16

Learning Probabilistic Finite Automata

> En efecto, las computadoras parten del sofisma, políticamente inaceptable, de que dos y dos son cuatro. Su conservadurismo es feroz en este respeto.
>
> **Pablo de la Higuera**, In, Out, Off... ¡Uf!

> The promiscuous grammar has high a priori probability, but assigns low probability to the data. The ad-hoc grammar has very low a priori probability, but assigns probability one to the data. These are two extreme grammar types: The best choice is usually somewhere between them.
>
> **Ray Solomonoff**, in [Sol97]

A language is a set of strings, and a string either belongs to a language or does not. An alternative is to define a distribution over the set of all strings, and in some sense the probability of a string in this set is indicative of its importance in the language. This distribution, if learnt from data, can in turn be used to disambiguate, by finding the most probable string corresponding to a pattern, or to predict by proposing the next symbol for a given prefix.

We propose in this chapter to investigate ways of learning distributions representable by probabilistic automata from strings. No extra knowledge is usually provided. In particular the structure of the automaton is unknown. The case where the structure is known corresponds to the problem of *probability estimation* and is discussed in Chapter 17.

We only work in this chapter with deterministic probabilistic finite au-

tomata: Positive learning results for distributions defined by other classes of grammars are scarce; we nevertheless comment upon some in the last section of the chapter. Definitions of the probabilistic automata can be found in Chapter 5, whereas the convergence criteria were introduced and discussed in Chapter 10, where also a number of negative results were studied.

16.1 Issues

There are a number of reasons for which one may want to learn a probabilistic language and it should be emphasised that these reasons are independent and therefore do not necessarily add up.

16.1.1 Dealing with noise

If the task is about learning a language in a noisy setting, the algorithms presented in earlier chapters for learning from text, from an informant or with an Oracle have shown not to be suited. Probabilities offer what looks like a nice alternative: Why not learn a probabilistic language in which (hopefully) the weights of the strings in the language will be big and if any noisy string interferes in the learning process the probability of this string will be small and therefore not penalise too much the learnt hypothesis?

But then, and this is a common mistake made, the probabilities we are thinking about in this noisy setting correspond to those indicating if a string belongs or not to the language. The classes of probabilistic languages to be learnt (see Chapter 5) do not use this idea of stochasticity (one should look into fuzzy automata for this, a class for which there are very few learning results).

16.1.2 Prediction

There are situations when what is really important is the symbol and not the string itself. An important and typical question will then be:

Having seen so far `abbaba`, what is the next symbol?

If we are able to access a distribution of probabilities over $\Sigma^\star$ (and thus the notation $Pr_{\mathcal{D}}(w)$ makes sense), we are able to deduce a distribution over Σ in a given *context*. If we denote by $Pr_{\mathcal{D}}(a|w)$ the probability of generating an a after having generating w when following distribution $\mathcal{D}$ (and $Pr(a|w)$

when the distribution is clear),

$$Pr_{\mathcal{D}}(a|w) = \frac{Pr_{\mathcal{D}}(wa\,\Sigma^\star)}{Pr_{\mathcal{D}}(w\,\Sigma^\star)}$$

The converse is true also: If prediction is possible, then we can use the ***chain rule*** in order to define an associated distribution over Σ^n, for each value of n, and also $\Sigma^\star$ provided the probability of halting given a string is also defined. For this we usually introduce symbol \$ used to terminate strings (strings will be in set $\Sigma^\star\,\$$). Formally, given a string $a_1a_2\ldots a_n\$$,

$$Pr_{\mathcal{D}}(w\$) =$$
$$Pr(a_1|\lambda)\cdot Pr(a_2|a_1)\cdots Pr(a_{i+1}|a_1\cdots a_i)\cdots Pr(a_n|a_1\cdots a_{n-1})\cdot Pr(\$|w)$$

16.2 Probabilities and frequencies

If the goal is to learn probabilistic finite automata, in practice we have no access to probabilities, but to frequencies. We will be told that in the sample, 150 out of the 500 strings start by the letter `a`, not that $\frac{3}{10}$ of the strings in the language start with `a`.

The first crucial issue is that even if mathematically $\frac{1}{2} = \frac{500}{1000}$, this does not fit the picture when we think of the fraction as a relative frequency. More precisely if this fraction is supposed to indicate the empirical probability of an event, then the first fraction would mean 'we have made two tests and out of these two, one was heads', whereas the second fraction is supposed to indicate that 1000 tests were done and 500 resulted in heads.

The difference is that if wanting to say something like 'I guess we get heads half of the time', there are probably more reasons to say this in the second case rather than the first.

The second point is that the automaton the learning algorithm will manipulate during the inference process should both keep track of the frequencies (because of the above reasons) and have an easy and immediate probabilistic interpretation.

For this reason, we introduce a new type of automata, which we call ***frequency finite automata*** (FFA).

We will call *frequency* the number of times an event occurs, and *relative frequency* the number of times an event occurs over the number of times it could have occurred.

Definition 16.2.1 *A **deterministic frequency finite automaton** (DFFA) is a tuple $\mathcal{A} = \langle \Sigma, Q, \mathbb{I}_{fr}, \mathbb{F}_{fr}, \delta_{fr}\rangle$ where*

- *Σ is the alphabet;*

- Q *is a finite set of* states*;*
- $\mathbb{I}_{fr} : Q \to \mathbb{N}$ *(initial-state frequencies); since the automaton is deterministic there is exactly one state* q_λ *for which* $\mathbb{I}_{fr}(q) \neq 0$*;*
- $\mathbb{F}_{fr} : Q \to \mathbb{N}$ *(final-state frequencies);*
- $\delta_{fr} : Q \times \Sigma \times Q \to \mathbb{N}$ *is the transition frequency function;*
- $\delta_{\mathcal{A}} : Q \times \Sigma \to Q$ *is the associated transition function.*

The definition is intended to be adjusted to that of DPFA (Definition 5.2.3, page 103). The notation $\delta_{fr}(q, a, q') = n$, can be interpreted as 'there is a transition from q to q' labelled with a that is used n times'. Obviously, because of determinism, there is at most one state q' such that $\delta_{fr}(q, a, q') > 0$, and this state is such that $q' = \delta_{\mathcal{A}}(q, a)$.

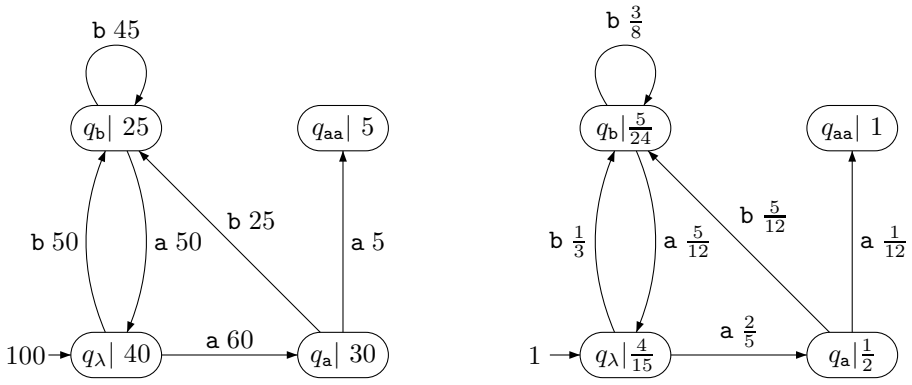
Algorithm 16.1: CONSTRUCTING a DPFA from a DFFA.

Input: a DFFA $\mathcal{A} = \langle \Sigma, Q, \mathbb{I}_{fr}, \mathbb{F}_{fr}, \delta_{fr}, \delta_{\mathcal{A}} \rangle$
Output: a DPFA $\mathcal{B} = \langle \Sigma, Q, q_\lambda, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}}, \delta_{\mathcal{B}} \rangle$
for $q \in Q$ **do** `/* Final probabilities */`
  $\text{FREQ}[q] \leftarrow \mathbb{F}_{fr}(q)$;
  for $a \in \Sigma$ **do** $\text{FREQ}[q] \leftarrow \text{FREQ}[q] + \delta_{fr}(q, a, \delta_{\mathcal{A}}(q, a))$
  $\mathbb{F}_{\mathbb{P}}(q) \leftarrow \frac{\mathbb{F}_{fr}(q)}{\text{FREQ}[q]}$;
  for $a \in \Sigma$ **do**
    $\delta_{\mathbb{P}}(q, a, q') \leftarrow \frac{\delta_{fr}(q,a,q')}{\text{FREQ}[q]}$;
    $\delta_{\mathcal{B}}(q, a) \leftarrow \delta_{\mathcal{A}}(q, a)$
  end
end
return $\mathcal{B}$

There is a relationship, analogous to what can be found in flow diagrams, between the frequencies on the transitions leading to a state and those leaving a state:

Definition 16.2.2
A DFFA $\mathcal{A} = \langle \Sigma, Q, \mathbb{I}_{fr}, \mathbb{F}_{fr}, \delta_{fr}, \delta_{\mathcal{A}} \rangle$ *is said to be* ***consistent*** *or* ***well defined*** *if*$_{def}$ $\forall q \in Q$, $\mathbb{I}_{fr}(q) + \sum_{\substack{q' \in Q \\ a \in \Sigma}} \delta_{fr}(q', a, q) = \mathbb{F}_{fr}(q) + \sum_{\substack{q' \in Q \\ a \in \Sigma}} \delta_{fr}(q, a, q')$.

When the DFFA is well defined, the number of strings entering and leaving a given state is identical. We denote by $\text{FREQ}(q)$ the total number both of entering strings and of leaving strings at state q.



(a) DFFA. (b) DPFA.

Fig. 16.1. FFA to PFA.

