5

Representing distributions over strings with automata and grammars

> If your experiment needs statistics, you ought to have done a better experiment.
>
> ***Ernest Rutherford***

> 'I think you're begging the question,' said Haydock, 'and I can see looming ahead one of those terrible exercises in probability where six men have white hats and six men have black hats and you have to work it out by mathematics how likely it is that the hats will get mixed up and in what proportion. If you start thinking about things like that, you would go round the bend. Let me assure you of that!'
>
> ***Agatha Christie**, The Mirror Crack'd from Side to Side (1962)*
>
>

Instead of defining a language as a set of strings, there are good reasons to consider the seemingly more complex idea of defining a distribution over strings. The distribution can be regular, in which case the strings are then generated by a probabilistic regular grammar or a probabilistic finite automaton. We are also interested in the special case where the automaton is deterministic.

Once distributions are defined, distances between the distributions and the syntactic objects they represent can be defined and in some cases they can be conveniently computed.

5.1 Distributions over strings

Given a finite alphabet Σ, the set $\Sigma^\star$ of all strings over Σ is enumerable, and therefore a distribution can be defined.

5.1.1 Distributions

A ***probabilistic language*** $\mathcal{D}$ is a probability distribution over $\Sigma^\star$.

The probability of a string $x \in \Sigma^\star$ under the distribution $\mathcal{D}$ is denoted by a positive value $Pr_{\mathcal{D}}(x)$ and the distribution $\mathcal{D}$ must verify

$$\sum_{x \in \Sigma^\star} Pr_{\mathcal{D}}(x) = 1.$$

If the distribution is modelled by some syntactic machine $\mathcal{A}$, the probability of x according to the probability distribution defined by $\mathcal{A}$ is denoted $Pr_{\mathcal{A}}(x)$. The distribution modelled by a machine $\mathcal{A}$ will be denoted $\mathcal{D}_{\mathcal{A}}$ and simplified to $\mathcal{D}$ if the context is not ambiguous.

If L is a language (a subset of $\Sigma^\star$), and $\mathcal{D}$ a distribution over $\Sigma^\star$, $Pr_{\mathcal{D}}(L) = \sum_{x \in L} Pr_{\mathcal{D}}(x)$.

Two distributions $\mathcal{D}$ and $\mathcal{D}'$ are equal (denoted by $\mathcal{D} = \mathcal{D}'$) $if_{def} \forall\, w \in \Sigma^\star$, $Pr_{\mathcal{D}}(w) = Pr_{\mathcal{D}'}(w)$.

In order to avoid confusion, even if we are defining languages (albeit probabilistic), we will reserve the notation L for sets of strings and denote the probabilistic languages as distributions, thus with symbol $\mathcal{D}$.

5.1.2 About the probabilities

In theory all the probabilistic objects we are studying could take arbitrary values. There are even cases where negative (or even complex) values are of interest! Nevertheless to fix things and to make sure that computational issues are taken into account we will take the simplified view that all probabilities will be rational numbers between 0 and 1, described by fractions. Their encoding then depends on the encoding of the two integers composing the fraction.

5.2 Probabilistic automata

We are concerned here with generating strings following distributions. If, in the non-probabilistic context, automata are used for parsing and recognising, they will be considered here as *generative* devices.

5.2.1 Probabilistic finite automata (PFA)

The first thing one can do is add probabilities to the non-deterministic finite automata:

Definition 5.2.1 (Probabilistic finite automaton (PFA)) A **probabilistic finite automaton (PFA)** is a tuple $\mathcal{A}=\langle \Sigma, Q, \mathbb{I}_{\mathbb{P}}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}} \rangle$, where:

- Q is a finite set of **states**; these will be labelled $q_1,\ldots,q_{|Q|}$ unless otherwise stated,
- Σ is the alphabet,
- $\mathbb{I}_{\mathbb{P}} : Q \to \mathbb{Q}^+ \cap [0, 1]$ (initial-state probabilities),
- $\mathbb{F}_{\mathbb{P}} : Q \to \mathbb{Q}^+ \cap [0, 1]$ (final-state probabilities),
- $\delta_{\mathbb{P}} : Q \times (\Sigma \cup \{\lambda\}) \times Q \to \mathbb{Q}^+$ is a transition function; the function is complete: $\delta_{\mathbb{P}}(q, a, q') = 0$ can be interpreted as 'no transition from q to q' labelled with a'. We will also denote (q, a, q', P) instead of $\delta_{\mathbb{P}}(q, a, q') = P$ where P is a probability.

$\mathbb{I}_{\mathbb{P}}$, $\delta_{\mathbb{P}}$ and $\mathbb{F}_{\mathbb{P}}$ are functions such that:

$$\sum_{q \in Q} \mathbb{I}_{\mathbb{P}}(q) = 1,$$

and $\forall q \in Q$,

$$\mathbb{F}_{\mathbb{P}}(q) + \sum_{a \in \Sigma \cup \{\lambda\},\ q' \in Q} \delta_{\mathbb{P}}(q, a, q') = 1.$$

In contrast with the definition of NFA, there are no accepting (or rejecting) states. As will be seen in the next section, the above definition of automata describes models that are generative in nature. Yet in the classical setting automata are usually introduced in order to parse strings rather than to generate them, so there may be some confusion here. The advantages of the definition are nevertheless that there is a clear correspondence with probabilistic left (or right) context-free grammars. But on the other hand if we were requiring a finite state machine to parse a string and somehow come up with some probability that the string belongs or not to a given language, we would have to turn to another type of machine.

Figure 5.2 shows a ***graphical representation*** of a PFA with four states, $Q = \{q_1, q_2, q_3, q_4\}$, two initial states, q_1 and q_2, with respective probabilities of being chosen of 0.4 and 0.6, and a two-symbol alphabet, $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. The numbers on the edges are the transition probabilities. The initial and final probabilities are drawn inside the state (as in Figure 5.1) before and after the name of the state. The transitions whose weights are 0 are not drawn. We will introduce this formally later, but we will also say that the automaton ***respects the following set of constraints***: $\{(q_1, \mathtt{a}, q_2),\ (q_1, \mathtt{b}, q_1),\ (q_2, \lambda, q_3),\ (q_2, \mathtt{b}, q_4),\ (q_3, \mathtt{b}, q_3),\ (q_3, \mathtt{b}, q_4),\ (q_4, \mathtt{a}, q_1),\ (q_4, \mathtt{a}, q_2)\}$. This means that these are the only transitions with non-null weight in the automaton.

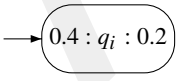


Fig. 5.1. Graphical representation of a state q_i. $\mathbb{I}_{\mathbb{P}}(q_i) = 0.4$, $\mathbb{F}_{\mathbb{P}}(q_i) = 0.2$.

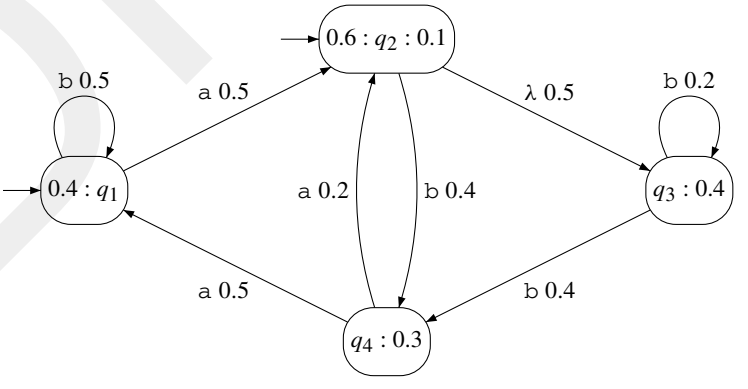


Fig. 5.2. Graphical representation of a PFA.

The above definition allows λ-transitions and λ-loops. In Figure 5.2, there is a λ-transition between state q_2 and state q_3. Such λ-transitions make things difficult for parsing. We show in Section 5.2.4 that λ-transitions and λ-loops can actually be removed in polynomial time without changing the distribution. When needing to differentiate we shall call the former λ-PFA, and the latter (when λ-transitions are forbidden) λ-free PFA.

Definition 5.2.2 For any λ-PFA $\mathcal{A} = \langle \Sigma, Q, \mathbb{I}_{\mathbb{P}}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}} \rangle$:

- a **λ-transition** is any transition labelled by λ,
- a **λ-loop** is a transition of the form (q, λ, q, P),
- a **λ-cycle** is a sequence of λ-transitions from $\delta_{\mathbb{P}}$ with the same starting and ending state: $(q_{i_1}, \lambda, q_{i_2}, P_1) \dots (q_{i_j}, \lambda, q_{i_{j+1}}, P_j) \dots (q_{i_k}, \lambda, q_{i_1}, P_k)$.

*5.2.2 Deterministic probabilistic finite-state automata (*DPFA*)*

As in the non-probabilistic case we can restrict the definitions in order to make parsing deterministic. In this case 'deterministic' means that there is only one way to generate each string at each moment, i.e. that in each state, for each symbol, the next state is unique.

The main tool we will use to generate distributions is deterministic probabilistic finite (state) automata. These will be the probabilistic counterparts of the deterministic finite automata.

Definition 5.2.3 (Deterministic probabilistic finite automata) A PFA $\mathcal{A} = \langle \Sigma, Q, \mathbb{I}_{\mathbb{P}}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}} \rangle$ is a **deterministic probabilistic finite automaton (DPFA)** *if*$_{def}$

- $\exists q_1 \in Q$ (**unique initial state**) such that $\mathbb{I}_{\mathbb{P}}(q_1) = 1$;
- $\delta_{\mathbb{P}} \subseteq Q \times \Sigma \times Q$ (no λ-transitions);
- $\forall q \in Q,\ \forall a \in \Sigma,\ |\{q' : \delta_{\mathbb{P}}(q, a, q') > 0\}| \leq 1$.

In a DPFA, a transition (q, a, q', P) is completely defined by q and a. The above definition is cumbersome in this case and we will associate with a DPFA a non-probabilistic transition function:

Definition 5.2.4 Let $\mathcal{A}$ be a DPFA. $\delta_{\mathcal{A}} : Q \times \Sigma \to Q$ is **the transition function** with $\delta_{\mathcal{A}}(q, a) = q' :\ \delta_{\mathbb{P}}(q, a, q') \neq 0$.

This function is extended (as in Chapter 4) in a natural way to strings:

$$\delta_{\mathcal{A}}(q, \lambda) = q$$
$$\delta_{\mathcal{A}}(q, a \cdot u) = \delta_{\mathcal{A}}(\delta_{\mathcal{A}}(q, a), u).$$

The probability function also extends easily in the deterministic case to strings:

$$\delta_{\mathbb{P}}(q, \lambda, q) = 1$$
$$\delta_{\mathbb{P}}(q, a \cdot u, q') = \delta_{\mathbb{P}}(q, a, \delta_{\mathcal{A}}(q, a)) \cdot \delta_{\mathbb{P}}(\delta_{\mathcal{A}}(q, a), u, q').$$

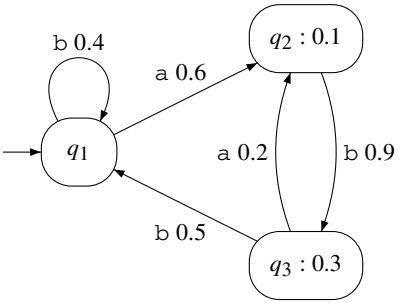


Fig. 5.3. Graphical representation of a DPFA.

