The L* Algorithm

Suppose we are presented with a language like $L = \{x \in \{0,1\}^* \mid \text{The third bit from the right end is a } 1\}$. We say we intuitively understand this language. Given a string we know if it is in A or not.
Coming up with a DFA for a language from scratch is somewhat more difficult. It would take some dedication. We might end up drawing a few incorrect DFAs during our search.

The L* algorithm is one way of writing a formal method which we use in finding a DFA for a language.

The Problem

The L* algorithm can be thought of as a game between two players - a *Teacher* and a *Learner*.
We play the role of the Learner, who wants to learn a regular language L from the Teacher. To that end, we can ask the Teacher two types of questions.

- **Membership Query**: Whether the given word w belongs to L or not. The Teacher answers with a boolean - True if $w \in L$, False otherwise.
- **Equivalence Query**: Whether the given DFA H accepts L or not. If H accepts T, the Teacher answers True. If H does not accept L, the Teacher answers with a counterexample $x \in L \backslash L(H) \cup L(H) \backslash L$.

In order to successfully *learn* the language L, we have to guess the correct DFA within a finite number of queries.

Broadly, the strategy is like this: We will maintain two sets of strings Q and T.
The set Q will be the set of states for our DFA. The set T (sometimes called a *test word set*) is used to distinguish different states. We will repeatedly construct DFAs, make an equivalence query, use the counterexample to extend T, and then repeat until we find a correct DFA.

In order to formalise the strategy, we define the following:

T-equivalent: Given any two strings $u, v \in \Sigma^*$ and a set $T \subseteq \Sigma^*$, we say that u, v are *T-equivalent*, and write $u \equiv_T v$, if $\forall t \in T,\ u \cdot t \in L \iff v \cdot t \in L$. Otherwise, we say that they are *T-distinguishable*.

Note that $\equiv_T$ is an equivalence relation, and for $T = \Sigma^*$, this relation is the same as I_L, the indistinguishability relation of L.
Also, for every $T_1 \subseteq T_2 \subseteq \Sigma^*$, $\equiv_{T_2}$ is a refinement of $\equiv_{T_1}$. As T_2 has more strings in it, it has a better chance of distinguishing any given pair of strings
Thus, for a sequence of sets $T_1 \subseteq T_2 \subseteq \cdots \subseteq \Sigma^*$, we have $\equiv_{T_2}$ refines $\equiv_{T_1}$, $\equiv_{T_3}$ refines $\equiv_{T_2}$, and so on.
But Σ^* is a superset of all the T's, so I_L refines every $\equiv_T$.

In the rest of our discussion we will assume that both Q and T are of finite size. Let us now state the strategy formally.

The following properties help us define a transition function for our DFA.

Separable: A set $Q \subseteq \Sigma^*$ is said to be *separable* with respect to T, if the elements of Q are pairwise T-distinguishable.

Closed: A set Q is said to be *closed* with respect to T, if $\forall q \in Q\ \forall a \in \Sigma,\ \exists q' \in Q$ such that $q \cdot a \equiv_T q'$.

Lemma 1. *If Q is closed and separable with respect to T, the transition function $\delta : (q, a) \to q' \in Q$ such that $q' \equiv_T q \cdot a$, is well defined.*

Proof. By definition of closure property, $\forall q \in Q\ \forall a \in \Sigma$ a q' always exists which is T-equivalent to $q \cdot a$. By definition of separability, this q' is unique. □

This lets us construct a hypothesis DFA $H = (Q, \Sigma, \delta, \varepsilon, F = \{q \in Q | q \in L\})$. The following two lemmas let us form the main loop of the L^* algorithm.

