12

Informed learners

Understanding is compression, comprehension is compression!

Greg Chaitin *(Chaitin, 2007)*

Comprendo. Habla de un juego donde las reglas no sean la línea de salida, sino el punto de llegada ¿No?

Arturo Pérez-Reverte*, el pintor de batallas*

'Learning from an informant' is the setting in which the data consists of labelled strings, each label indicating whether or not the string belongs to the target language.

Of all the issues which grammatical inference scientists have worked on, this is probably the one on which most energy has been spent over the years. Algorithms have been proposed, competitions have been launched, theoretical results have been given. On one hand, the problem has been proved to be on a par with mighty theoretical computer science questions arising from combinatorics, number theory and cryptography, and on the other hand cunning heuristics and techniques employing ideas from artificial intelligence and language theory have been devised.

There would be a point in presenting this theme with a special focus on the class of context-free grammars with a hope that the theory for the particular class of the finite automata would follow, but the history and the techniques tell us otherwise. The main focus is therefore going to be on the simpler yet sufficiently rich question of learning deterministic finite automata from positive and negative examples.

We shall justify this as follows:

- On one hand the task is hard enough, and, through showing what doesn't work and why, we will have a precious insight into more complex classes.
- On the other hand anything useful learnt on DFAs can be nicely transferred thanks to reductions (see Chapter 7) to other supposedly richer classes.
- Finally, there are historical reasons: on the specific question of learning DFAs from an informed presentation, some of the most important algorithms in grammatical inference have been invented, and many new ideas have been introduced due to this effort.

The specific question of learning context-free grammars and languages from an informant will be studied as a separate problem, in Chapter 15.

12.1 The prefix tree acceptor (PTA)

We shall be dealing here with *learning samples* composed of labelled strings:

Definition 12.1.1 Let Σ be an alphabet. An **informed learning sample** is made of two sets S_+ and S_- such that $S_+ \cap S_- = \emptyset$. The sample will be denoted as $S = \langle S_+, S_- \rangle$.

We will alternatively denote $(x, 1) \in S$ for $x \in S_+$ and $(x, 0) \in S$ for $x \in S_-$. Let $\mathcal{A} = \langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R}, \delta \rangle$ be a DFA.

Definition 12.1.2 $\mathcal{A}$ is **weakly consistent** with the sample $S = \langle S_+, S_- \rangle$ *if*$_{def}$ $\forall x \in S_+,\ \delta(q_\lambda, x) \in \mathbb{F}_\mathbb{A}$ and $\forall x \in S_-,\ \delta(q_\lambda, x) \notin \mathbb{F}_\mathbb{A}$.

Definition 12.1.3 $\mathcal{A}$ is **strongly consistent** with the sample $S = \langle S_+, S_- \rangle$ *if*$_{def}$ $\forall x \in S_+,\ \delta(q_\lambda, x) \in \mathbb{F}_\mathbb{A}$ and $\forall x \in S_-,\ \delta(q_\lambda, x) \in \mathbb{F}_\mathbb{R}$.

Example 12.1.1 The DFA from Figure 12.1 is only weakly consistent with the sample $\{(\mathtt{aa}, 1), (\mathtt{abb}, 0), (\mathtt{b}, 0)\}$ which can also be denoted as:

$$S_+ = \{\mathtt{aa}\}$$
$$S_- = \{\mathtt{abb}, \mathtt{b}\}.$$

String abb ends in state q_λ which is unlabelled (neither accepting nor rejecting), and string b (from S_-) cannot be entirely parsed. Therefore the DFA is not strongly consistent. On the other hand the same automaton can be shown to be strongly consistent with the sample $\{(\mathtt{aa}, 1), (\mathtt{aba}, 1)(\mathtt{abab}, 0)\}$.

A ***prefix tree acceptor*** (PTA) is a tree-like DFA built from the learning sample by taking all the prefixes in the sample as states and constructing the smallest DFA which is a tree $(\forall q \in Q, |\ \{q' : \delta(q', a) = q\}| \leq 1\})$, strongly consistent with the sample. A formal algorithm (BUILD-PTA) is given (Algorithm 12.1).

An example of a PTA is shown in Figure 12.2.

Note that we can also build a PTA from a set of positive strings only. This corresponds to building the PTA $(\langle S_+, \emptyset \rangle)$. In that case, for the same sample we would get the PTA represented in Figure 12.3.

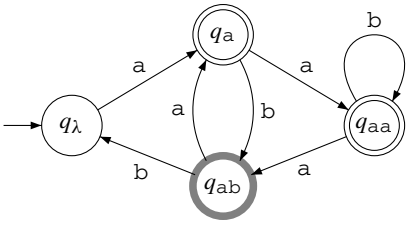


Fig. 12.1. A DFA.

Algorithm 12.1: BUILD-PTA.

Input: a sample $\langle S_+, S_- \rangle$
Output: $\mathcal{A} = \text{PTA}(\langle S_+, S_- \rangle) = \langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R}, \delta \rangle$
$\mathbb{F}_\mathbb{A} \leftarrow \emptyset$;
$\mathbb{F}_\mathbb{R} \leftarrow \emptyset$;
$Q \leftarrow \{q_u : u \in \text{PREF}(S_+ \cup S_-)\}$;
for $q_{u \cdot a} \in Q$ **do** $\delta(q_u, a) \leftarrow q_{ua}$;
for $q_u \in Q$ **do**
  if $u \in S_+$ **then** $\mathbb{F}_\mathbb{A} \leftarrow \mathbb{F}_\mathbb{A} \cup \{q_u\}$;
  if $u \in S_-$ **then** $\mathbb{F}_\mathbb{R} \leftarrow \mathbb{F}_\mathbb{R} \cup \{q_u\}$
end
return $\mathcal{A}$

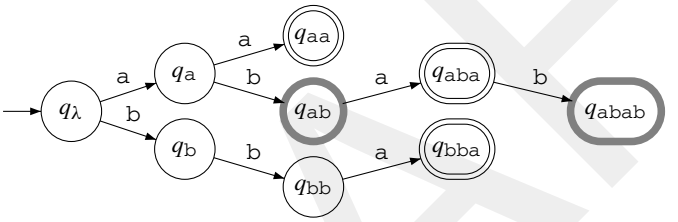


Fig. 12.2. PTA ({(aa, 1), (aba, 1), (bba, 1), (ab, 0), (abab, 0)}).

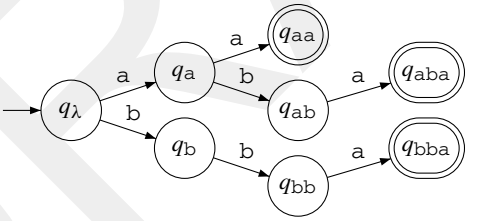


Fig. 12.3. PTA ({(aa, 1), (aba, 1), (bba, 1)}).

In Chapter 6 we will consider the problem of grammatical inference as the one of searching inside a space of admissible biased solutions, in which case we will introduce a non-deterministic version of the PTA.

Most algorithms will take the PTA as a starting point and try to generalise from it by merging states. In order not to get lost in the process (and not undo merges that have been made some time ago) it will be interesting to divide the states into three categories:

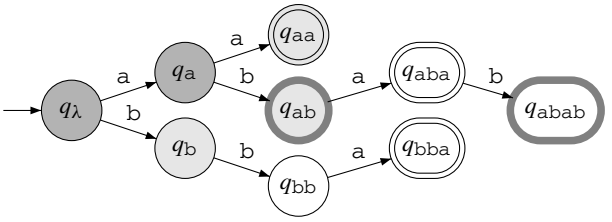


Fig. 12.4. Colouring of states: RED $= \{q_\lambda, q_\mathtt{a}\}$, BLUE $= \{q_\mathtt{b}, q_\mathtt{aa}, q_\mathtt{ab}\}$, and all the other states are WHITE.

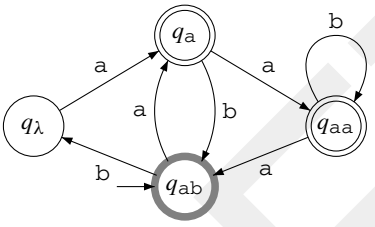


Fig. 12.5. The DFA $\mathcal{A}_{q_\mathtt{ab}}$.

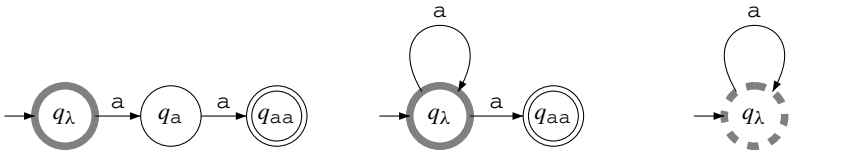
- The RED states which correspond to states that have been analysed and which will not be revisited; they will be the states of the final automaton.
- The BLUE states which are the ***candidate*** states: they have not been analysed yet and it should be from this set that a state is drawn in order to consider merging it with a RED state.
- The WHITE states, which are all the others. They will in turn become BLUE and then RED.

Example 12.1.2 We conventionally draw the RED states in dark grey and the BLUE ones in light grey as in Figure 12.4, where RED $= \{q_\lambda, q_\mathtt{a}\}$ and BLUE $= \{q_\mathtt{b}, q_\mathtt{aa}, q_\mathtt{ab}\}$.

We will need to describe the suffix language in any state q, consisting of the language recognised by the automaton when taking this state q as initial. We denote this automaton formally by $\mathcal{A}_q$ with $\mathbb{L}(\mathcal{A}_q) = \{w \in \Sigma^\star : \delta(q, w) \in \mathbb{F}_\mathbb{A}\}$. In Figure 12.5 we have used the automaton $\mathcal{A}$ from Figure 12.1 and chosen state $q_\mathtt{ab}$ as initial.

12.2 The basic operations

We first describe some operations common to many of the ***state merging techniques***. State merging techniques iteratively consider an automaton and two of its states and aim to ***merge*** them. This will be done when these states are ***compatible***. Sometimes, when noticing that a particular state cannot be merged, it gets ***promoted***. Furthermore at any moment all states are either RED, BLUE or WHITE. Let us also suppose that the current automaton is ***consistent*** with the sample. The starting point is the ***prefix tree acceptor*** (PTA). Initially, in the PTA, the unique RED state is q_λ whereas the BLUE states are the immediate successors of q_λ.



(a) PTA({aa, 1), (λ, 0)}. (b) After merging q_λ and $q_{\mathtt{a}}$. (c) After merging $q_\lambda, q_{\mathtt{a}}$ and $q_{\mathtt{aa}}$.