Definition 16.2.3 $\forall q \in Q$,

$$\begin{aligned} \text{FREQ}(q) &= \mathbb{I}_{fr}(q) + \sum_{\substack{q' \in Q \\ a \in \Sigma}} \delta_{fr}(q', a, q) \\ &= \mathbb{F}_{fr}(q) + \sum_{\substack{q' \in Q \\ a \in \Sigma}} \delta_{fr}(q, a, q'). \end{aligned}$$

In Definition 16.2.2, consistency is defined as *maintaining the flows*: Any strings that enters a state (or starts in a state) has to leave it (or end there). If this condition is fulfilled the corresponding PFA can be constructed by Algorithm 16.1.

Example 16.2.1 *We depict in Figure 16.1 a* FFA *(16.1(a)) and its associated* PFA *(16.1(b)).*

On the other hand, we will sometimes need to compare frequencies over a FFA. But for this we would have to be able to associate to a FFA a sample in order to say something about the sample. But this is impossible usually, even if the automaton is deterministic (see Exercise 16.1). If the automaton is not deterministic, the situation corresponds to that of estimating the probabilities of a model given a sample. This crucial question will be discussed in more detail in Chapter 17.

When needed, since the transformation from a DFFA to a DPFA is straightforward with Algorithm 16.1, if we denote for a given FFA $\mathcal{A}$ the corresponding PFA by $\mathcal{B}$, $Pr_{\mathcal{A}}(w) = Pr_{\mathcal{B}}(w)$. Therefore we will be allowed to compute directly, for a DFFA $\mathcal{A}$ the probability of any string.

If concentrating on a specific state q of a FFA $\mathcal{A}$, we will denote by $Pr_{\mathcal{A},q}(w)$ the probability of string w obtained when parsed (on the associated PFA from state q with initial probability one. Formally: let $\mathcal{B} =$ FFA_TO_PFA($\mathcal{A}$)$=\langle \Sigma, Q, \mathbb{I}_{\mathbb{P}}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}}, \delta_{\mathcal{A}} \rangle$,

$$Pr_{\mathcal{A},q}(w) = Pr_{\mathcal{C},q}(w) \text{ where } \mathcal{C} = \langle \Sigma, Q, \mathbb{I}_{\mathbb{P}q}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}}, \delta_{\mathcal{C}} \rangle$$

and $\mathbb{I}_{\mathbb{P}q}(q) = 1$, $\forall q' \neq q$, $\mathbb{I}_{\mathbb{P}q}(q') = 0$.

Example 16.2.2 *In the* DPFA $\mathcal{A}$ *represented in Figure 16.1(b),* $Pr_{\mathcal{A},q_\lambda}(\mathtt{ab}) = Pr_{\mathcal{A}}(\mathtt{ab}) = 1 \cdot \frac{2}{5} \cdot \frac{5}{12} \cdot \frac{5}{24} = \frac{5}{144}$ *whereas* $Pr_{\mathcal{A},q_2}(\mathtt{ab}) = 1 \cdot \frac{5}{12} \cdot \frac{1}{3} \cdot \frac{5}{24} = \frac{25}{864}$.

16.3 State merging algorithms

As in the case of learning finite state automata, the best studied algorithms to learn probabilistic automata rely on state merging.

16.3.1 From samples to FFA

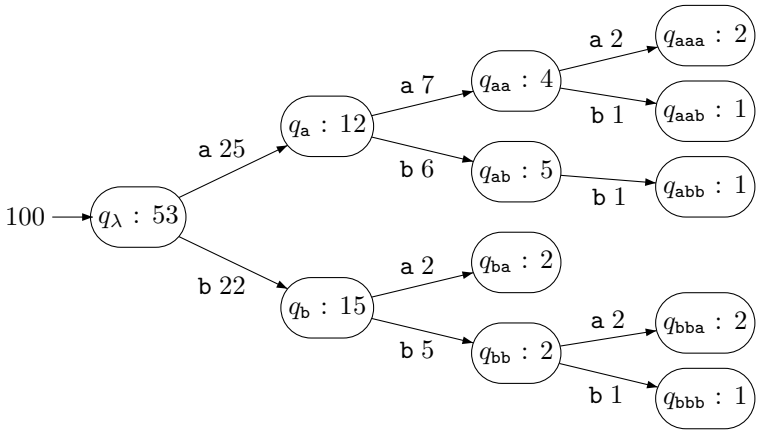


Fig. 16.2. A FPTA constructed from 100 strings.

Definition 16.3.1 *Let S be a multiset of strings from $\Sigma^\star$. The* ***frequency prefix tree acceptor*** FPTA(S) *is the* DFFA : $\langle \Sigma, Q, \mathbb{I}_{fr}, \mathbb{F}_{fr}, \delta_{fr}, \delta_{\mathcal{A}} \rangle$ *where*

- $Q = \{q_u : u \in \text{PREF}(S)\}$,
- $\mathbb{I}_{fr}(q_\lambda) = |S|$,
- $\forall ua \in \text{PREF}(S)$, $\delta_{fr}(q_u, a, q_{ua}) = |S|_{ua\,\Sigma^\star}$,
- $\forall ua \in \text{PREF}(S)$, $\delta(q_u, a) = q_{ua}$,
- $\forall u \in \text{PREF}(S)$, $\mathbb{F}_{fr}(q_u) = |S|_u$.

For example, in Figure 16.2, we have represented the FPTA built from sample S, which is a multiset of size 100, $S = \{(\lambda, 53), (\mathtt{a}, 12), (\mathtt{b}, 15), (\mathtt{aa}, 4), (\mathtt{ab}, 5), (\mathtt{ba}, 2), (\mathtt{bb}, 2), (\mathtt{aaa}, 2), (\mathtt{aab}, 1), (\mathtt{abb}, 1), (\mathtt{bba}, 2), (\mathtt{bbb}, 1)\}$.

16.3.2 Merging and folding

As in Chapter 12, we describe algorithms that start with a prefix tree acceptor and iteratively try to merge states, with a recursive folding process.

Given a DFFA the merging operation takes two states q and q', where q is a RED state and $q' = \delta(q_f, a)$ is a BLUE state, root of a tree. This also means that $\delta(q_f, a) \neq q_\lambda$. The operation is described by Algorithm STOCHASTIC-MERGE (16.2). First, transition $\delta_{\mathcal{A}}(q_f, a)$ is redirected to q. Then the folding of the tree rooted in q' into the automaton at state q is recursively called.

Algorithm 16.2: STOCHASTIC-MERGE.

Input: a FFA $\mathcal{A}$, 2 states $q \in$ RED, $q' \in$ BLUE
Output: $\mathcal{A}$ updated
Let (q_f, a) be such that $\delta_{\mathcal{A}}(q_f, a) = q'$;
$n \leftarrow \delta_{fr}(q_f, a, q')$;
$\delta_{\mathcal{A}}(q_f, a) \leftarrow q$;
$\delta_{fr}(q_f, a, q) \leftarrow n$;
$\delta_{fr}(q_f, a, q') \leftarrow 0$;
return STOCHASTIC-FOLD$(\mathcal{A}, q, q')$

For the resulting automaton to be deterministic, recursive folding is required. In this case Algorithm STOCHASTIC-FOLD (16.3) is used.

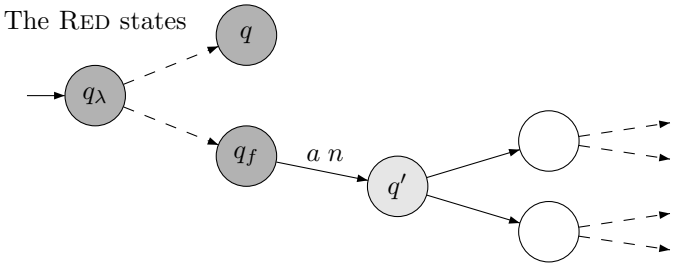
The situation is schematically represented in Figures 16.3(a) and 16.3(b): Suppose we want to merge state q' (BLUE) with state q (RED). First q_f is the unique predecessor of q', and the transition from q_f labelled by a (and used n times) is redirected to q. The folding can then take place.

Example 16.3.1 *Let us examine the merge-and-fold procedure with more care on an example. Suppose we are in the situation represented in Figure 16.4. We want to merge state $q_{\mathtt{aa}}$ with q_λ.*

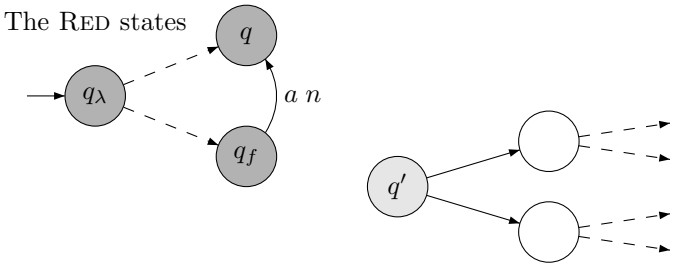
Once the redirection has taken place (Figure 16.5), state $q_{\mathtt{aa}}$ is effectively merged (Figure 16.6) with q_λ (leading to modification of the values of $\text{FREQ}_{\mathcal{A}}(q_\lambda)$, $\delta_{fr}(q_\lambda, \mathtt{a})$ and $\delta_{fr}(q_\lambda, \mathtt{b})$). In dashed lines the recursive merges that are still to be done.

Algorithm 16.3: STOCHASTIC-FOLD.

Input: a DFFA $\mathcal{A}$, 2 states q and q'
Output: $\mathcal{A}$ updated, where subtree in q' is folded into q
$\mathbb{F}_{fr}(q) \leftarrow \mathbb{F}_{fr}(q) + \mathbb{F}_{fr}(q')$;
for $a \in \Sigma$ *such that* $\delta_{\mathcal{A}}(q', a)$ *is defined* **do**
  if $\delta_{\mathcal{A}}(q, a)$ *is defined* **then**
    $\delta_{fr}(q, a, \delta_{\mathcal{A}}(q, a)) \leftarrow \delta_{fr}(q, a, \delta_{\mathcal{A}}(q, a)) + \delta_{fr}(q', a, \delta_{\mathcal{A}}(q', a))$;
    $\mathcal{A} \leftarrow$STOCHASTIC-FOLD$(\mathcal{A}, \delta_{\mathcal{A}}(q, a), \delta_{\mathcal{A}}(q', a))$
  else
    $\delta_{\mathcal{A}}(q, a) \leftarrow \delta_{\mathcal{A}}(q', a)$;
    $\delta_{fr}(q, a, \delta_{\mathcal{A}}(q, a)) \leftarrow \delta_{fr}(q', a, \delta_{\mathcal{A}}(q', a))$
  end
end
return $\mathcal{A}$



(a) The situation before merging.



