equivalence of states $p, q \in Q$ can be characterised by their standard signature. The proof is shown in [MQ12, p. 9-11].

Lemma 3. (Almost-equivalence by signature) [MQ12, Lemma 11] Let $p, q \in Q$. If $sig_p = sig_q$ then $p \approx q$.

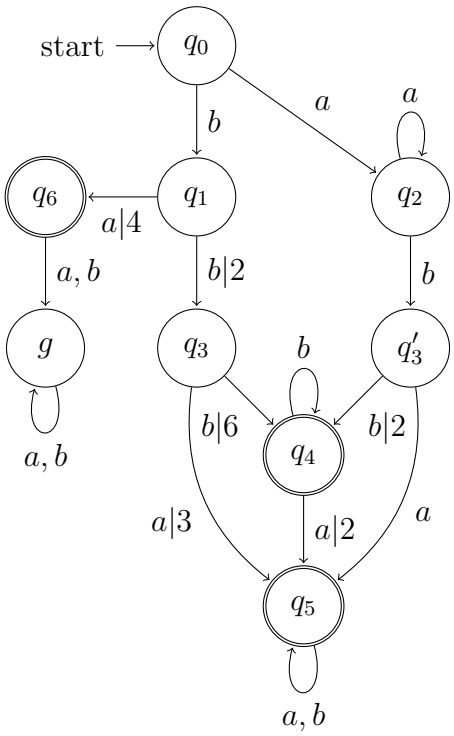
The weighted merge of two states q into q' using a scaling factor k is shown in Definition 6. This first defines, if q was the initial state, that the values for q'_0 and k'_0 are updated. Secondly, all transitions, and corresponding weights, that go to q need to updated to go to q' instead. Note that the merging of kernel states into another state does not produce an almost-equivalent automaton in hyper-minimisation. Implicitly, this input is rejected.

Definition 6. (State merge) [MQ12, def. 5] Let $q, q' \in Q$ and $k \in \mathbb{K}$ with $q \neq q'$. The k-weighted merge of q' into q for WDFA M is $merge_M(q' \xrightarrow{k} q) = (Q \setminus \{q'\}, \Sigma, q'_0, k'_0, \delta', wt', F \setminus \{q'\})$ such that for all $r \in Q$ and $\sigma \in \Sigma$:

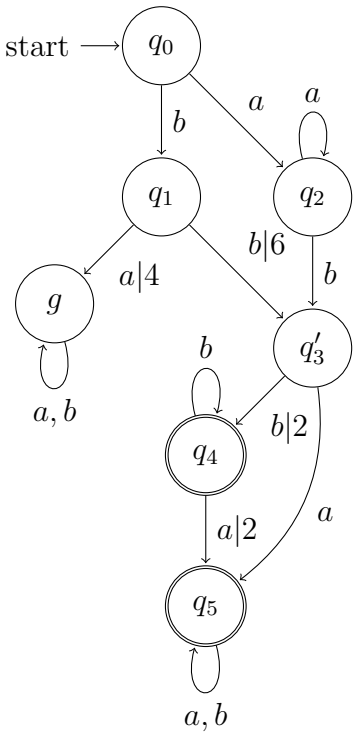
$$q'_0 = \begin{cases} q & \text{if } q_0 = q' \\ q_0 & \text{otherwise} \end{cases} \qquad k'_0 = \begin{cases} k \cdot k'_0 & \text{if } q_0 = q' \\ k_0 & \text{otherwise} \end{cases}$$

$$\delta'(r, \sigma) = \begin{cases} q & \text{if } \delta(r, \sigma) = q' \\ \delta(r, \sigma) & \text{otherwise} \end{cases} \qquad wt'(r, \sigma) = \begin{cases} k \cdot wt(r, \sigma) & \text{if } \delta(r, \sigma) = q' \\ wt(r, \sigma) & \text{otherwise} \end{cases}$$

Consider the example of hyper-minimisation of WDFA M_1 in Figure 5. We calculate the signature of q_1. Note, q_6 is co-preamble and q_3 is co-kernel. The signature map first reads a, then b, where b is the smallest symbol σ_0 because $wt(q_1, b) = 2$ and q_3 is co-kernel. Then, $sig_{q_1} = \langle (\bot, 1), (q_3, \frac{2}{2}) \rangle$. Now for q_3. We see q_4 and q_5 are both co-kernel, where $wt(q_3, a) = 3$ is smaller then $wt(q_3, b) = 6$. Then, $sig_{q_3} = \langle (q_5, \frac{3}{3}), (q_4, \frac{6}{3}) \rangle$. Idem., q'_3 where $\sigma_0 = a$ we have $sig'_{q_3} = \langle (q_5, \frac{1}{1}), (q_4, \frac{2}{1}) \rangle$. We see that $sig_{q_3} = sig_{q'_3}$. By Lemma 3 we conclude that $q_3 \approx q'_3$ and that these states can be merged. For the merge by Definition 6. $merge_{M_1}(q_3 \xrightarrow{k} q'_3)$ we take $k = \frac{3}{1} = 3$. We also find that



(a) A minimal WDFA M_1 of which q_3 and q'_3 are almost-equivalent and q_6 and g are almost-equivalent.



(b) The hyper-minimal WDFA M_2 of M_1 where q_3 merged into q'_3 with scaling factor 3 and q_6 merged into g with scaling factor 1.

Figure 5: An example of hyper-minimisation on a WDFA.

$sig_{q_6} = \langle(\bot, 1), (\bot, 1)\rangle = sig_g$.

By Lemma 3 the almost-equivalence is computed in Algorithm 2 as also described in [MQ12, p. 11]. This algorithm takes the automaton M and the set of co-kernel states $\overline{K}$ as input, and returns a partition of Q with the almost-equivalence for states, as well as the scaling map that defines the scaling factor for the states that need to be merged.