Fig. 12.6. Incompatibility is not a local affair.

There are three basic operations that shall be systematically used and need to be studied independently of the learning algorithms: COMPATIBLE, MERGE and PROMOTE.

12.2.1 COMPATIBLE: deciding equivalence between states

The question here is of deciding if two states are compatible or not. This is the same as deciding equivalence for the Nerode relation, but with only partial knowledge about the language. As obviously we do not have the entire language to help us decide upon this, but only the learning sample, the question is to know if merging these two states will not result in creating confusion between accepting and rejecting states. Typically the compatibility might be tested by:

$$q \backsimeq_{\mathcal{A}} q' \iff \mathbb{L}_{\mathbb{F}_{\mathbb{A}}}(\mathcal{A}_q) \cap \mathbb{L}_{\mathbb{F}_{\mathbb{R}}}(\mathcal{A}_{q'}) = \emptyset \text{ and } \mathbb{L}_{\mathbb{F}_{\mathbb{R}}}(\mathcal{A}_q) \cap \mathbb{L}_{\mathbb{F}_{\mathbb{A}}}(\mathcal{A}_{q'}) = \emptyset.$$

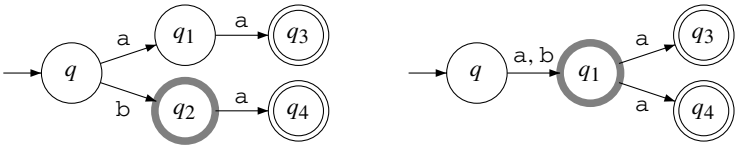
But this is usually not enough as the following example (Figure 12.6) shows. Consider the three-state PTA (Figure 12.6(a)) built from the sample $S = \{(\mathtt{aa}, 1),\ (\lambda, 0)\}$. Deciding equivalence between states q_λ and q_a through the formula above is not sufficient. Indeed languages $\mathbb{L}(\mathcal{A}_{q_\lambda})$ and $\mathbb{L}(\mathcal{A}_{q_{\mathtt{a}}})$ are weakly consistent, but if q_λ and $q_{\mathtt{a}}$ are merged together (Figure 12.6(b)), the state $q_{\mathtt{aa}}$ must also be merged with these (Figure 12.6(c)) to preserve determinism. This results in a problem: is the new unique state accepting or rejecting?

Therefore more complex operations will be needed, involving merging, folding and **then** testing consistency.

12.2.2 MERGE: merging two states

The merging operation takes two states from an automaton and merges them into a single state. It should be noted that the effect of the merge is that a deterministic automaton (see Figure 12.7(a)) will possibly lose the determinism property through this (Figure 12.7(b)). Indeed this is where the algorithms can reject a merge. Consider for instance automaton 12.8(a). If states q_1 and q_2 are merged, then to ensure determinism, states q_3 and q_4 will also have to be merged, resulting in automaton 12.8(b). If we have in our learning sample string aba (in S_-), then the merge should be rejected.

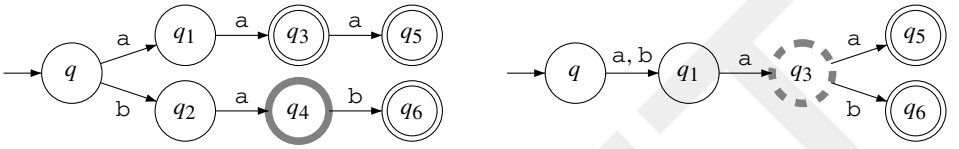
Algorithm 12.2 is given an NFA (with just one initial state, for simplicity), and two states. It updates the automaton.



(a) Before merging states q_1 and q_2.

(b) After merging the states.

Fig. 12.7. Merging two states may result in the automaton not remaining deterministic.



(a) Before merging states 1 and 2.

(b) After merging the states recursively (from 1 and 2).

Fig. 12.8. About merging.

Algorithm 12.2: MERGE.

Input: an NFA : $\mathcal{A} = \langle \Sigma, Q, \mathbb{I}, \mathbb{F}_{\mathbb{A}}, \mathbb{F}_{\mathbb{R}}, \delta_N \rangle$, q_1 and q_2 in Q, with $q_2 \notin \mathbb{I}$
Output: an NFA : $\mathcal{A} = \langle \Sigma, Q, \mathbb{I}, \mathbb{F}_{\mathbb{A}}, \mathbb{F}_{\mathbb{R}}, \delta_N \rangle$ in which q_1 and q_2 have been merged into q_1

for $q \in Q$ **do**
  for $a \in \Sigma$ **do**
    if $q_2 \in \delta_N(q, a)$ **then** $\delta_N(q, a) \leftarrow \delta_N(q, a) \cup \{q_1\}$;
    if $q \in \delta_N(q_2, a)$ **then** $\delta_N(q_1, a) \leftarrow \delta_N(q_1, a) \cup \{q\}$
  end
end
if $q_2 \in \mathbb{I}$ **then** $\mathbb{I} \leftarrow \mathbb{I} \cup \{q_1\}$;
if $q_2 \in \mathbb{F}_{\mathbb{A}}$ **then** $\mathbb{F}_{\mathbb{A}} \leftarrow \mathbb{F}_{\mathbb{A}} \cup \{q_1\}$;
if $q_2 \in \mathbb{F}_{\mathbb{R}}$ **then** $\mathbb{F}_{\mathbb{R}} \leftarrow \mathbb{F}_{\mathbb{R}} \cup \{q_1\}$;
$Q \leftarrow Q \setminus \{q_2\}$;
return $\mathcal{A}$

Since non-deterministic automata are in many ways cumbersome, we will attempt to avoid having to use these to define merging when manipulating only deterministic automata.

12.2.3 PROMOTE: promoting a state

Promotion is another deterministic and greedy decision. The idea here is that having decided, at some point, that a BLUE candidate state is different from all the RED states, it

Algorithm 12.3: PROMOTE.

Input: a DFA : $\mathcal{A} = \langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R}, \delta \rangle$, a BLUE state q_u, sets RED, BLUE
Output: a DFA : $\mathcal{A} = \langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R}, \delta \rangle$, sets RED, BLUE updated
RED $\leftarrow$ RED $\cup \{q_u\}$;
for $a \in \Sigma$: q_{ua} *is not* RED **do** add q_{ua} to BLUE;
return $\mathcal{A}$

should become RED. We call this a ***promotion*** and describe the process in Algorithm 12.3. The notations that are used here apply to the case where the states involved in a promotion are the basis of a tree. Therefore, the successors of node q_u are named q_{ua} with a in Σ.

12.3 Gold's algorithm

The first non-enumerative algorithm designed to build a DFA from informed data is due to E. Mark Gold, which is why we shall simply call this algorithm GOLD. The goal of the algorithm is to find the minimum DFA consistent with the sample. For that, there are two steps. The first is deductive: from the data, find a set of prefixes that have to lead to different states for the reasons given in Section 12.2.2 above, and therefore represent an incompressible set of states. The second step is inductive: alas, after finding the incompressible set of states, we are not done because it is not usually easy or even possible to 'fold in' the rest of the states. Since a direct construction of the DFA from there is usually impossible, (contradictory) decisions have to be taken. This is where artificial intelligence techniques might come in (see Chapter 14) as the problems one has to solve are proved to be intractable (in Chapter 6).

But as more and more strings become available to the learning algorithm (i.e. in the identification in the limit paradigm), the number of choices left will become more and more restricted, with, finally, just one choice. This is what allows convergence.

12.3.1 The key ideas

The main ideas of the algorithm are to represent the data (positive and negative strings) in a table, where each row corresponds to a string, some of which will correspond to the RED states and the others to the BLUE states. The goal is to create through promotion as many RED states as possible. For this to be of real use, the set of strings denoting the states will be *closed* by prefixes, i.e. if q_{uv} is a state so is q_u. Formally:

Definition 12.3.1 (Prefix- and suffix-closed sets) A set of strings S is **prefix-closed** (respectively **suffix-closed**) *if*$_{def}$ $uv \in S \implies u \in S$ (respectively if $uv \in S \implies v \in S$).

No inference is made during this phase of representation of the data: the algorithm is purely deductive.

The table then expresses an inequivalence relation between the strings and we should aim to complete this inequivalence related to the Nerode relation that defines the language:

$$x \equiv y \iff \left[\forall w \in \Sigma^\star\, xw \in L \iff yw \in L\right].$$

Once the RED states are decided, the algorithm 'chooses' to merge the BLUE states that are left with the RED ones and then checks if the result is consistent. If it is not, the algorithm returns the PTA.

In the following we will voluntarily accept a confusion between the states themselves and the names or labels of the states.

The information is organised in a table $\langle \text{STA}, \text{EXP}, \text{OT}\rangle$, called an ***observation table***, used to compare the candidate states by examining the data, where the three components are:

- $\text{STA} \subset \Sigma^\star$ is a finite set of (labels of) states. The states will be denoted by the indexes (strings) from a finite prefix-closed set. Because of this labelling, we will often conveniently use string terminology when referring to the states. Set STA will therefore both refer to the set of states and to the set of labels of these states, with context always allowing us to determine which.

  We partition STA as follows: $\text{STA} = \text{RED} \cup \text{BLUE}$. The BLUE states (or state labels) are those u in STA such that $uv \in \text{STA} \implies v = \lambda$. The RED states are the others. $\text{BLUE} = \{ua \notin \text{RED} : u \in \text{RED}\}$ is the set of states successors of RED that are not RED.
- $\text{EXP} \subset \Sigma^\star$ is the experiment set. This set is closed by suffixes, i.e. if uv is an experiment, so is v.
- $\text{OT} : \text{STA} \times \text{EXP} \to \{0, 1, *\}$ is a function that indicates if making an experiment in state q_u is going to result into an accepting, a rejecting or an unknown situation: the value $\text{OT}[u][e]$ is then respectively 1,0 or $*$. In order to improve readability we will write it as a table indexed by two strings, the first indicating the label of the state from which the experiment is made and the second being the experiment itself:

$$\text{OT}[u][e] = \begin{cases} 1 & \text{if } ue \in L \\ 0 & \text{if } ue \notin L \\ * & \text{otherwise (not known).} \end{cases}$$

  Obviously, the table should be ***redundant*** in the following sense. Given three strings u, v and w, if $\text{OT}[u][vw]$ and $\text{OT}[uv][w]$ are defined (i.e. $u, uv \in \text{STA}$ and $w, vw \in \text{EXP}$), then $\text{OT}[u][vw] = \text{OT}[uv][w]$.

