Symbolic learning of automata

Peter Habermehl

Université Paris Cité
Faculté des Sciences
UFR Informatique
Laboratoire IRIF
Peter.Habermehl@irif.fr

Organisation

- Webpage of this part of the course: `http://www.irif.fr/~haberm/cours/mpri`
- **Warning : The slides do not contain everything (far from that)**
- Schedule: Mondays 8h45 - 11h45
- Grades: Written exam
- Required knowledge: Basic formal language and automata theory (DFA, NFA)

General references

- Colin de la Higuera. Grammatical Inference. Learning Automata and Grammars. Cambridge University Press. 2010
  www.cambridge.org/core/books/grammatical-inference/CEEB229AC5A80DFC6436D860AC79434F
  pagesperso.lina.univ-nantes.fr/∼cdlh/book/
- Sicco Verwer. Efficient Identification of Timed Automata - Theory and Practice. PhD Thesis. TU Delft (Netherlands). 2010.
  repository.tudelft.nl/islandora/object/uuid:61d9f199-7b01-45be-a6ed-04498113a212/?collection=research

Introduction

on the board

Learning (Identifying) languages

The setting:

- $\mathcal{L}$: a language class
- $\mathcal{G}$: a class of representations of objects in a language class
- $L : \mathcal{G} \mapsto \mathcal{L}$: a naming function ($L(G)$ is the language denoted, accepted, recognised, represented by G, a "grammar").
- For example, regular (rational) languages over a finite alphabet Σ form a language class $\mathcal{REG}(\Sigma)$ and can be represented by $\mathcal{DFA}(\Sigma)$ or $\mathcal{NFA}(\Sigma)$ or $\mathcal{UFA}(\Sigma)$ or $\mathcal{AFA}(\Sigma)$ or $\mathcal{REGEXP}(\Sigma)$ or etc.
- Important decision problems:
  - Membership: Given $w \in \Sigma^*$ and $G \in \mathcal{G}$ is $w \in L(G)$?
  - Equivalence: Given G_1 and G_2 in $\mathcal{G}$ is $L(G_1) = L(G_2)$?

Learning (Identifying) languages

- What class of languages to learn ?
- How languages are represented ?
- Which information is made available ?
- How the information is made available ?

Which information is made available and how ?

- Passive learning
  - A presentation for a language is given
    - (Infinite) sequence of information about the language
  - The learning algorithm uses the information to infer a representation
  - The learner has no control over the information
- Active learning (Query learning)
  - The learner can query an oracle

Identification in the limit (Gold 67)

- Let $\mathcal{L}$ be a language class
- A presentation is a function $\varphi : \mathbb{N} \mapsto X$ where X is a set. $Pres(\mathcal{L})$ is the set of all allowed presentations.
- There exists a function $YIELDS : Pres(\mathcal{L}) \mapsto \mathcal{L}$
- $Pres(L) := \{\varphi \in Pres(\mathcal{L}) : YIELDS(\varphi) = L\}$
- Examples of presentations of a language L:
  - $TEXT(L) = \{\varphi : \mathbb{N} \mapsto \Sigma^* \mid \varphi(\mathbb{N}) = L\}$
  - $INFORMANT(L) = \{\varphi : \mathbb{N} \mapsto \Sigma^* \times \{0,1\} \mid \varphi(\mathbb{N}) = L \times \{1\} \cup \overline{L} \times \{0\}\}$
- The setting is said to be valid when given two presentations φ and ψ, whenever their range is equal (i.e. if $\varphi(\mathbb{N}) = \psi(\mathbb{N})$) then $YIELDS(\varphi) = YIELDS(\psi)$.
- One learns a representation of a language and not the language itself.
- Can be generalised to other concepts to be learnt (for example logical formulas)

Identification in the limit

- Given a presentation φ we denote $\varphi_n = \{\varphi(i) \mid i \leq n\}$
- G is called consistent with φ_n if G does not contradict φ_n
- A learning algorithm Alg is a program which takes $\varphi_n = \{\varphi(i) \mid i \leq n\}$ as input and produces a grammar G
- Definition: $\mathcal{G}$ is identifiable in the limit from $Pres(\mathcal{G})$ if there exists a learning algorithm Alg such that for all $G \in \mathcal{G}$ and any presentation $\varphi \in Pres(\mathcal{G})$ of $L(G)$ there exists n such that for all $m \geq n$: $L(Alg(\varphi_m)) = L(G)$ and $Alg(\varphi_m) = Alg(\varphi_n)$.
- for behaviourally correct identification, the last point is not needed.
- A learning algorithm is called consistent, if it changes its mind as soon as the current hypothesis is erroneous with the presented element.

Two general results

Gold. Language identification in the limit. Information and Control 1967
www.sciencedirect.com/science/article/pii/S0019995867911655

- A super-finite class of languages is a class which contains all finite languages and at least an infinite one.
- No super-finite class of languages is identifiable in the limit from text
- Any recursively enumerable class of recursive languages is identifiable in the limit from an informant (by which learning algorithm ?)

Complexity aspects

- Counting time
- Counting the number of examples
- Counting the number of mind changes

Counting time

- An algorithm Alg is said to have overall polynomial time if there exists a polynomial $p()$ such that $\forall G \in \mathcal{G}\ \forall n \geq p(||G||)\ \forall \varphi \in Pres(L(G)).L(Alg(\varphi_n)) = L(G)$.
- An algorithm Alg is said to have polynomial update time if there is a polynomial $p()$ such that, for every presentation φ and every integer n, constructing $Alg(\varphi_n)$ requires $O(p(||\varphi_n||))$ time.

Counting the number of examples

- Counting the number of examples needed to identify
- Counting the number of good examples to identify
- A grammar class $\mathcal{G}$ admits polynomial characteristic samples, if there exist an algorithm Alg and a polynomial $p()$ such that $\forall G \in \mathcal{G}$, $\exists CS \subseteq X$ such that
  1. $||CS|| \leq p(||G||)$ and
  2. $\forall \varphi \in Pres(L(G))\ \forall n \in \mathbb{N} : CS \subseteq \varphi_n$ implies $L(Alg(\varphi_n)) = L(G)$.