(b) The situation after merging and before folding.

Fig. 16.3. Merge-and-fold.

Then (Figure 16.7) $q_{\mathtt{aaa}}$ *is folded into* $q_{\mathtt{a}}$*, and finally* $q_{\mathtt{aab}}$ *into* $q_{\mathtt{b}}$*. The result is represented in Figure 16.8.*

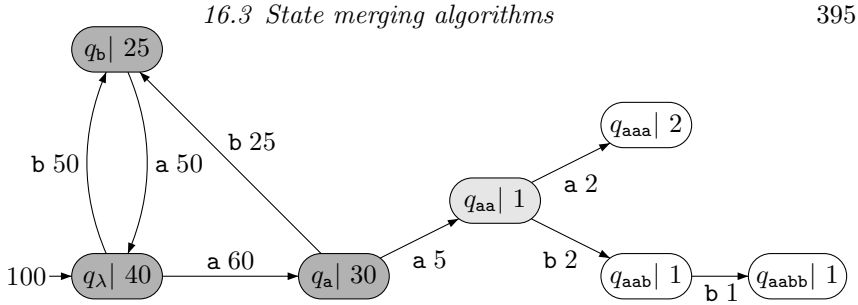


Fig. 16.4. Before merging $q_{\mathtt{aa}}$ with q_λ and redirecting transition $\delta_{\mathcal{A}}(q_{\mathtt{a}}, \mathtt{a})$ to q_λ.

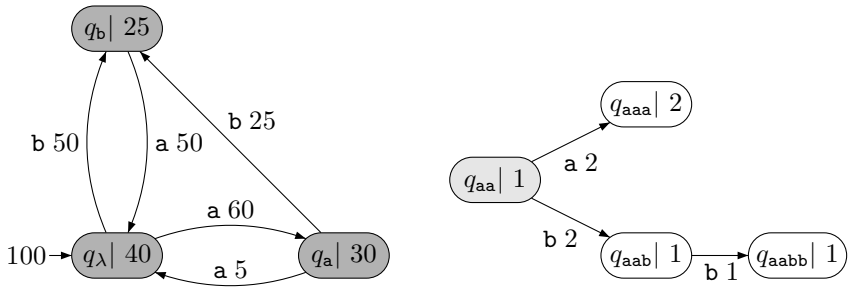


Fig. 16.5. Before folding $q_{\mathtt{aa}}$ into q_λ.

16.3.3 Deciding equivalence

The next difficult problem is that of deciding if two states q_u and q_v should be merged. This usually corresponds to testing in an automaton if the two distributions obtaining by using the same automaton but changing the initial frequency function are equivalent (better said, if they are equivalent when taking into account the fact that these two distributions are only approximatively described by samples).

So the question is really about deciding if two states of a given DFFA $\mathcal{A} = \langle \Sigma, Q, \mathbb{I}_{fr}, \mathbb{F}_{fr}, \delta_{fr}, \delta_{\mathcal{A}} \rangle$ (one BLUE and one RED) if these should be

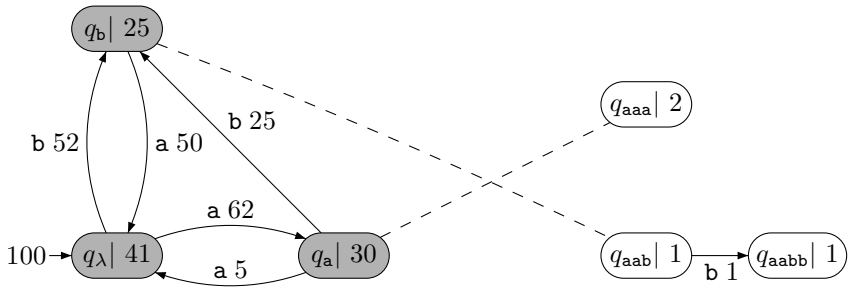


Fig. 16.6. Folding in $q_{\mathtt{aa}}$.

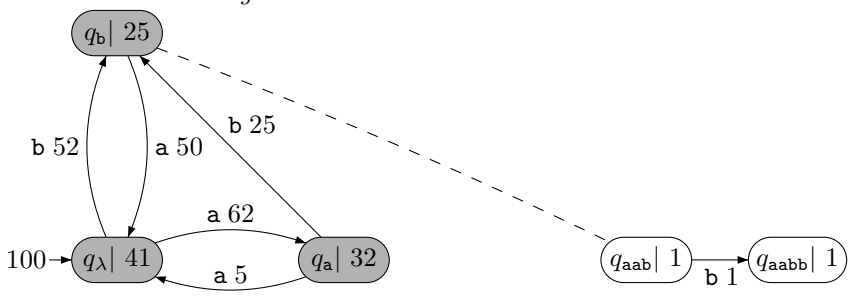


Fig. 16.7. Folding in $q_{\mathtt{aaa}}$.

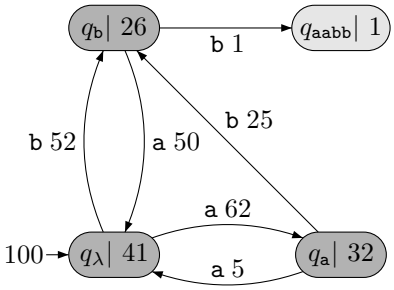


Fig. 16.8. Folding in $q_{\mathtt{aab}}$.

merged, which really means discovering from both the sampling process and the merges that have been done up to that point if $\mathcal{D}_{\mathcal{A},q_u}$ and $\mathcal{D}_{\mathcal{A},q_v}$ are sufficiently close. One possibility is to use here the results and algorithms described in Section 5.4 where we studied distances between distributions, and in a way, this is what is done.

16.3.4 Equivalence on prefixes

Let us describe the key idea behind the test used by Algorithm Alergia, which will be described in detail in Section 16.4. Obviously, if the two distributions given by automata $\mathcal{A}_1$ and $\mathcal{A}_2$ (whether Pfa or Ffa) are equivalent, then we should have:

$$\forall w \in \Sigma^\star,\ Pr_{\mathcal{A}_1}(w) = Pr_{\mathcal{A}_2}(w)$$

or at least, when testing over random samples, these observed relative frequencies should be sufficiently close:

$$\forall w \in S,\ Pr_{\mathcal{A}_1}(w) \approx Pr_{\mathcal{A}_2}(w).$$

But this is not going to be possible to test, partly because the number of

strings is infinite, but mainly because the values involved can be very small whenever the distribution is over many different strings.

A better idea is to compare the prefixes:

$$\forall w \in \Sigma^\star \ \ Pr_{\mathcal{A}_1}(w\,\Sigma^\star) \approx Pr_{\mathcal{A}_2}(w\,\Sigma^\star).$$

This test will be used (over a finite sample) but it should be noticed that it can mean having to test on all possible strings w; so we want to do this in such a way as to keep the number of tests small (and somehow bounded by the size of the sample).

16.3.5 Added bias

A second altogether different way of testing if two distributions are similar corresponds to taking the bet (we call it bias) that there will be a distinguishing string. This idea will be exploited in the Algorithm DSAI described in Section 16.5. Notice that the above quantities should diminish at exponential rate with the length of the prefixes. This means that if the difference between two states depends on a long prefix, we may not have enough information in the sample to be able to notice this difference. In order to deal with this problem, we can hope that two states are different because of some string which is unequally probable when reading from each of these states. We call this a *distinguishing string*. Obviously if it is normal that such distinguishing strings exist, the bias lies is in the fact of believing that the difference of probabilities is large.

Definition 16.3.2 *Given $\mu > 0$, a* DPFA *(or* DFFA*) $\mathcal{A} = \langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{P}, \delta_\mathbb{P} \rangle$ is* ***μ-distinguishable*** *if$_{def}$ for every pair of different states q_u and q_v in Q there exists a string w in $\Sigma^\star$ such that $|Pr_{\mathcal{A},q_u}(w) - Pr_{\mathcal{A},q_v}(w)| \geq \mu$.*

This means that one can check, during the merge-and-fold process, if two states are μ-distinguished or not. It should be noticed that if two states are μ-distinguishable, then every time we will want to merge two states q_u and q_v, with high enough probability there is a string w such that $|Pr_{\mathcal{A}_{q_u}}(w) - Pr_{\mathcal{A}_{q_v}}(w)| > \mu$.

In order for this test to make sense, a threshold (denoted by t_0) will be used. The threshold puts a limit: How many strings entering the BLUE state are needed in order to have enough statistical evidence to consider making a merge. Only states having enough evidence will be checked.

An important question concerns the future of those states that do not meet this quantity. There are a number of possibilities, some more theoretically founded (like just merging in a unique state all the remaining states at the

end of the learning process), and some using all types of heuristics. We shall not specifically recommend any here.

16.3.6 Promoting

Just as in the RPNI case, promotion takes place when the current (BLUE) state can be merged with no red state (RED). In that case, first the BLUE state is promoted and becomes RED. Then all the successors of the RED states that are not RED themselves become BLUE. Notice also that no frequencies are modified.

In practice it is possible to avoid exploring all the states in order to redefine the colourings, as, usually, only the successor of the promoted state can change to BLUE.