But, in the deterministic case, one should note that $\delta_{\mathbb{P}}(q, \lambda, q) = 1$ is not referring to the introduction of a λ-loop with weight 1 (and therefore the loss of determinism), but to the fact that the probability of having λ as a prefix is 1.

The size of a DPFA depends on the number n of states, the size $|\Sigma|$ of the alphabet and the number of bits needed to encode the probabilities. This is also the case when dealing with a PFA or λ-PFA.

In the DPFA depicted in Figure 5.3, the initial probabilities need not be represented: there is a unique initial state (here q_1) with probability of being initial equal to 1.

5.2.3 Parsing with a PFA

PFAs are adequate probabilistic machines to generate strings of finite length. Given a PFA (or λ-PFA) $\mathcal{A}$, the process is described in Algorithm 5.1. For this we suppose we have two functions:

Algorithm 5.1: Generating a string with a PFA.

Data: a PFA $\mathcal{A} = \langle \Sigma, Q, \mathbb{I}_{\mathbb{P}}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}} \rangle$, $\mathbf{Random}_Q$, $\mathbf{Random}_{\mathbb{I}_{\mathbb{P}}}$
Result: a string x
$x \leftarrow \lambda$;
$q \leftarrow \mathbf{Random}_{\mathbb{I}_{\mathbb{P}}}(\mathcal{A})$;
$(a, q') \leftarrow \mathbf{Random}_Q(q, \mathcal{A})$;
while $a \neq H$ **do** `/* generate another symbol */`
  $x \leftarrow x \cdot a$;
  $q \leftarrow q'$;
  $(a, q') \leftarrow \mathbf{Random}_Q(q, \mathcal{A})$
end
return x

- **Random**$_Q(q, \mathcal{A})$ takes a state q from a PFA $\mathcal{A}$ and returns a pair (a, q') drawn according to the distribution at state q in PFA $\mathcal{A}$ from all pairs in $\Sigma \times Q$ including the pair (H, q), where H is a special character not in Σ, whose meaning is 'halt'.
- **Random**$_{\mathbb{I}_\mathbb{P}}(\mathcal{A})$, which returns an initial state depending on the distribution defined by $\mathbb{I}_\mathbb{P}$.

Inversely, given a string and a PFA, how do we compute the probability of the string? We will in this section only work with λ-free automata. If the automaton contains λ-transitions, parsing requires us to compute the probabilities of moving freely from one state to another. But in that case, as this in itself is expensive, the elimination of the λ-transitions (for example with the algorithm from Section 5.2.4) should be considered. There are two classical algorithms that compute the probability of a string by dynamic programming.

The FORWARD algorithm (Algorithm 5.2) computes $Pr_\mathcal{A}(q_s, w)$, the probability of generating string w and being in state q_s after having done so. By then multiplying by the final probabilities (Algorithm 5.3) and summing up, we get the actual probability of the string. The BACKWARD algorithm (Algorithm 5.4) works in a symmetrical way. It computes Table B, whose entry B[0][s] is $Pr_\mathcal{A}(w|q_s)$, the probability of generating string

Algorithm 5.2: FORWARD.

Data: a PFA $A = \langle \Sigma, Q, \mathbb{I}_\mathbb{P}, \mathbb{F}_\mathbb{P}, \delta_\mathbb{P} \rangle$, a string $x = a_1 a_2 \cdots a_n$
Result: the probabilities F$[n][s] = Pr_\mathcal{A}(x, q_s)$

```
for j : 1 ≤ j ≤ |Q| do                                   /* Initialise */
    F[0][j] ← I_P(q_j);
    for i : 1 ≤ i ≤ n do F[i][j] ← 0
end
for i : 1 ≤ i ≤ n do
    for j : 1 ≤ j ≤ |Q| do
        for k : 1 ≤ k ≤ |Q| do
            F[i][j] ←F[i][j]+F[i − 1][k] · δ_P(q_k, a_i, q_j)
        end
    end
end
return F
```

Algorithm 5.3: Computing the probability of a string with FORWARD.

Data: the probabilities F$[n][s] = Pr_\mathcal{A}(x, q_s)$, a PFA $\mathcal{A}$
Result: the probability T$=Pr_\mathcal{A}(x)$

```
T← 0;
for j : 1 ≤ j ≤ |Q| do T← T+F[n][j] · F_P[j] ;
return T
```

Algorithm 5.4: BACKWARD

Data: a PFA $\mathcal{A} = \langle \Sigma, Q, \mathbb{I}_\mathbb{P}, \mathbb{F}_\mathbb{P}, \delta_\mathbb{P} \rangle$, a string $x = a_1 a_2 \cdots a_n$
Result: the probabilities $\mathrm{B}[0][s] = Pr_\mathcal{A}(x|q_s)$
for $j : 1 \leq j \leq |Q|$ **do** /* Initialise */
  $\mathrm{B}[n][j] \leftarrow \mathbb{F}_\mathbb{P}[j]$;
  for $i : 1 \leq i \leq n$ **do** $\mathrm{B}[i][j] \leftarrow 0$
end
for $i : n-1 \geq i \geq 0$ **do**
  for $j : 1 \leq j \leq |Q|$ **do**
    for $k : 1 \leq k \leq |Q|$ **do**
      $\mathrm{B}[i][j] \leftarrow \mathrm{B}[i][j] + \mathrm{B}[i+1][k] \cdot \delta_\mathbb{P}(q_j, a_i, q_k)$
    end
  end
end
return B

Algorithm 5.5: Computing the probability of a string with BACKWARD.

Data: the probabilities $\mathrm{B}[0][s] = Pr_\mathcal{A}(x|q_s)$, a PFA $\mathcal{A}$
Result: the probability $Pr_\mathcal{A}(x)$
$\mathrm{T} \leftarrow 0$;
for $j : 1 \leq j \leq |Q|$ **do** $\mathrm{T} \leftarrow \mathrm{T} + \mathrm{B}[0][j] \cdot \mathbb{I}_\mathbb{P}[j]$;
return T

w when starting in state q_s (which has initial probability 1). If we then run Algorithm 5.5 to this we also get the value $Pr_\mathcal{A}(w)$.

These probabilities will be of use when dealing with parameter estimation questions in Chapter 17.

Another parsing problem, that of finding the most probable path in the PFA that generates a string w, is computed through the VITERBI algorithm (Algorithm 5.6).

The computation of Algorithms 5.2, 5.4 and 5.6 have a time complexity in $\mathcal{O}(|x| \cdot |Q|^2)$. A slightly different implementation allows us to reduce this complexity by only visiting the states that are really successors of a given state. A vector representation allows us to make the space complexity linear.

In the case of DPFAs, the algorithms are simpler than for non-deterministic PFAs. In this case, the computation cost of Algorithm 5.6 is in $\mathcal{O}(|x|)$, that is, the computational cost does not depend on the number of states since at each step the only possible next state is computed with a cost in $\mathcal{O}(1)$.

Algorithm 5.6: VITERBI.

Data: a PFA $\mathcal{A} = \langle \Sigma, Q, \mathbb{I}_{\mathbb{P}}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}} \rangle$, a string $x = a_1 a_2 \cdots a_n$
Result: a sequence of states bestpath$= q_{p_0} \ldots q_{p_n}$ reading x and maximising:
$\mathbb{I}_{\mathbb{P}}(q_{p_0}) \cdot \delta_{\mathbb{P}}(q_{p_0}, a_1, q_{p_1}) \cdot \delta_{\mathbb{P}}(q_{p_1}, a_2, q_{p_2}) \ldots \delta_{\mathbb{P}}(q_{p_{n-1}}, a_n, q_{p_n}) \cdot \mathbb{F}_{\mathbb{P}}(q_{p_n})$

```
for j : 1 ≤ j ≤ |Q| do                                   /* Initialise */
    V[0][j] ← I_P(q_j);
    Vpath[0][j] ← λ;
    for i : 1 ≤ i ≤ n do
        V[i][j] ← 0;
        Vpath[i][j] ← λ
    end
end
for i : 1 ≤ i ≤ n do
    for j : 1 ≤ j ≤ |Q| do
        for k : 1 ≤ k ≤ |Q| do
            if V[i][j] < V[i − 1][k] · δ_P(q_k, a_i, q_j) then
                V[i][j] ← V[i − 1][k] · δ_P(q_k, a_i, q_j);
                Vpath[i][j] ← Vpath[i − 1][j] · q_j
            end
        end
    end
end
bestscore← 0;          /* Multiply by the halting probabilities */
for j : 1 ≤ j ≤ |Q| do
    if V[n][j] · F_P[j] >bestscore then
        bestscore← V[n][j] · F_P[j];
        bestpath←Vpath[n][j]
    end
end
return bestpath
```

We compute the probability of string ab in the automaton of Figure 5.4:

$$\begin{aligned} Pr_{\mathcal{A}}(\mathtt{ab}) &= \mathbb{I}_{\mathbb{P}}(q_1) \cdot \delta_{\mathbb{P}}(q_1, \mathtt{a}, q_2) \cdot \delta_{\mathbb{P}}(q_2, \mathtt{b}, q_3) \cdot \mathbb{F}_{\mathbb{P}}(q_3) \\ &\quad + \mathbb{I}_{\mathbb{P}}(q_2) \cdot \delta_{\mathbb{P}}(q_2, \mathtt{a}, q_4) \cdot \delta_{\mathbb{P}}(q_4, \mathtt{b}, q_2) \cdot \mathbb{F}_{\mathbb{P}}(q_2) \\ &\quad + \mathbb{I}_{\mathbb{P}}(q_2) \cdot \delta_{\mathbb{P}}(q_2, \mathtt{a}, q_4) \cdot \delta_{\mathbb{P}}(q_4, \mathtt{b}, q_1) \cdot \mathbb{F}_{\mathbb{P}}(q_1) \\ &= 0.4 \cdot 0.5 \cdot 0.2 \cdot 0.5 + 0.6 \cdot 0.2 \cdot 0.2 \cdot 0.6 + 0.6 \cdot 0.2 \cdot 0.5 \cdot 0 \\ &= 0.0344. \end{aligned}$$

Table 5.1. *Details of the computation of B and F for string* ab *on Automaton 5.4 by Algorithms* BACKWARD *and* FORWARD.