  Such a set CS is called a characteristic sample of G for Alg. If such an algorithm Alg exists, we say that Alg identifies $\mathcal{G}$ in the limit in CS-polynomial time.
- Note: the sample is specific for an algorithm and the size $||CS||$ takes into account also the size of strings

Counting the number of mind changes

- Given a learning algorithm Alg and a presentation φ, we say that Alg changes its mind at time n, if $Alg(\varphi_n) \neq Alg(\varphi_{n-1})$.
- An algorithm that never changes its mind when the current hypothesis is consistent with the new presented element is said to be conservative.
- Algorithm Alg makes a polynomial number of mind changes (MC) if there is a polynomial $p()$ such that, for each grammar G and each presentation φ of $L = L(G)$,
  $|\{k \in \mathbb{N} \mid Alg(\varphi_k) \neq Alg(\varphi_{k+1})\}| \leq p(||G||)$.
- An algorithm Alg identifies a class $\mathcal{G}$ in the limit in MC-polynomial time if
  1. Alg identifies $\mathcal{G}$ in the limit,
  2. Alg has polynomial update time,
  3. Alg makes a polynomial number of mind changes.

Learning from text

In this context, φ_n is typically denoted as a sample S, a finite set of words included in the language to be learnt.

Some examples of languages classes identifiable in the limit from text

- The class $\mathcal{SINGLE}(\Sigma)$ of all singleton languages of the form $L = \{w\}$ where $w \in \Sigma^*$
  - Exercise: Give a learning algorithm. What are its properties ?
- The class $\mathcal{FINITE}(\Sigma)$ of all finite languages over some alphabet Σ
  - Exercise: Give a learning algorithm. What are its properties ?
- The class $\mathcal{ABO}(\Sigma)$ of all "all-but-one"-languages L of the form $L = \Sigma^* \setminus \{w\}$ where $w \in \Sigma^*$
  - Exercise: Give a learning algorithm. What are its properties ? Characteristic sample ?

Some other examples of learning algorithms from text

- Regular languages can not be identified in the limit from text
  - follows from Gold's general result
- Subclasses of regular languages
  - k-testable languages
  - reversible languages
- Pattern languages

(strictly) k-testable languages (aka window languages)

- A k-testable language is given by four sets $I, F, T, C \subseteq \Sigma^*$ with
  - $I, F \subseteq \Sigma^{k-1}$ (prefixes and suffixes of length $k-1$)
  - $C \subseteq \Sigma^{<k}$ (short strings) such that $I \cap F = C \cap \Sigma^{k-1}$
  - $T \subseteq \Sigma^k$ (allowed segments)
- Given such a representation the language is $C \cup (I\Sigma^* \cup \Sigma^* F) \setminus (\Sigma^*(\Sigma^k \setminus T)\Sigma^*)$
- Example: $k = 2$, $I = \{a, b\}$, $F = \{b\}$, $C = \{b\}$ and $T = \{ab, bb\}$ represents the language: $bb^* + abb^*$
- Exercice: Give an algorithm to build an automaton directly from I, F, C, T
- Exercice: Propose a learning algorithm from text for k-testable languages and apply it on the sample $S = \{\lambda, a, abba, abbbba\}$ for $k = 1$, $k = 2$ and $k = 3$. What are the properties of your algorithm ?

Reversible languages

Dana Angluin. Inference of reversible languages. Journal of the ACM, 1982. dl.acm.org/doi/pdf/10.1145/322326.322334

- Given a DFA A, A^T is the automaton obtained by reversing the transition relation (and the initial and final states).
- A DFA A is reversible if A^T is deterministic.
- A regular language L is reversible if there exists a DFA A with $L(A) = L$ which is reversible.
- Sketch of a learning algorithm given a sample S:
  - Build prefix-tree acceptor (see below) for S
  - Merge all final states
  - Merge states q, q' as long as there is a transition with the same letter to q and q' or from q and q'
  - There is a polynomial-size CS (Which one ?)
  - It can be made incremental (How ?)

Pattern languages

- Let Σ be an alphabet and $X = \{x_1, x_2, \ldots\}$ a set of variables. A pattern is a string over $\Sigma \cup X$.
- A matching is a function $\sigma : X \mapsto \Sigma^*$. σ is extended to a pattern $\pi = \pi_1\pi_2 \ldots \pi_n$ by $\sigma(\pi) = \sigma(\pi_1)\sigma(\pi_2)\ldots\sigma(\pi_n)$ For a letter $a \in \Sigma$, $\sigma(a) = a$.
- A string $w \in \Sigma^*$ fits a pattern π if there is a matching σ such that $\sigma(\pi) = w$.
- The language defined by a pattern π (noted $L(\pi)$) is the set of all words $w \in \Sigma^*$ which fit π
- The pattern is called non-erasing if only $\sigma : X \mapsto \Sigma^+$ is allowed
- Let $\mathcal{PATTERNS}(\Sigma)$ be the class of non-erasing pattern languages over Σ.
- Exercice: Show that $\mathcal{PATTERNS}(\Sigma)$ is identifiable in the limit from text.
- Example sample: $S = \{abcbb, aabba, aacbbac, aaaba, acbbbac\}$.

Learning from an informant

Recall: Any recursively enumerable class of recursive languages is identifiable in the limit from an informant.

- A partial presentation is a sample $S = (S^+, S^-)$ with $S^+, S^- \subseteq \Sigma^*$ such that $S^+ \cap S^- = \emptyset$.
- A DFA $A = (\Sigma, Q, q_\lambda, F_a, F_r, \delta)$ is a finite-state automaton defined as usual.
  - F_a is the set of accepting states and
  - F_r the set of rejecting states (not always used)
- A is consistent with $S = (S^+, S^-)$ if $\delta(q_\lambda, w) \in F_a$ for all $w \in S^+$ and $\delta(q_\lambda, w) \notin F_a$ for all $w \in S^-$
- A is strongly consistent with $S = (S^+, S^-)$ if $\delta(q_\lambda, w) \in F_a$ for all $w \in S^+$ and $\delta(q_\lambda, w) \in F_r$ for all $w \in S^-$
- A learning algorithm typically constructs an automaton (strongly) consistent with S at each stage.