Lemma 2. *Given a hypothesis DFA $H = (Q, \Sigma, \delta, \varepsilon, F)$ where Q is closed and separable with respect to T, and a counterexample $w = w_1 w_2 \cdots w_m$, we can find strings q_{n+1} and t such that $Q' = Q \cup \{q_{n+1}\}$ is separable with respect to $T' = T \cup \{t\}$.*

Proof. Define $p_i = \delta^*(\varepsilon, w_1 w_2 \cdots w_i)$. We say a state p_i is *correct* if $p_i w_{i+1} \cdots w_m \in L \iff w \in L$. Now, ε is correct trivially, and p_m is not correct since w is a counterexample. Thus, there is some k such that p_{k-1} is correct but p_k is not. Then $t = w_{k+1} \cdots w_m$ distinguishes p_k and $p_{k-1} w_k$.
Since $p_{k-1} w_k \equiv_T p_k$ and $p_k \in Q$, by separability of Q, $p_{k-1} w_k$ is T-distinguishable from every element of $Q \backslash p_k$. As T' refines T, it is also T'-distinguishable from every string in $Q \backslash p_k$. And by construction, it is T'-distinguishable from p_k. Every pair of elements in Q is T'-distinguishable due to T' refining T.
Thus, Q' is separable with respect to T'. □

Lemma 3. *If Q is separable with respect to T, it is possible to add finitely many strings to Q resulting in a set Q' which is closed and separable with respect to T.*

Proof. Since $|T|$ is finite, we can check if two strings are T-equivalent with finitely many membership queries.
If Q is not closed, we can find a string $q \in Q$ and a letter $a \in \Sigma$ such that $q \cdot a$ is T-distinguishable from every element of Q. Then $Q \cup \{qa\}$ is still separable with respect to T.
We can repeat this process and eventually reach a set Q' which is closed. This will only take finitely many steps as
$|Q| \leq$ the index of $\equiv_T \leq$ the index of $I_L < \infty$.
The first inequality follows from separability of Q. The second, because I_L refines $\equiv_T$. The third, because L is regular. □

The L^* algorithm alternatingly uses lemmas 2 and 3 to slowly expand Q until $|Q|$ becomes equal to the index of I_L. At this point, Q is the set of equivalence classes of I_L and the hypothesis DFA is simply the minimal DFA accepting L.
After receiving a counterexample w from the Teacher, we create a new hypothesis DFA. It is possible that this DFA still does not correctly classify w. In this case we do not need to make another equivalence query; we can just pretend that the Teacher returned w as a counterexample again.

Following is a possible implementation of the L^* algorithm.

L*

```
1: function LEARN_DFA(Membership_Query, Equivalence_Query)
2:     Q=[ε], T=[ε]
3:     while True do
4:         Q, δ = CLOSE(Q, T)
5:         H = (Q, Σ, δ, ε, Q.filter(Membership_Query))
6:         result = EQUIVALENCE_QUERY(H)
7:         if result = True then
8:             return H
9:         new_state, new_test_word = ADD_TEST_WORD(Q, T, result)
10:        Q.append(new_state), T.append(new_test_word)
11: function ARE_INDISTINGUISHABLE(T, w1, w2)
12:    for t in T do
13:        if MEMBERSHIP_QUERY(w1·t) ≠ MEMBERSHIP_QUERY(w2·t) then
14:            return False
15:    return True
16: function CLOSE(Q, T)
17:    δ = {}
18:    i=1
19:    while i ≤ Q.length do
20:        for a in Σ do
21:            q = Q[i]
22:            for r in Q do
23:                if ARE_INDISTINGUISHABLE(q·a, r, T) then
24:                    δ[(q, a)] = r
25:                    break
26:            if (q, a) not in δ then
27:                Q.append(q·a)
28:                δ[(q, a)] = q·a
           i++
29:    return Q, δ
30: function ADD_TEST_WORD(Q, δ, w)
31:    q=ε, i=1
32:    while True do
33:        if MEMBERSHIP_QUERY(δ[q,w[i]]·w[i+1:]) ≠ MEMBERSHIP_QUERY(w) then
34:            return q·w[i], w[i+1:]
35:        else
36:            i++
37:            q = δ[q,w[i]]
```

Citations

- James Worrell, Exact learning of deterministic finite automata, Lecture notes by James Worrell, University of Oxford.
  https://www.cs.ox.ac.uk/people/james.worrell/DFA-learning.pdf

- Henrik Björklund, Johanna Björklund, and Wim Martens, Handbook of Automata Theory,

Chapter 11.
https://www.ems-ph.org/books/book.php?proj_nr=248

- Dana Angluin (1987), Learning Regular Sets from Queries and Counterexamples, *INFORMATION AND COMPUTATION 75, 87-106*
https://people.eecs.berkeley.edu/ dawnsong/teaching/s10/papers/angluin87.pdf