State	B	F
q_1	B[2][1] $= Pr(\lambda\|0) = 0$	F[0][1] $= Pr(\lambda, 1) = 0.4$
q_2	B[2][2] $= Pr(\lambda\|1) = 0.6$	F[0][2] $= Pr(\lambda, 2) = 0.6$
q_3	B[2][3] $= Pr(\lambda\|2) = 0.5$	F[0][3] $= Pr(\lambda, 3) = 0$
q_4	B[2][4] $= Pr(\lambda\|3) = 0.3$	F[0][4] $= Pr(\lambda, 4) = 0$
q_1	B[1][1] $= Pr(\mathtt{b}\|0) = 0$	F[1][1] $= Pr(\mathtt{a}, 1) = 0$
q_2	B[1][2] $= Pr(\mathtt{b}\|1) = 0.1$	F[1][2] $= Pr(\mathtt{a}, 2) = 0.2$
q_3	B[1][3] $= Pr(\mathtt{b}\|2) = 0.06$	F[1][3] $= Pr(\mathtt{a}, 3) = 0$
q_4	B[1][4] $= Pr(\mathtt{b}\|3) = 0.12$	F[1][4] $= Pr(\mathtt{a}, 4) = 0.12$
q_1	B[0][1] $= Pr(\mathtt{ab}\|0) = 0.05$	F[2][1] $= Pr(\mathtt{ab}, 1) = 0.06$
q_2	B[0][2] $= Pr(\mathtt{ab}\|1) = 0.024$	F[2][2] $= Pr(\mathtt{ab}, 2) = 0.024$
q_3	B[0][3] $= Pr(\mathtt{ab}\|2) = 0.018$	F[2][3] $= Pr(\mathtt{ab}, 3) = 0.04$
q_4	B[0][4] $= Pr(\mathtt{ab}\|3) = 0$	F[2][4] $= Pr(\mathtt{ab}, 4) = 0$

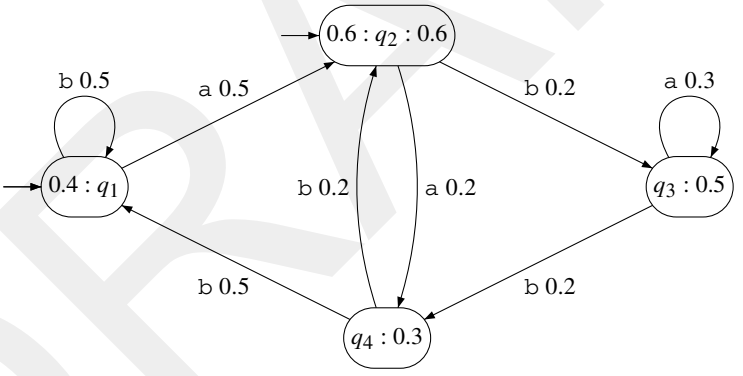


Fig. 5.4. A PFA $\mathcal{A}$, $Pr_{\mathcal{A}}(\mathtt{ab}) = 0.0344$.

We can also trace the BACKWARD and FORWARD computations in Table 5.1. We obtain:

$$0.05 \cdot 0.4 + 0.024 \cdot 0.6 + 0.018 \cdot 0 + 0 \cdot 0 =$$
$$0.06 \cdot 0 + 0.024 \cdot 0.6 + 0.04 \cdot 0.5 + 0 \cdot 0.3 = 0.0264.$$

We conclude this section by defining classes of string distributions on the basis of the corresponding generating automata.

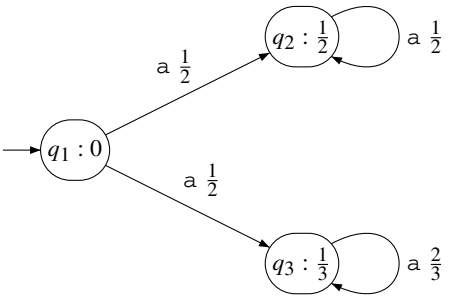


Fig. 5.5. A regular distribution that is not deterministic.

Definition 5.2.5 A distribution is **regular** *if*$_{def}$ it can be generated by some PFA. We denote by $\mathcal{REG}_{\mathcal{P}}(\Sigma)$ the class of all regular distributions (over alphabet Σ).

Definition 5.2.6 A distribution is **regular deterministic** *if*$_{def}$ it can be generated by some DPFA. We denote by $\mathcal{DET}_{\mathcal{P}}(\Sigma)$ the class of all regular deterministic distributions (over alphabet Σ).

Definition 5.2.7 Two PFAs are **equivalent** *if*$_{def}$ they generate the same distribution.

From the definitions of PFA and DPFA the following hierarchy is obtained:

Proposition 5.2.1 *A regular deterministic distribution is also a regular distribution. There exist distributions which are regular but not regular deterministic.*

Proof It can be checked that the distribution defined by the PFA from Figure 5.5 is not regular deterministic. □

5.2.4 Eliminating λ-transitions in a PFA

The algorithms from the previous section were able to parse whenever the PFA contained no λ-transitions. But even if a specific parser using λ-transitions can be built, it is tedious to compute with it or to generate strings in a situation where the λ-transitions may even be more frequent than the other transitions. We prove in this section that λ-transitions are not necessary in probabilistic finite automata and that there is no added power in automata that use λ-transitions compared to those that don't. To do so we propose an algorithm that takes a λ-PFA as input and returns an equivalent PFA with no more states, but where the (rational) probabilities may require a polynomial number only of extra bits for encoding.

Remember that a λ-PFA can not only have λ-transitions (including λ-loops), but also various initial states. Thus $\mathbb{I}_{\mathbb{P}}$ is a function $Q \to \mathbb{Q}^+$, such that $\sum_{q \in Q} \mathbb{I}_{\mathbb{P}}(q) = 1$.

Theorem 5.2.2 *Given a* λ-PFA $\mathcal{A}$ *representing distribution* $\mathcal{D}_{\mathcal{A}}$*, there exists a λ-free* PFA $\mathcal{B}$ *such that* $\mathcal{D}_{\mathcal{A}} = \mathcal{D}_{\mathcal{B}}$*. Moreover* $\mathcal{B}$ *is of total size at most n times the total size of* $\mathcal{A}$*, where n is the number of states in* $\mathcal{A}$*.*

Algorithm 5.7: Transforming the λ-PFA into a λ-PFA with just one initial state.

Input: a λ-PFA : $\langle \Sigma, Q, \mathbb{I}_{\mathbb{P}}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}} \rangle$
Output: a λ-PFA : $\langle \Sigma, Q \cup \{q_{new}\}, \mathbb{I}_{\mathbb{P}}{}', \mathbb{F}_{\mathbb{P}}{}', \delta_{\mathbb{P}}{}' \rangle$ with one initial state
$Q' \leftarrow Q \cup \{q_{new}\}$;
for $q \in Q$ **do**
  $\delta_{\mathbb{P}}{}'(q_{new}, \lambda, q) \leftarrow \mathbb{I}_{\mathbb{P}}(q)$;
  $\mathbb{I}_{\mathbb{P}}{}'(q) \leftarrow 0$
end
$\mathbb{I}_{\mathbb{P}}{}'(q_{new}) \leftarrow 1$;
$\mathbb{F}_{\mathbb{P}}{}'(q_{new}) \leftarrow 0$;
return $\langle \Sigma, Q \cup \{q_{new}\}, \mathbb{I}_{\mathbb{P}}{}', \mathbb{F}_{\mathbb{P}}{}', \delta_{\mathbb{P}}{}' \rangle$

Proof To convert $\mathcal{A}$ into an equivalent PFA $\mathcal{B}$ there are two steps. The starting point is a PFA $\mathcal{A} = \langle \Sigma, Q, \mathbb{I}_{\mathbb{P}}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}} \rangle$ where the set of states Q is labelled by numbers. We will update this numbering as we add more states.

Step 1: If there is more than one initial state, add a new initial state and λ-transitions from this state to each of the previous initial states, with probability equal to that of the state being initial. Notice that this step can be done in polynomial time and that the resulting automaton contains at most one more state than the initial one. This is described by Algorithm 5.7.

Step 2: Algorithm 5.8 iteratively removes a λ-loop if there is one, and if not, the λ-transition with maximal extremity. In a finite number of steps, the algorithm terminates.

To prove that in a finite number of steps all λ-transitions are removed, we first associate with any automaton $\mathcal{A}$ the value $\mu(\mathcal{A})$ corresponding to the largest number of a state in which a λ-transition ends.

$$\mu(\mathcal{A}) = \max\{i : q_i \in Q \text{ and } \delta_{\mathbb{P}}(q, \lambda, q_i) \neq 0\}.$$

Now let $\rho(\mathcal{A})$ be the number of λ-transitions ending in $q_{\mu(\mathcal{A})}$.

$$\rho(\mathcal{A}) = \big|\{q \in Q : \delta_{\mathbb{P}}(q, \lambda, q_{\mu(\mathcal{A})}) \neq 0\}\big|.$$

Finally, the value $v(\mathcal{A}) = \langle \mu(\mathcal{A}), \rho(\mathcal{A}) \rangle$ will decrease at each step of the algorithm, thus ensuring the termination and the convergence. In Figure 5.6, we have $\mu(\mathcal{A}) = 3$, $\rho(\mathcal{A}) = 2$ and $v(\mathcal{A}) = \langle 3, 2 \rangle$.

Algorithm 5.8 takes the current PFA and eliminates a λ-loop if there is one. If not it chooses a λ-transition ending in the state with largest number and eliminates it.

One can notice that in Algorithm 5.8, new λ-loops can appear when such a λ-transition is eliminated, but that is no problem because they will only appear in states of smaller index than the current state that is being considered.

Algorithm 5.8: Eliminating λ-transitions.

Input: a $\lambda-$ PFA : $\langle \Sigma, Q, \{q_1\}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}} \rangle$ with only one initial state
Output: a λ-free PFA : $\langle \Sigma, Q, \{q_1\}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}} \rangle$
while *there still are λ-transitions* **do**
  if *there exists a λ-loop* (q, λ, q, P) **then**
    for *all transitions* (q, a, q') $((a, q') \neq (\lambda, q))$ **do**
      $\delta_{\mathbb{P}}(q, a, q') \leftarrow \delta_{\mathbb{P}}(q, a, q') \cdot \frac{1}{1-\delta_{\mathbb{P}}(q,\lambda,q)}$
    end
    $\mathbb{F}_{\mathbb{P}}(q) \leftarrow \mathbb{F}_{\mathbb{P}}(q) \cdot \frac{1}{1-\delta_{\mathbb{P}}(q,\lambda,q)}$;
    $\delta_{\mathbb{P}}(q, \lambda, q) \leftarrow 0$
  else `/* there are no λ-loops */`
    let (q, λ, q_m) be a λ-transition with m maximal;
    foreach $(q_m, \lambda, q_n, P_\lambda)$ **do** `/*` $n < m$ `*/`
      $\delta_{\mathbb{P}}(q, \lambda, q') \leftarrow \delta_{\mathbb{P}}(q, \lambda, q_n) + \delta_{\mathbb{P}}(q, \lambda, q_m) \cdot \delta_{\mathbb{P}}(q_m, \lambda, q_n)$
    end
    foreach (q_m, a, q_n, P_a) **do** `/*` $a \in \Sigma$ `*/`
      $\delta_{\mathbb{P}}(q, a, q_n) \leftarrow \delta_{\mathbb{P}}(q, a, q_n) + \delta_{\mathbb{P}}(q, \lambda, q_m) \cdot \delta_{\mathbb{P}}(q_m, a, q_n)$
    end
    $\mathbb{F}_{\mathbb{P}}(q) \leftarrow \mathbb{F}_{\mathbb{P}}(q) + \delta_{\mathbb{P}}(q, \lambda, q_m) \cdot \mathbb{F}_{\mathbb{P}}(q_m)$;
    $\delta_{\mathbb{P}}(q, \lambda, q_m) \leftarrow 0$
  end
end
return $\langle \Sigma, Q, \{q_1\}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}} \rangle$

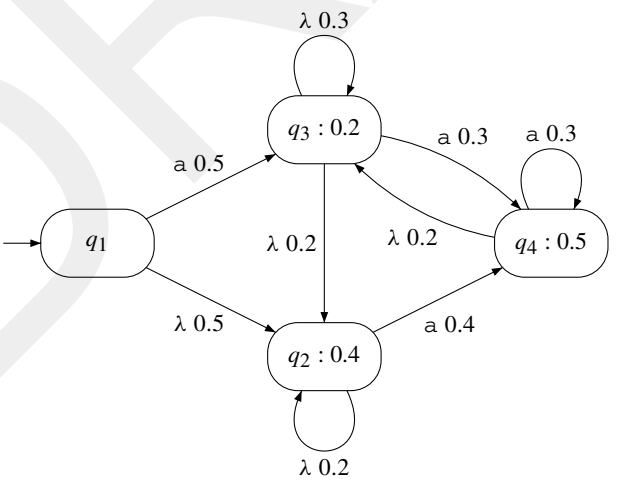


Fig. 5.6. A PFA.