Example 12.3.1 In observation table 12.1 we can read:

- $\text{STA} = \{\lambda, \mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{ab}\}$ and among these $\text{RED} = \{\lambda, \mathtt{a}\}$.
- $\text{OT}[\mathtt{aa}][\lambda] = 0$ so $\mathtt{aa} \notin L$.
- On the other hand we have $\text{OT}[\mathtt{b}][\lambda] = 1$ which means that $\mathtt{b} \in L$.
- Note also that the table is redundant: for example, $\text{OT}[\mathtt{aa}][\lambda] = \text{OT}[\mathtt{a}][\mathtt{a}] = 0$, and similarly $\text{OT}[\lambda][\mathtt{a}] = [\mathtt{a}][\lambda] = *$. This is only due to the fact that the table is an observation of the data. It does not compute or invent new information.

In the following, we will be only considering legal tables, i.e. those that are based on a set STA prefix-closed, a set EXP suffix-closed and a redundant table. Legality can be

Table 12.1. *An observation table.*

	λ	a	b
λ	0	∗	1
a	∗	0	0
b	1	1	∗
aa	0	0	∗
ab	0	∗	∗

Algorithm 12.4: GOLD-check legality.

Input: a table ⟨STA, EXP, OT⟩
Output: a Boolean indicating if the table is legal or not
OK ← **true**;
for $s \in$ STA **do** `/* check if STA is prefix-closed */`
  if PREF(s) $\not\subset$ STA **then** OK ← **false**
end
for $e \in$ EXP **do** `/* check if EXP is suffix-closed */`
  if PREF(e) $\not\subset$ EXP **then** OK ← **false**
end
for $p \in$ STA **do** `/* check if all is legal */`
  for $e \in$ EXP **do**
    for $p \in$ PREF(e) **do**
      if $[sp \in \text{STA} \wedge \text{OT}[s][e] \neq \text{OT}[sp][p^{-1}e]$ **then** OK ← **false**
    end
  end
end
return OK

checked with Algorithm 12.4. The complexity of the algorithm can be easily improved (see Exercise 12.1). In practice, a good policy is to stock the data in an association table, with the string as key and the label as value, and to manipulate in the observation table just the keys to the table. This makes the legality issue easy to deal with.

Definition 12.3.2 (Holes)
A **hole** in a table ⟨STA, EXP, OT⟩ is a pair (u, e) such that $\text{OT}[u][e] = *$.

A hole corresponds to a missing observation.

Definition 12.3.3 (Complete table)
The table ⟨STA, EXP, OT⟩ is **complete** (or has **no holes**) if_{def} $\forall u \in$ STA, $\forall e \in$ EXP, $\text{OT}[u][e] \in \{0, 1\}$.

	λ	a
λ	0	1
a	1	0
b	1	0
aa	0	0
ab	1	0

(a) An observation table, which is not closed, because of row aa.

	λ	a
λ	0	1
a	1	0
b	0	1
aa	0	1
ab	1	0

(b) A complete and closed observation table.

Fig. 12.9.

Definition 12.3.4 (Rows)
We will refer to OT[u] as the **row** indexed by u and will say that two rows OT[u] and OT[v] are **compatible** for OT (or u and v are consistent for OT) $if_{def}\ \nexists e \in \text{EXP} : \big(\text{OT}[u][e] = 0$ and $\text{OT}[v][e] = 1\big)$ or $\big(\text{OT}[u][e] = 1$ and $\text{OT}[v][e] = 0\big)$. We denote this by $u \backsimeq_{\text{OT}} v$.

The goal is not going to be to detect if two rows are compatible, but if they are not.

Definition 12.3.5 (Obviously different rows)
Rows u and v are **obviously different** (OD) for OT (we also write OT[u] and OT[v] are obviously different) $if_{def}\ \exists e \in \text{EXP} : \text{OT}[u][e]$, $\text{OT}[v][e] \in \{0, 1\}$ and $\text{OT}[u][e] \neq \text{OT}[v][e]$.

Example 12.3.2 Table 12.1 is incomplete since it has holes. Rows OT[λ] and OT[a] are incompatible (and OD), but row OT[aa] is compatible with both OT[λ] and OT[a].

12.3.2 Complete tables

We now consider the ideal (albeit unrealistic) setting where there are no holes in the table.

Definition 12.3.6 (Closed table)
A table OT is **closed** $if_{def}\ \forall u \in \text{BLUE},\ \exists s \in \text{RED} : \text{OT}[u] = \text{OT}[s]$.

The table presented in Figure 12.9(a) is not closed (because of row aa) but Table 12.9(b) is. Being closed means that every BLUE state can be matched with a RED one.

12.3.3 Building a DFA from a complete and closed table

Building an automaton from a table ⟨STA, EXP, OT⟩ can be done very easily as soon as certain conditions are met:

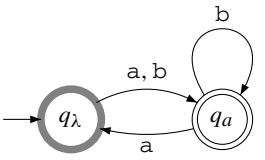
- The set of strings marking the states in STA is prefix-closed,
- The set EXP is suffix-closed,
- The table should be complete: holes correspond to undetermined pieces of information,
- The table should be closed.

	λ	a
λ	0	1
a	1	0
b	1	0
aa	0	1
ab	1	0

(a) The observation table.

	a	b
q_λ	$q_\mathtt{a}$	$q_\mathtt{a}$
$q_\mathtt{a}$	q_λ	$q_\mathtt{a}$

(b) The transition table.



(c) The automaton.

Fig. 12.10. A table and the corresponding automaton.

Once these conditions hold we can use Algorithm GOLD-BUILDAUTOMATON (12.5) and convert the table into a DFA.

Algorithm 12.5: GOLD-BUILDAUTOMATON.

Input: a closed and complete observation table (STA, EXP, OT)
Output: a DFA $\mathcal{A} = \langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R}, \delta \rangle$
$Q \leftarrow \{q_r : r \in \text{RED}\}$;
$\mathbb{F}_\mathbb{A} \leftarrow \{q_{we} : we \in \text{RED} \wedge \text{OT}[w][e] = 1\}$;
$\mathbb{F}_\mathbb{R} \leftarrow \{q_{we} : we \in \text{RED} \wedge \text{OT}[w][e] = 0\}$;
for $q_w \in Q$ **do**
  for $a \in \Sigma$ **do** $\delta(q_w, a) \leftarrow q_u : u \in \text{RED} \wedge \text{OT}[u] = \text{OT}[wa]$
end
return $\langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R}, \delta \rangle$

Example 12.3.3 Consider Table 12.10(a). We can apply the construction from Algorithm 12.5 and obtain $Q = \{q_\lambda, q_\mathtt{a}\}$, $\mathbb{F}_\mathbb{A} = \{q_\mathtt{a}\}$, $\mathbb{F}_\mathbb{R} = \{q_\lambda\}$ and δ is given by the transition table 12.10(b). Automaton 12.10(c) can be built.

Definition 12.3.7 (Consistent table) Given an automaton $\mathcal{A}$ and an observation table $\langle$STA, EXP, OT$\rangle$, $\mathcal{A}$ is **consistent** with $\langle$STA, EXP, OT$\rangle$ when the following holds:

- $\text{OT}[u][e] = 1 \implies ue \in \mathbb{L}_{\mathbb{F}_\mathbb{A}}(\mathcal{A})$,
- $\text{OT}[u][e] = 0 \implies ue \in \mathbb{L}_{\mathbb{F}_\mathbb{R}}(\mathcal{A})$.

$\mathbb{L}_{\mathbb{F}_\mathbb{A}}(\mathcal{A})$ is the language recognised by $\mathcal{A}$ by accepting states, whereas $\mathbb{L}_{\mathbb{F}_\mathbb{R}}(\mathcal{A})$ is the language recognised by $\mathcal{A}$ by rejecting states.

Theorem 12.3.1 (Consistency theorem) *Let* $\langle$STA, EXP, OT$\rangle$ *be an observation table closed and complete. If* STA *is prefix-closed and* EXP *is suffix-closed then* GOLD-BUILDAUTOMATON*(*$\langle$STA, EXP, OT$\rangle$*) is consistent with the information in* $\langle$STA, EXP, OT$\rangle$.

Proof The proof is straightforward as GOLD-BUILDAUTOMATON builds a DFA directly from the data from ⟨STA, EXP, OT⟩. □

12.3.4 Building a table from the data

The second question is that of obtaining a table from a sample. At this point we want the table to be consistent with the sample, and to be just an alternative representation of the sample. Given a sample S and a set of states RED prefix-closed, it is always possible to build a set of experiments EXP such that the table ⟨STA, EXP, OT⟩ contains all the information in S (and no other information!). There can be many possible tables, one corresponding to each set of RED states we wish to consider. And, of course, in most cases, these tables are going to have holes.

Algorithm 12.6: GOLD-BUILDTABLE.

Input: a sample $S = \langle S_+, S_- \rangle$, a set RED prefix-closed
Output: table ⟨STA, EXP, OT⟩
EXP $\leftarrow$ SUFF(S);
BLUE $\leftarrow$ RED $\cdot\ \Sigma \setminus$ RED;
for $p \in$ RED $\cup$ BLUE **do**
  for $e \in$ EXP **do**
    if $p.e \in S_+$ **then** OT$[p][e] \leftarrow 1$
    else
      if $p.e \in S_-$ **then** OT$[p][e] \leftarrow 0$ **else** OT$[p][e] \leftarrow *$
    end
  end
end
return ⟨STA, EXP, OT⟩

Algorithm GOLD-BUILDTABLE (12.6) builds the table corresponding to a given sample and a specific set RED.

Example 12.3.4 Table 12.2 constructed for the sample $S = \{(\texttt{aa}, 1), (\texttt{bbaa}, 1), (\texttt{aba}, 0)\}$ and for the set of RED states $\{\lambda, \texttt{a}\}$ is given here. We have not entered the '$*$' symbols to increase readability (i.e. an empty cell denotes a symbol $*$).

12.3.5 Updating the table

We notice the following:

Proposition 12.3.2 *If* $\exists t \in$ BLUE *such that* OT$[t]$ *is obviously different from any* OT$[s]$ *(with* $s \in$ RED*), then whatever way we fill the holes in* ⟨STA, EXP, OT⟩*, the table will not be closed.*

Table 12.2. GOLD-BUILDTABLE $\big(\{(\mathtt{aa}, 1), (\mathtt{bbaa}, 1), (\mathtt{aba}, 0)\}), \{\lambda, \mathtt{a}\}\big)$.

	λ	a	aa	ba	aba	baa	bbaa
λ			1		0		1
a		1		0			
b						1	
aa	1						
ab		0					

In other words, if one BLUE state is obviously different from all the RED states, then even guessing each $*$ correctly is not going to be enough. This means that this BLUE state should be promoted before attempting to fill in the holes.