16.3.7 Withdrawing

There is, in the case of probabilistic automata learning, a last operation that didn't make much sense when working with non-probabilistic automata. At some moment the algorithm may have to take a decision about a state (or an edge) used by very few strings. This may seem like a trifling (there is not much data so even if we made a mistake it could seem it would not be a costly one), but this is not true: To take an extreme example suppose that from a state just one string is generated, perhaps $\mathtt{a}^n$ with n very large (much larger than the actual size of the sample). Then making a mistake and perhaps merging this state with some RED state can change in a far too drastic way the frequencies because the same string will be folded many times into the automaton. For details, see Exercise 16.2.

This phenomenon would deserve to be analysed with much more care, but for the moment we will just solve the question by using a threshold t_0 with each algorithm: If the amount of data reaching a BLUE state is less than the threshold, then the state (and all its successors) is withdrawn. It remains in the FFA to maintain consistency, but will not be checked for merging.

16.4 ALERGIA

In Chapter 12 we described algorithm RPNI. Algorithm ALERGIA follows the same ideas: A predefined ordering of the states, a compatibility test and then a merging operation. The general algorithm visits the states through two loops and attempts (recursively) to merge states.

16.4.1 The algorithm

Algorithm 16.4: ALERGIA-TEST.

Input: a FFA $\mathcal{A}$, f_1, n_1, f_2, n_2, $\alpha > 0$
Output: a Boolean indicating if the frequencies $\frac{f_1}{n_1}$ and $\frac{f_2}{n_2}$ are sufficiently close
$\gamma \leftarrow \left|\frac{f_1}{n_1} - \frac{f_2}{n_2}\right|$;
return $\left(\gamma < (\sqrt{\frac{1}{n_1}} + \sqrt{\frac{1}{n_2}}) \cdot \sqrt{\frac{1}{2}\ln\frac{2}{\alpha}}\right)$

The compatibility test makes use of the Hoeffding bounds. The algorithm ALERGIA-COMPATIBLE (16.5) calls test ALERGIA-TEST (16.4) as many times as needed, this number being finite due to the fact that the recursive calls visit a tree.

Algorithm 16.5: ALERGIA-COMPATIBLE.

Input: a FFA $\mathcal{A}$, two states q_u, q_v, $\alpha > 0$
Output: q_u and q_v compatible?
Correct$\leftarrow$ **true**;
if ALERGIA-TEST($\mathbb{F}_{\mathbb{P}_\mathcal{A}}(q_u)$, $\text{FREQ}_\mathcal{A}(q_u)$, $\mathbb{F}_{\mathbb{P}_\mathcal{A}}(q_v)$, $\text{FREQ}_\mathcal{A}(q_v), \alpha$) **then**
Correct$\leftarrow$ **false**;
for $a \in \Sigma$ **do**
  if ALERGIA-TEST($\delta_{fr}(q_u, a)$, $\text{FREQ}_\mathcal{A}(q_u)$, $\delta_{fr}(q_v, a)$, $\text{FREQ}_\mathcal{A}(q_v), \alpha$) **then** Correct$\leftarrow$ **false**
end
return Correct

The basic function CHOOSE is as follows: Take the smallest state in an ordering that has been done at the beginning (on the FPTA). But it has been noticed by different authors that using a data-driven approach offered better guarantees. The test that is used to decide if the states are to be merged or not (function COMPATIBLE) is based on the Hoeffding test made on the relative frequencies of the empty string and of each prefix.

The quantities that are compared are:

$$\frac{\mathbb{F}_{fr}(q)}{\text{FREQ}(q)} \text{ and } \frac{\mathbb{F}_{fr}(q')}{\text{FREQ}(q')}$$

and also for each symbol a in the alphabet Σ,

$$\frac{\delta_{fr}(q,a)}{\text{FREQ}(q)} \text{ and } \frac{\delta_{fr}(q',a)}{\text{FREQ}(q')}.$$

Algorithm 16.6: ALERGIA.

Input: a sample S, $\alpha > 0$
Output: a FFA $\mathcal{A}$
Compute t_0, threshold on the size of the multiset needed for the test to be be statistically significant;
$\mathcal{A} \leftarrow \text{FPTA}(S)$;
$\text{RED} \leftarrow \{q_\lambda\}$;
$\text{BLUE} \leftarrow \{q_a : a \in \Sigma \cap \text{PREF}(S)\}$;
while CHOOSE q_b *from* BLUE *such that* $\text{FREQ}(q_b) \geq t_0$ **do**
  if $\exists q_r \in \text{RED} : \text{ALERGIA-COMPATIBLE}(\mathcal{A}, q_r, q_b, \alpha)$ **then**
    $\mathcal{A} \leftarrow \text{STOCHASTIC-MERGE}(\mathcal{A}, q_r, q_b)$
  else
    $\text{RED} \leftarrow \text{RED} \cup \{q_b\}$
  end
  $\text{BLUE} \leftarrow \{q_{ua} : ua \in \text{PREF}(S) \wedge q_u \in \text{RED}\} \setminus \text{RED}$
end
return $\mathcal{A}$

16.4.2 Running the algorithm

Running probabilistic automata learning algorithms in a clear and understandable way is a difficult task: On one hand the targets should be meaningful and therefore contain cycles, and on the other hand the samples should be large enough to offer statistical evidence for the decisions that are taken by the algorithm. This results in a number of states in the FPTA much too large to be represented graphically.

In order to explain the way ALERGIA works, we shall use a sample of 1000 strings of (artificially) bounded length (just in order to make things simpler, but without loss of generality). The sample is represented in Table 16.1. The target, supposed to have generated the sample, is represented in Figure 16.9. The starting point is the DFFA given in Figure 16.10. State q_λ is RED and states $q_{\mathtt{a}}$ and $q_{\mathtt{b}}$ are BLUE.

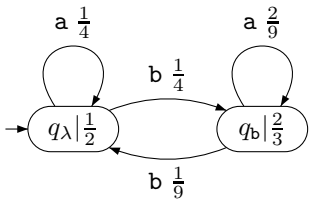


Fig. 16.9. The target.

Table 16.1. *The data.*

λ	490	abb	4	abab	2	aaaaa	1
a	128	baa	9	abba	2	aaaab	1
b	170	bab	4	abbb	1	aaaba	1
aa	31	bba	3	baaa	2	aabaa	1
ab	42	bbb	6	baab	2	aabab	1
ba	38	aaaa	2	baba	1	aabba	1
bb	14	aaab	2	babb	1	abbaa	1
aaa	8	aaba	3	bbaa	1	abbab	1
aab	10	aabb	2	bbab	1		
aba	10	abaa	2	bbba	1		

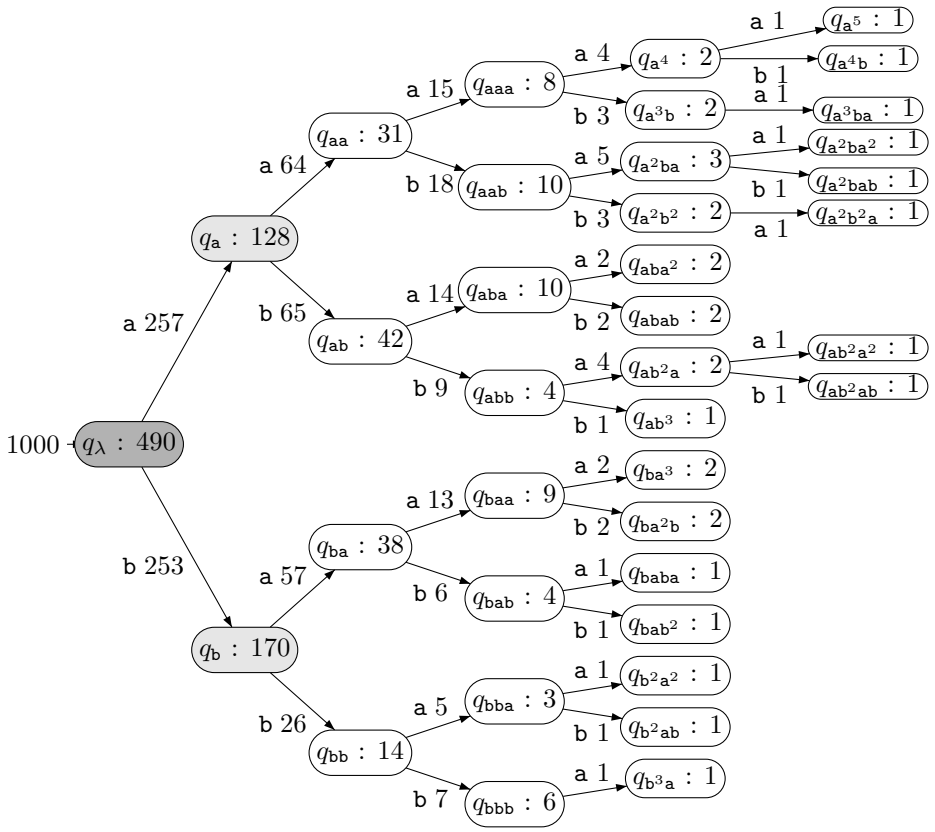


Fig. 16.10. The full FPTA, with coloured states.

Parameter α is arbitrarily set to 0.05. We choose 30 as a value for threshold t_0. In theory, the value can be computed: It corresponds to a value for

which both errors (to make a wrong merge or not to merge at all) can be bounded (see Section 16.9).

ALERGIA tries merging $q_{\mathtt{a}}$ and q_λ, which means comparing $\frac{490}{1000}$ with $\frac{128}{257}$, $\frac{257}{1000}$ with $\frac{64}{257}$, $\frac{253}{1000}$ with $\frac{65}{257}$, ...

For example, $\left|\frac{\delta_{fr}(q_\lambda,\mathtt{a})}{\text{FREQ}_{\mathcal{A}_q}(q_\lambda)} - \frac{\delta_{fr}(q_{\mathtt{a}},\mathtt{a})}{\text{FREQ}_{\mathcal{A}_{q_{\mathtt{a}}}}(q_{\mathtt{a}})}\right| = \left|\frac{257}{1000} - \frac{64}{257}\right| = 0.008$ is less than $\left(\sqrt{\frac{1}{1000}} + \sqrt{\frac{1}{257}}\right)\sqrt{\frac{1}{2}\ln\frac{2}{\alpha}}$ which for $\alpha = 0.05$ is about 0.1277.