Line by line, Algorithm 2 does the following execution: Lines 1-2 initialises the variables for the almost-equivalence partition π to be "the trivial partition, in which each state forms its own block" [HM10, p. 8] and the scaling map f for each state. Line 3 initialises the empty hash map h for the signature map, and the set of remaining states I to loop over. Line 5-16 is the loop body for states $q \in I$, where q is implicitly removed from I at the end of a loop iteration. Line 5 calculates the signature map by Definition 5. If the hash map already contains this exact signature, then q is almost-equivalent to some other state by Lemma 3. Line 7 retrieves the block of states with the same signature as q. Line 8-9 is an efficiency optimisation where the state representing the larger block (q) is merged into [Mal11, p. 41]. Line 10 adds the predecessors of the state that gets merged back to I, in the case that merging q' changes the almost-equivalence status of its predecessors. Line 11 calculates the scaling factor for the merging state q'. Line 12 performs the merge as defined by Definition 6 (where q' is rejected if $q' \in K$). Line 13 adds q' to the block of q. Lines 14-15 recompute the scaling factors for all almost-equivalent states that were in the block of q', so that

these can be merged later by Algorithm 3. Line 16 adds the signature map for state q, with which we started the for-loop, to the hash map.

The merging that happens in Algorithm 3 simply goes over each block in the almost-equivalence partition that it is given as input. Then, it merges all preamble states into some selected state in the block with the appropriate scaling factor.

Algorithm 2 Algorithm computing the almost-equivalence $\sim$

Require: a minimal WDFA M anbd its co-kernel states $\overline{K}$
Return: almost-equivalence $\sim$ as a partition and scaling map $f : Q \to \mathbb{K}$

for all $q \in Q$ **do**
  $\pi(q) \leftarrow \{q\} : f(q) \leftarrow 1$ // trivial initial blocks
$h \leftarrow \emptyset, i \leftarrow Q$ // hash map of type $h : Q^{\Sigma} \to Q$
for all $q \in I$ **do**
  $succ \leftarrow sig_q$ // compute standardised signature using δ and $\mathbb{K}$
  if $HasValue(h, succ)$ **then**
    $q' \leftarrow Get(h, succ)$ // retrieve state in bucket 'succ' of h
    if $|\pi(q')| \geq |\pi(q)|$ **then**
      $Swap(q, q')$ //exchange roles of q and q'
    $I \leftarrow I \cup \{r \in Q \setminus \{q'\} \mid \exists \sigma \in \Sigma : \delta(r, \sigma) = q'\}$ // add predecessor of q'
    $f(q') \leftarrow \frac{wt(q', \sigma_0)}{wt(q, \sigma_0)}$ // add precessors of q'
    $M \leftarrow merge_M(q' \xrightarrow{f(q')} q)$ // merge q' into q
    $\pi(q) \leftarrow \pi(q) \cup \pi(q')$ // q and q' are almost-equivalent
    for all $r \in \pi(q')$ **do**
      $f(r) \leftarrow f(r) \cdot f(q')$ // recompute scaling factors
  $h \leftarrow Put(h, succ, q)$ // store q in h under key '$succ$'
return (h, f)

Table 2: Algorithm for calculating almost-equivalence and the corresponding scaling map, as in [MQ12] for WDFAs.

The principles that are used for the computation of almost-equivalent states for the weighted setting of WDFAs states are transferable to the probabilistic setting of PDFAs.
For the almost-equivalence relation we let the almost-equivalence for languages, PDFAs and states be the same as for WDFAs (Definition 5) with adjustment to the possible set of scalars $\mathbb{R}_{>0}$ and the appropriate definition of right support.

Definition 7. (Almost-equivalence) for PDFAs:

- Two probabilistic languages φ_1, φ_2 are almost-equivalent if for almost all $s \in \Sigma^*$ holds $\varphi_1(s) = k \cdot \varphi_2(s)$ for some factor $k \in \mathbb{R}_{>0}$.
- Two PDFAs M_1 and M_2 are almost-equivalent if φ_{M_1} is almost-equivalent to φ_{M_2} with factor $k = 1$.

Algorithm 3 Merging almost-equivalent states

Require: a minimal WDFA M, its kernal states K, its almost-equivalence $\approx$ and a scaling map $f : Q \to \mathbb{K}$

Return: hyper-minimal WDFA M that is almost-equivalent to the input WDFA

for all $B \in (Q/\approx)$ **do**
  select $q \in B$ such that $q \in K$ if possible
  for all $q' \in B \setminus K$ **do**
    $M \leftarrow merge_M(q' \xrightarrow{\frac{f(q')}{f(q)}} q)$
return M

Table 3: The algorithm for merging almost-equivalent states, as in [MQ12] for WDFAs.

- Two states $p, q \in Q$ are almost-equivalent if for a finite amount of strings $s \in \Sigma^*$ holds that $\overrightarrow{q}(s) \neq k \cdot \overrightarrow{p}(s)$ for some $k \in \mathbb{R}_{>0}$.

The reason why the principles for almost-equivalence can be transferred is because both weighted settings share a *monoid* in their *semiring*.
The weighted setting for WDFAs has its weights taken over a commutative semifield $\langle \mathbb{K}, +, \cdot, 0, 1 \rangle$ [MQ12] where $\mathbb{K}$ is a set of weights. This semifield includes the commutative monoid $\langle \mathbb{K}, \cdot, 1 \rangle$. The probabilistic semifield is $\langle [0,1], max, \cdot, 0, 1 \rangle$ [MQ12]. Evidently, if we take $\mathbb{K} = [0,1]$ then these semifields share the commutative monoid $\langle [0,1], \cdot, 1 \rangle$. Because WDFAs and PDFAs, as well as the algorithm for hyper-minimisation of WDFAs, exclusively use the monoid for $\cdot$ we transfer the knowledge gained from hyper-minimisation for WDFAs to that of PDFAs.

Precisely, the definition of the signature map in Definition 5 and the corresponding Lemma 3 for almost-equivalence of states is identical for PDFAs. Both WDFAs and PDFAs have a ratio of weights by which the signature map is able to recognise the almost-equivalence between states. This follows from the practically identical definitions for almost-equivalent states between Definition 4 and 5.

Arguably, the construction for a WDFA for $\mathbb{K} = [0,1]$ is equivalent to that of PDFAs with exception for the initial weight, final probabilities and reduced stochastic property. We will show that the first two exceptions are trivial differences. The reduced stochastic property is a major differentiating property, that, as shown through a counter-example in Section Examples, can not be preserved for all state merges.

Lemma 4. (Initial weight) The initial weight in a WDFA and non-existent initial weight in a PDFA are not differentiating properties for hyper-minimisation.