12.3.6 Algorithm GOLD

The general algorithm can now be described.

It is composed of four steps requiring four sub-algorithms:

(i) Given sample S, Algorithm GOLD-BUILDTABLE (12.6, page 248) builds an initial table with RED $= \{\lambda\}$, BLUE $= \Sigma$ and $E =$ SUFF(S).

(ii) Find a BLUE state obviously different (OD) with all RED states. Promote this BLUE state to RED and repeat.

(iii) Fill in the holes that are left (using Algorithm GOLD-FILLHOLES). If the filling of the holes fails, return the PTA (using Algorithm BUILD-PTA (12.1, page 239)).

(iv) Using Algorithm GOLD-BUILDAUTOMATON (12.5, page 247), build the automaton. If it is inconsistent with the original sample, return the PTA instead.

12.3.7 A run of the algorithm

Example 12.3.5 We provide an example run of Algorithm GOLD (12.7).

We use the following sample:

$$S_+ = \{\mathtt{bb}, \mathtt{abb}, \mathtt{bba}, \mathtt{bbb}\}$$
$$S_- = \{\mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{bab}\}.$$

We first build the observation table corresponding to RED $= \{\lambda\}$.

Now, Table 12.3 is not closed because of row OT[b]. So we promote $q_{\mathtt{b}}$ and update the table, obtaining Table 12.4.

Table 12.3. *The table for* $S_+ = \{\mathtt{bb}, \mathtt{abb}, \mathtt{bba}, \mathtt{bbb}\}$ $S_- = \{\mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{bab}\}$ *and* RED $= \{\lambda\}$.

	λ	a	b	aa	ab	ba	bb	abb	bab	bba	bbb
λ		0	0	0			1	1	0	1	1
a	0	0					1				
b	0		1		0	1	1				

Algorithm 12.7: GOLD for DFA identification.

Input: a sample S
Output: a DFA consistent with the sample
RED $\leftarrow \{\lambda\}$;
BLUE $\leftarrow \Sigma$;
$\langle$STA, EXP, OT$\rangle \leftarrow$ GOLD-BUILDTABLE(S,RED);
while $\exists x \in$ BLUE *such that* OT[x] *is* OD **do**
  RED $\leftarrow$ RED $\cup \{x\}$;
  BLUE $\leftarrow$ BLUE $\cup \{xa : a \in \Sigma\}$;
  for $u \in$ STA **do**
    for $e \in$ EXP **do**
      if $ue \in S_+$ **then** OT[u][e] $\leftarrow 1$
      else if $ue \in S_-$ **then** OT[u][e] $\leftarrow 0$
      else OT[u][e] $\leftarrow *$
    end
  end
end
OT $\leftarrow$ GOLD-FILLHOLES(OT);
if fail then return BUILD-PTA(S)
else
  $\mathcal{A} \leftarrow$ GOLD-BUILDAUTOMATON($\langle$STA, EXP, OT$\rangle$);
  if CONSISTENT($\mathcal{A}$,S) **then return** $\mathcal{A}$
  else return BUILD-PTA(S)
end

Table 12.4. *The table for* $S_+ = \{$bb, abb, bba, bbb$\}$ $S_- = \{$a, b, aa, bab$\}$ *and* RED $= \{\lambda, \mathtt{b}\}$.

	λ	a	b	aa	ab	ba	bb	abb	bab	bba	bbb
λ		0	0	0			1	1	0	1	1
b	0		1		0	1	1				
a	0	0					1				
ba			0								
bb	1	1	1								

But Table 12.4 is not closed because of OT[bb]. Since $q_{\mathtt{bb}}$ is obviously different from both q_λ (because of experiment λ) and $q_{\mathtt{b}}$ (because of experiment b), we promote $q_{\mathtt{bb}}$ and update the table to Table 12.5.

At this point there are no BLUE rows that are obviously different from the RED rows. Therefore all that is needed is to fill the holes. Algorithm GOLD-FILLHOLES is now used to make the table complete.

Table 12.5. *The table for* $S_+ = \{\mathtt{bb}, \mathtt{abb}, \mathtt{bba}, \mathtt{bbb}\}$ $S_- = \{\mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{bab}\}$ *and* RED $= \{\lambda, \mathtt{b}, \mathtt{bb}\}$.

	λ	a	b	aa	ab	ba	bb	abb	bab	bba	bbb
λ		0	0	0			1	1	0	1	1
b	0		1		0	1	1				
bb	1	1	1								
a	0	0					1				
ba			0								
bba	1										
bbb	1										

Table 12.6. *The table for* $S_+ = \{\mathtt{bb}, \mathtt{abb}, \mathtt{bba}, \mathtt{bbb}\}$ $S_- = \{\mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{bab}\}$ *after running the first phase of* GOLD-FILLHOLES.

	λ	a	b	aa	ab	ba	bb	abb	bab	bba	bbb
λ	1	0	0	0			1	1	0	1	1
b	0	0	1		0	1	1				
bb	1	1	1								
a	0	0					1				
ba			0								
bba	1										
bbb	1										

Table 12.7. *The table for* $S_+ = \{\mathtt{bb}, \mathtt{abb}, \mathtt{bba}, \mathtt{bbb}\}$ $S_- = \{\mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{bab}\}$ *after phase 2 of* GOLD-FILLHOLES.

	λ	a	b	aa	ab	ba	bb	abb	bab	bba	bbb
λ	1	0	0	0	1	1	1	1	0	1	1
b	0	0	1	1	0	1	1	1	1	1	1
bb	1	1	1	1	1	1	1	1	1	1	1
a	0	0					1				
ba			0								
bba	1										
bbb	1										

This algorithm first fills the rows corresponding to the RED rows by using the information contained in the BLUE rows which are compatible (in the sense of Definition 12.3.4). In this case there are a number of possibilities which may conflict. For example, we have $\mathtt{a} \preceq_{\mathrm{OT}} \lambda$ but also $\mathtt{a} \preceq_{\mathrm{OT}} \mathtt{b}$. And we are only considering pairs where the first prefix/state is BLUE and the second is RED.

We suppose that in this particular case, the algorithm has selected $\mathtt{a} \preceq_{\mathrm{OT}} \mathtt{b}$, $\mathtt{ba} \preceq_{\mathrm{OT}} \lambda$, $\mathtt{bba} \preceq_{\mathrm{OT}} \lambda$ and $\mathtt{bbb} \preceq_{\mathrm{OT}} \mathtt{bb}$. This results in building first Table 12.6. Then all the holes in the RED rows are filled by 1s (Table 12.7).

Table 12.8. *The complete table for* $S_+ = \{\texttt{bb}, \texttt{abb}, \texttt{bba}, \texttt{bbb}\}$ $S_- = \{\texttt{a}, \texttt{b}, \texttt{aa}, \texttt{bab}\}$ *after running* GOLD-FILLHOLES.

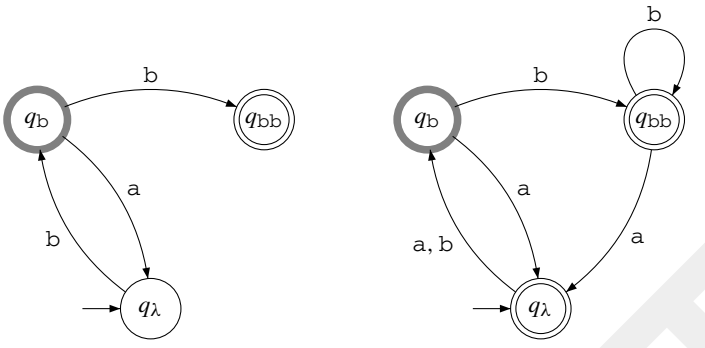
	λ	a	b	aa	ab	ba	bb	abb	bab	bba	bbb
λ	1	0	0	0	1	1	1	1	0	1	1
b	0	0	1	1	0	1	1	1	1	1	1
bb	1	1	1	1	1	1	1	1	1	1	1
a	0	0	1	1	0	1	1	1	1	1	1
ba	1	0	0	0	1	1	1	1	0	1	1
bba	1	0	0	0	1	1	1	1	0	1	1
bbb	1	1	1	1	1	1	1	1	1	1	1

Algorithm 12.8: GOLD-FILLHOLES.

Input: a table ⟨STA, EXP, OT⟩
Output: the table OT updated, with holes filled
for $p \in$ BLUE **do** /* First fill in all the RED lines */
  if $\exists r \in$ RED : $p \backsimeq_{\text{OT}} r$ **then** /* Find **a** compatible RED */
    for $e \in$ EXP **do**
      if $\text{OT}[p][e] \neq *$ **then** $\text{OT}[r][e] \leftarrow \text{OT}[p][e]$
    end
  else
    return fail
  end
end
for $r \in$ RED **do**
  for $e \in$ EXP **do** **if** $\text{OT}[r][e] = *$ **then** $\text{OT}[r][e] \leftarrow 1$
end
for $p \in$ BLUE **do** /* Now fill in all the BLUE lines */
  if $\exists r \in$ RED : $p \backsimeq_{\text{OT}} r$ **then** /* Find **a** compatible RED */
    for $e \in$ EXP **do**
      if $\text{OT}[p][e] = *$ **then** $\text{OT}[p][e] \leftarrow \text{OT}[r][e]$
    end
  else
    return fail
  end
end
return ⟨STA, EXP, OT⟩

The second part of the algorithm again visits the BLUE states, tries to find a compatible RED state and copies the corresponding RED row. This results in Table 12.8.

Finally, the third part of Algorithm 12.8 fills the remaining holes of the BLUE rows. This results in Table 12.8.



(a) The 'safe' information. (b) Guessing the missing information.

Fig. 12.11. The final filling of the holes for GOLD.

Next, Algorithm GOLD-BUILDAUTOMATON (12.5) is run on the table, with the resulting automaton depicted in Figure 12.11(b). The DFA accepts aa which is in S_+, therefore the PTA is returned instead.

One might consider that GOLD-FILLHOLES is far too trivial for such a complex task as that of learning automata. Indeed, when considering Table 12.5 a certain number of safe decisions about the automaton can be made. These are depicted in Figure 12.11(a). The others have to be guessed:

- equivalent lines for a could be λ and b (so $q_{\mathtt{a}}$ could be either q_λ or $q_{\mathtt{b}}$),
- possible candidates for bba are λ and bb (so $q_{\mathtt{bba}}$ could be either q_λ or $q_{\mathtt{bb}}$),
- possible candidates for bbb are λ and bb (so $q_{\mathtt{bbb}}$ could be either q_λ or $q_{\mathtt{bb}}$).