The merge is accepted and the new automaton is depicted in Figure 16.11.

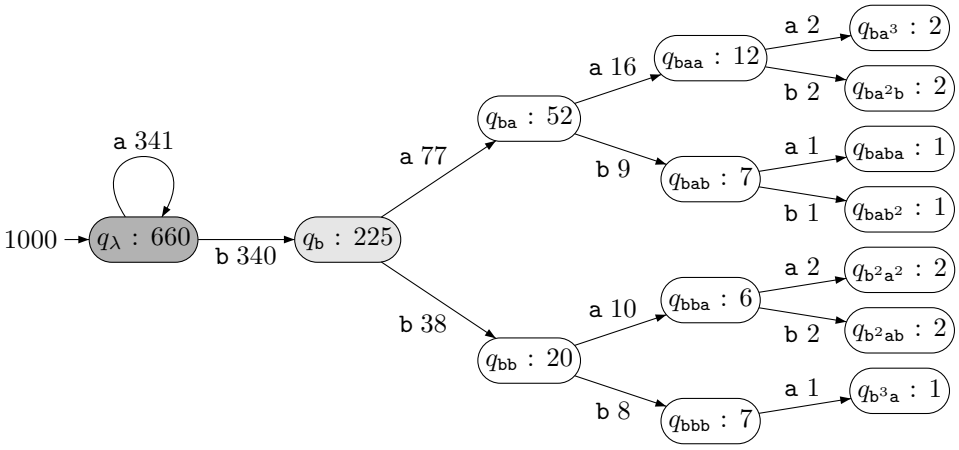


Fig. 16.11. After merging $q_{\mathtt{a}}$ and q_λ.

The algorithm tries merging $q_{\mathtt{b}}$ (the only BLUE state at that stage) and q_λ. This fails after comparing $\frac{\mathbb{F}_{fr}(q_\lambda)}{\text{FREQ}(q_\lambda)} = \frac{661}{1340}$ and $\frac{\mathbb{F}_{fr}(q_{\mathtt{b}})}{\text{FREQ}(q_{\mathtt{b}})} = \frac{222}{339}$ (giving in the test 0.162 (for the γ) against 0.111). So state $q_{\mathtt{b}}$ gets promoted; the current automaton is represented in Figure 16.12.

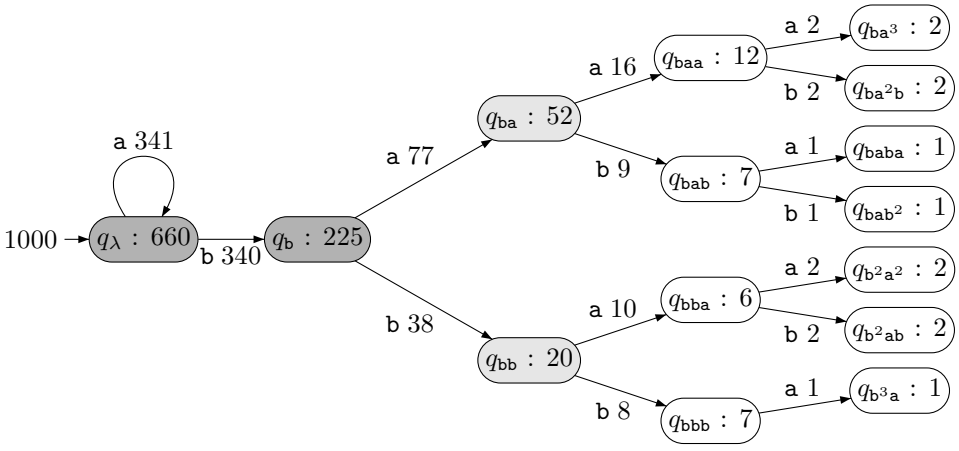


Fig. 16.12. Promoting $q_{\mathtt{b}}$.

The algorithm now tries merging $q_{\mathtt{ba}}$ and q_λ. This fails when comparing $\frac{660}{1341}$ with $\frac{52}{77}$. The difference between the two relative frequencies is too large, and when algorithm ALERGIA-TEST is called (with as before a value for α of 0.05), the result is negative.

The algorithm now tries merging $q_{\mathtt{ba}}$ and $q_{\mathtt{b}}$. This merge is accepted, partly because there is little information to tell us otherwise. The new automaton is depicted in Figure 16.13.

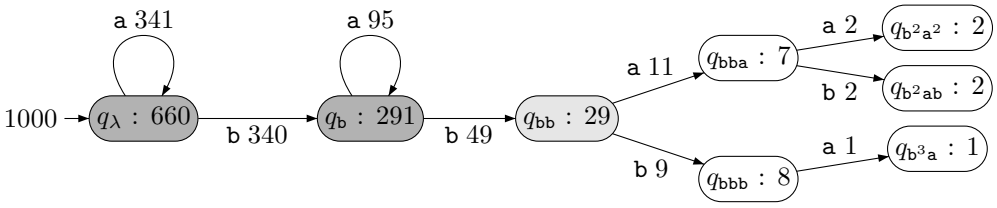


Fig. 16.13. After merging $q_{\mathtt{ba}}$ and $q_{\mathtt{b}}$.

We now try merging $q_{\mathtt{bb}}$ and q_λ. This is also accepted. The new automaton is depicted in Figure 16.14.

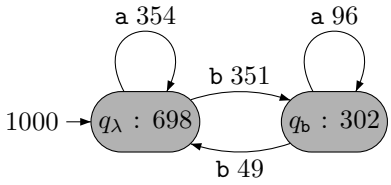


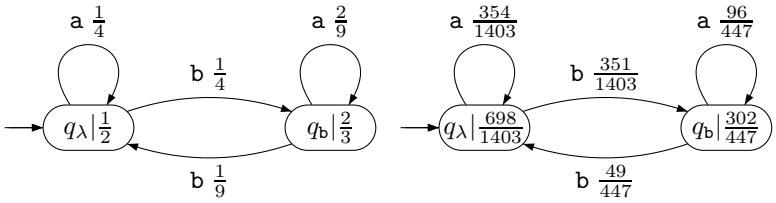
Fig. 16.14. Merging $q_{\mathtt{bb}}$ and q_λ.

Only two states are left. By using Algorithm DFFA_TO_DPFA (16.1) we obtain the DPFA represented in Figure 16.15(b), which can be compared with the target (16.15(a)). One should notice by how little the bad merges have been avoided, in a relatively simple case. This tends to show that the number of examples needed to run the algorithm in reasonable conditions has to be much larger.

16.4.3 Why ALERGIA *works*

We do not intend to prove the properties of algorithm ALERGIA here. These can be found (albeit for a more theoretical version of the algorithm) in different research papers. Nevertheless we can state the following facts:

Proposition 16.4.1 *Algorithm* ALERGIA *identifies any* DPFA *in the limit with probability one and runs in time polynomial in* $\|S\|$.



(a) The target. (b) The result.

Fig. 16.15. Comparing the target and the result.

The compatibility test is in fact much cheaper than the one for RPNI. No merges need to be done before the test. Therefore this can be done through counting and in time linear with the size of the tree rooted in the BLUE node. This is the reason for which the data driven approach described in Section 14.5.2 (page 345) has been used with good results.

Since this is done inside a double loop the overall complexity is very excessively bounded by a cubic polynomial. Users report that the practical complexity is linear.

Convergence is much more of a tricky matter. There are two issues here:

- The first is to identify the structure;
- The second is to identify the probabilities. This can be done through different techniques, such as Algorithm 10.2 for Stern-Brocot trees (page 241).

There have been many variants of algorithm ALERGIA with modifications of:

- the order of the merges,
- the statistical test which is used,
- the way we deal with the uncertain states.

16.5 Using distinguishing strings

A different test can be used provided an extra bias is added: Given a DPFA $\mathcal{A}$, two states q and q' are said to be *μ-distinguishable* if there exists a string x such that $\left|Pr_{\mathcal{A},q}(x) - Pr_{A,q'}(x)\right| \geq \mu$ where $Pr_{\mathcal{A},q}(x)$ is the probability of generating x by $\mathcal{A}$ when taking q as the initial state. This is equivalent to saying

$$\mathbf{L}_\infty(\mathcal{D}_{\mathcal{A},q}, \mathcal{D}_{\mathcal{A},q'}) \geq \mu.$$

The chosen bias corresponds to believing that when things are different,

then it should be so for explainable or visible reasons: At least one string should testify to why there are two states instead of one. One may sometimes relax this condition in several ways, for instance by asking this to be true only for states that themselves have a fair chance of being reached.

When trying in an algorithm to detect this, it means that we decide upon a merge by testing whether:

'Does there exist in $\Sigma^\star$ a string x such that
$\left|Pr_{\mathcal{A},q}(x) - Pr_{\mathcal{A},q'}(x)\right| \geq \mu$?'

Notice that again it is an 'optimistic' argument: A decision of merging is taken when there is nothing saying it is a bad idea to do so.

16.5.1 The algorithm

We describe here an algorithm we call DSAI (***Distinguishing Strings Automata Inference***), of which several variants exist. The one we present here is simple, but in order to obtain convergence results, one should be prepared to handle a number of extra parameters.

The idea is, given a RED state q_r and a BLUE state q_b (which is visited a minimum number of times), to compute $\mathbf{L}_\infty(\mathcal{D}_{\mathcal{A},q_r}, \mathcal{D}_{\mathcal{A},q_b})$. If this quantity is less than $\frac{\mu}{2}$ the two states are compatible and are merged. A state that is not reached enough times is kept till the end in order to avoid an accumulation of small errors.

In the version presented here, these less important states are just kept as a tree. For obvious reasons, merging together all the states reached a number of times inferior to the threshold may be a better strategy.