Proof: The initial weight of the WDFA can be set to equal one, such that it is the identity of the monoid and no longer has an impact on the language accepted by the WDFA. Then, it matches the description for the non-existent initial weight of a PDFA.

Lemma 5. (Accepting states) The accepting states in a WDFA and accepting states with finalisation probabilities in a PDFA are not differentiating properties for hyper-minimisation.

Proof: All accepting states $f \in F$ of a WDFA can be seen as states for which the final probabilities in a PDFA would be greater than zero: $F[f] > 0$. Then, all appropriate definitions such as right language and right support for PDFAs would recognise f as an accepting state. It should be mentioned that the definitions for valid paths to reach a state in both PDFAs and WDFAs have the constraint that the weight of the path can not be equal to zero. This equal constraint follows for left languages and right languages, as well as their supports. For the almost-equivalence of states by Definition 4 and 7 we note that the right support remains equal, since the exact same set of strings are accepted by the exact same transition map and set of accepting states. It follows that the value of the final probability need not be considered for the signature map by Definition 5 for its translation to PDFAs.

3.2 Examples

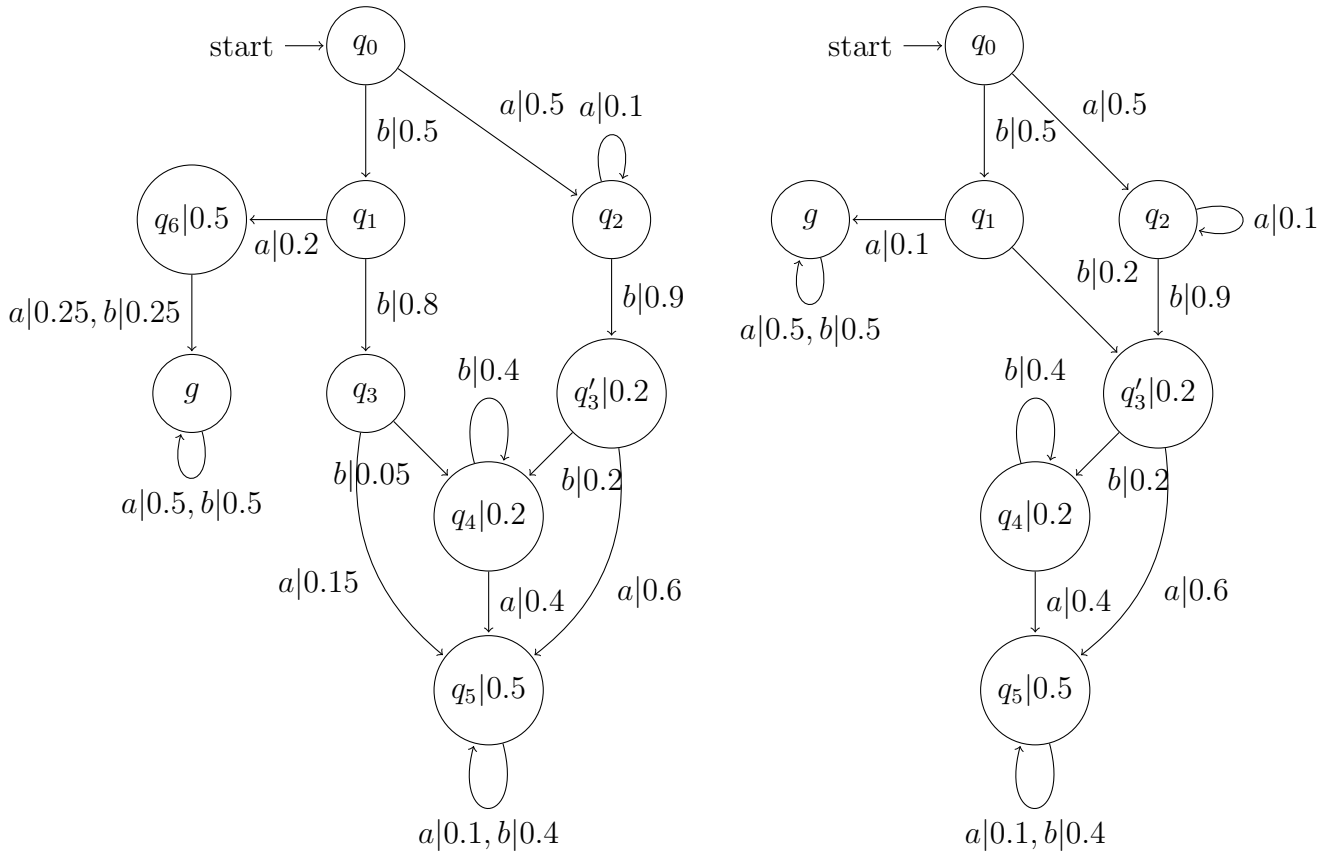
The hyper-minimisation algorithm for WDFAs, as stated, can largely be transferred over to PDFAs. However, there are instances of merging states with a scaling factor where the resulting PDFA no longer upholds the reduced stochastic property. In this section we will first go over an example for which the algorithm for WDFAs can be applied with no issue. Then we give an example where the reduced stochastic property causes an issue, that can be alleviated by redistributing its probabilities. Finally, we give a counter-example for a PDFA for which a preamble state is almost-equivalent to another state, but cannot be merged. For this last example, we define *illegal merges*. The oppositive of an illegal merge is a legal merge.

Definition 8. (Illegal merge) An **illegal merge** is a merge of two states, at least one of which being preamble, for which no probability distribution P of PDFA M and no scaling factor exists, such that the merge obtains an almost-equivalent automaton.

In Figure 6 we see the example where the algorithm for a WDFA obtains the hyper-minimal PDFA M_2 of M_1. In Table 4 we see see the signature map for M_1. We note that $sig(g) = sig(q_0)$ and $sig(q_3) = sig'_3$. By the algorithm we end up with the non-singleton blocks of $\pi = \{\{q_6, g\}, \{q'_3.q_3\}\}$. We merge the states by their respective blocks. The scaling factor k for the merge of $q_6 \xrightarrow{k} g$ is $k = \frac{P(q_1,a)}{P(g,a)} = \frac{0.25}{0.5} = \frac{1}{2}$. The scaling factor k for the merge $q_3 \xrightarrow{k} q'_3$ is $k = \frac{1}{4}$.