Therefore the quantity $v(\mathcal{A})$ decreases (for the lexicographic order over $\mathbb{N}^2$) at each round of the algorithm, since a new λ-transition only appears as a combination of a (q, λ, q_m) and a (q_m, λ, q_n) and $n < m$ with $m = \mu(\mathcal{A})$.

As clearly this can only take place a finite number of times, the algorithm converges.

Then we prove that at each step $v(\mathcal{A})$ does not increase (for the lexicographic order), and that there can only be a finite number of consecutive steps where $v(\mathcal{A})$ remains equal. Summarising, at each step one of the following holds:

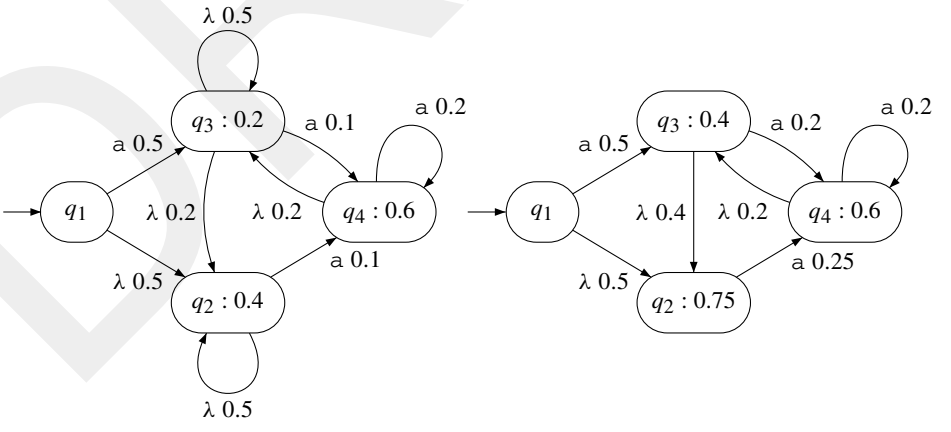
- A λ-loop is erased. Then $v(\mathcal{A})$ is left untouched because no new λ-transition is introduced, but the number of λ-loops is bounded by the number of states of the PFA. So only a finite number of λ-loop elimination steps can be performed before having no λ-loops left.
- A λ-transition (that is not a loop) is replaced. This transition is a (q, λ, q_m, P) with $\mu(\mathcal{A}) = m$. Therefore only λ-transitions with terminal vertex of index smaller than m can be introduced. So $v(\mathcal{A})$ diminishes. □

Also, clearly, if the probabilities are rational, they remain so.

A run of the algorithm. We suppose the λ-PFA contains just one initial state and is represented in Figure 5.7(a). First, the λ-loops at states q_2 and q_3 are eliminated and several transitions are updated (Figure 5.7(b)).

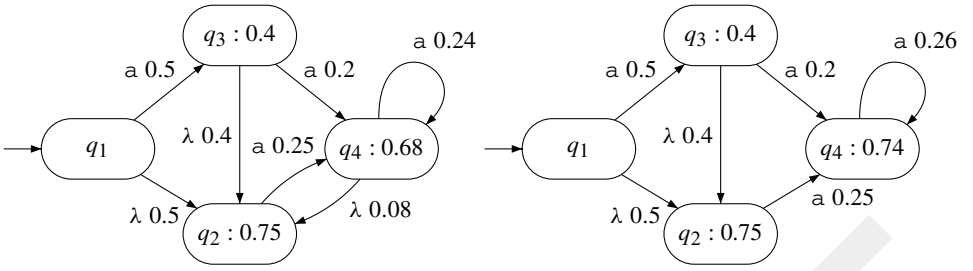
The value of $v(\mathcal{A})$ is $\langle 3, 1\rangle$. At that point the algorithm eliminates transition (q_4, λ, q_3) because $\mu(\mathcal{A}) = 3$. The resulting PFA is represented in Figure 5.8(a). The new value of $v(\mathcal{A})$ is $\langle 2, 3\rangle$ which, for the lexicographic order, is less than the previous value.

The PFA is represented in Figure 5.8(b). The new value of $v(\mathcal{A})$ is $\langle 2, 2\rangle$. Then transition (q_3, λ, q_2) is eliminated resulting in the PFA represented in Figure 5.9(a) whose value



(a) A PFA. (b) PFA after eliminating λ-loops.

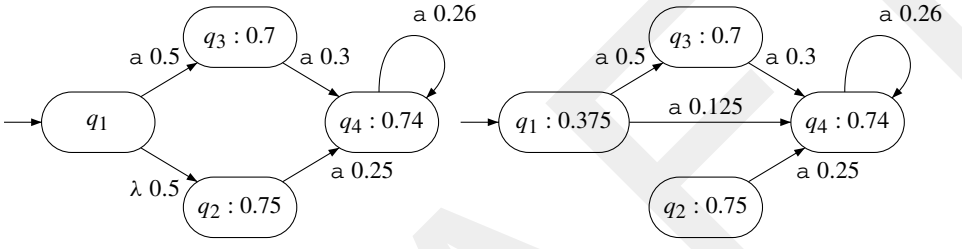
Fig. 5.7.



(a) PFA after eliminating transition (q_4, λ, q_3).

(b) PFA after eliminating transition (q_4, λ, q_2).

Fig. 5.8.



(a) PFA after eliminating transition (q_3, λ, q_2).

(b) PFA after eliminating transition (q_1, λ, q_2).

Fig. 5.9.

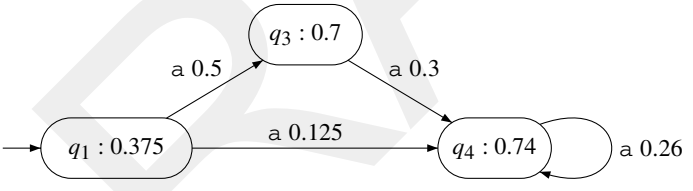


Fig. 5.10. PFA after simplification.

$v(\mathcal{A})$ is $\langle 2, 1\rangle$. Finally the last λ-transition is removed and the result is represented in Figure 5.9(b). At this point, since state q_2 is no longer reachable, the automaton can be pruned as represented in Figure 5.10.

Complexity issues. The number of states in the resulting automaton is identical to the initial one. The number of transitions is bounded by $|Q|^2 \cdot |\Sigma|$. But each multiplication of probabilities can result in the summing of the number of bits needed to encode each probability. Since the number of λ-loops to be removed on one path is bounded by n, this also gives a polynomial bound to the size of the resulting automaton.

Conclusion. Automatic construction of a PFA (through union operations for example) will lead to the introduction of λ-transitions or multiple initial states.

Algorithms 5.7 and 5.8 remove λ-transitions, and an equivalent PFA with just one initial state can be constructed for a relatively small cost.

It should be remembered that not all non-determinism can be eliminated, as it is known that PFAs are strictly more powerful than their deterministic counterparts (see Proposition 5.2.1, page 95).

5.3 Probabilistic context-free grammars

One can also add probabilities to grammars, but the complexity increases. We therefore only survey here some of the more elementary definitions.

Definition 5.3.1 A **probabilistic context-free grammar (PCFG)** G is a quintuple $< \Sigma, V, R, P, N_1 >$ where Σ is a finite alphabet (of terminal symbols), V is a finite alphabet (of variables or non-terminals), $R \subset V \times (\Sigma \cup V)^*$ is a finite set of production rules, $P : R \to \mathbb{R}^+$ is the probability function, and N_1 $(\in V)$ is the axiom.

As in the case of automata, we will restrict ourselves to the case where the probabilities are rational (in $\mathbb{Q} \cap [0, 1]$).

Given a PCFG G, a string w, and a left-most derivation $d = \alpha_0 \Rightarrow \ldots \Rightarrow \alpha_k$ where

$$\begin{aligned}
&\alpha_0 = N_{i_0} = N_1,\\
&\alpha_k = w,\\
&\alpha_j = l_j N_{i_j} \gamma \text{ with } l_j \in \Sigma^\star, \gamma \in (\Sigma \cup V)^*, \text{and } j < k,\\
&\alpha_{j+1} = l_j \beta_{i_j} \gamma \text{ with } (N_{i_j}, \beta_{i_j}) \in R
\end{aligned}$$

we define the weight of derivation d as: $Pr_{G,d}(w) = \prod_{0 \le j < k} P(N_{i_j}, \beta_{i_j})$ where (N_{i_j}, β_{i_j}) is the rule used to rewrite α_j into α_{j+1}. Notice that it is essential to count only the left-most derivations in order not to count the same derivation various times. The quantity $Pr_{G,d}(w)$ is the probability of generating string w **and** of doing this using derivation d.

Now we sum over all (left-most) derivations: $Pr_G(w) = \sum_d Pr_{G,d}(w)$.

Example 5.3.1 Consider the grammar $G = <\{\mathtt{a}, \mathtt{b}\}, \{N_1\}, R, P, N_1>$ with $R = \{(N_1,\ \mathtt{a}N_1\mathtt{b}), (N_1,\ \mathtt{a}N_1), (N_1,\ \lambda)\}$ and the probabilities of the rules given by $P(N_1,\ \lambda) = \frac{1}{6}$, $P(N_1,\ \mathtt{a}N_1\mathtt{b}) = \frac{1}{2}$ and $P(N_1,\ \mathtt{a}N_1) = \frac{1}{3}$.

Then string $\mathtt{aab}$ can be generated through $d_1 = N_1 \Rightarrow \mathtt{a}N_1\mathtt{b} \Rightarrow \mathtt{aa}N_1\mathtt{b} \Rightarrow \mathtt{aab}$ and $d_2 = N_1 \Rightarrow \mathtt{a}N_1 \Rightarrow \mathtt{aa}N_1\mathtt{b} \Rightarrow \mathtt{aab}$. Both derivations have probability $Pr_{G,d_i}(\mathtt{aab}) = \frac{1}{36}$, and therefore $Pr_G(\mathtt{aab}) = \frac{1}{18}$.

As with PFAs, we can give the algorithms allowing us by dynamic programming to effectively sum over all the paths. Instead of using BACKWARD and FORWARD, we define INSIDE and OUTSIDE, two algorithms that compute respectively $Pr_G(w|N)$, the

Algorithm 5.9: INSIDE.