In general, computing $\mathbf{L}_\infty$ between two deterministic DPFA or FFA is not simple. But in this case the computation is between a DFFA and a FPTA. Let k be the number of strings in the FPTA with root q_b Let us denote by S_b the set of all strings x such that $Pr_{\mathcal{A},q_b}(x) \neq 0$. Note that $k < |S_b| \leq |S|$. $\mathbf{L}_\infty(\mathcal{D}_{\mathcal{A},q_r}, \mathcal{D}_{\mathcal{A},q_b})$ is either reached by a string x in S_b, and there are only a finite number of such strings, or by one of the $k+1$ most probable strings for $\mathcal{D}_{\mathcal{A},q_r}$. We denote by $\text{mps}(\mathcal{A}_{q_r}, k+1)$ this set.

Given a DPFA $\mathcal{A}$ and an integer n, one can compute in polynomial time the set of the n most probable strings in $\mathcal{D}_\mathcal{A}$ (see Exercise 5.4).

Therefore, in this case, since q_b is a BLUE state, and therefore the root of a tree, DSAI-COMPATIBLE computes this distance ($\mathbf{L}_\infty(\mathcal{D}_{\mathcal{A},q_r}, \mathcal{D}_{\mathcal{A},q_b})$).

Algorithm DSAI (16.8) is structured similarly to ALERGIA.

The key idea is that on one hand, if two states are not to be merged, the

Algorithm 16.7: DSAI-COMPATIBLE.

Input: a FFA $\mathcal{A}$, two states $q \in$ RED and $q' \in$ BLUE, $\mu > 0$
Output: a Boolean indicating if q and q' are compatible
for $x \in \Sigma^\star : Pr_{\mathcal{A},q'}(x) \neq 0$ **do**
  if $\left|Pr_{\mathcal{A},q}(x) - Pr_{\mathcal{A},q'}(x)\right| > \frac{\mu}{2}$ **then return false**
end
for *each* x *in* $\text{mps}(\mathcal{A}_{q_r}, k+1)$ **do**
  if $Pr_{\mathcal{A},q'}(x) > \frac{\mu}{2}$ **then return false**
end
return true

Algorithm 16.8: DSAI.

Input: a sample S, $\mu > 0$
Output: a FFA $\mathcal{A}$
Compute t_0, threshold on the size of the multiset needed for the test to be be statistically significant;
$\mathcal{A} \leftarrow$ FPTA(S) ;
RED $\leftarrow \{q_\lambda\}$;
BLUE $\leftarrow \{q_a : a \in \text{PREF}(S)\}$;
while CHOOSE q_b *from* BLUE *such that* FREQ$(q_b) \geq t_0$ **do**
  if $\exists q_r \in$ RED : DSAI-COMPATIBLE$(\mathcal{A}, q_r, q_b, \mu)$ **then**
    $\mathcal{A} \leftarrow$ STOCHASTIC-MERGE$(\mathcal{A}, q_r, q_b)$
  else
    RED $\leftarrow$ RED $\cup \{q_b\}$
  end
  BLUE $\leftarrow \{q_{ua} : ua \in \text{PREF}(S) \wedge q_u \in \text{RED}\} \setminus$ RED
end
return $\mathcal{A}$

distinguishing string exists and will be seen, and, on the other hand, if the two states compared correspond in fact to the same state in the target, then the μ bound is sufficiently big to avoid reaching the wrong conclusion by 'bad luck'.

16.5.2 Running the algorithm

We run the algorithm DSAI on the same sample as in Section 16.4.2. The sample, of size 1000, is recalled in Table 16.2, whereas the FPTA is recalled in Figure 16.16.

Table 16.2. *The data.*

λ	490	abb	4	abab	2	aaaaa	1
a	128	baa	9	abba	2	aaaab	1
b	170	bab	4	abbb	1	aaaba	1
aa	31	bba	3	baaa	2	aabaa	1
ab	42	bbb	6	baab	2	aabab	1
ba	38	aaaa	2	baba	1	aabba	1
bb	14	aaab	2	babb	1	abbaa	1
aaa	8	aaba	3	bbaa	1	abbab	1
aab	10	aabb	2	bbab	1		
aba	10	abaa	2	bbba	1		

The starting point is represented in Figure 16.16.

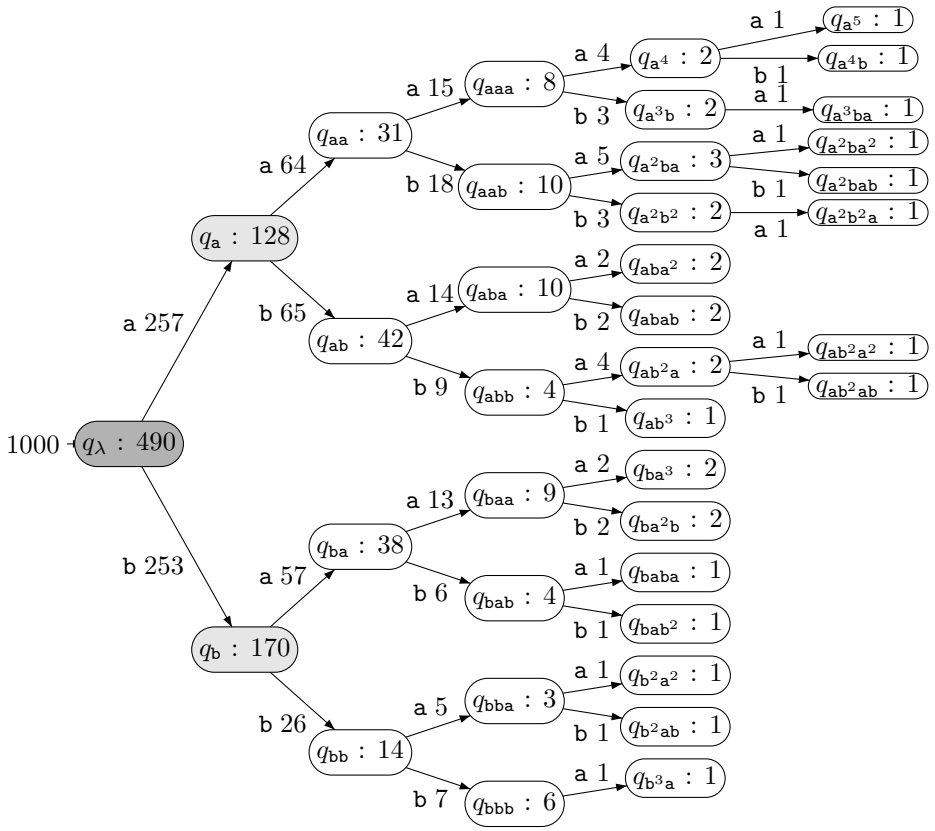


Fig. 16.16. The FPTA, with coloured states.

We set μ to 0.2 and the threshold has value $t_0 = 30$. This means that each

time the difference between the two relative frequencies is less than $\mu/2$ we are reasonably sure that this is because the two probabilities are not further apart than μ, and therefore they are identical.

Algorithm DSAI then tries merging $q_{\mathtt{a}}$ and q_λ. This corresponds to computing $\mathbf{L}_\infty(\mathcal{D}_{\mathcal{A},q_\lambda}, \mathcal{D}_{\mathcal{A},q_{\mathtt{a}}})$. The maximum value of $|Pr_{\mathcal{A}_{q_\lambda}}(x) - Pr_{\mathcal{A}_{q_{\mathtt{a}}}}(x)|$ is obtained for $x = \lambda$ and is 0.008. The merge is accepted.

The new automaton is depicted in Figure 16.17.

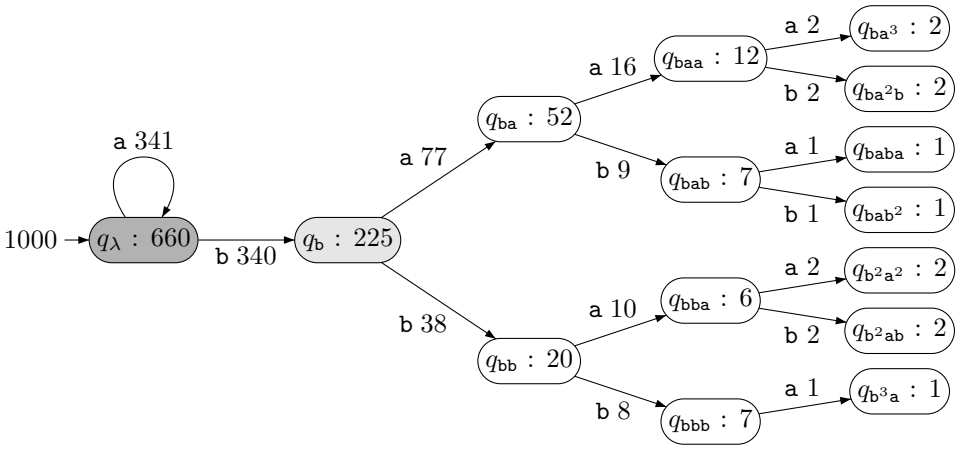


Fig. 16.17. After merging $q_{\mathtt{a}}$ and q_λ.

DSAI tries merging $q_{\mathtt{b}}$ and q_λ. But for string λ we have $Pr_{\mathcal{A}_{q_\lambda}}(\lambda) = \frac{660}{1341}$, $Pr_{\mathcal{A}_{q_{\mathtt{b}}}}(\lambda) = \frac{225}{340}$ and the difference is 0.170 which is more than $\frac{\mu}{2}$. State $q_{\mathtt{b}}$ is therefore promoted.