Signature map	
$\langle q_5, 4\rangle, \langle q_5, 1\rangle$	q_5
$\langle q_4, 1\rangle, \langle q_5, 1\rangle$	q_4
$\langle q_5, 3\rangle, \langle q_4, 1\rangle$	q'_3
$\langle q_5, 3\rangle, \langle q_4, 1\rangle$	q_3
$\langle q_2, 1\rangle, \langle q'_3, 9\rangle$	q_2
$\langle \bot, 1\rangle, \langle q_3, 4\rangle$	q_1
$\langle \bot, 1\rangle, \langle \bot, 1\rangle$	g
$\langle \bot, 1\rangle, \langle \bot, 1\rangle$	q_6
$\langle q_2, 1\rangle, \langle q_1, 1\rangle$	q_0

Table 4: The signature map for M_1 in Figure 6a.



(a) A minimal PDFA M_1 of which q_3 and q'_3 are almost-equivalent and q_6 and g are almost-equivalent.

(b) A hyper-minimal PDFA M_2 where q_3 merged into q'_3 with scaling factor $\frac{1}{4}$ and q_6 merged into g with scaling factor $\frac{1}{2}$.

Figure 6: An example of hyper-minimisation on a PDFA.

Next we consider the example M_3 in Figure 7. In this PDFA the states q_3 and q'_3 are almost-equivalent because for almost all $s \in \Sigma^* : \overrightarrow{q'_3}(s) = 6.25 \cdot \overrightarrow{q_3}(s)$. Precisely, only for one string $s = \varepsilon$ it is true that $\overrightarrow{q'_3}(s) \neq 6.25 \cdot \overrightarrow{q_3}(s)$. However, when attempting to merge the preamble state q_3 into q'_3 for $k = 6.25$ we find that the probability for $\delta'(q_1, b) = q'_3$ becomes $P'(q_1, b) = 6.25 \cdot P(q_1, b) = 6.25 \cdot 0.2 = 1.25$. By the definition of probabilities and by the definition of the reduced stochastic property it can not be that a probability of a state exceeds one.

We conclude that the merging algorithm can not be applied to M_3 as is. Yet, it is possible to obtain the hyper-minimal automaton by making some adjustments. In Figure 8 we see an almost-equivalent PDFA M_4 to M_3 with a different probability distribution over its transitions and accepting states. Intuitively, distributing the probabilities of M_3 in such a way that the scaling factor is exactly one, assures that two states can always be merged.

The probability redistribution in M_4 makes use of the definition of almost-equivalence to maximise the probabilities on the path towards q_3. Observe that we increase the outgoing probability as much as possible for the transition from q_0 in the path that reaches q_3, and for the path the path does not reach q_3 we reduce the probability as much as possible. This is allowed by almost-equivalence as long as the probability of the right language from the initial state towards an accepting kernel

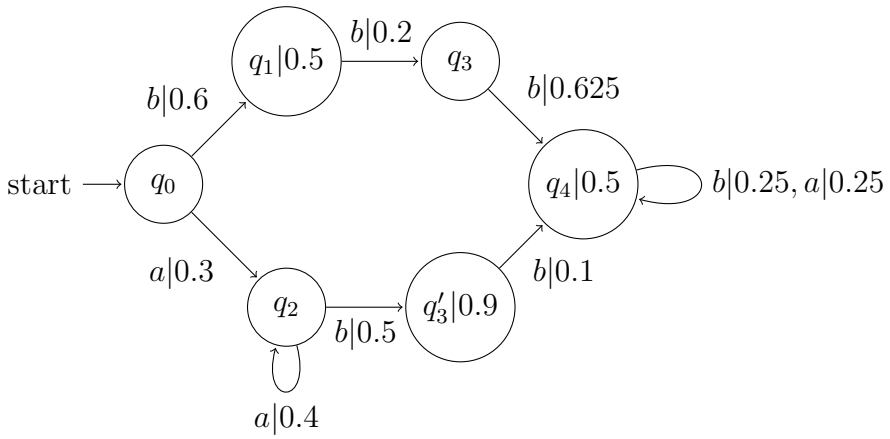


Figure 7: A PDFA M_3 where q_3 and q_3' are almost-equivalent, but can not be merged for the scaling factor $k = 6.25$.

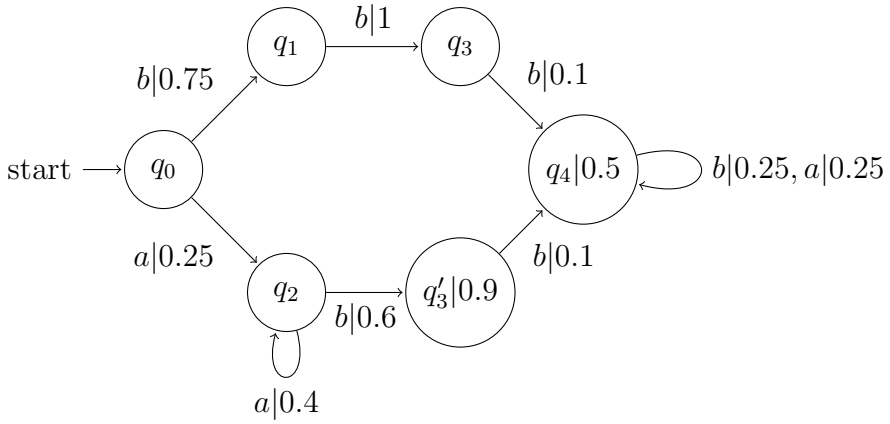


Figure 8: An almost-equivalent PDFA M_4 to M_3 after a probability redistribution.

state is equivalent for almost all strings, and as long as all states adhere to the reduced stochastic property.

By Lemma 6 we can remove the final probability of preamble state q_1, which allows us to shift the finalisation probability onto the probability from q_1 to q_3. Then, by shifting the probability $P(q_0, a)$ onto the probability from $P(q_2, b)$ we lower the sum of probabilities in state q_0, allowing the probability from q_0 to q_1 to be raised such that the sum of probabilities in q_0 equals one. Evidently, for all $s \in supp(\overleftarrow{q_3'})$ it stands that $\varphi_{M_3}(s) = 0.3{\cdot}0.4^{|s|-2}{\cdot}0.5{\cdot}0.9 = 0.25{\cdot}0.4^{|s|-2}{\cdot}0.6{\cdot}0.9 = \varphi_{M_3}(s)$, for all $s = \{bbbz \mid z \in \{a, b\}^*\}$ we have $\varphi_{M_2}(s) = 0.6{\cdot}0.2{\cdot}0.625{\cdot}0.25^{|s|-3} = 0.75{\cdot}1{\cdot}0.1{\cdot}0.25^{|s|-3} = \varphi_{M_3}(s)$, idem., for all $s \in supp(\overleftarrow{q_4}) : \varphi_{M_3}(s) = \varphi_{M_4}(s)$. Thus, M_3 and M_4 are almost-equivalent by Definition 7.