Data: a PCFG $G =< \Sigma, V, R, P, N_1 >$ in quadratic normal form, with
$V = \{N_1, \ldots, N_{|V|}\}$, a string $x = a_1 a_2 \cdots a_n$,
Result: The probabilities $\mathrm{I}[i][j][k] = Pr_G(a_j \cdots a_k | N_i)$

```
for j : 1 ≤ j ≤ n do                                   /* Initialise */
    for k : j ≤ k ≤ n do
        for i : 1 ≤ i ≤ |V| do  I[i][j][k] ← 0
    end
end
for (N_i, b) ∈ R do
    for j : 1 ≤ j ≤ n do
        if a_j = b then I[i][j][j] ← P(N_i, b)
    end
end
for m : 1 ≤ m ≤ n − 1 do
    for j : 1 ≤ j ≤ n − m do
        for k : j ≤ k ≤ j + m − 1 do
            for (N_i, N_{i_1} N_{i_2}) ∈ R do  I[i][j][j + m] ←
            I[i][j][j + m]+I[i_1][j][k]·I[i_2][k + 1][j + m − 1] · P(N_i, N_{i_1} N_{i_2})
        end
    end
end
return I
```

probability of generating string w from non-terminal N, and $Pr_G(uNv)$, the probability of generating uNv from the axiom.

Notice that the OUTSIDE algorithm needs the computation of the INSIDE one.

One should be careful when using PCFGs to generate strings: the process can diverge. If we take for example grammar G= $< \{\mathtt{a}, \mathtt{b}\}, \{N_1\}, R, P, N_1 >$ with rules $N_1 \rightarrow N_1 N_1$ (probability $\frac{1}{2}$) and $N_1 \rightarrow \mathtt{a}$ (probability $\frac{1}{2}$), then if generating with this grammar, although everything looks fine (it is easy to check that $Pr(\mathtt{a}) = \frac{1}{2}$, $Pr(\mathtt{aa}) = \frac{1}{8}$ and $Pr(\mathtt{aaa}) = \frac{1}{16}$), the generation process diverges. Let x be the estimated length of a string generated by G. The following recursive relation must hold:

$$x = \frac{1}{2} \cdot 2x + \frac{1}{2} \cdot 1$$

and it does not accept any solution.

In other words, as soon as there are more than two occurrences of the symbol 'N_1', then in this example, the odds favour an exploding and non-terminating process.

Algorithm 5.10: OUTSIDE.

Data: a PCFG $G =< \Sigma, V, R, P, N_1 >$ in quadratic normal form, with
$V = \{N_1, \ldots, N_{|V|}\}$, string $u = a_1 a_2 \cdots a_n$
Result: the probabilities $\mathrm{O}[i][j][k] = Pr_G(a_1 \cdots a_j N_i a_k \cdots a_n)$
for $j : 0 \leq k \leq n+1$ **do**
  for $k : j+1 \leq k \leq n+1$ **do**
    for $i : 1 \leq i \leq |V|$ **do** $\mathrm{O}[i][j][k] \leftarrow 0$
  end
end
$\mathrm{O}[1][0][n+1] \leftarrow 1$;
for $e : 0 \leq e \leq n$ **do**
  for $j : 1 \leq j \leq n-e$ **do**
    $k \leftarrow n+1-j+e$;
    for $(N_i, N_{i_1} N_{i_2}) \in R$ **do**
      for $s : j \leq s \leq k$ **do**
      $\mathrm{O}[i_1][j][s] \leftarrow \mathrm{O}[i_1][j][s] + P(N_i, N_{i_1} N_{i_2}) \cdot \mathrm{O}[i][j][k] \cdot \mathrm{I}[i_2][s][k-1]$;
      $\mathrm{O}[i_2][s][k] \leftarrow \mathrm{O}[i_2][s][k] + P(N_i, N_{i_1} N_{i_2}) \cdot \mathrm{I}[i_1][j+1][s] \cdot \mathrm{O}[i][j][k]$
    end
  end
end
return O

5.4 Distances between two distributions

There are a number of reasons for wanting to measure a distance between two distributions:

- In a testing situation where we want to compare two resulting PFAs obtained by two different algorithms, we may want to measure the distance towards some ideal target.
- In a learning algorithm, one option may be to decide upon merging two nodes of an automaton. For this to make sense we want to be able to say that the distributions at the nodes (taking each state as the initial state of the automaton) are close.
- There may be situations where we are faced by various candidate models, each describing a particular distribution. Some sort of nearest neighbour approach may then be of use, if we can measure a distance between distributions.
- The relationship between distances and kernels can also be explored. An attractive idea is to relate distances and kernels over distributions.

Defining similarity measures between distributions is the most natural way of comparing them. Even if the question of exact equivalence (discussed in Section 5.4.1) is of interest, in practical cases we wish to know if the distributions are close or not. In tasks involving

the learning of PFAs or DPFAs we may want to measure the quality of the result or of the learning process. For example, when learning takes place from a sample, measuring how far the learnt automaton is from the sample can also be done by comparing distributions since a sample can be encoded as a DPFA.

5.4.1 Equivalence questions

We defined equality between two distributions earlier; equivalence between two models is true when the underlying distributions are equal. But suppose now that the distributions are represented by PFAs, what more can we say?

Theorem 5.4.1 *Let $\mathcal{A}$ and $\mathcal{B}$ be two* PFA*s. We can decide in polynomial time if $\mathcal{A}$ and $\mathcal{B}$ are equivalent.*

Proof If $\mathcal{A}$ and $\mathcal{B}$ are DPFAs the proof can rely on the fact that the Myhill-Nerode equivalence can be redefined in an elegant way over regular deterministic distributions, which in turn ensures there is a canonical automaton for each regular deterministic distribution. It being canonical means that for any other DPFA generating the same distribution, the states can be seen as members of a partition block, each block corresponding to state in the canonical automaton.

The general case is more complex, but contrarily to the case of non-probabilistic objects, the equivalence is polynomially decidable. One way to prove this is to find a polynomially computable metric. Having a null distance (or not) will then correspond exactly to the PFAs being equivalent. Such a polynomially computable metric exists (the $\mathbf{L}_2$ norm, as proved in the Proposition 5.5.3, page 110) and the algorithm is provided in Section 5.5.3. □

But in practice we will often be confronted with either a sample and a PFA or two samples and we have to decide this equivalence upon incomplete knowledge.

5.4.2 Samples as automata

If we are given a sample S drawn from a (regular) distribution, there is an easy way to represent this empirical distribution as a DPFA. This automaton will be called the ***probabilistic prefix tree acceptor*** (PPTA(S)).

The first important thing to note is that a sample drawn from a probabilistic language is not a set of strings but a multiset of strings. Indeed, if a given string has a very high probability, we can expect this string to be generated various times. The total number of *different* strings in a sample S is denoted by $\uparrow S \uparrow$, but we will denote by $|S|$ the total number of strings, including repetitions. Given a language L, we can count how many occurrences of strings in S belong to L and denote this quantity by $|S|_L$. To count the number of occurrences of a string x in a sample S we will write $\text{cnt}_S(x)$, or $|S|_x$. We

will use the same notation when counting occurrences of prefixes, suffixes and substrings in a string: for example $|S|_{\Sigma^\star \mathtt{ab} \Sigma^\star}$ counts the number of strings in S containing ab as a substring.

The empirical distribution associated with a sample S drawn from a distribution $\mathcal{D}$ is denoted $\widehat{S}$, with $Pr_{\widehat{S}}(x) = \frac{\mathrm{cnt}_S(x)}{|S|}$.

Example 5.4.1 Let $S = \{\mathtt{a}(3), \mathtt{aba}(1), \mathtt{bb}(2), \mathtt{babb}(4), \mathtt{bbb}(1)\}$ where babb(4) means that there are 4 occurrences of string babb. Then $\uparrow S \uparrow = 5$, but $|S| = 11$ and $|S|_{b^*} = 3$.

Definition 5.4.1 Let S be a multiset of strings from $\Sigma^\star$. The **probabilistic prefix tree acceptor** PPTA(S) is the DPFA $\langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{P}, \delta_\mathbb{P} \rangle$ where

- $Q = \{q_u : u \in \textsc{Pref}(S)\}$.
- $\forall ua \in \textsc{Pref}(S) :\ \delta_\mathbb{P}(q_u, a, q_{ua}) = \frac{|S|_{ua\,\Sigma^\star}}{|S|_{u\,\Sigma^\star}}$.
- $\forall u \in \textsc{Pref}(S) :\ \mathbb{F}_\mathbb{P}(q_u) = \frac{|S|_u}{|S|_{u\,\Sigma^\star}}$.

Example 5.4.2 Let $S = \{\mathtt{a}(3), \mathtt{aba}(1), \mathtt{bb}(2), \mathtt{babb}(4), \mathtt{bbb}(1)\}$. Then the corresponding probabilistic prefix tree acceptor PPTA(S) is depicted in Figure 5.11. The fractions are represented in a simplified way (1 instead of $\frac{4}{4}$). This nevertheless corresponds to a loss of information: $\frac{4}{4}$ is 'different' statistically than $\frac{400}{400}$; the issue will be dealt with in Chapter 16.

The above transformation allows us to define in a unique way the distances between regular distributions over $\Sigma^\star$. In doing so they implicitly define distances between automata, but also between automata and samples, or even between samples.

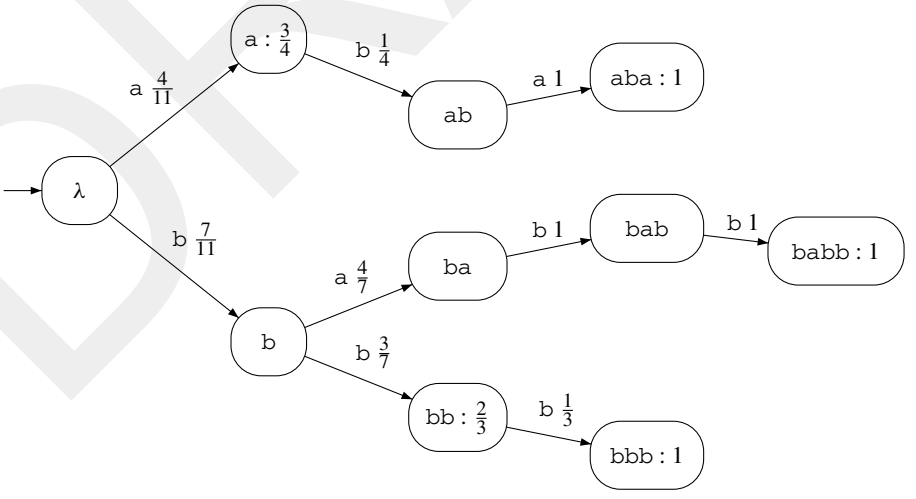


Fig. 5.11. PPTA for $S = \{\mathtt{a}(3), \mathtt{aba}(1), \mathtt{bb}(2), \mathtt{babb}(4), \mathtt{bbb}(1)\}$.