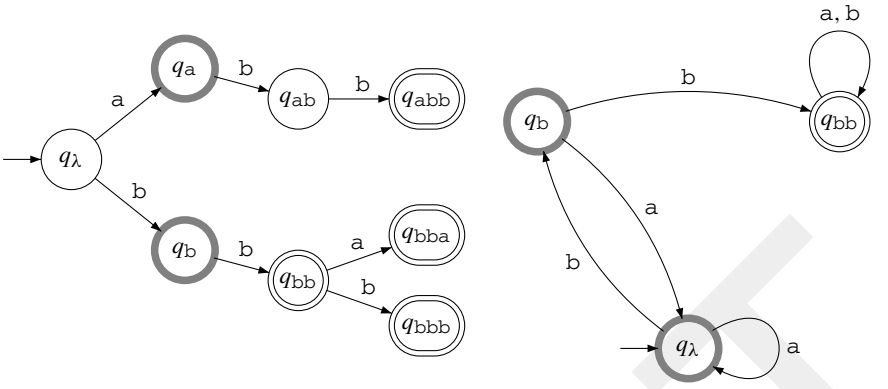
In the choices above not only should the possibilities before merging be considered, but also the interactions between the merges. For example, even if in theory both states $q_{\mathtt{a}}$ and $q_{\mathtt{bba}}$ could be merged into state q_λ, they both cannot be merged together! We will not enter here into how this guessing can be done (greediness is one option).

Therefore the algorithm, having failed, returns the PTA depicted in Figure 12.12(a). If it had guessed the holes correctly an automaton consistent with the data (see Figure 12.12(b)) might have been returned.

12.3.8 Proof of the algorithm

We first want to prove that, alas, filling in the holes is where the real problems start. There is no tractable strategy that will allow us to fill the holes easily:

Theorem 12.3.3 (Equivalence of problems) *Let* RED *be a set of states prefix-closed, and S be a sample. Let* ⟨STA, EXP, OT⟩ *be an observation table consistent with all the data in S, with* EXP *suffix-closed.*



(a) The PTA. (b) The correct solution.

Fig. 12.12. The result and the correct solution.

The question:

> ***Name:*** *Consistent*
> ***Instance:*** *A sample S, a prefix-closed set* RED
> ***Question:*** *Does there exist a* DFA $= \langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R}, \delta \rangle$ *with* $Q = \{q_u : u \in \text{RED}\}$*, and if* $ua \in$ RED, $\delta(q_u, a) = q_{ua}$*, consistent with S?*

is equivalent to:

> ***Name:*** *Holes*
> ***Instance:*** *An observation table* ⟨STA, EXP, OT⟩
> ***Question:*** *Can we fill the holes in such a way as to have* ⟨STA, EXP, OT⟩ *closed?*

Proof If we can fill the holes and obtain a closed table, then a DFA can be constructed which is consistent with the data (by Theorem 12.3.1). If there is a DFA with the states of RED then we can use the DFA to fill the holes. □

From this we have:

Theorem 12.3.4 *The problem:*

> ***Name:*** *Minimum consistent* DFA *reachable from* RED
> ***Instance:*** *A sample S, a set* RED *prefix-closed such that each* OT[s] *(*$s \in$ RED*) is obviously different from all the others with respect to S, and a positive integer n*
> ***Question:*** *Is there a* DFA $= \langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R}, \delta \rangle$ *with the conditions* $\{q_u : u \in \text{RED}\} \subseteq Q$, $|Q| = n$*, consistent with S?*

is $\mathcal{NP}$*-complete.*

And as a consequence:

Corollary 12.3.5 *Given S and a positive integer n, the question:*

> ***Name:*** *Minimum consistent* DFA
> ***Instance:*** *A sample S and a positive integer n*
> ***Question:*** *Is there a* DFA *with n states consistent with S?*

is $\mathcal{NP}$-complete.

Proof We leave the proofs that these problems are $\mathcal{NP}$-hard to Section 6.2 (page 119).

Proving that either of the problems belongs to $\mathcal{NP}$ is not difficult: simply producing a DFA and checking consistency is going to take polynomial time only as it consists of parsing the strings in S. □

On the positive side, identification in the limit can be proved:

Theorem 12.3.6 *Algorithm* GOLD*, given any sample $S = \langle S_+, S_- \rangle$:*

- *outputs a* DFA *consistent with S,*
- *admits a polynomial-sized characteristic sample,*
- *runs in time and space polynomial in $\|S\|$.*

Proof

- Remember that in the worst case the PTA is returned.
- The characteristic sample can be constructed in such a way as to make sure that all the states are found to be OD. The number of such strings is quadratic in the size of the target automaton. Furthermore it can be proved that none of these strings needs to be of length more than n^2. It should be noted that for this to work, the order in which the BLUE states are explored for promotion matters. If the size of the canonical acceptor of the language is n, then there is a characteristic sample CS_L with $\|\mathrm{CS}_L\| = 2n^2(|\Sigma| + 1)$, such that GOLD($S$) produces the canonical acceptor for all $S \supseteq \mathrm{CS}_L$.
- Space complexity is in $\mathcal{O}(\|S\| \cdot n)$ whereas time complexity is in $\mathcal{O}(n^2 \cdot \|S\|)$. We leave for Exercise 12.5 the question of obtaining a faster algorithm. □

Corollary 12.3.7 *Algorithm* GOLD *identifies $\mathcal{DFA}(\Sigma)$ in* POLY-CS *time.*

Identification in the limit in POLY-CS polynomial time follows from the previous remarks.

12.4 RPNI

In Algorithm GOLD, described in Section 12.3, there is more than a real chance that after many iterations the final problem of 'filling the holes' is not solved at all (and perhaps cannot be solved unless more states are added) and the PTA is returned. Even if this is mathematically admissible (since identification in the limit is ensured), in practice one

would prefer an algorithm that does some sort of generalisation in all circumstances, and not just in the favourable ones.

This is what is proposed by algorithm RPNI (Regular Positive and Negative Inference). The idea is to greedily create clusters of states (by merging) in order to come up with a solution that is always consistent with the learning data. This approach ensures that some type of generalisation takes place and, in the best of cases (which we can characterise by giving sufficient conditions that permit identification in the limit), returns the correct target automaton.

12.4.1 The algorithm

We describe here a generic version of Algorithm RPNI. A number of variants have been published that are not exactly equivalent. These can be studied in due course. Basically, Algorithm RPNI (12.13) starts by building PTA(S_+) from the positive data (Algorithm BUILD-PTA (12.1, page 239)), then iteratively chooses possible merges, checks if a given merge is correct and is made between two compatible states (Algorithm RPNI-COMPATIBLE (12.10)), makes the merge (Algorithm RPNI-MERGE (12.11)) if admissible and promotes the state if no merge is possible (Algorithm RPNI-PROMOTE (12.9)).

The algorithm has as a starting point the PTA, which is a deterministic finite automaton. In order to avoid problems with non-determinism, the merge of two states is immediately followed by a folding operation: the merge in RPNI always occurs between a RED state and a BLUE state. The BLUE states have the following properties:

- If q is a BLUE state, it has exactly one predecessor, i.e. whenever $\delta(q_1, a_1) = \delta(q_2, a_2) = q$, then necessarily $q_1 = q_2$ and $a_1 = a_2$.
- q is the root of a tree, i.e. if $\delta(q, u \cdot a)$=$\delta(q, v \cdot b)$ then necessarily $u = v$ and $a = b$.

Algorithm 12.9: RPNI-PROMOTE.

Input: a DFA $\mathcal{A} = \langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R}, \delta \rangle$, sets RED, BLUE $\subseteq Q$, $q_u \in$ BLUE
Output: $\mathcal{A}$, RED, BLUE updated
RED $\leftarrow$ RED $\cup \{q_u\}$;
BLUE $\leftarrow$ BLUE $\cup \{\delta(q_u, a), a \in \Sigma\}$;
return $\mathcal{A}$, RED, BLUE

Algorithm RPNI-PROMOTE (12.9), given a BLUE state q_u, promotes this state to RED and all the successors in $\mathcal{A}$ of this state become BLUE.

Algorithm RPNI-COMPATIBLE (12.10) returns YES if the current automaton cannot parse any string from S_- but returns NO if some counter-example is accepted by the current automaton. Note that the automaton $\mathcal{A}$ is deterministic.

Algorithm 12.10: RPNI-COMPATIBLE.

Input: $\mathcal{A}$, S_-
Output: a Boolean, indicating if $\mathcal{A}$ is consistent with S_-
for $w \in S_-$ **do**
  if $\delta_{\mathcal{A}}(q_\lambda, w) \cap \mathbb{F}_{\mathbb{A}} \neq \emptyset$ **then return false**
end
return true

Algorithm RPNI-MERGE (12.11) takes as arguments a RED state q and a BLUE state q'. It first finds the unique pair (q_f, a) such that $q' = \delta_{\mathcal{A}}(q_f, a)$. This pair exists and is unique because q' is a BLUE state and therefore the root of a tree. RPNI-MERGE then redirects $\delta(q_f, a)$ to q. After that, the tree rooted in q' (which is therefore disconnected from the rest of the DFA) is folded (RPNI-FOLD) into the rest of the DFA. The possible intermediate situations of non-determinism (see Figure 12.7, page 242) are dealt with during the recursive calls to RPNI-FOLD. This two-step process is shown in Figures 12.13 and 12.14.

Algorithm 12.11: RPNI-MERGE.

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

Algorithm RPNI (12.13) depends on the choice of the function CHOOSE. Provided it is a deterministic function (such as one that chooses the minimal $\langle u, a \rangle$ in the lexicographic order), convergence is ensured.

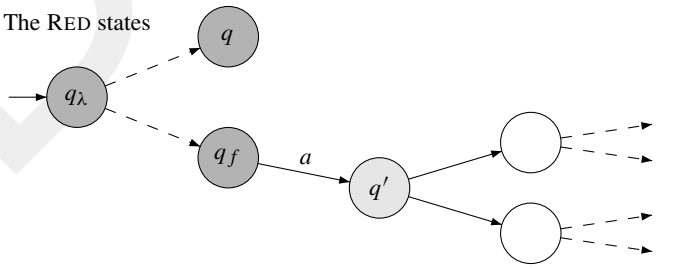


Fig. 12.13. The situation before merging.

Algorithm 12.12: RPNI-FOLD.

