Outline

- Learning Regular Languages
  - Passive
  - Active
- Beyond Language Learning
  - Refine Black-Box Test Suites
  - Learn Small MDP Strategies
Learning Regular Languages
Finite Automaton

- **Automaton** \( A = \langle \Sigma, Q, q_0, \delta, F \rangle \)
  - Alphabet \( \Sigma \)
  - Finite set of states \( Q \)
  - Initial state \( q_0 \in Q \)
  - Transition relation \( \delta : Q \times \Sigma \rightarrow X \)
    - Deterministic \( X = Q \)
    - Non-deterministic \( X = 2^Q \)
    - Probabilistic \( X = \mathcal{D}(Q) \)
  - Accepting states \( F \)
From Automaton to Language

- Extend transition relation
  - $\hat{\delta}(\epsilon) := q_0$
  - $\hat{\delta}(wa) := \delta(w, a)$

- Language $\mathcal{L}(A) := \{ w \in \Sigma^* \mid \hat{\delta}(w) \in F \}$

- $\mathcal{L}(A)$ is a regular language
From Language to Automaton

- Given a regular language \( \mathcal{L} \)
- Nerode relation
  - \( u \equiv_{\mathcal{L}} v \) iff for all \( w \in \Sigma^* : uw \in \mathcal{L} \iff vw \in \mathcal{L} \)
- Canonical DFA \( \langle \Sigma, Q_{\mathcal{L}}, q_{0\mathcal{L}}, \delta_{\mathcal{L}}, F_{\mathcal{L}} \rangle \)
  - \( Q_{\mathcal{L}} = \Sigma^*/\equiv_{\mathcal{L}} \)
  - \( q_{0\mathcal{L}} = [\varepsilon] \)
  - \( \delta_{\mathcal{L}} \) defined by \( \delta_{\mathcal{L}}([u], a) = [ua] \)
  - \( F_{\mathcal{L}} = \{ [u] \mid u \in \mathcal{L} \} \)
Language Learning Problem

- Finite sets of example strings $\mathcal{X} \subseteq \Sigma^*$
- Binary labeling $f : \mathcal{X} \rightarrow \{0, 1\}$
- Construct $n$-state DFA $A$, such that
  - $\mathcal{X}^1 := \{x \mid f(x) = 1\} \subseteq \mathcal{L}(A)$ and
  - $\mathcal{X}^0 := \{x \mid f(x) = 0\} \subseteq \Sigma^* \setminus \mathcal{L}(A)$
- Active vs passive learning
- Exact vs approximate
- Minimal vs near minimal
Complexity

- **Passive Learning NP - complete**
  - Exact & Approximate
  - Minimal & Polynomial in minimal
  - Polynomial if input strings are complete up to length $n$
    - Input is exponential
- **Active Learning polynomial**
  - Depends on teacher!
Passive Learning

- No teacher
- Just labeled samples
k- tails Algorithm

- **k- equivalence relation**
  \[ u \equiv_k v \text{ iff for all } w \in \Sigma^k : uw \in \mathcal{X} \iff vw \in \mathcal{X} \text{ and} \]
  \[ \text{if } uw, vw \in \mathcal{X} : f(uw) = f(vw) \]

- Induces (non-)deterministic automaton
- Solves language learning problem for minimal \( n \), if \( k \) is greater than longest string in \( \mathcal{X} \)
- Often called Biermann‘s Algorithm
k-tails Algorithm as CSP

- Solution of language learning problem as a solution of the following CSP
- For each $u \in \mathcal{X}$ introduce variable $S_u$
  - $S_u \neq S_{u'} \ | \ u \in \mathcal{X}^0, u' \in \mathcal{X}^1$
  - $S_u = S_{u'} \Rightarrow S_{ua} = S_{u'a} \ | \ ua, u'a \in \mathcal{X}$
- Solve CSP over $\{1, \ldots, k\}$
Learning Probabilistic Automata