Lemma 6. (Accepting preamble states) A PDFA M with preamble state $q \in Q$ and $F[q] > 0$ is almost-equivalent to the PDFA M' with $M' = M$ where $F'[q] = 0$.

Proof: : Let $q \in Q$ be preamble. Then $supp(\overleftarrow{q})$ is a finite set. It follows that for all $s \in supp(\overleftarrow{q})$: $\varphi_M(s) \neq \varphi_{M'}(s)$ is true for finite s. Conclusion: $M \approx M'$ by Definition 7.

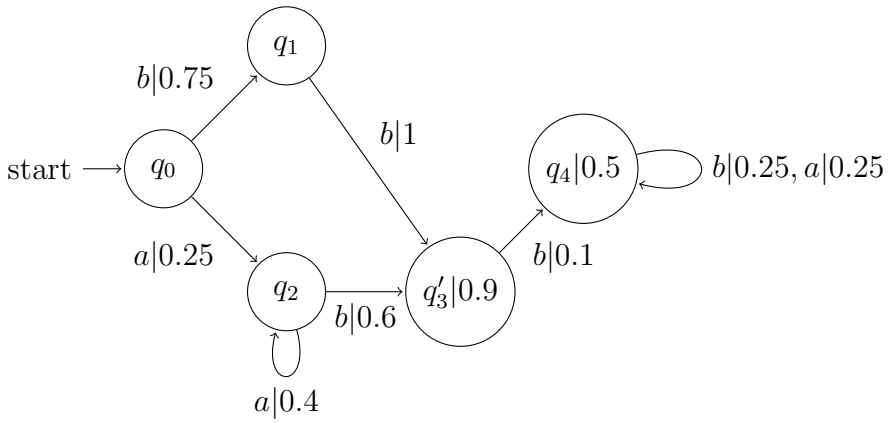


Figure 9: The hyper-minimal PDFA M_5 for M_3.

The PDFA M_4 can have state q_3 merged into q_3' with a scaling factor equalling one, without the reduced stochastic property being violated (see Figure 9). Thus, there exist PDFAs that can not be hyper-minimised by the classic WDFA hyper-minimisation algorithm, yet can be made hyper-minimal by performing probability redistributions to another (almost-equivalent) PDFA.

Whereas the previous example shows that it is sometimes possible to find a probability distribution for which a merge is legal, such distribution is not always possible to find. For this we show the PDFA in Figure 10. By a proof we show that the two almost-equivalent states, one of which preamble, can never be legally merged for any probability distribution. With this counter-example we show, by the characterisation of hyper-minimality in Theorem 1, that the hyper-minimal PDFA cannot be obtained by any state-reduction technique.

The PDFA M_6 is an example where q_3 and q_3' are almost-equivalent states. In Table 5 we see the probabilities if we calculate $\overrightarrow{q_3}$ and $\overrightarrow{q_3'}$. Then for $k = 10$ it holds for almost all strings $s \in \Sigma^*$ that $\overrightarrow{q_3}(s) = 10 \cdot \overrightarrow{q_3'}(s)$.

However, q_3 cannot be merged into q_3' for its necessary scaling factor. This is because the new transition b from q_1 to q_3' would need to exceed the maximum probability of one. We can attempt to perform a probability redistribution in a new PDFA $M' = M_6$ as in Figure 8. This does not make it possible to execute the merge, however, as seen in the proof.

Proof: The merge of q_3 into q_3' for PDFA M_6 is an illegal merge.

1. $\varphi_{M'} \approx \varphi_{M_6}$ if and only if for all $s \in supp(q_3')$ it is true that $\varphi_{M'}(s) = \varphi_{M_6}(s)$. This stands because q_3' is kernel and accepting.

2. There does not exists a distribution $P' \neq P$ with $P'(q_0, a) < P(q_0, a)$ where, for all $s \in supp(q_3')$ it is true that $\varphi_{M'}(s) = \varphi_{M_6}(s)$. This is because probability on the loop can not be changed, and because the sum of probabilities on q_2 equals one.

3. Let $P'(q_3', b) > P(q_3', b)$. Then by 1. and 2. this probability needs to be pushed forward to q_4. This is an accepting kernel state, so we find that $P'(q_3', b) \cdot F'[q_4] = P(q_3', b) \cdot F[q_4]$. Then for the almost-equivalence of PDFAs we require $F'[q_4] = F[q_4] \cdot \frac{P(q_3', b)}{P'(q_3', b)}$. Consequently, for $P'(q_3, b)F'[q_4] = P(q_3, b)F[q_4]$ to hold we require $P'(q_3, b) = \frac{P(q_3, b) \cdot F[q_4]}{F'[q_4]}$. Then for the scaling factor $k = P'(q_3, b) \cdot P'(q_3', b)^{-1} = \frac{P(q_3, b) \cdot F[q_4]}{F'[q_4]} \cdot (\frac{P(q_3', b) \cdot F[q_4]}{F'[q_4]})^{-1} = P(q_3, b) \cdot P(q_3', b)^{-1}$. In conclusion, the scaling factor remains the same after pushing the probability $P(q_3', b)$ forward.

4. By 1. and 3. we conclude that the scaling factor must be $k = 10$. In order for M_6 to be almost-equivalent after a merge $q_3 \xrightarrow{10} q_3'$ there must exist two probabilities $i, j \in [0, 1]$ for which $i \cdot j \cdot P(q_3', b) = P(q_0, b) \cdot P(q_1, b) \cdot P(q_3, b)$. Then $i \cdot j = 5$ which is impossible for probabilities. In fact, $P(q_0, b) \cdot P(q_1, b)$ is already maximal.