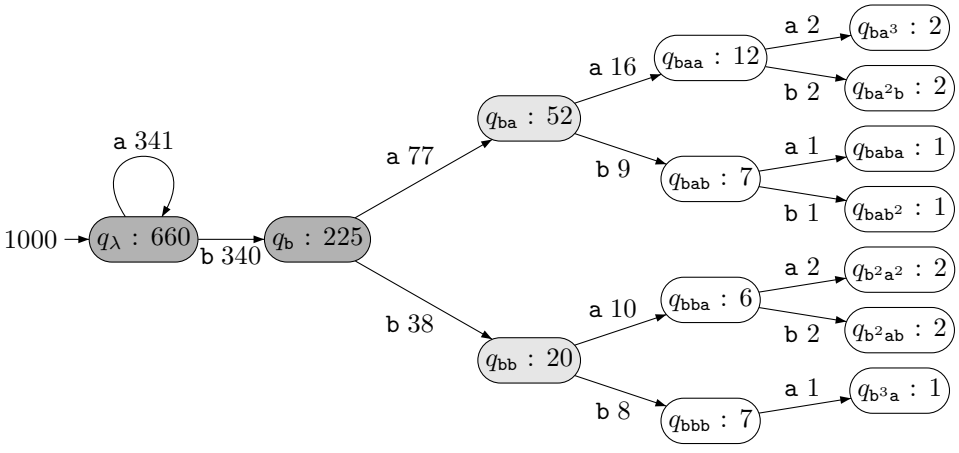
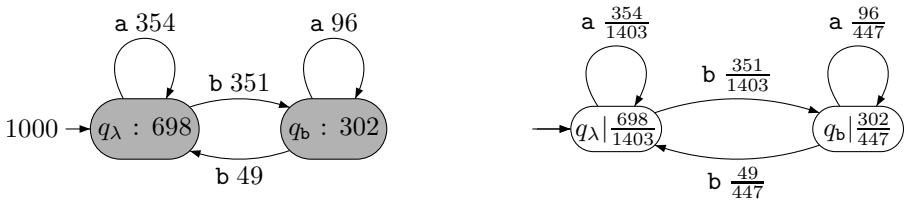


Fig. 16.18. After merging $q_{\mathtt{ba}}$ and $q_{\mathtt{b}}$.

The algorithm now tries merging $q_{\mathtt{ba}}$ and q_λ. This fails because $|Pr_{\mathcal{A}_{q_\lambda}}(\lambda) - Pr_{\mathcal{A}_{q_{\mathtt{ba}}}}(\lambda)| = 0.675 - 0.492 > \frac{\mu}{2}$. But the possible merge between $q_{\mathtt{ba}}$ and $q_{\mathtt{b}}$ is accepted (the largest difference is again for string λ, with 0.014). The new DFFA is represented in Figure 16.18. We now try merging $q_{\mathtt{bb}}$ and q_λ. This is accepted, since $\mathbf{L}_\infty(\mathcal{D}_{\mathcal{A},q_\lambda}, \mathcal{D}_{\mathcal{A},q_{\mathtt{bb}}}) = |Pr_{\mathcal{A}_{q_\lambda}}(\lambda) - Pr_{\mathcal{A}_{q_{\mathtt{bb}}}}(\lambda)|$=0.044.

The final automaton is shown in Figure 16.19 (DFFA on the left, DPFA on the right).



(a) The DFFA. (b) The DPFA.

Fig. 16.19. After merging $q_{\mathtt{ba}}$ and q_λ.

The result is exactly the same as that returned by ALERGIA. But, of course, this need not be so in general.

16.5.3 Why it works

In the literature, a version of Algorithm DSAI, in which the merges are only tested if they do not introduce cycles, is proved to have a number of features, of which some positive PAC learning properties. Their results do not extend to the general case where there is no condition set on the topology of the automata.

Nevertheless, we do have the following result provided some small changes are made:

Proposition 16.5.1 *Algorithm* DSAI *identifies any μ-distinguishable* DPFA *in the limit with probability one and runs in polynomial time.*

Proof If we fix the threshold, with a number of strings increasing, the tests are going to return **true**, for all but a finite number of values. Then, with techniques studied in earlier sections, the probabilities can be identified instead of being estimated.

The complexity of the algorithm is clearly polynomial with the size of the sample. □

16.6 Hardness results regarding ALERGIA and DSAI

16.6.1 A difficult target for ALERGIA *and* DSAI

Let us note two things concerning the approaches presented in Sections 16.4 and 16.5: Limiting oneself to acyclic automata offers an important theoretical advantage. This allows one to consider effectively that the strings have been randomly and independently sampled (which is a necessary condition for the statistical tests considered). On the other hand, in the case of ALERGIA, from the moment a merge has introduced a cycle, the same string is going to be used over and over. Intuitively this is probably not a bad thing, as the example is going to reinforce, if anything, the next set of tests.

In both cases convergence is achieved, but in the second case identification is met from a polynomial number of examples. Yet the μ parameter is important. It is then not too difficult to find a class of languages for which everything will fail: This family is indexed by an integer n, and consists of languages generated by the (indexed) probabilistic grammar G_n which generates:

- if $|u| < n$ $Pr_G(u) = 0$,
- if $|u| = n$ and $m \in \mathtt{a}\Sigma^*$ $Pr_G(u) = 2^{-n}$,
- if $|u| = n$ and $m \in \mathtt{b}\Sigma^*$ $Pr_G(u) = 0$,
- if $|u| = n+1$ and $m \in \mathtt{a}\Sigma^*$ $Pr_G(u) = 0$,
- if $|u| = n+1$ and $u \in \mathtt{b}\Sigma^*$ $Pr_G(u) = 2^{-n+1}$,
- if $|u| > n+1$ and $u \in \Sigma^*$ $Pr_G(u) = 0$.

We represent in Figure 16.20 the automaton corresponding to the case where $n = 5$.

States 1 et 2 are not μ-distinguishable, or better said are only so for values of μ that increase exponentially with n, yet the different languages are very different. So an exponential number of examples will be needed for

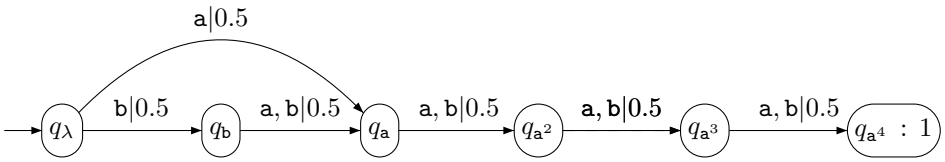


Fig. 16.20. A probabilistic automaton hard to learn, for $n = 5$.

identification (of the structure) to take place.

16.6.2 The equivalence problem revisited

Finally let us attempt to generalise here both approaches in a common way, by using the intersection with regular sets. When studying algorithms ALERGIA and DSAI, the outside structure is similar, and only the compatibility test changes. Each time a state merging algorithm takes a decision concerning compatibility of two states q and q', we are attempting to compare the distributions when using q and q' as initial states. In one case (ALERGIA), we do this by testing prefixes, and in the case of Algorithm DSAI we are testing the significant strings. In other words, we are selecting a language L, and compute the value $\left|Pr_{\mathcal{A},q}(L) - Pr_{\mathcal{A},q'}(L)\right|$. This value is compared with some value depending on the evidence, by typically using the Hoeffding bounds.

In the case of ALERGIA, L is either just one string or a set $u\,\Sigma^\star$. In the case of DSAI, L is just one string.

A generalised test, therefore consists in:

(i) Select some regular language L
(ii) Using the Hoeffding bounds, compare $\left|Pr_{\mathcal{A},q}(L) - Pr_{\mathcal{A},q'}(L)\right|$ with $\left(\sqrt{\frac{1}{\text{FREQ}(q)}} + \sqrt{\frac{1}{\text{FREQ}(q')}}\right) \cdot \sqrt{\frac{1}{2}\ln\frac{2}{\alpha}}$.

16.7 MDI and other heuristics

Algorithms ALERGIA and DSAI decided upon merging (and thus generalisation) through a local test: Substring frequencies are compared and if it is not unreasonable to merge, then merging takes place. A more pragmatic point of view could be to merge whenever doing so is going to give us an advantage: The goal is of course to reduce the size of the hypothesis while keeping the predictive qualities of the hypothesis (at least with respect to the learning sample) as good as possible. For this we can use the likelihood (see Section 5.5.5). The goal is to obtain a good balance between the gain in size and the loss in perplexity.

Attempting to find a good compromise between these two values is the main idea of algorithm MDI (Minimum Divergence Inference).

16.7.1 The scoring

These ideas follow those presented in the MDL inspired algorithms (see Section 14.4), so we need to define a score for an automaton that would somehow reflect the balance we have described. This can be done with respect to the learning sample by computing the perplexity of the sample (see Section 5.5.5,

page 127) and dividing it by the size of the automaton, when counting the number of states.

$$\text{score}(S, \mathcal{A}) = \frac{\sum_{w \in S} \text{cnt}_S(w) \log Pr_{\mathcal{A}} w}{\|\mathcal{A}\|}.$$

We defined and studied these measures in Section 5.5.5. Optimising the denominator will lead to merge, whereas the optimum perplexity is obtained by the Fpta(S).

16.7.2 The algorithm

The structure of the algorithm is similar to that used by the algorithms presented in the previous sections, but we modify the compatibility test. Algorithm Mdi (16.10) uses the function Mdi-Compatible (Algorithm 16.9).

The key difference is that the recursive merges are to be made inside Algorithm Mdi-Compatible (16.9) and before the new score is computed instead of in the main algorithm.

Algorithm 16.9: Mdi-Compatible.

Input: a Ffa $\mathcal{A}$, two states q and q', S, $\alpha > 0$
Output: a Boolean indicating if q and q' compatible?
$\mathcal{B} \leftarrow$Stochastic-Merge$(\mathcal{A}, q, q')$;
return $\big(\text{score}(S, \mathcal{B}) < \alpha\big)$

Several variants of Mdi can be tried: One may prefer trying several possible merges and keeping the best, or using a parameter α that can change in time and depend on the quantity of information available. Of course, the score function can also be modified. Different things that need finer tuning: Instead of taking the sum of the logarithms of the probabilities, one may want to take the perplexity directly.

