Learning regular sets from queries and counterexamples (1987)

Dana Angluin,
presented by Marc-Olivier Buob

LINCS, april 2019

About this presentation

This work is based on the original article:

> Learning regular sets from queries and counterexamples
>
> D Angluin - Information and computation, 1987 - Elsevier
>
> The problem of identifying an unknown regular set from examples of its members and nonmembers is addressed. It is assumed that the regular set is presented by a minimally adequate Teacher, which can answer membership queries about the set and can also test a conjecture and indicate whether it is equal to the unknown set and provide a counterexample if not.(A counterexample is a string in the symmetric difference of the correct set and the conjectured set.) A learning algorithm L∗ is described that correctly learns any …
>
> Cité 2060 fois   Autres articles   Les 20 versions
>
> Résultat de recherche le plus pertinent Voir tous les résultats

and *The L^* algorithm, Nir Levental, May 2018*.

Huge thanks to Anne Bouillard for her help ;-)

Outline

1. Context
2. Preliminaries: words, languages, DFA
3. Building automaton from observation
4. The L^* algorithm

Introduction

Context

Goal: How to describe an infinite set of strings in a program?
Many applications:

- Text search,
- Spell checking,
- Routing,
- ...

Background

Many data structures:

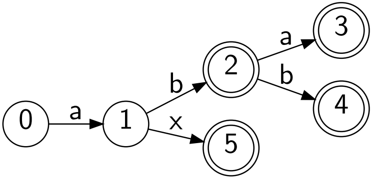

Figure: Trie recognizing $\{ab, aba, abb, ax\}$

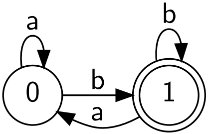

Figure: DFA recognizing $\{ab, abb, abab, \ldots\}$

In this talk we will talk about Deterministic Finite Automaton (DFA).

Goal

In this talk, a *learner* tries to learn a DFA M by querying a *teacher*.

- The learner can only ask two types of question to the teacher:
  1. Does a word w belong to the language represented by M?
  2. Does an automaton H corresponds to M? If not, the teacher gives a counter example.

Goal: design an efficient "learner" algorithm.

Preliminaries

Words and languages

- An *alphabet* Σ is a finite the set of symbols (or *letters*).
  - *Ex:* $\Sigma = \{a, b\}$
- A *word* is a sequence of letters.
  - *Ex:* "*a*", "*aba*", "*aaabbbaa*"
- ε denotes the *empty word*.
- The set of all the words over Σ is denoted by Σ^*.
- Any subset of words $L \subseteq \Sigma^*$ is called *language*.
  - *Ex:* $\{aaa, bbb\}$, all the words starting by "*aba*", all words formed by repeating "*ab*".

Prefixes and suffixes

Consider a word $w = w_0 \dots w_n$:

- Each word $w_0 \dots w_k, 0 \leq k \leq n$ is a *prefix* of w, as well as ε.
  - *Ex:* $\mathrm{prefixes}(\text{"}hello\text{"}) = \{\varepsilon, \text{"}h\text{"}, \text{"}he\text{"}, \text{"}hel\text{"}, \text{"}hell\text{"}, \text{"}hello\text{"}\}$
- Each word $w_k \dots w_n, 0 \leq k \leq n$ is a *suffix* of w, as well as ε.
  - *Ex:* $\mathrm{suffixes}(\text{"}hello\text{"}) = \{\text{"}hello\text{"}, \text{"}ello\text{"}, \text{"}llo\text{"}, \text{"}lo\text{"}, \text{"}o\text{"}, \varepsilon\}$

DFA: definition (1/4)

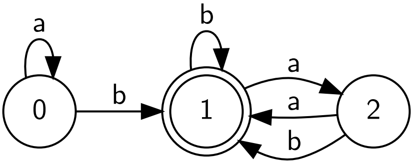

Figure: A small DFA.

A DFA can be represented by a tuple $(Q, \Sigma, q_0, F, \delta)$.

- Q is the set of states.
- Σ is the alphabet.
- q_0 is the initial state.
- $\delta : Q \times \Sigma \to Q \cup \{\perp\}$ is the transition function.
- $F \subseteq Q$ is the set of final states.

DFA: example (2/4)

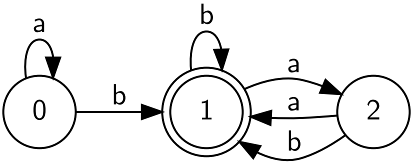

Figure: A small DFA.