5. By 4. we conclude that it not possible to merge q_3 into q_3' for the scaling factor of 10 for any distribution P.

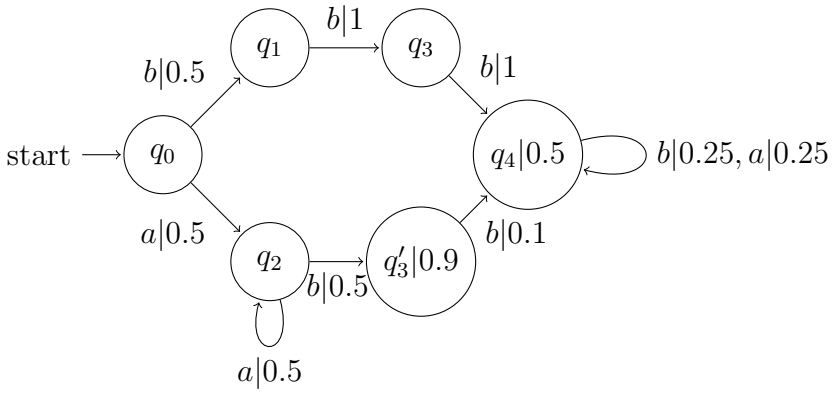


Figure 10: A PDFA M_6 where q_3 and q_3' are almost-equivalent, but can not be merged.

string	q_3	q_3'
ε	0	0.9
a	0	0
b	0.5	0.05
ba	0.125	0.0125
baa	0.03125	0.003125
baa...	etc	etc
bb	==	==

Table 5: The final probabilities of strings starting in states q_3 and q_3''

By this proof we conclude that there exist PDFAs which contain preamble states that are almost-equivalent to another state, for which there is no state-reduction technique that obtains a PDFA without such preamble states. By counter-example we find that the hyper-minimal PDFA, by the classic notion of hyper-minimal for DFAs / WDFAs, cannot always be obtained.

Illegal merges are a problem unique to the (reduced) stochastic property of PDFAs for merges of almost-equivalent states. To illustrate this we look at an example for a classic merge for equivalence. In figure 11 we see two equivalent states q_3 and q_3'. Once more, we find that the merge $q_3 \xrightarrow{6} q_3'$ cannot occur. However, for equivalence states we can merge kernel states into preamble states without it changing the language. So $q_3' \xrightarrow{\frac{1}{6}} q_3$ is a legal merge to obtain a minimised PDFA. By swapping the direction of the merge we can find always find a scaling factor $k \leq 1$ such that the reduced stochastic property holds after the merge.

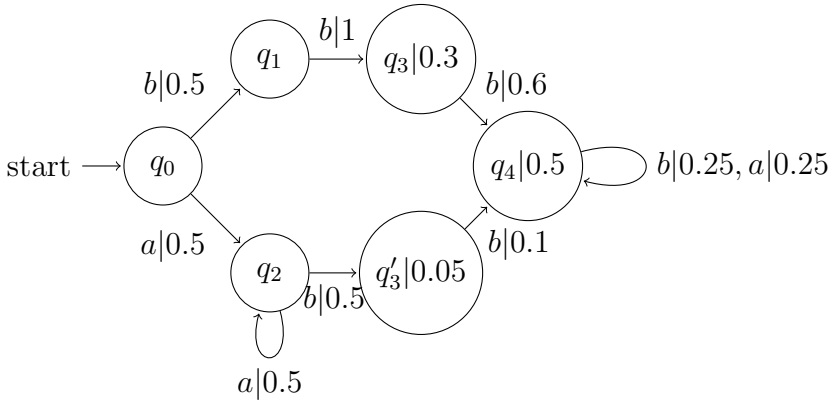


Figure 11: A PDFA M_6 where q_3 and q_3' are equivalent.

4 Merging algorithm

We have shown that the algorithm for computing almost-equivalent states in Algorithm 2 can be used to compute the almost-equivalent states for PDFAs. But, in the previous section we have also shown that the merging of states sometimes requires a probability redistribution over the transitions and accepting states of the current PDFA. For that, we make an alteration to the merging algorithm. As shown in our counter-example, this altered merging algorithm cannot assure that merges can occur.

The principle behind the merging algorithm is that we perform a probability redistribution over the PDFA before each merge, such that the scaling factor for the merge is less or equal to one. By Algorithm 2 (line 10) we calculate that the scaling factor for a merge $q' \xrightarrow{k} q$ is $k = \frac{P(q',\sigma_0)}{P(q,\sigma_0)}$. Intuitively, we reduce this scaling factor by redistributing the probabilities of the PDFA such that $P(q', \sigma_0)$ is as small as possible (minimal) and $P(q, \sigma_0)$ as large as possible (maximal). We accompany this by Theorem 2.

Theorem 2. (Legal merge) A merge $q' \xrightarrow{k} q$ in PDFA M is legal if and only if there exists an almost-equivalent probability distribution of M such that $k \leq 1$.

Proof: Assume $q' \xrightarrow{k} q$ is legal for some k. Then there exists a probability distribution such that all states in the PDFA M obtained after the merge adhere to the reduced stochastic property. Let M' be the PDFA with probability distribution P' that is almost-equivalent to M, where $P'(q_3, \sigma_0)$ is

Algorithm 4 Merging almost equivalent states of PDFA

Require: A minimal PDFA, its kernel states K, its almost-equivalence $\approx$ and a scaling map $F : Q \to \mathbb{R}_{>0}$

Return: A hyper-minimised PDFA with no preamble states that are almost-equivalent to another state, for which a legal merge exists.

for all $B \in (Q/\approx)$ **do**
2. select $q \in B$ such that $q \in K$ if possible
for all $q' \in B \setminus K$ **do**
4. $(M', k) \leftarrow RedistributeProbabilities(M, q', q)$ // compute probability redistribution and new scaling factor k
if $k \leq 1$ **do**
6. $M \leftarrow merge_{M'}(q' \xrightarrow{k} q)$
return M

Table 6: The merging algorithm 3 (3) adjusted for PDFAs.

minimal and $P'(q'_3, \sigma_0)$ is maximal. Then at least one predecessor p of q_3 must be maximal. Then for some $\sigma \in \Sigma$ and $\delta(p, \sigma) = q'_3$ it the case that p no longer holds by the reduced stochastic property if $P'(p, \sigma) \cdot k'$ for some scaling factor $k' > 1$. By assumption, the merge is legal, so there must be a scaling factor k' such that p adheres to the reduced stochastic property. Because p is maximal, this is only possible if this $k' \leq 1$. Conclusion: if $q' \xrightarrow{k} q$ is legal, then there exists a probability distribution of P such that $k \leq 1$.