5.4.3 Distances between automata

- The most general family of distances is referred to as the $\mathbf{L}_n$ distances or distances for the norm $\mathbf{L}_n$. The general definition goes as follows:

$$\mathbf{L}_n(\mathcal{D}, \mathcal{D}') = \left(\sum_{x \in \Sigma^\star} \left| Pr_{\mathcal{D}}(x) - Pr_{\mathcal{D}'}(x) \right|^n \right)^{\frac{1}{n}}.$$

- For n=1 we get a natural distance also known as the $\mathbf{L}_1$ distance, the ***variation*** or ***Manhattan*** distance, or distance for the norm $\mathbf{L}_1$.

$$\mathbf{L}_1(\mathcal{D}, \mathcal{D}') = \sum_{x \in \Sigma^\star} \left| Pr_{\mathcal{D}}(x) - Pr_{\mathcal{D}'}(x) \right|.$$

- And in the special case where $n = 2$ we obtain:

$$\mathbf{L}_2(\mathcal{D}, \mathcal{D}') = \sqrt{\sum_{x \in \Sigma^\star} (Pr_{\mathcal{D}}(x) - Pr_{\mathcal{D}'}(x))^2}.$$

  This is also known as the quadratic distance or the Euclidean distance. Notice that with $n = 2$ the absolute values of the definition of $\mathbf{L}_n$ vanish, which will allow computation to be easier.
- The following distance (sometimes denoted also as d_* or $d_{\max}$) for the $\mathbf{L}_\infty$ norm is the limit when $n \to \infty$ of the $\mathbf{L}_n$:

$$\mathbf{L}_\infty(\mathcal{D}, \mathcal{D}') = \max_{x \in \Sigma^\star} \left| Pr_{\mathcal{D}}(x) - Pr_{\mathcal{D}'}(x) \right|.$$

When concerned with very small probabilities such as those that may arise when an infinite number of strings have non-null probability, it may be more useful to use logarithmic probabilities. In this way a string with very small probability may influence the distance because the relative probabilities in both distributions are very different. Suppose $Pr_1(x) = 10^{-7}$ and $Pr_2(x) = 10^{-9}$, then the effect for $\mathbf{L}_1$ of this particular string will be of $99 \cdot 10^{-9}$ whereas for the logarithmic distance the difference will be the same as if the probabilities had been 10^{-1} and 10^{-3}.

The *logarithmic distance* is

$$d_{\log}(\mathcal{D}, \mathcal{D}') = \max_{x \in \Sigma^\star} \left| \log Pr_{\mathcal{D}}(x) - \log Pr_{\mathcal{D}'}(x) \right|.$$

But the logarithmic distance erases all the differences between the large and the less large probabilities. A good compromise is the ***Kullback-Leibler divergence***:

$$d_{\mathrm{KL}}(\mathcal{D}, \mathcal{D}') = \sum_{x \in \Sigma^\star} Pr_{\mathcal{D}}(x) \cdot \log \frac{Pr_{\mathcal{D}}(x)}{Pr_{\mathcal{D}'}(x)}.$$

We set in a standard way that $0 \log 0 = 0$ and $\frac{0}{0} = 1$.

The ***Kullback-Leibler divergence*** is the sum over all strings of the logarithmic loss weighted down by the actual probability of a string. The divergence is an asymmetrical measure: the distribution $\mathcal{D}$ is used twice as it clearly is the 'real' distribution, the one that assigns the weights in the sum.

It should be noticed that in the case where some string has a null probability in $\mathcal{D}'$, but not in $\mathcal{D}$, the denominator $Pr_{\mathcal{D}'}(x)$ is null. As it is supposed to divide the expression, the Kullback-Leibler divergence is infinite. This is obviously a crucial issue:

- A model that assigns a null probability to a possible string is bad. The KL-measure tells us it is worse than bad, as no matter what the rest of the distribution looks like, the penalty is infinite and cannot be compensated.
- A first way to deal with this serious problem is through *smoothing*: the model is generalised in order to assign (small) probabilities to events that were not regarded as possible during the learning phase. The crucial question is how to do this and is a research field in itself.
- The second way consists of using a metric that does not have the inconveniences of the KL-divergence. This is theoretically sound, but at the same time does not address the practical issues of assigning a null probability to a possible event.

Rewriting the Kullback-Leibler divergence as

$$d_{\mathrm{KL}}(\mathcal{D}, \mathcal{D}') = \sum_{x \in \Sigma^\star} Pr_{\mathcal{D}}(x) \cdot \log Pr_{\mathcal{D}}(x) - \sum_{x \in \Sigma^\star} Pr_{\mathcal{D}}(x) \cdot \log Pr_{\mathcal{D}'}(x), \tag{5.1}$$

one can note the first term is the entropy of $\mathcal{D}$ and does not depend on $\mathcal{D}'$, and the second term is the cross-entropy of $\mathcal{D}$ given $\mathcal{D}'$. One interpretation (from information theory) is that we are measuring the difference between the optimal number of bits needed to encode a random message (the left-hand side) and the average number of bits when using distribution $\mathcal{D}'$ to encode the data.

5.4.4 Some properties

- $\mathbf{L}_n$ $(\forall n)$, $\mathbf{L}_\infty$ are metrics, i.e. they comply with the usual properties: $\forall \mathcal{D}, \mathcal{D}',\ \forall d \in \{\mathbf{L}_n, \mathbf{L}_\infty\}$,
  1. $d(\mathcal{D}, \mathcal{D}') = 0 \iff \mathcal{D} = \mathcal{D}'$,
  2. $d(\mathcal{D}, \mathcal{D}') = d(\mathcal{D}', \mathcal{D})$,
  3. $d(\mathcal{D}, \mathcal{D}') + d(\mathcal{D}', \mathcal{D}'') \geq d(\mathcal{D}, \mathcal{D}'')$.
- d_{KL} is not a metric (Definition 3.1.1, page 47) but can be adapted to comply with those conditions. This is usually not done because the asymmetrical aspects of d_{KL} are of interest in practical and theoretical settings.
  It nevertheless verifies the following properties: $\forall \mathcal{D}, \mathcal{D}'$,
  1. $d_{\mathrm{KL}}(\mathcal{D}, \mathcal{D}') \geq 0$,
  2. $d_{\mathrm{KL}}(\mathcal{D}, \mathcal{D}') = 0 \iff \mathcal{D} = \mathcal{D}'$.
- Obviously $\forall \mathcal{D}, \mathcal{D}', \mathbf{L}_\infty(\mathcal{D}, \mathcal{D}') \leq \mathbf{L}_1(\mathcal{D}, \mathcal{D}')$.
- Pinsker's inequality states that:

$$d_{\mathrm{KL}}(\mathcal{D}, \mathcal{D}') \geq \frac{1}{2 \ln 2} \mathbf{L}_1(\mathcal{D}, \mathcal{D}')^2.$$

These inequalities help us to classify the distances over distributions in the following (informal) way:

$$d_{\log} \precsim D_{\mathrm{KL}} \precsim \mathbf{L}_1 \precsim \ldots \precsim \mathbf{L}_n \precsim \mathbf{L}_{n+1} \precsim \mathbf{L}_\infty.$$

This can be read as:

- From left to right we find the distances that attach most to least importance to the relative differences between the probabilities. The further to the right, the more important are the absolute differences.
- If you want to measure closeness of two distributions by the idea that you want to be close on the important probabilities, then you should turn to the right-hand side of the expression above.
- If you want all the probabilities to count and to count this in a relative way, you should use measures from the left-hand side of the expression above.

Alternatively, we can consider a sample as a *random variable*, in which case we can consider the probability that a sample of a given size has such or such property. And the notation $Pr_{\mathcal{D}}(f(S_n))$ will be used to indicate the probability that property f holds over a sample of size n sampled following $\mathcal{D}$.

Lemma 5.4.2 *Let $\mathcal{D}$ be any distribution on $\Sigma^\star$. Then $\forall a > 1$, the probability that a sample S of size n has*

$$\mathbf{L}_\infty(\mathcal{D}, \widehat{S}) \leq \sqrt{6a(\log n)/n}$$

is at least $4n^{-a}$.

Essentially the lemma states that the empirical distribution converges (for the $\mathbf{L}_\infty$ distance) to the true distance.

5.5 Computing distances

When we have access to the models (the automata) for the distributions, an exact computation of the distance is possible in certain cases. This can also solve two other problems:

- If the distance we are computing respects the conditions $d(x, y) = 0 \iff x = y$ and $d(x, y) \geq 0$, computing the distance allows us to solve the equivalence problem.
- Since samples can easily be represented by PFAs, we can also compute the distance between two samples or between a sample and a generator through these techniques. Error bounds corresponding to different statistical risks can also be computed.

5.5.1 Computing prefixial distances between states

Let $\mathcal{A}$ and $\mathcal{B}$ be two PFAs (without λ-transitions) with $\mathcal{A} = \langle \Sigma, Q_{\mathcal{A}}, \mathbb{I}_{\mathbb{P}\mathcal{A}}, \mathbb{F}_{\mathbb{P}\mathcal{A}}, \delta_{\mathbb{P}\mathcal{A}} \rangle$ and $\mathcal{B} = \langle \Sigma, Q_{\mathcal{B}}, \mathbb{I}_{\mathbb{P}\mathcal{B}}, \mathbb{F}_{\mathbb{P}\mathcal{B}}, \delta_{\mathbb{P}\mathcal{B}} \rangle$ with associated probability functions $Pr_{\mathcal{A}}$ and $Pr_{\mathcal{B}}$.

We will denote in a standard fashion by $\mathcal{A}_q$ the PFA obtained from $\mathcal{A}$ when taking state q as the unique initial state. Alternatively it corresponds to using function $\mathbb{I}_{\mathbb{P}\mathcal{A}}(q) = 1$ and $\forall q' \neq q,\ \mathbb{I}_{\mathbb{P}\mathcal{A}}(q') = 0$.

We define now η_q as the probability of reaching state q in $\mathcal{A}$. Computing η_q may seem complex but is not:

$$\eta_q = \mathbb{I}_{\mathbb{P}\mathcal{A}}(q) + \sum_{s \in Q_\mathcal{A}} \sum_{a \in \Sigma} \eta_s \cdot \delta_{\mathbb{P}\mathcal{A}}(s, a, q).$$

In a similar way, we define $\eta_{qq'}$ as the probability of jointly reaching (with the same string) state q in $\mathcal{A}$ and state q' in $\mathcal{B}$. This means summing over all the possible strings and paths. Luckily, the recursive definition is much simpler and allows a polynomial implementation by dynamic programming.

By considering on one hand the only zero-length prefix λ and on the other hand the other strings which are obtained by reading all but the last character, and then the last one, $\forall q, q' \in Q_\mathcal{A} \times Q_\mathcal{B}$,

$$\eta_{qq'} = \mathbb{I}_{\mathbb{P}\mathcal{A}}(q)\mathbb{I}_{\mathbb{P}\mathcal{B}}(q') + \sum_{s \in Q_\mathcal{A}} \sum_{s' \in Q_\mathcal{B}} \sum_{a \in \Sigma} \eta_{s,s'} \cdot \delta_{\mathbb{P}\mathcal{A}}(s, a, q) \cdot \delta_{\mathbb{P}\mathcal{B}}(s', a, q'). \tag{5.2}$$

The above equation is one inside a system of linear equations. A cubic algorithm (in the number of equations and variables) can solve this, even if specific care has to be taken with manipulation of the fractions. In certain cases, where the PFAs have reasonably short probable strings, it can be noticed that convergence is very fast, and a fixed number of iterations is sufficient to approximate the different values closely.

We will use this result in Section 5.5.3 to compute $\mathbf{L}_2$.