Here:

- $Q = \{0, 1, 2\}, q_0 = 0, \Sigma = \{a, b\}, F = \{1\}$.
- $\delta(0, a) = 0, \delta(0, b) = 1,$
  $\delta(1, a) = 2, \delta(1, b) = 1,$
  $\delta(2, a) = 1, \delta(2, b) = 1$

DFA: properties (3/4)

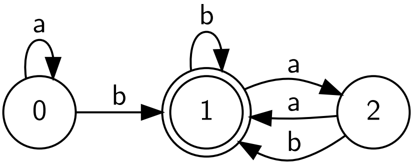

Figure: A small DFA.

Properties of this automaton:

- *Deterministic:* for all q, egress transition are labeled by distinct symbol.
- *Complete:* $\forall q \in Q, \forall a \in \Sigma, \delta(q, a) \in Q$.
- *Finite:* Q is finite.
- *Minimal:* M is minimal, because there is no smaller DFA accepting the same set of words.

DFA: language (4/4)

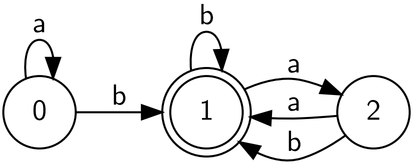

Figure: A small DFA.

- δ naturally extends to $Q \times \Sigma^*$:
  - $\forall w \in \Sigma^*, \forall a \in \Sigma, \delta(q, w.a) = \delta(\delta(q, w), a)$
  - *Ex:* $\delta(0, ba) = 2, \delta(0, bbbabaa) = 1$
- If $\delta(q_0, w) \in F$, then the word is said to be *accepted*.
  - *Ex:* "*bbb*","*abbaaab*" are accepted, but not ε,"*a*","*bba*".
- $\mathcal{L}(M)$ is called the (regular) *language* of M and gathers all the words accepted words by M.

Building automaton from observations

Problem statement

- The learner try to infer a DFA M kept secret by the *teacher*.
- The learner can only ask two types of queries:
  - *membership queries:* $w \in \mathcal{L}(M)$?
  - *equivalence queries:* $\mathcal{L}(H) \neq \mathcal{L}(M)$?
    - ...where H denotes a hypothesis automaton.
    - If $\mathcal{L}(H) \neq \mathcal{L}(M)$, the teacher returns a word in $\mathcal{L}(M) \backslash \mathcal{L}(H) \cup \mathcal{L}(H) \backslash \mathcal{H}(M)$
- At the end of the L^* algorithm, the learner has discovered the *minimal complete DFA* verifying $\mathcal{L}(M) = \mathcal{L}(H)$.

Observation table

To keep track of her queries, the learner maintains a triple (S, E, T) where:

- S is a set of prefixes.
- E is a set of suffixes.
- $T : (S \cup S.\Sigma) \times E \to \{0, 1\}$ is the observation table defined by

$$T(s, e) = 1 \text{ iff } s.e \in \mathcal{L}(M)$$

To be able to derive an hypothesis automaton, T must meet two properties: *closeness* and *separability*.

- *Separability* is required to identify each state of H by a prefix.
- *Closeness* is required to identify the target of each transition.

Let's see what are those properties...

Closeness

Definition:

$$\forall s \in S, \forall a \in \Sigma, \exists s' \in S \mid \mathrm{row}(s.a) = \mathrm{row}(s')$$

Examples: Assume $\Sigma = \{a, b\}, S = \{\varepsilon, a\}$ and $E = \{\varepsilon\}$:

	ε
ε	1
a	0
b	1
aa	1
ab	0

Figure: T is closed.

	ε
ε	1
a	1
b	1
aa	1
ab	0

Figure: T is not closed because $\mathrm{row}(ab) \notin \{\mathrm{row}(\varepsilon), \mathrm{row}(a)\}$.

Separability

Definition:

$$\forall s \in S, \forall s' \in S, (s \neq s') \implies (\mathrm{row}(s) \neq \mathrm{row}(s'))$$

Examples: Assume $\Sigma = \{a, b\}, S = \{\varepsilon, a\}$ and $E = \{\varepsilon\}$:

	ε
ε	1
a	0
b	1
aa	1
ab	0

Figure: T is separable.

	ε
ε	1
a	1
b	1
aa	1
ab	0

Figure: T is not separable because $\mathrm{row}(\varepsilon) = \mathrm{row}(a)$.