As show in for Figure 10 it is not always possible to find a distribution of probabilities that allows for a legal merge. If a distribution exists for a legal merge, then the proof for merging in the hyper-minimisation of WDFAs suffices [MQ12].

Because the almost-equivalence relation is transitive [BGS09, Lemma 2.8, p. 73] we may redistribute the probabilities of a PDFA M such that the obtained M' is almost-equivalent to M. By transitivity it follows that the hyper-minimised PDFA of M' is also almost-equivalent to M. Since it does not need to be equivalent we can formulate an algorithm that is greedier than for weight redistributions for weighted automata that need to be equivalent (e.g. [Moh09, p29]).

Lemma 7. **($P, \overline{P}$ Almost-equivalence)** States that are co-preamble and preamble are almost-equivalent to the garbage state and can merged / removed freely.

Proof: The proof follows from the definition of the standard signature (Definition 5).

4.1 Probability redistribution algorithm

The probability redistribution computation occurs in the next three steps. The algorithm receives a minimal PDFA M and the states q', q where q' merges into q. It returns the PDFA such that the outgoing transitions of q' are minimal and the outgoing transitions of q are maximal.

Step 1. Prune the automaton by Lemma 6. and Lemma 7. such that there no final probabilities on preamble states, and no states that are both preamble and co-preamble. Remove all inaccessible states.

Step 2. Do a forward walk where the probabilities are pushed. Perform each step only after the previous step is finished.

1. Start in the initial state q_0
2. For all outgoing transitions $\delta(current, \sigma)$ (with a strictly positive probability) of the current state that go to a state p from which q' is not reachable do: Push probabilities forward such that the outgoing transition $P(current, \sigma)$ becomes minimal, $P(q, \sigma_0)$ is maximal and M remains almost-equivalent. For pushing probabilities forward on branching paths, track how much the probability has increased on each path such that for all $p_1, p_2 \in \{\delta(current, \sigma) \mid \sigma \in \Sigma\}$ holds $\frac{P_{new}(p_1,s)}{P_{old}(p_1,s)} = \frac{P_{new}(p_2,s)}{P_{old}(p_2,s)}$ for all $s \in \Sigma^*$ and such that at least one path $P(p, s)$ is maximal. Repeat for each branch.
3. For all outgoing transitions $\delta(current, \sigma)$ (with a strictly positive probability) of the current that go to a state p from which q' is reachable: Maximise each of these branching paths to q' by repeating step 2 and 3. Track how much the probability has increased on each path such that for all $p_1, p_2 \in \{\delta(current, \sigma) \mid \sigma \in \Sigma\}$ holds $\frac{P_{new}(p_1,s)}{P_{old}(p_1,s)} = \frac{P_{new}(p_2,s)}{P_{old}(p_2,s)}$ for all $s \in \Sigma^*$ and such that at least one path $P(p, s)$ is maximal.

Step 3. Return the probability redistributed automaton and $k = \frac{P(q',\sigma_0)}{P(q,\sigma_0)}$.

For the time complexity of the probability redistribution algorithm, where n is the amount of states of M, we find that Step 1. can be accomplished in $O(n)$ by simply checking all $q \in Q$ of M. Step 2. and 3., under the assumption that maximising the probabilities over branches can be accomplished by a depth-first search, can be accomplished with a forward walk in $O(n)$. Lastly, Step 3. simply returns in one line, with time complexity $O(1)$.

The total time complexity of the probability redistribution algorithm is $O(n)$, which is nested in the merge loop of Algorithm 4 combining for a total time complexity of $O(n^2)$.

5 Conclusions and further research

In this thesis we have shown that the hyper-minimisation algorithm using state merges for WDFAs can be transferred to the probabilistic setting of PDFAs to find an almost-equivalent, hyper-minimised automaton. This transferral follows from the shared algebraic structures between the two types of automata, such that the algorithmic components of hyper-minimisation can be transferred to PDFAs. That such an algorithm exists means that hyper-minimisation is a lossy compression technique that can be applied in practice to a PDFAs to obtain an automaton that is smaller, thus improving the efficiency of that model. However, there are circumstances where the hyper-minimal PDFA can not be obtained, due to the existence of illegal merges formed by the stochastic constraint. By analysis and by the proposed probability redistribution algorithm we provide a calculation and characterisation for when a state-reduction can be successfully applied. This work thus provides a

foundation for understanding the trade-offs between state-reduction and distributional fidelity in probabilistic deterministic finite automata.

The existence of illegal merges lowers the effectiveness of the hyper-minimisation technique in its capacity to reduce the size of the automaton. For this reason hyper-minimisation may always not be the best option for the lossy compression of PDFAs. Instead, an algorithm that works on minimising a PFA where equivalences of probabilities are permitted a certain error, such as [KW14], may give better results. That said, these different compression techniques do not necessarily invalidate each other, and sometimes can be applied sequentially to obtain an even smaller PDFA. Furthermore, there may be probabilistic data models which prefer its compression to be onto finite strings, which hyper-minimisation is purposed for.

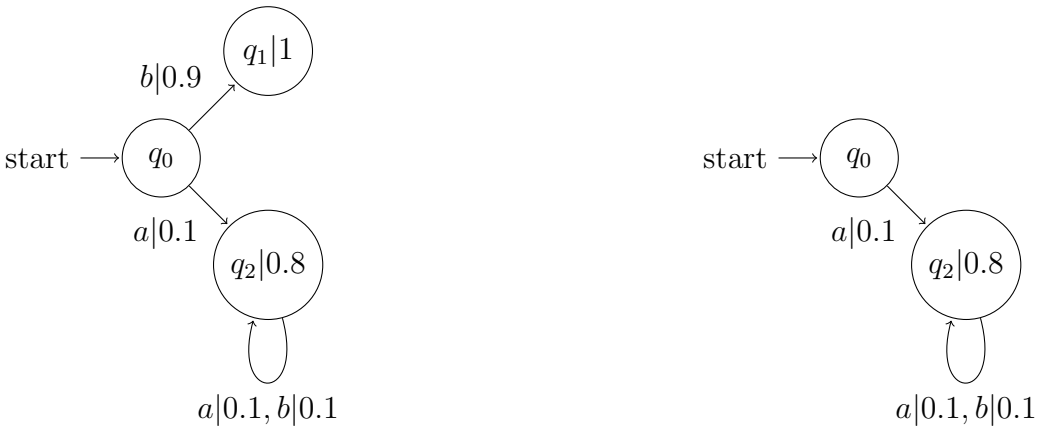
A consideration for hyper-minimisation on PDFAs as a compression technique is that it does not regard high probability words as important. In Figure 12 we can see that the string 'b' has a corresponding probability of 0.9. After hyper-minimisation we no longer accept any string 'b' because hyper-minimisation does not retain such finite strings. However, practical applications of PDFA may have a preference for retaining words with high probability. For these applications hyper-minimisation can end up producing an automaton with too much information loss.