5.5.2 Computing the KL-divergence

We recall the second definition of the KL-divergence and adapt it for two DPFAs (Equation 5.1):

$$d_{\mathrm{KL}}(\mathcal{A}, \mathcal{B}) = \sum_{x \in \Sigma^\star} Pr_\mathcal{A}(x) \cdot \log Pr_\mathcal{A}(x) - \sum_{x \in \Sigma^\star} Pr_\mathcal{A}(x) \cdot \log Pr_\mathcal{B}(x).$$

It follows that we need to know how to compute $\sum_{x \in \Sigma^\star} Pr_\mathcal{A}(x) \cdot \log Pr_\mathcal{B}(x)$. Then the first part of the formula can be computed by simply taking $\mathcal{B} = \mathcal{A}$. We denote by $Pr_\mathcal{A}(a|x)$ the probability of reading a after having read x and by $Pr_\mathcal{A}(a|x)$ the probability of halting after having read x.

Let us, in the case of DPFAs, show that we can compute $\sum_{x \in \Sigma^\star} Pr_\mathcal{A}(x) \cdot \log Pr_\mathcal{B}(x)$. If we take any string $x = a_1 \cdots a_n$, we have $\log Pr_\mathcal{B}(x) = \sum_{i \in [n]} \log Pr_\mathcal{B}(a_i|a_1 \cdots a_{i-1}) + \log Pr_\mathcal{B}(\lambda|x)$. Each term $\log Pr_\mathcal{B}(a_i)$ therefore appears in every string (and every time) containing letter a_i. We can therefore factorise:

$$\sum_{x \in \Sigma^\star} Pr_\mathcal{A}(x) \cdot \log Pr_\mathcal{B}(x) = \sum_{x \in \Sigma^\star} \sum_{a \in \Sigma} Pr_\mathcal{A}(xa\,\Sigma^\star) \cdot \log Pr_\mathcal{B}(a|x) + \sum_{x \in \Sigma^\star} Pr_\mathcal{A}(x) \cdot \log Pr_\mathcal{B}(\lambda|x).$$

Since $\mathcal{A}$ and $\mathcal{B}$ are DPFAs, another factorisation is possible, by taking together all the strings x finishing in state q in $\mathcal{A}$ and in state q' in $\mathcal{B}$. We recall that $\eta_{q,q'}$ is the probability of reaching simultaneously state q in $\mathcal{A}$ and state q' in $\mathcal{B}$. The value of $\sum_{x\in\Sigma^\star} Pr_{\mathcal{A}}(x) \cdot \log Pr_{\mathcal{B}}(x)$ is:

$$\sum_{q\in Q_{\mathcal{A}}} \sum_{q'\in Q_{\mathcal{B}}} \sum_{a\in\Sigma} \eta_{q,q'} Pr_{\mathcal{A}}(a|q) \cdot \log Pr_{\mathcal{B}}(a|q') + \sum_{q\in Q_{\mathcal{A}}} \sum_{q'\in Q_{\mathcal{B}}} \eta_{q,q'} \mathbb{F}_{\mathbb{P}_{\mathcal{A}}}(q) \cdot \log \mathbb{F}_{\mathbb{P}_{\mathcal{B}}}(q').$$

Since the $\eta_{q,q'}$ can be computed thanks to Equation 5.2, we can state:

Proposition 5.5.1 *Let $\mathcal{A}$, $\mathcal{B}$ be two* DPFA*s. Then $d_{\mathrm{KL}}(\mathcal{A},\mathcal{B})$ can be computed in polynomial time.*

This proposition will be the key to understanding algorithm MDI, given in Section 16.7.

5.5.3 Co-emission

If a direct computation of a distance between two automata is difficult, one can compute instead the probability that both automata generate the same string at the same time. We define the co-emission between $\mathcal{A}$ and $\mathcal{B}$:

Definition 5.5.1 (Co-emission probability) The **co-emission probability** of $\mathcal{A}$ and $\mathcal{B}$ is

$$\mathbf{coem}(\mathcal{A},\mathcal{B}) = \sum_{w\in\Sigma^\star} \left(Pr_{\mathcal{A}}(w) \cdot Pr_{\mathcal{B}}(w)\right).$$

This is the probability of emitting w simultaneously from automata $\mathcal{A}$ and $\mathcal{B}$. The main interest in being able to compute the co-emission is that it can be used to compute the $\mathbf{L}_2$ distance between two distributions:

Definition 5.5.2 ($\mathbf{L}_2$ distance between two models) The **distance for the $\mathbf{L}_2$ norm** is defined as:

$$\mathbf{L}_2(\mathcal{A},\mathcal{B}) = \sqrt{\sum_{w\in\Sigma^\star} (Pr_{\mathcal{A}}(w) - Pr_{\mathcal{B}}(w))^2}.$$

This can be computed easily by developing the formula:

$$\mathbf{L}_2(\mathcal{A},\mathcal{B}) = \sqrt{\mathbf{coem}(\mathcal{A},\mathcal{A}) + \mathbf{coem}(\mathcal{B},\mathcal{B}) - 2\,\mathbf{coem}(\mathcal{A},\mathcal{B})}.$$

If we use $\eta_{qq'}$ (the probability of jointly reaching state q in $\mathcal{A}$ and state q' in $\mathcal{B}$),

$$\mathbf{coem}(\mathcal{A},\mathcal{B}) = \sum_{q\in Q} \sum_{q'\in Q'} \eta_{qq'} \cdot \mathbb{F}_{\mathbb{P}_{\mathcal{A}}}(q) \cdot \mathbb{F}_{\mathbb{P}_{\mathcal{B}}}(q').$$

Since the computation of the $\eta_{qq'}$ could be done in polynomial time (by Equation 5.2, page 108), this in turns means that the computation of the co-emission and of the $\mathbf{L}_2$ distance is polynomial.

Concluding:

Theorem 5.5.2 *If $\mathcal{D}$ and $\mathcal{D}'$ are given by* PFA*s,* $\mathbf{L}_2(\mathcal{D}, \mathcal{D}')$ *can be computed in polynomial time.*

Theorem 5.5.3 *If $\mathcal{D}$ and $\mathcal{D}'$ are given by* DPFA*s,* $d_{\mathrm{KL}}(\mathcal{D}, \mathcal{D}')$ *can be computed in polynomial time.*

5.5.4 Some practical issues

In practical settings we are probably going to be given a sample (drawn from the unknown distribution $\mathcal{D}$) which we have divided into two subsamples in a random way. The first sample has been used to learn a probabilistic model ($\mathcal{A}$) which is supposed to generate strings in the same way as the (unknown) target sample does.

The second sample is usually called the ***test*** sample (let us denote it by S) and is going to be used to measure the quality of the learnt model $\mathcal{A}$.

There are a number of options using the tools studied up to now:

- We could use the $\mathbf{L}_2$ distance between $\mathcal{A}$ and $\widehat{S}$. The empirical distribution $\widehat{S}$ can be represented by a PPTA and we can compute the $\mathbf{L}_2$ distance between this PPTA and $\mathcal{A}$ using Proposition 5.5.1.
- Perhaps the $\mathbf{L}_1$ distance is better indicated and we would like to try to do the same with this. But an exact computation of this distance (between two DPFAs) cannot be done in polynomial time (unless $\mathcal{P} = \mathcal{NP}$), so only an approximation can be obtained.
- The Kullback-Leibler divergence is an alternative candidate. But as $\mathcal{D}$ is unknown, again the distance has to be computed over the empirical distribution $\widehat{S}$ instead of $\mathcal{D}$. In this case we may notice that in Equation 5.1 the first part is constant and does not depend on $\mathcal{A}$, so we are really only interested in minimising the second part (the relative entropy).
- The alternative is to use *perplexity* instead.

5.5.5 Perplexity

If we take the equation for the KL-divergence, we can measure the distance between a hypothesis distribution H and the real distribution $\mathcal{D}$. If we only have a sample S of the real distribution, then it may still be possible to use this sample (or the corresponding empirical distribution $\widehat{S}$) instead of the real distribution $\mathcal{D}$. We would then obtain (from Equation 5.1):

$$d_{KL}(\widehat{S}, H) = \sum_{x \in \Sigma^\star} Pr_{\widehat{S}}(x) \cdot \log Pr_{\widehat{S}}(x) - \sum_{x \in \Sigma^\star} Pr_{\widehat{S}}(x) \cdot \log Pr_H(x), \qquad (5.3)$$

In the above equation the first term doesn't depend on the hypothesis. We can therefore remove it when what we want is to compare different hypotheses. For those strings not in the sample, the probability in $\widehat{S}$ is 0, so they need not be counted either.

Simplifying, we get the estimation of the ***divergence*** (DIV) between S and the hypothesis H:

$$\mathrm{DIV}(S|H) = -\frac{1}{|S|}\sum_{x\in S}\mathrm{cnt}_S(x)\log(Pr_H(x)). \tag{5.4}$$

This in turn helps us define the ***perplexity*** of H with respect to the sample:

$$\begin{aligned} PP(S|H) &= \left[\prod_{x\in S} Pr_H(x)\right]^{-\frac{\mathrm{cnt}_S(x)}{|S|}} \\ &= \sqrt[n]{\prod_{x\in S} Pr_H(x)^{\mathrm{cnt}_S(x)}}. \end{aligned} \tag{5.5}$$

The properties of the perplexity can be summarised as follows:

- Equation 5.4 tells us that the perplexity measures the average number of bits one must 'pay' by using the model H instead of $\mathcal{D}$ while coding the sample S.
- From Equation 5.5, the perplexity can be seen as the inverse of the geometric mean of the probabilities – according to the model – of the sample words.

In practical situations, the following issues must be carefully taken into account:

- Perplexity and entropy are infinite whenever there is a string in the sample for which $Pr_H(x) = 0$. In practice, this implies that the perplexity must be used on a *smoothed* model, i.e. a model that provides a non-null probability for any string of $\Sigma^\star$. One should add that this is not a defect of using perplexity. This is a normal situation: a model that cannot account for a possible string is a bad model.
- The perplexity can compare models only when using the same sample S.

5.6 Exercises

5.1 Compute the PPTA corresponding to the sample $S = \{\lambda(6), \mathtt{a}(2), \mathtt{aba}(1), \mathtt{baba}(4), \mathtt{bbbb}(1)\}$.

5.2 Consider the PFA from Figure 5.12. What is the probability of string `aba`? Of string `bbaba`? Which is the most probable string (in $\Sigma^\star$)?

5.3 Write a polynomial time algorithm which, given a DPFA $\mathcal{A}$, returns the most probable string in $\mathcal{D}_\mathcal{A}$.

5.4 Write a polynomial time algorithm which, given a DPFA $\mathcal{A}$ and an integer n, returns $\mathrm{mps}(\mathcal{A}, n)$, the n most probable string in $\mathcal{D}_\mathcal{A}$.

5.5 Prove that the probabilistic language from Figure 5.13 is not deterministic.

5.6 Build two DPFAs $\mathcal{A}$ and $\mathcal{B}$ with n states, each such that $\{w \in \Sigma^\star : Pr_\mathcal{A}(w) > 0 \wedge Pr_\mathcal{B}(w) > 0\} = \emptyset$ yet $\mathbf{L}_2(\mathcal{D}_\mathcal{A}, \mathcal{D}_\mathcal{B}) \leq 2^{-n}$.