A fundamental complexity result

- Problem: Given a sample $S = (S^+, S^-)$ of strings over some alphabet Σ and $n \in \mathbb{N}$, is there a DFA with n states consistent with S ?
- This problem is NP-complete for binary alphabets.
- There a several proofs in the literature which are wrong. see Lingg at al. Learning from Positive and Negative Examples: New Proof for Binary Alphabets. LearnAut 2022. arxiv.org/abs/2206.10025
- Proof on the board.
- Exercice: What about unary alphabets ?
- This means that learning a minimal DFA from a sample can (probably) not be done in polynomial time.
- However, we can still hope for CS-polynomial time.

Main ingredients of algorithms

- Given a sample $S = (S^+, S^-)$ the prefix-tree acceptor (PTA) for S is the smallest DFA with a tree-like structure where states are prefixes of the strings of S^+ and which is (strongly) consistent with S.
  - *BUILDPTA*(S) constructs the *PTA* for a sample S
- *RED* states: have been analysed, are part of the result
- *BLUE* states: not yet analysed, but are considered
- Myhill-Nerode congruence: $u \sim_L v$: for all $w \in \Sigma^*. uw \in L$ iff $vw \in L$. Two strings which are not equivalent must lead to different states in any DFA for L. In a minimal DFA all states are pairwise distinguishable by a word w which is accepted from one and rejected from the other.

Gold's algorithm

E Mark Gold. Complexity of Automaton identification from given data. Information and Computation 37, 1978.
www.sciencedirect.com/science/article/pii/S0019995878905624

- From $S = (S^+, S^-)$ find a set of prefixes which must lead to different states
- Try to fold in the rest of the states of the PTA.
- Example: $S^+ = \{bb, abb, bba, bbb\}$, $S^- = \{a, b, aa, bab\}$
- Main data structure: Observation table (STA, EXP, OT) where
  - $STA \subseteq \Sigma^*$ is a prefix closed disjoint union of $BLUE$ (no extension of a $BLUE$ state is $BLUE$) and RED (the others)
  - $EXP \subseteq \Sigma^*$ is a suffix closed set
  - A function $OT : STA \times EXP \mapsto \{0, 1, *\}$ defined as $OT[u][e] = 1$, if $ue \in S^+$, $OT[u][e] = 0$, if $ue \in S^-$, else $*$.
  - The table should be non-contradictory: $OT[u][vw] = OT[uv][w]$ (when defined) for all $u, v, w \in \Sigma^*$

Main concepts

- An observation table is complete if it has no holes ($OT[u][v] = *$)
- Two rows of an observation table are compatible ($u \sim_{OT} v$) if there is no e such that ($OT[u][e] = 0$ and $OT[v][e] = 1$) or ($OT[u][e] = 1$ and $OT[v][e] = 0$)
- Two rows are obviously different (*OD*) if they are not compatible.
- A complete observation table is closed if for all rows v in *BLUE* there is a row u in *RED* with $OT[u] = OT[v]$
- One can build an automaton from a closed and complete observation table: $BUILDAUTO(STA, EXP, OT)$
- This automaton is consistent with the data in the table.
- How to get a complete and closed observation table from S ?
- One can construct a table from S with holes and try to fill the holes, but that's difficult as it should not lead to a contradictory table.

$BUILDAUTO(STA, EXP, OT)$

input : A closed and complete observation table (STA, EXP, OT)
output: A DFA $A = (\Sigma, Q, q_\lambda, F_a, F_r, \delta)$
$Q \leftarrow \{q_u \mid u \in RED\}$;
$F_a \leftarrow \{q_{ue} \mid OT[u][e] = 1\}$;
$F_r \leftarrow \{q_{ue} \mid OT[u][e] = 0\}$;
for $q_u \in Q$ **do**
  for $a \in \Sigma$ **do**
    $\delta(q_u, a) \leftarrow q_v$ if $v \in RED$ and $OT[ua] = OT[v]$
  end
end
return A

Lemma: The automaton A is consistent with the information in STA, EXP, OT (i.e. $OT[u][v] = 1$ implies that A accepts uv and $OT[u][v] = 0$ implies that A rejects uv)

BUILDTABLE(*S*, *RED*)

input : A Sample $S = (S^+, S^-)$, A set of strings *RED* prefix-closed
output: An observation table (STA, EXP, OT)

```
EXP ← SUFFIXES(S);
BLUE ← RED.Σ \ RED;
for u ∈ RED ∪ BLUE do
    for e ∈ EXP do
        if ue ∈ S+ then OT[u][e] ← 1;
        else
            if ue ∈ S− then OT[u][e] ← 0 else OT[u][e] ← ∗;
        end
    end
end
return (RED ∪ BLUE, EXP, OT)
```

Gold's algorithm

input : A Sample S
output: A DFA consistent with S
$RED \leftarrow \{\lambda\}; BLUE \leftarrow \Sigma;$
$(STA, EXP, OT) \leftarrow BUILDTABLE(S, RED);$
while *there exist* $v \in BLUE$ *such that* v *is OD from all* RED **do**
  $RED \leftarrow RED \cup \{v\};$
  $BLUE \leftarrow (BLUE \setminus \{v\}) \cup \{va : a \in \Sigma\};$
  $UPDATETABLE(STA, EXP, OT);$
end
$A \leftarrow BUILDAUTOGOLD(STA, EXP, OT);$
if $CONSISTENT(A, S)$ **then return** A **else return** $BUILDPTA(S);$

$UPDATETABLE(STA, EXP, OT)$: fill the new rows with information from S
$CONSISTENT(A, S)$: checks that all strings in S are correctly classified by A

BUILDAUTOGOLD(STA, EXP, OT)

```
input  : An observation table (STA, EXP, OT)
output: A DFA A = (Σ, Q, q_λ, F_a, F_r, δ)
Q ← {q_u | u ∈ RED};
for q_u ∈ Q do
    if OT[u][λ] = 1 then Add q_u to F_a else if OT[u][λ] = 0
      then Add q_u to F_r else Add q_u to either F_a or F_r
end
for q_u ∈ Q do
    for a ∈ Σ do
        if ua ∈ RED then
            δ(q_u, a) ← q_ua
        else
            Choose v ∈ RED such that v ∼_OT ua;
            δ(q_u, a) ← q_v
        end
    end
end
return A
```

Properties of Gold's algorithm

- non-deterministic choices
- does not generalise at all sometimes (returns just the PTA)
- Given any sample (S^+, S^-) the algorithm
  - outputs a DFA consistent with S
  - admits a polynomial characteristic sample
  - runs in time and space polynomial in $||S||$
- Gold's algorithm identifies $\mathcal{DFA}(\Sigma)$ in CS-polynomial time
- What is a characteristic sample ?