Input: a DFA $\mathcal{A}$, states $q, q' \in Q$, q' being the root of a tree
Output: $\mathcal{A}$ updated, where subtree in q' is folded into q
if $q' \in \mathbb{F}_{\mathbb{A}}$ **then** $\mathbb{F}_{\mathbb{A}} \leftarrow \mathbb{F}_{\mathbb{A}} \cup \{q\}$;
for $a \in \Sigma$ **do**
  if $\delta_{\mathcal{A}}(q', a)$ *is defined* **then**
    if $\delta_{\mathcal{A}}(q, a)$ *is defined* **then**
      $\mathcal{A} \leftarrow$RPNI-FOLD($\mathcal{A}, \delta_{\mathcal{A}}(q, a),\ \delta_{\mathcal{A}}(q', a)$)
    else
      $\delta_{\mathcal{A}}(q, a) \leftarrow \delta_{\mathcal{A}}(q', a)$;
    end
  end
end
return $\mathcal{A}$

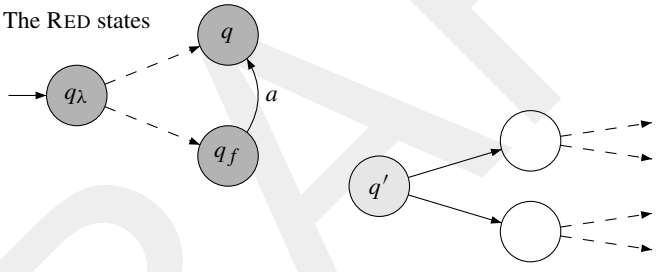


Fig. 12.14. The situation after merging and before folding.

12.4.2 The algorithm's proof

Theorem 12.4.1 *Algorithm* RPNI *(12.13) identifies in the limit* $\mathcal{DFA}(\Sigma)$ *in* POLY-CS *time.*

Proof We use here Definition 7.3.3 (page 154). We first prove that the algorithm computes a consistent solution in polynomial time. First note that the size of the PTA ($\|$PTA$\|$ being the number of states) is polynomial in $\|S\|$. The function CHOOSE can only be called at most $\|$PTA$\|$ number of times. At each call, compatibility of the running BLUE state will be checked with each state in RED. This again is bounded by the number of states in the PTA. And checking compatibility is also polynomial.

Then to prove that there exists a polynomial characteristic set we constructively add the examples sufficient for identification to take place. Let $\mathcal{A} = \langle \Sigma, Q, q_\lambda, \mathbb{F}_{\mathbb{A}}, \mathbb{F}_{\mathbb{R}}, \delta \rangle$ be the complete minimal canonical automaton for the target language L. Let $<_{\text{CHOOSE}}$ be the order relation associated with function CHOOSE. Then compute the minimum

Algorithm 12.13: RPNI.

Input: a sample $S = \langle S_+, S_- \rangle$, functions COMPATIBLE, CHOOSE
Output: a DFA $\mathcal{A} = \langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R}, \delta \rangle$
$\mathcal{A} \leftarrow$ BUILD-PTA(S_+);
RED $\leftarrow \{q_\lambda\}$;
BLUE $\leftarrow \{q_a : a \in \Sigma \cap \text{PREF}(S_+)\}$;
while BLUE $\neq \emptyset$ **do**
  CHOOSE($q_b \in$ BLUE);
  BLUE $\leftarrow$ BLUE $\setminus \{q_b\}$;
  if $\exists q_r \in$ RED *such that* RPNI-COMPATIBLE(RPNI-MERGE($\mathcal{A}$, q_r, q_b), S_-) **then**
    $\mathcal{A} \leftarrow$ RPNI-MERGE($\mathcal{A}$, q_r, q_b);
    BLUE $\leftarrow$ BLUE $\cup \{\delta(q, a) : q \in \text{RED} \wedge a \in \Sigma \wedge \delta(q, a) \notin \text{RED}\}$;
  else
    $\mathcal{A} \leftarrow$ RPNI-PROMOTE(q_b, $\mathcal{A}$)
  end
end
for $q_r \in$ RED **do** `/* mark rejecting states */`
  if $\lambda \in (\mathbb{L}(\mathcal{A}_{q_r})^{-1} S_-)$ **then** $\mathbb{F}_\mathbb{R} \leftarrow \mathbb{F}_\mathbb{R} \cup \{q_r\}$
end
return $\mathcal{A}$

distinguishing string between two states q_u, q_v (MD), and the shortest prefix of a state q_u (SP):

- $\text{MD}(q_u, q_v) = \min_{<_{\text{CHOOSE}}}\{w \in \Sigma^\star : \big(\delta(q_u, w) \in \mathbb{F}_\mathbb{A} \wedge \delta(q_v, w) \in \mathbb{F}_\mathbb{R}\big) \vee \big(\delta(q_u, w) \in \mathbb{F}_\mathbb{R} \wedge \delta(q_v, w) \in \mathbb{F}_\mathbb{A}\big)\}$.
- $\text{SP}(q_u) = \min_{<_{\text{CHOOSE}}}\{w \in \Sigma^\star : \delta(q_\lambda, w) = q_u\}$.

RPNI-CONSTRUCTCS($\mathcal{A}$) (Algorithm 12.14) uses these definitions to build a characteristic sample for RPNI, for the order $<_{\text{CHOOSE}}$, and the target language.

MD(q_u, q_v) represents the minimum suffix allowing us to establish that states q_u and q_v should never be merged. For example, if we consider the automaton in Figure 12.15, in which the states are numbered in order to avoid confusion, this string is `aa` for q_1 and q_2.

SP(q_u) is the ***shortest prefix*** in the chosen order that leads to state q_u. Normally this string should be u itself. For example, for the automaton represented in Figure 12.15, $\text{SP}(q_1) = \lambda$, $\text{SP}(q_2) = \texttt{a}$, $\text{SP}(q_3) = \texttt{aa}$ and $\text{SP}(q_4) = \texttt{b}$. □

12.4.3 A run of the algorithm

To run RPNI we first have to select a function CHOOSE. In this case we use the lex-length order over the prefixes leading to a state in the PTA. This allows us to mark the states once

Algorithm 12.14: RPNI-CONSTRUCTCS.

Input: $\mathcal{A} = \langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R}, \delta \rangle$
Output: $S = \langle S_+, S_- \rangle$
$S_+ \leftarrow \emptyset$;
$S_- \leftarrow \emptyset$;
for $q_u \in Q$ **do**
  for $q_v \in Q$ **do**
    for $a \in \Sigma$ *such that* $\mathbb{L}(\mathcal{A}_{q_v}) \cap a\,\Sigma^\star \neq \emptyset$ *and* $q_u \neq \delta(q_v, a)$ **do**
      $w \leftarrow \mathrm{MD}(q_u, q_v)$;
      if $\delta(q_\lambda, u \cdot w) \in \mathbb{F}_\mathbb{A}$ **then**
        $S_+ \leftarrow S_+ \cup \mathrm{SP}(q_u) \cdot w$; $S_- \leftarrow S_- \cup \mathrm{SP}(q_v)a \cdot w$
      else $S_- \leftarrow S_- \cup \mathrm{SP}(q_u) \cdot w$; $S_+ \leftarrow S_+ \cup \mathrm{SP}(q_v)a \cdot w$
    end
  end
end
return $\langle S_+, S_- \rangle$

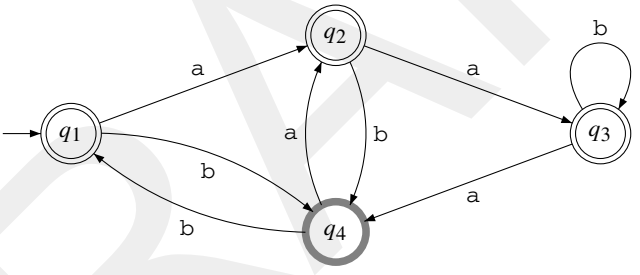


Fig. 12.15. Shortest prefixes of a DFA.

and for all. With this order, state q_1 corresponds to q_λ in the PTA, q_2 to $q_\mathtt{a}$, q_3 to $q_\mathtt{b}$,q_4 to $q_\mathtt{aa}$, and so forth.

The data for the run are:

$$S_+ = \{\mathtt{aaa}, \mathtt{aaba}, \mathtt{bba}, \mathtt{bbaba}\}$$
$$S_- = \{\mathtt{a}, \mathtt{bb}, \mathtt{aab}, \mathtt{aba}\}$$

From this we build PTA(S_+), depicted in Figure 12.16.

We now try to merge states q_1 and q_2, by using Algorithm RPNI-MERGE with values $\mathcal{A}$, q_1, q_2. Once transition $\delta_\mathcal{A}(q_1, \mathtt{a})$ is redirected to q_1, we reach the situation represented in Figure 12.17.

This is the point where Algorithm RPNI-FOLD is called, in order to fold the subtree rooted in q_2 into the rest of the automaton; the result is represented in Figure 12.18.

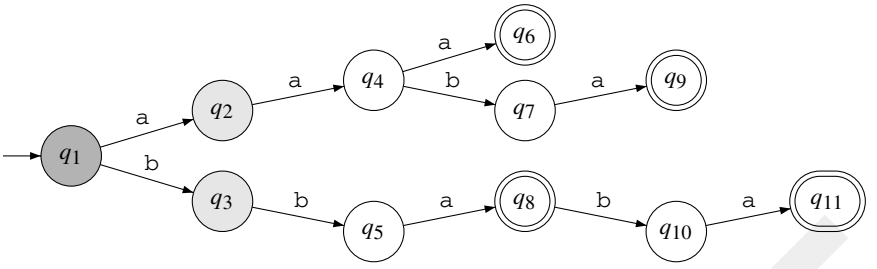


Fig. 12.16. PTA(S_+).

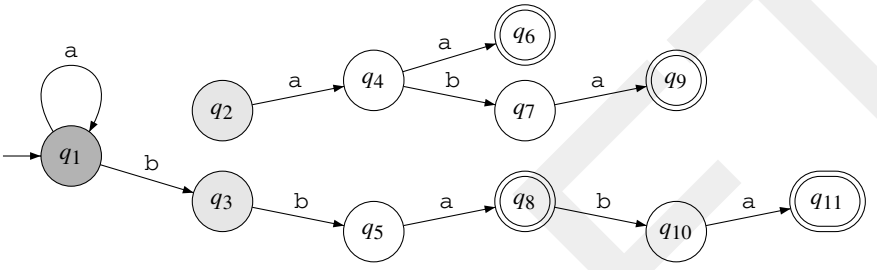


Fig. 12.17. After $\delta_{\mathcal{A}}(q_1, \mathtt{a}) = q_1$.