5.7 Let $\mathcal{D}$ and $\mathcal{D}'$ be two distributions. Prove that $\forall\alpha, \beta > 0 : \alpha + \beta = 1, \alpha\mathcal{D} + \beta\mathcal{D}'$ is a distribution.

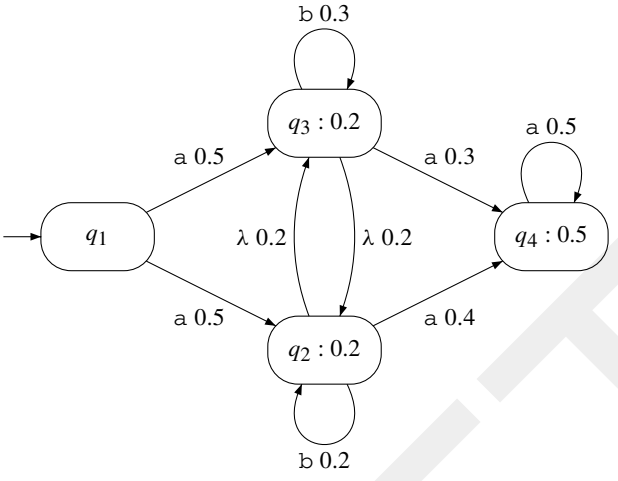


Fig. 5.12. A PFA.

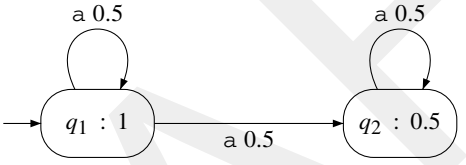


Fig. 5.13. A very simple distribution.

5.8 Consider the PFA from Figure 5.12. Eliminate the λ-transitions from it and obtain an equivalent λ-free PFA.

5.9 Let $S = \{\mathtt{a}(3), \mathtt{aba}(1), \mathtt{bb}(2), \mathtt{babb}(4), \mathtt{bbb}(1)\}$ and call $\mathcal{A}$ the PFA from Figure 5.12. Compute $\mathbf{L}_1(\mathcal{D}_\mathcal{A}, \mathcal{D}_\S)$ and $\mathbf{L}_2(\mathcal{D}_\mathcal{A}, \mathcal{D}_\S)$.

5.10 Prove that $\mathbf{L}_1(\mathcal{D}_\mathcal{A}, \mathcal{D}_\S)$ can be computed, given any sample S and any PFA $\mathcal{A}$.

5.11 Build a PFA $\mathcal{A}_n$ and a string x_n for which the VITERBI score of x is less than any non-null polynomial fraction of $Pr_\mathcal{A}(x)$.

5.12 Write a randomised algorithm which, given a PFA $\mathcal{A}$ and a value $\alpha > 0$ tells us whether there exists a string w whose probability is at least α.

5.7 Conclusions of the chapter and further reading

5.7.1 Bibliographical background

The main reference about probabilistic finite automata is Azaria Paz's book (Paz, 1971); two articles by Enrique Vidal *et al.*, present these automata from an engineering point of view (Vidal *et al.*, 2005a, 2005b). Extended proofs of some of the results presented here, like Proposition 5.2.1 (page 95), can be found there.

We have chosen here to use the term 'probabilistic automata' uniquely whereas a number of authors have sometimes used 'stochastic automata' for exactly the same objects. This

can create confusion and if one wants to access the different papers written on the subject, it has to be kept in mind.

Among the other probabilistic finite state machines we can find hidden Markov models (HMMs) (Jelinek, 1998, Rabiner, 1989), probabilistic regular grammars (Carrasco & Oncina, 1994b), Markov chains (Saul & Pereira, 1997), n-grams (Jelinek, 1998, Ney, Martin & Wessel, 1997), probabilistic suffix trees (Ron, Singer & Tishby, 1994), deterministic probabilistic automata (Carrasco & Oncina, 1994b) and weighted automata (Mohri, 1997). These are some names of syntactic objects which have been used to model distributions over sets of strings of possibly infinite cardinality, sequences, words, phrases but also terms and trees. Pierre Dupont *et al.* prove the links between HMMs and PFAs in (Dupont, Denis & Esposito, 2005), with an alternative proof in (Vidal *et al.*, 2005a, 2005b).

The parsing algorithms are now well known; the FORWARD algorithm is described by Leonard Baum *et al.* (Baum *et al.*, 1970). The VITERBI algorithm is named after Andrew Viterbi (Viterbi, 1967).

Another problem related to parsing is the computation of the probability of all strings sharing a given prefix, suffix or substring in a PFA (Fred, 2000).

More complicated is the question of finding the most probable string in a distribution defined by a PFA. In the general case the problem is intractable (Casacuberta & de la Higuera, 2000), with some associated problems undecidable (Blondel & Canterini, 2003), but in the deterministic case a polynomial algorithm can be written using dynamic programming (see Exercise 5.3). Curiously enough, randomised algorithms solve this question very nicely with small error.

Several other interesting questions are raised in Omri Guttman's PhD thesis (Guttman, 2006), for instance the question of knowing how reasonable it can be to attempt to approximate an unknown distribution with a regular one. He proves that for a fixed bounded number of states n, the best PFA with at most n states can be arbitrarily bad.

In Section 5.2.4 we consider the problem of eliminating λ-transitions; λ-PFAs appear as natural objects when combining distributions. λ-PFAs introduce specific problems, in particular, when sequences of transitions labelled with λ are considered. Some authors have provided alternative parsing algorithms to deal with λ-PFAs (Picó & Casacuberta, 2001). In that case parsing takes time that is no longer linear in the size of the string to be parsed but in the size of this string multiplied by the number of states of the PFA. In (Mohri, Pereira & Riley, 2000) Mehryar Mohri proposes to eliminate λ-transitions by means of first running the Floyd-Warshall algorithm in order to compute the λ-transition distance between pairs of edges before removal of these edges. The algorithm we give here can be found in (Hanneforth & de la Higuera, 2009). Thomas Hanneforth points out (Hanneforth, 2008) that when dealing with $\mathcal{A}$ obtained automatically, once the λ-transitions have been eliminated, there is pruning to be done, as some states can be no longer accessible.

We have only given some very elementary results concerning probabilistic context-free grammars (Section 5.3). The parsing algorithms INSIDE and OUTSIDE were introduced by Karim Lari and Steve Young and can be found (with non-trivial differences) in several places (Casacuberta, 1994, Lari & Young, 1990).

Several researchers have worked on distances between distributions (Section 5.4). The first important results were found by Rafael Carrasco *et al.* (Calera-Rubio & Carrasco, 1998, Carrasco, 1997). In the context of HMMs and with intended bio-informatics applications, intractability results were given by Rune Lyngsø *et al.* (Lyngsø & Pedersen, 2001, Lyngsø, Pedersen & Nielsen, 1999). Other results are those by Michael Kearns *et al.* (Kearns *et al.*, 1994). Properties about the distances between automata have been published in various places. Thomas Cover and Jay Thomas' book (Cover & Thomas, 1991) is a good place for these.

A first algorithm for the equivalence of PFAs was given by Vijay Balasubramanian (Balasubramanian, 1993). Then Corinna Cortes *et al.* (Cortes, Mohri & Rastogi, 2006) noticed that being able to compute the $\mathbf{L}_2$ distance in polynomial time ensures that the equivalence is also testable.

The algorithms presented here to compute distances have first appeared in work initiated by Rafael Carrasco *et al.* (Calera-Rubio & Carrasco, 1998, Carrasco, 1997) for the KL-divergence (for string languages and tree languages) and for the $\mathbf{L}_2$ between trees and Thierry Murgue (Murgue & de la Higuera, 2004) for the $\mathbf{L}_2$ between strings. Rune Lyngsø *et al.* (Lyngsø & Pedersen, 2001, Lyngsø, Pedersen & Nielsen, 1999) introduced the co-emission probability, which is an idea also used in kernels. Corinna Cortes *et al.* (Cortes, Mohri & Rastogi, 2006) proved that computing the $\mathbf{L}_1$ distance was intractable; this is also the case for each $\mathbf{L}_n$ with n odd. The same authors better the complexity of (Carrasco, 1997) in a special case and present further results for computations of distances.

Practical issues with distances correspond to topics in speech recognition. In applications such as language modelling (Goodman, 2001) or statistical clustering (Brown *et al.*, 1992, Kneser & Ney, 1993), perplexity is introduced.

5.7.2 Some alternative lines of research

If comparing samples co-emission may be null when large vocabulary and long strings are used. An alternative idea is to compare not only the whole strings, but all their prefixes (Murgue & de la Higuera, 2004):

Definition 5.7.1 (Prefixial co-emission probability) The **prefixial co-emission probability** of $\mathcal{A}$ and $\mathcal{B}$ is

$$\mathbf{coempr}(\mathcal{A},\mathcal{B}) = \sum_{w\in\Sigma^\star} \left(Pr_{\mathcal{A}}(w\,\Sigma^\star)\cdot Pr_{\mathcal{B}}(w\,\Sigma^\star)\right).$$

Definition 5.7.2 ($\mathbf{L}_2 pref$ distance between two models) The **prefixial distance for the $\mathbf{L}_2$ norm**, denoted by $\mathbf{L}_{2pref}$, is defined as:

$$\mathbf{L}_{2pref}(\mathcal{A},\mathcal{B}) = \sqrt{\sum_{w\in\Sigma^\star} (Pr_{\mathcal{A}}(w\,\Sigma^\star) - Pr_{\mathcal{B}}(w\,\Sigma^\star))^2}$$

which can be computed easily using:

$$\mathbf{L}_{2pref}(\mathcal{A}, \mathcal{B}) = \sqrt{\mathbf{coempr}(\mathcal{A}, \mathcal{A}) + \mathbf{coempr}(\mathcal{B}, \mathcal{B}) - 2\mathbf{coempr}(\mathcal{A}, \mathcal{B})}.$$

Theorem 5.7.1 $\mathbf{L}_{2pref}$ *is a metric over* $\Sigma^\star$.

Proof These proofs are developed in (Murgue & de la Higuera, 2004). □

5.7.3 Open problems and possible new lines of research

Probabilistic acceptors are defined in (Fu, 1982), but they have only seldom been considered in syntactic pattern recognition or in (probabilistic) formal language theory.

One should also look into what happens when looking at approximation issues. Let $\boldsymbol{\mathcal{D}}_1$ and $\boldsymbol{\mathcal{D}}_2$ be two classes of distributions, $\epsilon > 0$ and d a distance. Then $\boldsymbol{\mathcal{D}}_1$ d-ϵ approximates $\boldsymbol{\mathcal{D}}_2$ whenever given any distribution $\mathcal{D}$ from $\boldsymbol{\mathcal{D}}_1$, $\exists \mathcal{D} \in \boldsymbol{\mathcal{D}}_2$ such that $d(\mathcal{D}, \mathcal{D}') < \epsilon$. The question goes as follows: are regular distributions approximable by deterministic regular ones? For what value of ϵ? Along these lines, one should consider distributions represented by automata of up to some fixed size.

Analysing (in the spirit of (Guttman, 2006)) the way distributions can or cannot be approximated can certainly help us better understand the difficulties of the task of learning them.