RPNI (Regular Positive and Negative Inference)

- Gold's algorithm might just output the PTA
- RPNI starts from the PTA and greedily chooses states to merge while guaranteeing consistency with the sample
- Example: $S^+\{aaa, aaba, bba, bbaba\}$ et $S^- = \{a, bb, aab, aba\}$

RPNI basic ingredients

- *CHOOSE* chooses a blue state for possible merging
- $RPNIMERGE(A, q_r, q_b)$ merge a red state q_r with a blue state q_b and recursively merges any states reached by the same letter from q_r and q_b. A merge **fails** if an accepting and a rejecting state is merged.
- $RPNIPROMOTE(q_b, A)$ promotes a blue state q_b to red and adds all successors of q_b (**which lead to a state which can reach an accepting state**) to blue states
- Remark: In the original version the constructed DFA only contains accepting states and S^- is used to reject merges

$RPNIMERGE(A, q, q')$

input : A DFA A and two states q,q' from A
output: a boolean and A modified
if $q \in F_a$ *and* $q' \in F_r$ *or* $q' \in F_a$ *and* $q \in F_r$ **then return** false;
Add a new state q'' to A;
if $q \in F_a$ *or* $q' \in F_a$ **then** set $q'' \in F_a$;
if $q \in F_r$ *or* $q' \in F_r$ **then** set $q'' \in F_r$;
for *each occurrence of* q *(resp.* q'*) as source or target of transition* **do**
    replace q (resp. q') by q''
end
while A *contains non-det. choice with target states* q_n *and* q'_n **do**
    $b \leftarrow RPNIMERGE(A, q_n, q'_n)$;
    if *not* b **then** undo merge of q with q'; **return** false;
end
return true

RPNI algorithm

input : A Sample S
output: A DFA consistent with S
$A \leftarrow BUILDPTA(S)$; $RED \leftarrow \{q_\lambda\}$; $BLUE \leftarrow \{q_a | a \in \Sigma \cap PREF(S)\}$;
while $BLUE \neq \emptyset$ **do**
    $CHOOSE(q_b \in BLUE)$; $BLUE \leftarrow BLUE \setminus \{q_b\}$;
    for $q_r \in RED$ **do**
        $b \leftarrow RPNIMERGE(A, q_r, q_b)$;
        if b **then break** ;
    end
    if *not* b **then** $A \leftarrow RPNIPROMOTE(q_b, A)$;
end
return A

Remarks

- In this formulation branches of the PTA containing only rejecting states are not folded into the automaton A. One could do it by adding these states to BLUE but this does not correspond to the original RPNI algorithm.
- There are two non-deterministic choices: $CHOOSE(q_b \in BLUE)$ and $q_r \in RED$.
  - One can choose for example the lexlength-order of prefixes leading to states.
  - The order should be fixed from the beginning !
- How to get a characteristic sample ?
  - depends on the order states are choosen
  - for each pair of states q_u and q_v in the automaton to be learnt, where u and v are the shortest strings reaching the states and each letter $a \in \Sigma$ identify the shortest distinguishing string $w = DS(q_u, q_v)$ and add strings uw and vaw to S^+ or S^-.
- RPNI identifies $\mathcal{DFA}(\Sigma)$ in CS-polynomial time.

Other algorithms

- Evidence driven state merging
  - The order of merges is not fixed
  - Choose two states to merge and perform cascade of forced merges
  - if inconsistent, undo and choose two other states
  - Compute for each pair of red and blue states a score and choose the best one
  - A score could be the number of strings of S which would end up in the same state
- Other AI techniques: genetic programming, etc.
  - typically learn NFA

Active learning

Also called query learning.

- The learner makes queries answered by an oracle (teacher)
- Membership queries
  - Query: $w \in L$?
  - Answer: Yes/No
- Weak equivalence queries
  - Query: $L(H) = L$?
  - Answer: Yes/No
- (Strong) equivalence queries
  - Query: $L(H) = L$?
  - Answer: Yes/No plus a counterexample $w \in L \setminus H \;\cup\; H \setminus L$
- Subset queries
- etc.

Query learning

- $\mathcal{G}$ is identifiable in the limit with queries if there exists a learning algorithm A such that given any $G \in \mathcal{G}$, A returns a grammar G' equivalent to G and halts.
- Query complexity: How many queries are needed ?
- Complexity: if "everything" is polynomially bounded, then we say polynomially identifiable. Remark: The complexity depends on the size of counterexamples.
- If a class $\mathcal{L}$ contains a non empty set $L_\cap$ and n sets $L_1, \ldots, L_n$ such that $\forall i, j \in \{1, \ldots, n\}. L_i \cap L_j = L_\cap$, any algorithm using membership, weak equivalence and subset queries needs in the worst case to make $n - 1$ queries.
- $\mathcal{D}FA(\Sigma)$ can not be identified by a polynomial number of strong equivalence queries alone.