- Input is multiset of positive strings
- Replace $k$ equivalence by similarity
- Two nodes in the PFA $u$ and $v$ are $(0-)similar if
  \[ \left| \frac{m_u(\sigma)}{m_u} - \frac{m_v(\sigma)}{m_v} \right| \leq \sqrt{\frac{1}{2} \log \frac{1}{\epsilon}} \left( \frac{1}{\sqrt{m_u}} + \frac{1}{\sqrt{m_v}} \right) \]
  - $m_u$ ... number of strings arriving at $u$
  - $m_u(\sigma)$ ... number of strings leaving $u$ via $\sigma$
- PFA not efficiently learnable in general unless noisy parity is efficiently learnable
Mining Specifications

- Verification relies on the existence of specifications
- Manually writing specifications is complicated and error prone
- Learn specifications from runtime traces
  - Specification as PFA
  - Filter out „hot core“
  - Learn with similarity version of k-tails Algorithm
Mining Specifications

Program -> Tracer -> Instrumented program

Test inputs -> Run -> Traces

Run -> Flow dependence annotator

Flow dependence annotator -> Annotated traces

Scenario seeds -> Scenario extractor

Scenario extractor -> Abstract scenario strings

Scenario extractor -> Automaton learner

Automaton learner -> Specification
Example

A buggy program using the socket API

```c
1 int s = socket(AF_INET, SOCK_STREAM, 0);
2 ...
3 bind(s, &serv_addr, sizeof(serv_addr));
4 ...
5 listen(s, 5);
6 ...
7 while(1) {
8     int ns = accept(s, &addr, &len);
9     if (ns < 0) break;
10    do {
11        read(ns, buffer, 255);
12        ...
13        write(ns, buffer, size);
14        if (cond1) return;  
15    } while (cond2)
16    close(ns);
17 }
18 close(s);
```
Example

Instrumented call to socket

```c
int instrumented_socket(int domain,
                         int type,
                         int proto)
{
    int rc = socket(domain, type, proto);
    fprintf(the_trace_fp,
            "socket(domain = %d, type = %d, "
            "proto = %d, return = %d)\n",
            domain, type, proto, rc);
    return rc;
}
```
Example

One extracted scenario

1 socket (domain = 2, type = 1, proto = 0, return = 7)
2 bind (so = 7, addr = 0x400120, addr_len = 6, return = 0)
3 listen (so = 7, backlog = 5, return = 0)
4 accept (so = 7, addr = 0x400200, addr_len = 0x400240, return = 8) [seed]
5 read (fd = 8, buf = 0x400320, len = 255, return = 12)
6 write (fd = 8, buf = 0x400320, len = 12, return = 12)
7 read (fd = 8, buf = 0x400320, len = 255, return = 7)
8 write (fd = 8, buf = 0x400320, len = 7, return = 7)
9 close (fd = 8, return = 0)
Simplified scenario

1 socket(return = 7)
2 bind(so = 7)
3 listen(so = 7)
4 accept(so = 7, return = 8) [seed]
5 read(fd = 8)
6 write(fd = 8)
7 read(fd = 8)
8 write(fd = 8)
9 close(fd = 8)
Example

Corresponding scenario string

1. socket (return = x0:T0)  (A)
2. bind (so = x0:T0)  (B)
3. listen (so = x0:T0)  (C)
4. accept (so = x0:T0, return = x1:T0) [seed]  (D)
5. read (fd = x1:T0)  (E)
6. write (fd = x1:T0)  (F)
7. read (fd = x1:T0)  (E)
8. write (fd = x1:T0)  (F)
9. close (fd = x1:T0)  (G)
Example

Final specification

```c
1 int s = socket(AF_INET, SOCK_STREAM, 0);
2 ...  
3 bind(s, &serv_addr, sizeof(serv_addr));  
4 ...  
5 listen(s, 5);
6 ...  
7 while(1) {
8   int ns = accept(s, &addr, &len);
9   if (ns < 0) break;
10  do {
11     read(ns, buffer, 255);
12     ... 
13     write(ns, buffer, size);
14     if (cond1) return;  
15  } while (cond2)
16  close(ns);
17 }  
18 close(s);
```
Active Learning