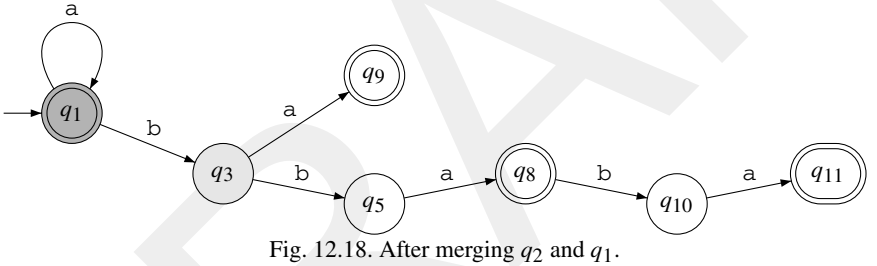


Fig. 12.18. After merging q_2 and q_1.

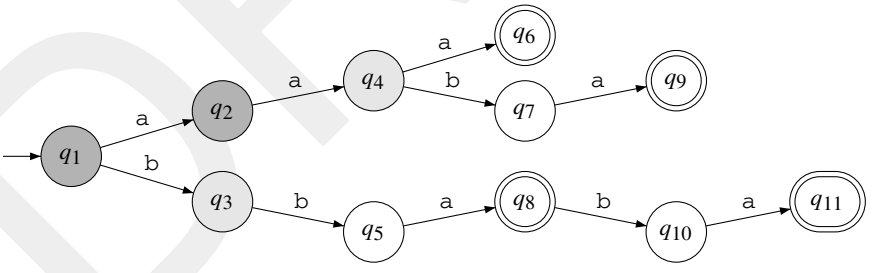


Fig. 12.19. The PTA with q_2 promoted.

The resulting automaton can now be tested for compatibility but if we try to parse the negative examples we notice that counter-example `a` is accepted. The merge is thus abandoned and we return to the PTA.

State q_2 is now promoted to RED, and its successor q_4 is BLUE (Figure 12.19).

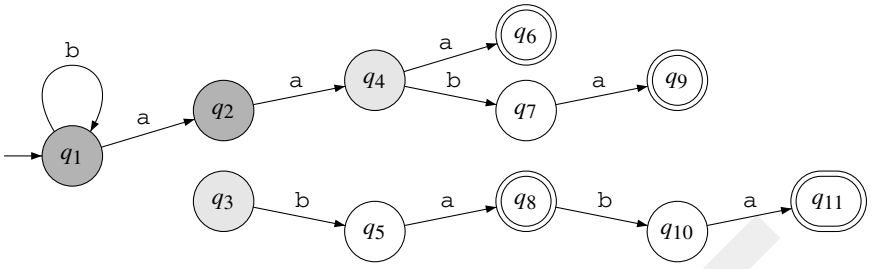


Fig. 12.20. Trying to merge q_3 and q_1.

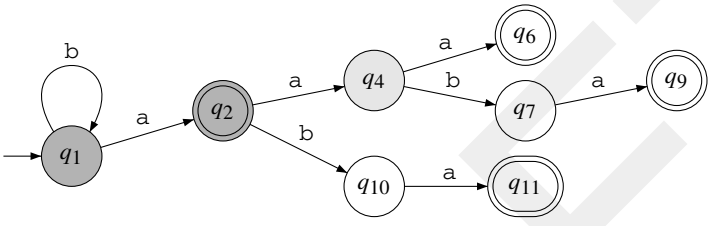


Fig. 12.21. After the folding.

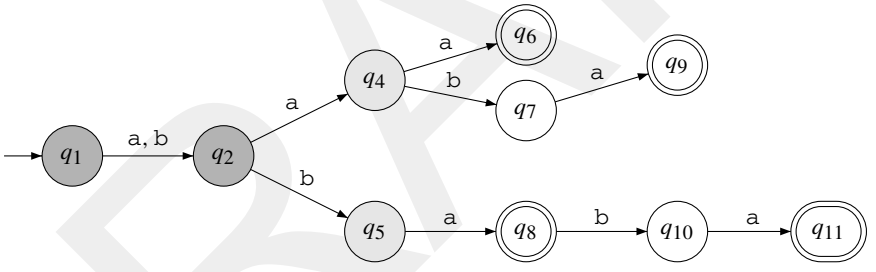


Fig. 12.22. Trying to merge q_3 and q_2.

So the next BLUE state is q_3 and we now try to merge q_3 with q_1. The automaton in Figure 12.20 is built by considering the transition $\delta_{\mathcal{A}}(q_1, \mathtt{b}) = q_1$. Then RPNI-FOLD($\mathcal{A}$, q_1, q_3) is called.

After folding, we get an automaton (see Figure 12.21) which again parses counter-example `a` as positive.

Therefore the merge $\{q_1, q_3\}$ is abandoned and we must now check the merge between q_3 and q_2. After folding, we are left with the automaton in Figure 12.22 which this time parses the counter-example `aba` as positive.

Since q_3 cannot be merged with any RED state, there is again a promotion: RED $= \{q_1, q_2, q_3\}$, and BLUE $= \{q_4, q_5\}$. The updated PTA is depicted in Figure 12.23.

The next BLUE state is q_4 and the merge we try is q_4 with q_1. But `a` (which is the distinguishing suffix) is going to be accepted by the resulting automaton (Figure 12.24).

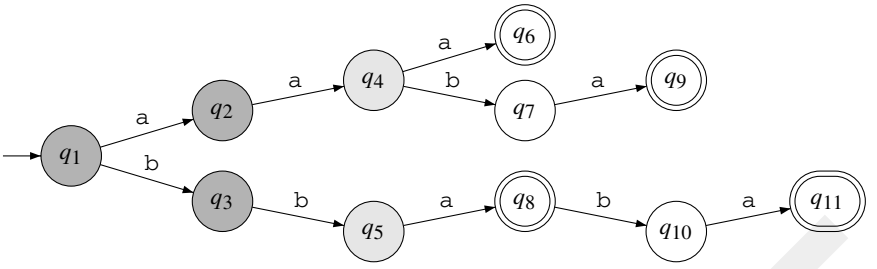


Fig. 12.23. The PTA with q_3 promoted.

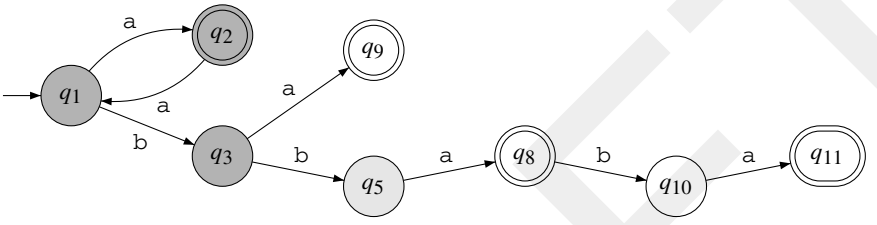


Fig. 12.24. Merging q_4 with q_1 and folding.

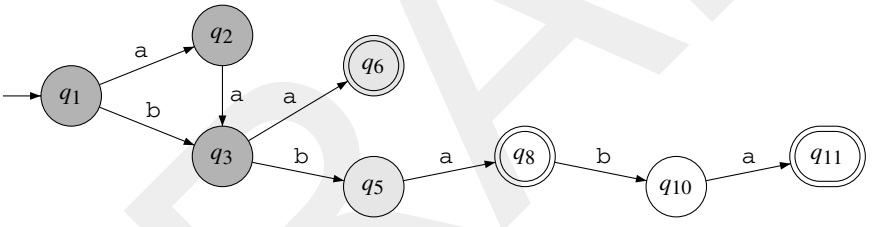


Fig. 12.25. Automaton after merging q_4 with q_3.

The merge between q_4 and q_2 is then tested and fails because of `a` now being parsed. The next merge (q_4 with q_3) is accepted. The resulting automaton is shown in Figure 12.25.

The next BLUE state is q_5; notice that state q_6 has the shortest prefix at that point, but what counts is the situation in the original PTA. The next merge to be tested is q_5 with q_1: it is rejected because of string `a` which is a counter-example that would be accepted by the resulting automaton (represented in Figure 12.26). Then the algorithm tries merging q_5 with q_2: this involves folding in the different states q_8, q_{10} and q_{11}. The merge is accepted, and the automaton depicted in Figure 12.27 is constructed.

Finally, BLUE state q_6 is merged with q_1. This merge is accepted, resulting in the automaton represented in Figure 12.28.

Last, the states are marked as final (rejecting). The final accepting ones are correct, but by parsing strings from S_-, state q_2 is marked as rejecting (Figure 12.29).

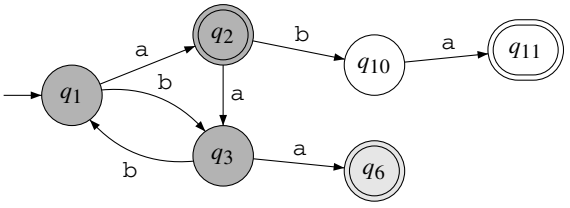


Fig. 12.26. Automaton after merging q_5 with q_1, and folding.

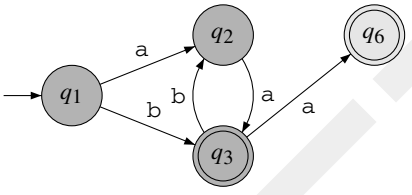


Fig. 12.27. Automaton after merging q_5 with q_2, and folding.

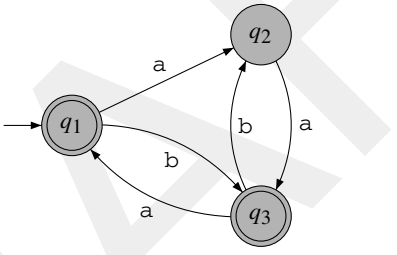


Fig. 12.28. Automaton after merging q_6 with q_1.

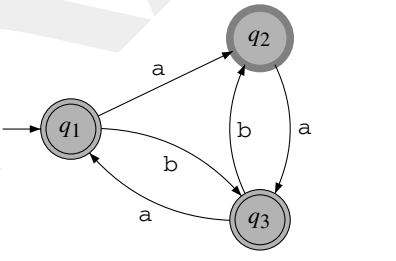


Fig. 12.29. Automaton after marking the final rejecting states.

12.4.4 Some comments about implementation

The RPNI algorithm, if implemented as described above, does not scale up. It needs a lot of further work done to it before reaching a satisfactory implementation. It will be necessary to come up with a correct data structure for the PTA and the intermediate automata. One should consider the solutions proposed by the different authors working in the field.