L^* algorithm

Dana Angluin. Learning regular sets from queries and counter examples. Information and Computation. 1987

people.eecs.berkeley.edu/∼dawnsong/teaching/s10/papers/angluin87.pdf

- “started” the field of query learning
- first query learning algorithm for regular languages
- introduces the MAT (minimally adequate teacher) model
  - answers membership and equivalence queries
- stochastic setting (PAC-learning) where equivalence queries are replaced by calls to a random sampling oracle

L^* algorithm

- Learn a regular language L (given by the minimal DFA A)
- Basic data structure: Observation table (similar to Gold's algorithm)
- Overview
  - find a closed and consistent observation table allowing to construct a DFA
  - submit an equivalence query with that DFA
  - use counterexample to update the table
  - use membership queries to make table closed and consistent
  - iterate
- Example

L^* algorithm

Main data structure: Observation table (STA, EXP, OT) where

- $STA \subseteq \Sigma^*$ a disjoint union of *BLUE* and *RED* states
- $BLUE = RED.\Sigma \setminus RED$
- $EXP \subseteq \Sigma^*$ is the experiment set
- A function $OT : STA \times EXP \mapsto \{0, 1, *\}$ defined as $OT[u][e] = 1$, if $ue \in L$, $OT[u][e] = 0$, if $ue \notin L$, else $*$.
- Additional properties:
  - STA is prefix-closed
  - EXP is suffix-closed
  - the table is complete if $OT[u][e]$ is always different from $*$ (it can be completed with membership queries). We suppose that it is always complete. In an implementation redundant entries are checked only once.

Key definitions

- Two rows u and v are equivalent (noted $u \equiv_{EXP} v$) if $OT[u] = OT[v]$.
- the table is closed if given any row $u \in BLUE$ there is a row $v \in RED$ such that $u \equiv_{EXP} v$
  - close table: promote $u \in BLUE$ to RED and add all ua to $BLUE$, iterate
- the table is consistent if for all $u, v \in RED$, $u \equiv_{EXP} v$ implies $ua \equiv_{EXP} va$ for all $a \in \Sigma$
  - Make table consistent: add ae to EXP if e separates ua and va

Key definitions

- if the table is closed and consistent, one can construct an automaton $H = A_{OT} = (\Sigma, Q, q_\lambda, F_a, F_r, \delta)$ from the table:
  - $Q = \{q_u \mid u \in RED \text{ and } \forall v < u.u \not\equiv_{EXP} v\}$
  - $F_a = \{q_u \mid OT[u][\lambda] = 1\}$, $F_r = \{q_u \mid OT[u][\lambda] = 0\}$
  - For all $q_u \in Q$ and $a \in \Sigma$, $\delta(q_u, a) = q_v$ for $v \equiv_{EXP} ua$
- Lemma: if STA is prefix-closed and EXP is suffix-closed, then the automaton A_{OT} is consistent with the data (i.e. $ue \in L(A_{OT})$ iff $OT[u][e] = 1$).
- Notation: $\lfloor q_u \rfloor_H = u$ and $\lfloor v \rfloor_H = \lfloor q_u \rfloor_H$ if $u = \delta^*_H(q_\lambda, v)$
  $\sigma_A(u) = 1$ if $u \in L(A)$ and $\sigma_A(u) = 0$ else.

L^* algorithm

```
input  : A regular language L (represented by min. DFA A)
output: A DFA H such that L(H) = L
(STA, EXP, OT) ← LSTARINITIALISE();
repeat
    while (STA, EXP, OT) is not closed or not consistent do
        if (STA, EXP, OT) is not closed then
            (STA, EXP, OT) ← LSTARCLOSE(STA, EXP, OT);
        if (STA, EXP, OT) is not consistent then
            (STA, EXP, OT) ← LSTARCONSISTENT(STA, EXP, OT);
    end
    H ← LSTARBUILDAUTO(STA, EXP, OT);
    ANSWER ← EQ(H);
    if ANSWER ≠ YES then
        (STA, EXP, OT) ← LSTARUSEEQ(STA, EXP, OT, ANSWER);
until ANSWER = YES;
return H;
```

s

L^* algorithm

- $LSTARINITIALISE()$: $RED = \{\lambda\}$, $BLUE = \Sigma$, $EXP = \{\lambda\}$, complete the table and make it closed if necessary
- $LSTARUSEEQ(STA, EXP, OT, ANSWER)$:
  - $ANSWER$ is a string w
  - add w and all its prefixes to RED
  - add all extensions with all $a \in \Sigma$ of new red w' to $BLUE$ if not in RED already
  - complete the table with membership queries

Properties of L^*

- Let H be the automaton constructed by $LSTARBUILDAUTO(STA, EXP, OT)$ with n states. Any automaton consistent with OT with n or less states is isomorphic to H.
- L^* terminates
- Any automaton consistent with an observation table (STA, EXP, OT) with n distinct rows must have at least n states.
- Let k be the size of the alphabet. Let n be the number of states of the minimal complete automaton of the language to be learnt, m the size of the biggest counter example returned.
  - $|RED| \leq n + m(n-1)$
  - $|STA| \leq (k+1)(n + m(n-1))$
  - Therefore, there are at most $(k+1)(n + m(n-1))n$ entries in the table
  - Strings are of size at most $m + 2n - 1$
  - Furthermore, there are at most $n-1$ equivalence queries and at most $O(k * n^2 * m)$ membership queries.

Variations of L^*

- [Maler/Pnueli 95] Handling of a counterexample w: add all suffixes of it to E. This insures that *RED* contains always distinct rows and consistency is not needed anymore (the table is always consistent by construction).
- [Rivest/Shapire 93] Handling of the counterexample
  - find a point in the counterexample $w = uav$ where the state (string) reached in H by ua is different from the one reached in H by u followed by a.
  - Formally: We have $\sigma_A(\lfloor\lambda\rfloor_H w) \neq \sigma_A(\lfloor w \rfloor_H)$. Therefore, there must be $a \in \Sigma$, $u, v \in \Sigma^*$ with $uav = w$ such that $\sigma_A(\lfloor u \rfloor_H av) \neq \sigma_A(\lfloor ua \rfloor_H v)$
  - search this **breaking point** in a binary way using membership queries
  - reduces the complexity from m to $log(m)$
  - but EXP is not suffix-closed anymore
    - the constructed automaton is not necessarily consistent with the data in the table ! Is this a problem ?
- [Kearns/Vazirani 94] Use of discrimination trees instead of observation table. See TTT algorithm.