- Provide a teacher
- Angluin’s minimally adequate teacher
  - Membership query
    - „What is the label of x?“
  - Equivalence query
    - „Is my answer equivalent to the correct one?“
Angluin’s Algorithm L* 

- $U$... prefix closed set „states“
- $V$... suffix closed set „distinguishing suffixes“
- $T : (U \cup U \circ \Sigma) \circ V \rightarrow \{0, 1\}$
- $row(s) : V \rightarrow \{0, 1\}; row(s)(e) = T(se)$
- Closed $\forall t \in U \circ \Sigma \exists s \in V : row(t) = row(s)$
- Consistent $\forall s_1, s_2 \in U : row(s_1) = row(s_2) \Rightarrow \forall a \in \Sigma : row(s_1a) = row(s_2a)$
- Runs in polynomial time in number of states & longest counterexample
Angluin’s Algorithm $L^*$

**Function** Angluin()

initialize $(U, V, T)$

repeat

\[ \textbf{while not(isClosed}((U, V, T)) \textbf{ or not(isConsistent}((U, V, T)) \textbf{) then} \]

\[ \text{if not(isConsistent}((U, V, T)) \text{ then} \]

\[ \text{find } a \in \Gamma, v \in V \text{ and } u, u' \in U \text{ such that} \]

\[ T(u) = T(u') \text{ and } T(ua)(v) \neq T(u'a)(v) \]

\[ \text{add } av \text{ to } V \]

\[ \text{for every } u \in U \cup U \Gamma \]

\[ \text{ask membership query for } uav \]

\[ \text{if not(isClosed}((U, V, T)) \text{ then} \]

\[ \text{find } u \in U, a \in \Gamma \text{ such that } T(ua) \neq T(u') \text{ for all } u' \in U \]

\[ \text{move } ua \text{ to } U \]

\[ \text{for every } a' \in \Gamma \text{ and } v \in V \]

\[ \text{ask membership query for } u'aav \]

\[ \text{construct hypothesized automaton } \mathcal{H} \]

\[ \text{do an equivalence query with hypothesis } \mathcal{H} \]

\[ \text{if the answer is a counterexample } u \text{ then} \]

\[ \text{add every prefix } u' \text{ of } u \text{ to } U. \]

\[ \text{for every } a \in \Gamma, v \in V \text{ and prefix } u' \text{ of } u \]

\[ \text{ask membership query for } u'v \text{ and } u'av. \]

\[ \text{until the answer is 'yes' to the hypothesis } \mathcal{H} \]

Output $\mathcal{H}$. 
Extensions & Applications

- **Approximate Learner**
  - Replace equivalence test by random sampling oracle
  - Make $\frac{1}{\epsilon} \left( \log \frac{1}{\delta} + \log 2 \times (i + 1) \right)$ calls
  - Obtain $\epsilon$-Approximation with probability $1 - \delta$

- **Rivest et al. 1993**:  
  - Search for counterexample that is a witness
  - Removes consistency check requirement

- **Learn context-free grammars**
- **Learning timed systems**
Learning Assumptions for Compositional Verification

- Labeled transition systems $M_1, M_2$
- Parallel Composition $M_1 \parallel M_2$
  - Synchronous & asynchronous actions
- Safety property $P$
  - LTS completed with error state
- Verify $M_1 \parallel M_2 \models P$
  - Do not want to compute $M_1 \parallel M_2$
Example

Verify: $Input \parallel Output \models Order$
Assume-guarantee Reasoning

- Assumption $A$
- Component $M$
- Property $P$

$\langle A \rangle M \langle P \rangle$ true if
- Whenever $M$ is part of a system that satisfies $A$
- $P$ is also satisfied
- Is error state reachable in $A \parallel M \parallel P_{err}$