Prior research into hyper-minimisation elaborated on **canonical languages**. These are a subset of languages that can be described by hyper-minimal automata. While this is open for further research, it remains an issue that the hyper-minimal PDFA can not always be obtained. Instead, further research could be to find and characterise the class of PDFAs that can always be hyper-minimised, and to find their corresponding set of languages.

This thesis can be expanded upon by translating its findings to Markov chains, where applicable by determinism. Before that, expanding hyper-minimisation to the non-deterministic setting, which does not yet see any advancements for classic finite automata or weighted finite automata, would be the place to start. There, the possibility that there exists a possibly more efficient hyper-minimisation algorithm without state merges for deterministic or non-deterministic FAs would also be of interest.

The algorithm for the hyper-minimisation of PDFAs can see more development by finding an **optimal** hyper-minimisation where the number of errors is as few as possible for a hyper-minimal automaton. The proposed probability redistribution algorithm makes use of a greedy pruning step, where all finalisation probabilities are removed before redistribution. The redistribution algorithm can be improved by only removing these finalisation probabilities when demanded for a legal merge.

The stochastic property is evident to be violated by state-reduction in certain scenarios. Then, further research could look to relax this constraint where allowed. For instance, it may be possible to form an almost-equivalent PDFA with probabilities that exceed one on some transitions, for as long as probabilistic distribution of the language remains less or equal to one.



(a) A minimal PDFA.

(b) The hyper-minimal counterpart.

Figure 12: Example of information loss of high probability after hyper-minimisation.

References

[BBCF24] Jean Berstel, Luc Boasson, Olivier Carton, and Isabelle Fagnot. Minimization of automata, 2024.

[BDN12] Frédérique Bassino, Julien David, and Cyril Nicaud. Average case analysis of moore's state minimization algorithm. *Algorithmica*, 63 (1-2), 2012.

[BGS09] Andrew Badr, Viliam Geffert, and Ian Shipman. Hyper-minimizing minimized deterministic finite state automata. *RAIRO-Theoretical Informatics and Applications*, 43(1):69–94, 2009.

[Bra68] Walter S Brainerd. The minimalization of tree automata. *Information and Control*, 13(5):484–491, 1968.

[Chu24] Wenjing Chu. *Automata Learning: from Probabilistic to Quantum*. PhD thesis, Leiden University, 2024.

[CIK93] Karel Culik II and Jarkko Kari. Image compression using weighted finite automata. *Computers & Graphics*, 17(3):305–313, 1993.

[dlHO13] Colin de la Higuera and Jose Oncina. Computing the most probable string with a probabilistic finite state machine. *11th international conference on finite-state methods and natural language processing*, 2013.

[DMPS18] Randal Douc, Eric Moulines, Pierre Priouret, and Philippe Soulier. *Markov chains*, volume 4. Springer, 2018.

[Eis03] Jason Eisner. Simpler and more general minimization for weighted finite-state automata. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 64–71, 2003.

[GKO15] Jan Friso Groote, Tim WDM Kouters, and Ammar Osaiweran. Specification guidelines to avoid the state space explosion problem. *Software Testing, Verification and Reliability*, 25(1):4–33, 2015.

[HM10] Markus Holzer and Andreas Maletti. An nlogn algorithm for hyper-minimizing a (minimized) deterministic automaton. *Theoretical computer science*, 411(38-39):3404–3413, 2010.

[HMU07] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 2007.

[Hop71] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971.

[Kim19] George Kim. The relationship between the chomsky hierarchy and automata. Part of The University of Chicago Mathematics REU, 2019.

[KW14] Stefan Kiefer and Björn Wachter. Stability and complexity of minimising probabilistic automata. In *International Colloquium on Automata, Languages, and Programming*, pages 268–279. Springer, 2014.

[Mal11] Andreas Maletti. Notes on hyper-minimization. In *AFL*, pages 34–49, 2011.

[Mar11] John C Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill, 4 edition, 2011.

[Moh09] Mehryar Mohri. Weighted automata algorithms. In *Handbook of weighted automata*, pages 213–254. Springer, 2009.

[MQ12] Andreas Maletti and Daniel Quernheim. Unweighted and weighted hyper-minimization. *International Journal of Foundations of Computer Science*, 23(06):1207–1225, 2012.

[MVV+18] Gunasekaran Manogaran, V Vijayakumar, Ramachandran Varatharajan, Priyan Malarvizhi Kumar, Revathi Sundarasekar, and Ching-Hsien Hsu. Machine learning based big data processing framework for cancer diagnosis using hidden markov model and gm clustering. *Wireless personal communications*, 102(3):2099–2116, 2018.

[PBW10] David Pfau, Nicholas Bartlett, and Frank Wood. Probabilistic deterministic infinite automata. *Advances in neural information processing systems*, 23, 2010.

[PR96] Fernando CN Pereira and Michael D Riley. Speech recognition by composition of weighted finite automata, 1996.

[PZ93] Amir Pnueli and Lenore D Zuck. Probabilistic verification. *Information and computation*, 103(1):1–29, 1993.

[Rab63] Michael O. Rabin. Probabilistic automata. *Information and control*, 6(3):230–245, 1963.

[RPB96] Giuseppe Riccardi, Roberto Pieraccini, and Enrico Bocchieri. Stochastic automata for language modeling. *Computer Speech & Language*, 10(4):265–293, 1996.

[Tar72] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.