TTT

Howar et al. The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning. RV 2014.

learnlib.de/wp-content/uploads/2017/10/ttt.pdf

- Experiments are organised in a discrimination tree (DT) instead of an observation table:
  - rooted binary tree, inner nodes are labelled by strings $v \in EXP$, the two children labelled by 0 (left) and 1 (right).
  - leaves are labeled by strings corresponding to states of the hypothesis automaton
  - $SIFT(u)$ into a tree: if leaf then return the label (state), else if node labelled by v check if $uv \in L$ then branch right else branch left
- Key steps:
  - Initialising the DT
  - Hypothesis construction
  - Hypothesis refinement
  - Hypothesis stabilisation
  - Discriminator finalisation

Key steps

- Initialising the DT: start with root labeled by λ and two children, the left or right leaf is labeled by λ depending on if $\lambda \in L$ or not.
- Hypothesis construction:
  - States of the automaton are the leaves
  - Transitions are determined by sifting: From u, there is a transition with a to the state given by $SIFT(ua)$
  - Accepting states are the ones in the left subtree of the root, rejecting states the others
- Hypothesis refinement:
  - given counterexample w use Rivest/Shapire's method to find $uav = w$ such that $\sigma_A(\lfloor u \rfloor_H av) \neq \sigma_A(\lfloor ua \rfloor_H v)$
  - $\lfloor ua \rfloor_H$ and $\lfloor u \rfloor_H a$ need to be split. Add new state $\lfloor u \rfloor_H a$ by adding v in EXP.
- Hypothesis stabilisation:
  - Check if hypothesis does not contradict information in the discrimination tree.
- Discriminator finalisation:

Learning symbolic automata

Drews and D'Antoni. Learning symbolic automata. TACAS 2017.
https://pages.cs.wisc.edu/$\sim$loris/papers/tacas17learning.pdf
Argyros and D'Antoni. The learnability of symbolic automata. CAV 2018.
pages.cs.wisc.edu/ loris/papers/cav18-learning.pdf

Symbolic Finite Automata

- Instead of letters from a finite alphabet, transitions are labeled by a formula from an effective boolean algebra $\mathcal{B}$
- Boolean algebra: $\mathcal{B} = (\mathcal{D}, \Psi, [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$
  - $\mathcal{D}$: a set of domain elements
  - Ψ: a set of predicates closed under boolean connectives with $\bot, \top \in \Psi$
  - $[\![\_]\!] : \Psi \to 2^{\mathcal{D}}$ a denotation function such that
    - $[\![\bot]\!] = \emptyset$
    - $[\![\top]\!] = \mathcal{D}$
    - for all $\psi, \phi \in \Psi . [\![\psi \vee \phi]\!] = [\![\psi]\!] \cup [\![\phi]\!]$ and $[\![\psi \cap \phi]\!] = [\![\psi]\!] \cap [\![\phi]\!]$ and $[\![\neg\psi]\!] = \mathcal{D} \setminus [\![\psi]\!]$
- Example: The equality algebra over some domain $\mathcal{D}$. Basic predicates are all formulas of the form $x = a$ where $a \in \mathcal{D}$. The set of all predicates is obtained by boolean combinations of these basic predicates. (Exercice: Show that predicates can be transformed into a simple normal form)

Symbolic Automata

A s-FA A is a tuple $(\mathcal{B}, Q, q_\lambda, F, \delta)$ with

- $\mathcal{B}$ a boolean algebra (called the alphabet)
- Q a finite set of states with $q_\lambda \in Q$ the initial state
- $F \subseteq Q$ the set of final states
- $\delta \subseteq Q \times \Psi_\mathcal{B} \times Q$ the transition relation containing a finite set of transitions

Symbolic automata

$A = (\mathcal{B}, Q, q_\lambda, F, \delta)$

- characters are elements of $\mathcal{D}_\mathcal{B}$
- words (strings) are elements of $\mathcal{D}_\mathcal{B}^*$
- A move $\rho = (q_1, \phi, q_2) \in \delta$ (written also $q_1 \xrightarrow{\phi} q_2$) is a transition from source state q_1 to target state q_2 where ϕ is the guard (or predicate) of the move. For a character $a \in \mathcal{D}_\mathcal{B}$, an a-move of A is a move $q_1 \xrightarrow{\phi} q_2$ such that $a \in [\![\phi]\!]$
- A is deterministic if for all transitions (q, ϕ, q_1) and $(q, \phi, q_2) \in \delta$, $q_1 \neq q_2$ implies $[\![\phi \wedge \psi]\!] = \emptyset$
- A is complete if for each character a there is an a-move out of each q.
- Exercice: Define the language of an s-FA.
- Theorem: Symbolic automata can be determinised, completed, minimized.

(Active) learning of symbolic automata

- Obviously, to learn a s-FA, one must be able to learn a formula of the boolean algebra
- Exercice: Give a (polynomial) active learning algorithm for the equality algebra. Hint: use only equivalence queries. What is its query complexity ?
- The automata learning algorithm uses as a blackbox an active learning algorithm Λ for the underlying boolean algebra

The MAT^* algorithm

input : $\mathcal{O}$: Membership oracle, $\mathcal{E}$: Equivalence oracle: Λ: algebra learning algorithm
output: A s-FA H
$T \leftarrow InitialiseDiscriminationTree(\mathcal{O})$;
$S_\Lambda \leftarrow InitialiseGuardLearners(T, \Lambda)$;
$H \leftarrow GetSFAModel(T, S_\Lambda, \mathcal{O})$;
while $\mathcal{E}(H)$ *does not succeed* **do**
  $w \leftarrow GetCounterexample(H)$;
  $T, S_\Lambda \leftarrow ProcessCounterexample(T, S_\Lambda, w, \mathcal{O})$;
  $H \leftarrow GetSFAModel(T, S_\Lambda, \mathcal{O})$;
end
return H;

Discrimination Tree (Recall)

- Experiments are organised in a discrimination tree (DT) instead of an observation table.
- rooted binary tree, inner nodes are labelled by strings $w' \in \mathcal{D}_{\mathcal{B}}^*$, the two children labelled by 0 (left) and 1 (right).
- leaves are labeled by strings $s \in \mathcal{D}_{\mathcal{B}}^*$ corresponding to states of the hypothesis automaton
- Main operation: $SIFT(w)$ into a tree starting from root: if at leaf then return the label (state), else if at a node labelled by w' then according to $\mathcal{O}(ww')$ branch right or branch left
- Initially, we have a root labelled by λ and a leaf labelled by λ according to $\mathcal{O}(\lambda)$. The other leaf is left unknown.
- When the other leaf is requested by the membership oracle (a string is sifted going to the corresponding branch) we add this string as leaf.