The presentation we have followed here avoids the heavy non-deterministic formalisms that can be found in the literature and that add an extra (and mostly unnecessary) difficulty to the implementation.

12.5 Exercises

12.1 Algorithm 12.4 (page 245) has a complexity in $\mathcal{O}(|\text{STA}|^2 + |\text{STA}| \cdot |E|^2)$. Find an alternative algorithm, with a complexity in $\mathcal{O}(|\text{STA}| \cdot |E|)$.

12.2 Run Gold's algorithm for the following data:

$$S_+ = \{\texttt{a}, \texttt{abb}, \texttt{bab}, \texttt{babbb}\}$$
$$S_- = \{\texttt{ab}, \texttt{bb}, \texttt{aab}, \texttt{b}, \texttt{aaaa}, \texttt{babb}\}$$

12.3 Take $\text{RED} = \{q_\lambda\}$, $\text{EXP} = \{\lambda, \texttt{b}, \texttt{bbb}\}$. Suppose $S_+ = \{\lambda, \texttt{b}\}$ and $S_- = \{\texttt{bbb}\}$. Construct the corresponding DFA, with Algorithm GOLD-BUILDAUTOMATON 12.5 (page 247). What is the problem?

12.4 Build an example where RED is not prefix-closed and for which Algorithm GOLD-BUILDAUTOMATON 12.5 (page 247) fails.

12.5 In Algorithm GOLD the complexity seems to depend on revisiting each cell in OT various times in order to decide if two lines are obviously different. Propose a data structure which allows the first phase of the algorithm (the deductive phase) to be in $\mathcal{O}(n \cdot \|S\|)$ where n is the size (number of states) of the target DFA.

12.6 Construct the characteristic sample for the automaton depicted in Figure 12.30(a) with Algorithm RPNI-CONSTRUCTCS (12.14).

12.7 Construct the characteristic sample for the automaton depicted in Figure 12.30(b), as defined in Algorithm RPNI-CONSTRUCTCS (12.14).

12.8 Run Algorithm RPNI for the order relations $\leq_{\textit{alpha}}$ and $\leq_{\textit{lex-length}}$ on

$$S_+ = \{\texttt{a}, \texttt{abb}, \texttt{bab}, \texttt{babbb}\}$$
$$S_- = \{\texttt{ab}, \texttt{bb}, \texttt{aab}, \texttt{b}, \texttt{aaaa}, \texttt{babb}\}.$$

12.9 We consider the following definition in which a learning algorithm is supposed to learn from its previous hypothesis and the new example:

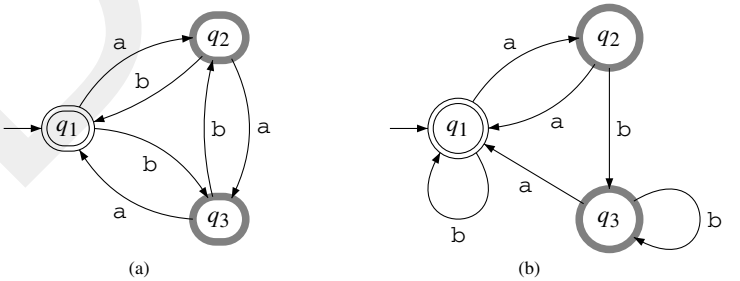


Fig. 12.30. Target automata for the exercises.

Definition 12.5.1 An algorithm **A incrementally** identifies grammar class $\mathcal{G}$ in the limit *if*$_{def}$ given any T in $\mathcal{G}$, and any presentation ϕ of $\mathbb{L}(T)$, there is a rank n such that if $i \geq n$, $\mathbf{A}(\phi(i), G_i) \equiv T$.

Can we say that RPNI is incremental? Can we make it incremental?

12.10 What do you think of the following conjecture?

Conjecture of non-incrementality of the regular languages. There exists no incremental algorithm that identifies the regular languages in the limit from an informant. More precisely, let **A** be an algorithm that given a DFA $\mathcal{A}_k$, a current hypothesis, and a labelled string w_k (hence a pair $\langle w_k, l(w_k)\rangle$), returns an automaton $\mathcal{A}_{k+1}$. In that case we say that **A** identifies in the limit an automaton T *if*$_{def}$ for no rank k above n, $\mathcal{A}_k \not\equiv T$.

Note that $l(w)$ is the label of string w, i.e. 1 or 0.

12.11 Devise a collusive algorithm to identify DFAs from an informant. The algorithm should rely on an encoding of DFAs over the intended alphabet Σ. The algorithm checks the data, and, if some string corresponds to the encoding of a DFA, this DFA is built and the sample is reconsidered: is the DFA minimal and compatible for this sample? If so, the DFA is returned. If not, the PTA is returned. Check that this algorithm identifies DFAs in POLY-CS time.

12.6 Conclusions of the chapter and further reading

12.6.1 Bibliographical background

In Section 12.1 we have tried to define in a uniform way the problem of learning DFAs from an informant. The notions developed here are based on the common notion of the prefix tree acceptor, sometimes called the augmented PTA, which has been introduced by various authors (Alquézar & Sanfeliu, 1994, Coste & Fredouille, 2003). It is customary to present learning in a more asymmetric way as generalising from the PTA and controlling the generalisation (i.e. avoiding over-generalisation) through the negative examples. This approach is certainly justified by the capacity to define the search space neatly (Dupont, Miclet & Vidal, 1994): we will return to it in Chapters 6 and 14.

Here the presentation consists of viewing the problem as a classification question and giving no advantage to one class over another. Among the number of reasons for preferring this idea, there is a strong case for manipulating three types of states, some of which are of unknown label. There is a point to be made here: if in ideal conditions, and when convergence is reached, the hypothesis DFA (being exactly the target) will only have final states (some accepting and some rejecting), this will not be the case when the result is incorrect. In that case deciding that all the non-accepting states are rejecting is bound to be worse than leaving the question unsettled.

The problem of identifying DFAs from an informant has attracted a lot of attention: E. Mark Gold (Gold, 1978) and Dana Angluin (Angluin, 1978) proved the intractability of

finding the smallest consistent automaton. Lenny Pitt and Manfred Warmuth extended this result to non-approximability (Pitt & Warmuth, 1993). Colin de la Higuera (de la Higuera, 1997) noticed that the notion of polynomial samples was non-trivial.

E. Mark Gold's algorithm (GOLD) (Gold, 1978) was the first grammatical inference algorithm with strong convergence properties. Because of its incapacity to do better than return the PTA the algorithm is seldom used in practice. There is nevertheless room for improvement. Indeed, the first phase of the algorithm (the deductive step) can be implemented with a time complexity of $\mathcal{O}(\|S\| \cdot n)$ and can be used as a starting point for heuristics.

Algorithm RPNI is described in Section 12.4. It was developed by Jose Oncina and Pedro García (Oncina & García, 1992). Essentially this presentation respects the original algorithm. We have only updated the notations and somehow tried to use a terminology common to the other grammatical inference algorithms. There have been other alternative approaches based on similar ideas: work by Boris Trakhtenbrot and Ya Barzdin (Trakhtenbrot & Bardzin, 1973) and by Kevin Lang (Lang, 1992) can be checked for details.

A more important difference in this presentation is that we have tried to avoid the non-deterministic steps altogether. By replacing the symmetrical merging operation, which requires determinisation (through a *cascade of merges*), by the simpler asymmetric folding operation, NFAs are avoided.

In the same line, an algorithm that doesn't construct the PTA explicitly is presented in (de la Higuera, Oncina & Vidal, 1996). The RED, BLUE terminology was introduced in (Lang, Pearlmutter & Price, 1998), even if does not coincide exactly with previous definitions: in the original RPNI the authors use *shortest prefixes* to indicate the RED states and elements of the *kernel* for some prefixes leading to the BLUE states. Another analysis of merging can be found in (Lambeau, Damas & Dupont, 2008).

Algorithm RPNI has been successfully adapted to tree automata (García & Oncina, 1993), and infinitary languages (de la Higuera & Janodet, 2004).

An essential reference for those wishing to write their own algorithms for this task is the datasets. Links about these can be found on the grammatical inference webpage (van Zaanen, 2003). One alternative is to generate one's own targets and samples. This can be done with the GOWACHIN machine (Lang, Pearlmutter & Coste, 1998).

12.6.2 Some alternative lines of research

Both GOLD and RPNI have been considered as good starting points for other algorithms. During the ABBADINGO competition, state merging was revisited, the order relation being built during the run of the algorithm. This led to heuristic EDSM (evidence driven state merging), which is described in Section 14.5.

More generally the problem of learning DFAs from positive and negative strings has been tackled by a number of other techniques (some of which are presented in Chapter 14).

12.6.3 Open problems and possible new lines of research

There are a number of questions that still deserve to be looked into concerning the problem of learning DFAs from an informant, and the algorithms GOLD and RPNI:

Concerning the problem of learning DFAs from an informant. Both algorithms we have proposed are incapable of adapting correctly to an incremental setting. Even if a first try was made by Pierre Dupont (Dupont, 1996), there is room for improvement. Moreover, one aspect of incremental learning is that we should be able to *forget* some of the data we are learning from during the process. This is clearly not the case with the algorithms we have seen in this chapter.

Concerning the GOLD algorithm. There are two research directions one could recommend here. The first corresponds to the deductive phase. As mentioned in Exercise 12.5, clever data structures should accelerate the construction of the table with as many obviously different rows as possible.

The second line of research corresponds to finding better techniques and heuristics to fill the holes.

Concerning the RPNI algorithm. The complexity of the RPNI algorithm remains loosely studied. The actual computation which is proposed is not convincing, and empirically, those that have consistently used the algorithm certainly do not report a cubic behaviour. A better analysis of the complexity (joined with probably better data structures) is of interest in that it would allow us to capture the parts where most computational effort is spent.

Other related topics. A tricky open question concerns the collusion issues. An alternative 'learning' algorithm could be to find in the sample a string which would be the encoding of a DFA, decode this string and check if the characteristic sample for this automaton is included. This algorithm would then rely on collusion: it needs a teacher to encode the automaton. Collusion is discussed in (Goldman & Mathias, 1996): for the learner-teacher model to be able to resist collusion, an adversary is introduced. This, here, corresponds to the fact that the characteristic sample is to be included in the learning sample for identification to be mandatory. But the fact that one can encode the target into a unique string, which is then correctly labelled and passed to the learner together with a proof that the number of states is at least n, which has been remarked in a number of papers (Castro & Guijarro, 2000, Denis & Gilleron, 1997, Denis, d'Halluin & Gilleron, 1996) remains troubling.