Building H from T (1/3)

Assume $\Sigma = \{a, b\}, S = \{\varepsilon, a\}, E = \{\varepsilon\}$ and T defined as follows:

	ε
ε	1
a	0
b	1
aa	1
ab	0

Figure: T is closed and separable, so we can build H.

Building H from T (2/3)

Build states:

- One state per distinct row related to prefixes $s \in S$
- q is final iff s.t. $T(s, \varepsilon) = 1$.

	ε
ε	1
a	0
b	1
aa	1
ab	0

Here, two states 0 and 1 are created corresponding namely to ε and a. $q_0 = 0$ as ε identifies 0.

Building H from T (2/3)

Build transitions:

- For each state q, get its suffix s.
- For each $a \in \Sigma$, retrieve $\mathrm{row}(s.a)$ and find the corresponding prefix $s' \in S$ s.t. $\mathrm{row}(s.a) = \mathrm{row}(s')$.
- Find q' the state identified by s'. Connect q to q' with a a-transition.

	ε
ε	1
a	0
b	1
aa	1
ab	0

Here, if state 1 corresponds is identified a, what is $\delta(1, b)$? To answer we extract $\mathrm{row}(ab)$ which corresponds to $\mathrm{row}(a)$. So $\delta(1, b) = 1$.

The L^* algorithm

Definitions: Closed-by-...

Consider a word $w = w_0 \dots w_n$, for example "hello":

- Each word $w_0 \dots w_k, 0 \leq k \leq n$ is a *prefix* of w, as well as ε.
    - *Ex:* ε, *"h", "he", "hel", "hell", "hello"*
- Each word $w_k \dots w_n, 0 \leq k \leq n$ is a *suffix* of w, as well as ε.
    - *Ex: "hello", "ello", "llo", "lo", "o",* ε

A language L is *closed-by-prefix* (resp. *closed-by-suffix*) iff for each word of L, its prefixes (resp. suffixes) also belong to L.

- *Ex:* $\{\varepsilon,$ *"h", "he", "hel", "hell", "hello"*$\}$ is closed-by-prefix.

Information maintained learner-side

- A set $S \subset \Sigma^*$ of words *closed-by-prefix*, initialized to $S \leftarrow \emptyset$.
- A set $E \subset \Sigma^*$ of words *closed-by-suffix*, initialized to $E \leftarrow \emptyset$.
- A mapping $T = \{0,1\}^{(S \cup S.\Sigma) \times E}$ indicating whether a word $w \in (S \cup S.\Sigma).E$ belongs to $\mathcal{L}(M)$ or not.
  - ... where $S.\Sigma = \{s.a \mid s \in S, a \in \Sigma\}$.
  - T is usually represented as a table where:
    - each row corresponds to a prefix in $(S \cup S.\Sigma) \times E$;
    - each column corresponds to a suffix in E.
  - $T(s,e) = 1 \iff s.e \in \mathcal{L}(M)$.

Algorithm 1: L^* algorithm.

Input: Σ the alphabet, EQ the equivalence query, MQ the membership query
Output: H the hypothesis automaton

```
(S, E) ← ({ε}, {ε}) ;
T(ε, ε) ← MQ(ε) ;
while true do
    while ¬(separable(S, E, T) ∧ closed(S, E, T)) do
        if ¬separable(S, E, T) then
            Find s ∈ S, s' ∈ S two non-separable prefixes of T ;
            Find a ∈ Σ, e ∈ E s.t. T(s.a, e) ≠ T(s'.a, e) ;
            E ← E ∪ {a.e} ;
        if ¬closed(S, E, T) then
            Find s ∈ S, a ∈ Σ s.t. row(s.a) ∉ {row(s), s ∈ S} ;
            S ← S ∪ {s.a} ;
        T(s, e) ← MQ(s.e) for each unset values of T;
    Derive H according to T ;
    if not EQ(H) then
        S ← S ∪ prefixes(t) where t is a counter-example;
        T(s, e) ← MQ(s.e) for each unset values of T;
    else
        break
Return H;
```

Demo!

Implementation in Python 3, available on demand.

- Rely on `numpy` (to manage T).
- Extends `pybgl`[1] (to manipulate automata).
- Uses `graphviz` and `jupyter-notebook` (to display automata H and M).
- Test suite realized with `pytest-3`.

[1] `https://github.com/nokia/pybgl`

Conclusion

- L^* algorithm allows a learner to discover the DFA of a teacher.
- It took me time to fully understand this work. Notations were confusing. The both articles used different terminologies and notations. And algorithms were not so detailed.
- Once again, huge thanks to Anne :-)
- See the original article for proofs and performances.