GetSFAModel$(T, S_\Lambda, \mathcal{O})$: Building a s-FA Hypothesis

- We start with an automaton with as many states as leaves of the discrimination tree.
- To obtain the guards of each transition, for each pair of states q_u and q_v we start a learner Λ^{q_u,q_v}
- If the learner Λ^{q_u,q_v} asks a membership query a, it is answered by sifting ua into the tree (if the result is v then yes else no).
  Special case (once typically at the beginning of the algorithm): if the discrimination tree is extended with a leaf, then we restart building an hypothesis with one more state.
- if the learner Λ^{q_u,q_v} asks an equivalence query it is suspended
- When all Λ learners are suspended we have to check that the resulting automaton is
  - deterministic
  - complete

Checking the hypothesis automaton

- Determinism: For each state q_u in the hypothesis automaton and each pair of moves $(q_u, \phi_1, q_v), (q_u, \phi_2, q_{v'})$ we verify $[\![\phi_1 \wedge \phi_2]\!] = \emptyset$. If there is a character a such that $a \in [\![\phi_1 \wedge \phi_2]\!]$, then let $m = SIFT(ua)$. Then, a must satisfy the guard of $u \rightarrow m$. Therefore, if $m = v$ (resp. $m = v'$) then we provide a as counterexample to the learner Λ^{q_u, q_v} (resp. $\Lambda^{q_u, q_{v'}}$)
- Completeness: For each state q_u in the hypothesis automaton let $S = \{\phi \mid (q_u, \phi, q') \in \delta_H\}$. We check that $[\![\bigvee_{\phi \in S} \phi]\!] = \mathcal{D}$. If a character $a \notin [\![\bigvee_{\phi \in S} \phi]\!]$ is found, let $v = sift(ua)$. a is provided as counterexample to Λ^{q_u, q_v}
- These two check are iterated until a deterministic and complete automaton is found.
- This automaton can then be submitted to the equivalence oracle.

Processing the counterexample

- Like Rivest/Shapire: Find a breaking point in the counterexample $w = uaw'$ where the state (string) reached in H by ua can be distinguished from the one reached in H by u followed by a, i.e. $\mathcal{O}(\lfloor u \rfloor_H aw') \neq \mathcal{O}(\lfloor ua \rfloor_H w')$.
- Contrary to the DFA case, here a counterexample does not always lead to a new state. It could be also that a guard is wrong.
- Let $u' = \lfloor u \rfloor_H$. Let q_v be the result of $sift(u'a)$. Consider transition $(q_{u'}, \phi, q_v)$.
- $a \notin [\![\phi]\!]$: That means that the guard is incorrect. We give a as a counterexample to the learner $\Lambda^{q_{u'}, q_v}$.
- $a \in [\![\phi]\!]$: We replace the leaf labelled by v in the discrimination tree by a subtree with a node w' and two leaves labeled by the states v and $u'a$ based on the results of $\mathcal{O}(v)$ and $\mathcal{O}(u'a)$ which are different.
- In the last case, all transitions directed to v might be wrong. We start fresh instances of the algebra learning algorithm for all these transitions as well as the new ones from and to $q_{u'a}$.

Properties of the MAT^* algorithm, Remarks

- If a state has several outgoing transitions, it might be that the learner learns first just one transition with the disjunction of all guards
- Let $(\mathcal{B}, Q, q_\lambda, F, \delta)$ and s-FA, Λ a learning algorithm for $\mathcal{B}$ and k be the maximum size that a predicate guard may take in any intermediate hypothesis.
- MAT^* learns A using $O(|Q|^2|\delta|C_m^\Lambda(k) + |Q|^2|\delta|C_e^\Lambda(k)log(m))$ membership and $O(|Q||\delta|C_e^\Lambda(k))$ equivalence queries where m is the length of the longest counterexample and $C_m^\Lambda(k)$ (resp. $C_e^\Lambda(k)$) are the number of membership (resp. equivalence) queries needed by the Λ learner to learn concepts of size k.

Learning Alternating Automata

Angluin et al. Learning Regular Languages via Alternating Automata. IJCAI 2015 www.cs.bgu.ac.il/~dana/documents/AEF_IJCAI15.pdf
generalises
Bollig, Habermehl, Kern, Leucker. Angluin-style learning of NFA. IJCAI 2009. www.ijcai.org/Proceedings/09/Papers/170.pdf
revisited by
Berndt et al. Learning residual alternating automata. Information and Computation 289 (2022)
www.sciencedirect.com/science/article/abs/pii/S0890540122001365

Alternating automata

- For a set S, $\mathcal{F}(S)$ denotes the set of all formulas over S with binary operators $\vee$, $\wedge$ and $\top$, $\bot$.
- Restrictions: $\mathcal{F}_\vee$ (only $\vee$ and $\top$) and $\mathcal{F}_\wedge$ (only $\wedge$ and $\bot$).
- An alternating automata AFA is a tuple $(\Sigma, Q, Q_0, F, \delta)$:
  - Σ: finite alphabet
  - Q: finite set of states
  - $Q_0 \in \mathcal{F}(Q)$: initial condition
  - $F \subset Q$: final states
  - $\delta : Q \times \Sigma \to \mathcal{F}(Q)$: transition function
- Special cases:
  - DFA: $Q_0 = q_\lambda$ and δ restricted to Q
  - NFA: Q_0 and δ restricted to $\mathcal{F}_\vee$
  - UFA (universal): Q_0 and δ restricted to $\mathcal{F}_\wedge$
- A transition $\delta(q, a)$ can be a nested formula. One can consider just formulas in DNF.