A difficult question is that of setting the tuning parameter (α): If set too high, merges will take place early, which will include perhaps a wrong merge, prohibiting later necessary merges, and the result can be bad. On the contrary, a small α will block all merges, including those that should take place, at least until there is little data left. This is the *safe* option, which leads in most cases to very little generalisation.

Algorithm 16.10: MDI.

Input: a sample S, $\alpha > 0$
Output: a FFA $\mathcal{A}$
Compute t_0, threshold on the size of the multiset needed for the test to be be statistically significant;
$\mathcal{A} \leftarrow \text{FPTA}(S)$;
$\text{RED} \leftarrow \{q_\lambda\}$;
$\text{BLUE} \leftarrow \{q_a : a \in \text{PREF}(S)\}$;
current_score$\leftarrow$ score$(S, \text{FPTA}(S))$;
while CHOOSE q_b *from* BLUE *such that* $\text{FREQ}(q_b) \geq t_0$ **do**
  if $\exists q_r \in \text{RED} : \text{MDI-COMPATIBLE}(q_r, q_b, \mathcal{A}, S, \alpha)$ **then**
    $\mathcal{A} \leftarrow \text{STOCHASTIC-MERGE}(\mathcal{A}, q_r, q_b)$
  else
    $\text{RED} \leftarrow \text{RED} \cup \{q_b\}$
  end
  $\text{BLUE} \leftarrow \{q_{ua} : ua \in \text{PREF}(S) \wedge q_u \in \text{RED}\} \setminus \text{RED}$
end
return $\mathcal{A}$

16.7.3 Why it works

MDI is known to work in practice, in which case it tends to work better that the algorithms we described earlier. On the other hand, no convergence property has been proved.

One of the key features of MDI is that the computation of the new scores (after a particular merge) can be done incrementally, which allows complexity to be kept low.

16.8 Exercises

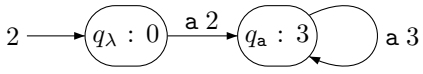


Fig. 16.21. What is the sample?

16.1 Consider FFA represented in Figure 16.21. Build two samples from which this FFA can be obtained.

16.2 Suppose we have a sample containing string $\mathtt{a}^{1000}$ and strings `b`, `bb` and `bbb` each appearing 100 times. Build the FPTA corresponding to

this sample and merge state $q_{\mathtt{a}}$ with state q_λ. What has happened? How can we avoid this problem?

16.3 Compute the number of examples needed in a Hoeffding test to allow for an error ϵ of 0.05 and $\delta = 0.01$. What happens if you wish to decrease to $\epsilon = 0.01$?

16.4 Construct the FFA corresponding to a 3-state automaton ($\{q_1, q_2, q_3\}$) with sample $\{\mathtt{aab}, \mathtt{aba}, \mathtt{bbb}(3)\}$ and for the two following sets of constraints:
$C_1 = \{(q_1, \mathtt{a}, q_2), (q_2, \mathtt{a}, q_3),\ (q_3, \mathtt{a}, q_1), (q_1, \mathtt{b}, q_2), (q_2, \mathtt{b}, q_3), (q_3, \mathtt{a}, q_3)\}$
$C_2 = \{(q_1, \mathtt{a}, q_2), (q_1, \mathtt{a}, q_3),\ (q_2, \mathtt{a}, q_2), (q_3, \mathtt{b}, q_3), (q_2, \mathtt{a}, q_1), (q_3, \mathtt{b}, q_1)\}$

16.9 Conclusion of the chapter and further reading

16.9.1 Bibliographical background

The first work in learning probabilistic grammars is due to Jim Horning [Hor69], who put up the setting and showed that with enumeration algorithms identification with probability one was possible. Between the other early papers, probabilistic automata and their theory are described by Azaria Paz in his book [Paz71]. Dana Angluin made an important (even if unpublished) contribution in her study on identification with probability one [Ang88]. In the mid-nineties a variety of results were shown: Andreas Stolcke [Sto94] proposed a method not only to estimate the probabilities but also to learn the structure of Hidden Markov Models. Naoki Abe and Manfred Warmuth exposed the hardness of learning or approximating distributions [AW92]; the results are combinatorial: Given a set of constraints, estimating probabilities is a hard problem.

The DFFA introduced in this chapter are inspired by the multiplicity automata that have been used in grammatical inference [BV96, BBB+00]. Only the semantics are different.

The ideas from Section 16.4 come from [CO94b]: ALERGIA was invented by Rafael Carrasco and Jose Oncina. They proved the convergence of a simpler version of that algorithm, called RLIPS[CO99]. An extension of ALERGIA by Colin de la Higuera and Franck Thollard not only identifies the structure, but also the actual probabilities [dlHT00].

Extensions of ALERGIA to the tree case was done by the same authors in [COCR01]. Another extension to deal with the richer class of probabilistic deterministic linear languages can be found in [dlHO03]. The same authors propose a study of the learnability of probabilistic languages for a variety of queries in [dlHO04].

Between the applications of algorithms that learn DPFA, one can find text and document analysis [YLT00], and web page classification [GBE96].

The use of distinguishing strings (Section 16.5) was introduced by Dana Ron *et al.* [RST95]. Another specificity of this algorithm (and we have not followed this here) is to learn only acyclic automata. In fact the reason for this lies more in the necessity of a proof. In order to do this two states q_u and q_v will be immediately declared incompatible if $|u| \neq |v|$. There have been several results following this paper. Improvements in [TC04, PG05, Gut06] concerned the different parameters that are needed in order to obtain PAC type results, but also on the bounds and the fact that acyclic conditions can be dropped. An incremental version was proposed in [GKPP06].

Algorithm MDI (Section 16.7) was invented by Franck Thollard *et al.* and used since then on a variety of tasks, with specific interest in language modelling [TD99, TDdlH00, Tho01].

16.9.2 Some alternative lines of research

A different approach is that of considering that the strings do not come from a set but from a sequence (each string can only be generated once). This has been analysed in [dlH98].

There are of course alternative ways to represent distributions over strings: Recurrent neural networks [CFS96] have been tried, but comparisons with the direct grammatical based approaches have not been made on large datasets.

We have concentrated here on deterministic automata, but there have been several authors attempting to learn non-deterministic probabilistic finite automata.

A first step has consisted in studying the class of the probabilistic residual finite state automata, introduced by Yann Esposito *et al.* [ELDD02], and being the probabilistic counterparts to the residual finite state automata introduced by François Denis *et al.* [DLT00, DLT01].

The richness of the class of the probabilistic finite automata has led to the introduction by François Denis *et al.* [HDE06, DEH06] of innovative algorithm DEES that learns a multiplicity automaton (the weights can be negative) by iteratively solving equations on the residuals. Conversion to a PFA is possible.

Christopher Kermorvant [KdlHD04] learns DPFA with additional knowledge. On trees [RJCRC00] Rico *et al.* learn k-testable trees and then computes probabilities.

Omri Guttman's thesis [GVW05, Gut06] and published work adds a geometric point of view to the hardness of learning PFA.

We left untouched in the discussion the hard question of smoothing. The idea is that after using any of the algorithms presented within this chapter, there will usually be strings whose probability, when parsing with the resulting PFA is going to be 0. This is source of all types of problems in practice. Pierre Dupont and Juan-Carlos Amengual [DA00], but also Franck Thollard [Tho01] studied this difficult question for which there still is a lot of room for answers.

16.9.3 Open problems and possible new lines of research

There are a number of directions one can follow and problems to be solved in this field:

(i) Fuzzy automata have sometimes also been called probabilistic automata [Rab66]. In these what is computed is not the weight of a string in the language but the probability that a string belongs to a language. The probability is thus that of being recognised, not of being generated. If there have been some (few) attempts to learn such objects, there have not been (to our knowledge) any systematic results here.

(ii) A central question in this section has been that of taking the decision to merge, or not, two states. This has depended on the idea the learner has that the two states are equivalent, idea based on the information it has at that moment *i.e.* the fact that some earlier merges have been done and the samples. But this in turn leads to a natural question: Given two samples S and S', have they been produced by the same automaton? An answer to this question (obviously taking specific bias) is necessary in order to better understand what is feasible and what is not. Work in the statistical community deserves to be looked into for this reason.

(iii) Learning probabilistic context-free grammars is an open question. There were some early studies [Hor69, Mar74], some heuristics (often relying on constructing a grammar and then estimating the probabilities using the INSIDE-OUTSIDE algorithm (see Section 17.5, page 430) but no 'usable' algorithm exists.

(iv) Markov decision processes (MDPs) are used in reinforcement learning. They allow to use finite state machines to manipulate, probabilities in an active learning setting.

Definitions A **POMPD** is defined by a set of states Q, an input alphabet Σ, an initial state q_λ and two functions:

- A probabilistic transition function $\delta : Q \times \Sigma \times Q \to \mathbb{R}^+$ with $\forall q \in Q$, $\sum_{q \in Q, a \in \Sigma} \delta(q, a, q') = 1$.
- A probabilistic ***reward*** function $r : Q \to \mathbb{R}$.

A POMPD is deterministic if function δ is deterministic, *i.e.* if $\forall q \in Q, \forall a \in \Sigma, \exists q' \in Q$ such that $\delta(q, a, q') = 1$. See Figure 16.22 for an example of a MDP. Notice that the rewards are also used as a goal function. POMPD's are used to model a situation: the 'learning' problem is then to construct the best strategy, given a POMPD. The other question, of finding the POMPD from some data (perhaps in a setting close to the one used in the introduction, page 7) has been left so far untouched.

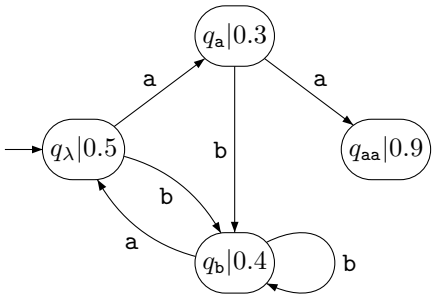


Fig. 16.22. Graphical representation of a deterministic MDP.