Alternating automata

- δ is extended to words $w \in \Sigma^*$ and formulas $\varphi \in \mathcal{F}(Q)$ in DNF ($\varphi = \bigvee_i M_i$ and $M_i = \bigwedge_j q_{i,j}$) by
  - $\delta(\varphi, \lambda) = \varphi$
  - $\delta(\varphi, a) = \bigvee_i \bigwedge_j \delta(q_{i,j}, a)$ for $a \in \Sigma$
  - $\delta(\varphi, wa) = \delta(\delta(\varphi, w), a)$ for $a \in \Sigma$ and $w \in \Sigma^*$
- The evaluation of a formula is defined by
  - $[\![\top]\!] = \top$, $[\![\bot]\!] = \bot$
  - $[\![q]\!] = \begin{cases} \top & \text{if } q \in F \\ \bot & \text{else} \end{cases}$
  - $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \vee [\![\psi]\!]$ and $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \wedge [\![\psi]\!]$
- $w \in \Sigma^*$ is accepted by an AFA if $[\![\delta(Q_0, w)]\!] = \top$.
- The language $L(A)$ is $\{w \in \Sigma^* \mid [\![\delta(Q_0, w)]\!] = \top\}$
- Given an AFA $A = (\Sigma, Q, Q_0, F, \delta)$, we write A_q for $A = (\Sigma, Q, q, F, \delta)$,

Residuality

Denis et al. Residual finite-state automata. STACS 2001.
link.springer.com/chapter/10.1007/3-540-44693-1_13

- Given a language $L \in \Sigma^*$ a residual language is a language $w^{-1}.L$ for some $w \in \Sigma^*$.
- An automaton $A = (\Sigma, Q, Q_0, F, \delta)$ is called residual, if for all $q \in Q$, $L(A_q)$ is a residual language of $L(A)$.
- RNFA, RUFA, RAFA are the residual restrictions of NFA, UFA, AFA.
- All DFA are trivially residual.
- RNFA and RUFA admit canonical minimal representatives.

Exercises and remarks

- Let $L_n = (a+b)^* a (a+b)^n$
- Exercise: Give an NFA with $n+2$ states for L_n.
- Exercise: How many states a DFA for L_n has at least ?
- Exercise: Give an UFA with $O(n)$ states for L_n.
- Let $L'_n = \{uwv\$w \mid u, v \in \{a,b\}^* \text{ and } w \in \{a,b\}^n\}$.
- Exercise: How many states an NFA for L'_n has at least ?
- Exercise: How many states a DFA for L'_n has at least ?
- Exercise: Give an AFA with $O(n)$ states recognizing L'_n
- NFA and UFA can be exponentially more succinct than DFA
- AFA can be double-exponentially more succinct than DFA
- AFA can be exponentially more succinct than NFA and UFA

Learning Alternating automata: AL^*

generalizes L^*. Specialised versions NL^* and UL^*.

- Main idea: Rows in the observation table can be composed by boolean operations to obtain other rows.
- Remember: in L^*, a table is closed, if for all BLUE rows there exists an equivalent (i.e.having the same entries) RED row.
- Generalisation of closedness: It is enough that each BLUE row can be obtained by boolean operations on RED rows.
- A row r of an observation table can be seen as vector over the binary alphabet $\{0, 1\}$ with the size of the experiment set as dimension
- We define $\sqcup$ and $\sqcap$ operations on rows as the extension of $\wedge$ and $\vee$ on vectors.
- Let R be a set of rows. For a formula $\varphi \in \mathcal{F}(R)$ we define its evaluation $[\![\varphi]\!]$ in the usual way using $\sqcup$ and $\sqcap$
- The set $P \subseteq R$ is a $(\sqcup, \sqcap)$-basis for R if $R \subseteq [\![\mathcal{F}(P)]\!]$
- A basis P is minimal if no set $P \setminus \{p\}$ is a basis. A minimal basis is not necessarily unique !

Learning Alternating automata

- An observation table (STA, EXP, OT) (with STA a disjoint union of *RED* an *BLUE*) is P-closed for a $P \subseteq RED$, if P is a basis for STA.
- Given $v \in EXP$. $M^P(v) := \bigwedge_{p \in P, p[v]=1} p$
- Given a row $r \in STA$. $b^P(r) := \bigvee_{v \in EXP, r[v]=1} M^P(v)$
- Notice, $[\![b^P(r)]\!] = r$ for $r \in P$.
- Given a P-closed observation table, one can construct an alternating automaton $(\Sigma, Q, Q_0, F, \delta)$
  - $Q = P$
  - $Q_0 = b^P(r_\lambda)$
  - $F = \{r \in P \mid r[\lambda] = 1\}$
  - For all $a \in \Sigma$ et $r \in Q$, $\delta(r, a) = b^P(ra)$

Learning algorithm AL^*

input : $\mathcal{O}$: Membership oracle, $\mathcal{E}$: Equivalence oracle, a language L
output: An AFA H such that $L(H) = L$

$(STA, EXP, OT) \leftarrow INITIALISE();$
while *true* **do**
    $P \leftarrow RED$;
    while (STA, EXP, OT) *is not P-closed* **do**
        find a row $r \in BLUE$ with $r \notin [\![\mathcal{F}(P)]\!]$;
        add ua to RED and P; complete table using $\mathcal{O}$; $P \leftarrow RED$
    end
    construct a minimal basis P and AFA H for P; check with $\mathcal{E}$;
    if *ok* **then return** H;
    else
        get a counterexample w; add all suffixes of w to EXP;
        complete table using $\mathcal{O}$;
    end
end

Remarks

- The way the counterexample is analysed guarantees that the algorithm stops. Each counterexample will add at least one different column.
- The status of rows might switch during the algorithm between being in the basis or not, as more information becomes available.
- A minimal basis is not necessarily of minimal size. However, it can be obtained easily greedily.
- Computing a basis of minimal size is NP-complete
- One can use approximation algorithms to obtain a basis of almost minimal size in polynomial time
- One obtains variants UL^*, NL^* by restricting the formulas to conjunctions (resp. disjonctions)
  - in this case, it is easy to obtain basis of minimal size.
- The resulting automaton is not necessarily a RAFA. The algorithm can be changed for that.