Proof rule

\[
\begin{align*}
\langle A \rangle M_1 \langle P \rangle \\
\langle \text{true} \rangle M_2 \langle A \rangle \\
\hline
\langle \text{true} \rangle M_1 \parallel M_2 \langle P \rangle
\end{align*}
\]
Learning the right Assumption

Membership queries by simulating words on $M_1 \parallel P_{err}$

Conjecture test by model checking twice

Simulate counterexample on $M_1 \parallel P_{err}$
Example

Verify: \( \text{Input} \parallel \text{Output} \models \text{Order} \)

Construct Assumption \( A \)

\[ \langle A \rangle \text{Input} \langle Order \rangle \text{ and } \langle \text{true} \rangle \text{Output} \langle A \rangle \]

\[ \Sigma = \{ \text{send}, \text{ack}, \text{output} \} \]
Example

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$T_1$</th>
<th>$E_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>$\lambda$</td>
<td>true</td>
</tr>
<tr>
<td>output</td>
<td></td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S_1 \cdot \Sigma$</th>
<th>$\lambda$</th>
<th>$E_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ack</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>output</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>send</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>output, ack</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>output, output</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>output, send</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

Assumption A1

Counter example: (input, send, ack, input)
Example

<table>
<thead>
<tr>
<th>$S_2$</th>
<th>$T_2$</th>
<th>$E_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>$\lambda$</td>
<td>true</td>
</tr>
<tr>
<td>output</td>
<td></td>
<td>true</td>
</tr>
<tr>
<td>send</td>
<td></td>
<td>false</td>
</tr>
</tbody>
</table>

Assumption A2

![Diagram](image-url)
Example

Assumption A2

\[ \langle A_2 \rangle \text{Input} \langle Order \rangle \]
\[ \langle \text{true} \rangle \text{Output} \langle A_2 \rangle \]

\[ \text{Input} \parallel \text{Output} \models \text{Order} \]
Beyond Language Learning
Refine Black-Box Test Suites

- Given a Black-Box program
- Analyse given Test Suite for the program
  - Redundant?
  - Covering?
- Understand and refine Test Suite
  - Category Partition
  - Decision Trees
Triangle program

- Input: sides (a,b,c)
- Output:
  - Equilateral Triangle (1,1,1)
  - Isosceles Triangle (2,2,1)
  - Irregular Triangle (1,2,3)
  - No Triangle (1,1,3)
  - Illegal Values (-1,0,’x‘)
Category Partition

- **Parameters & environment conditions**
  - Specify the behavior of the program
  - E.g. a, b, c

- **Categories**
  - Characterize parameter
  - E.g. a compares to b and c

- **Choices**
  - Partitions the categories
  - E.g. \( a \leq b + c \) / \( a > b + c \)
(a=1, b=2, c=2) abstracted to
(a <= b + c, b = c, isosceles)

Enter abstract tests in Decision Tree

```
1  (a vs. b) = a! = b
2   | (c vs. a+b) = c <= a+b
3   |   | (a vs. b+c) = a <= b+c
4   |   |   | (b vs. a+c) = b <= a+c
5   |   |   |   | (b vs. c) = b = c
6   |   |   |   |   | (a) = a > 0: Isosceles (22.0)
```

- Are tests covering?
- Are tests redundant?
Learn Small MDP Strategies

- Markov Decision Process
  - States $S$
  - Actions $A$
  - Non-determinism
  - Probabilistic transitions
- Reachability objective $P \subseteq S$
- Strategy
  - Function $S \rightarrow A$
  - Maximum reachability probability of $P$
Learn Small MDP Strategies

- Markov Decision Process
  - States $S$
  - Actions $A$
  - Non-determinism
  - Probabilistic transitions
- Reachability objective $P \subseteq S$
- Strategy
  - Function $S \rightarrow A$
  - $\epsilon$-optimal reachability probability of $P$
Learn Small MDP Strategies

- $\varepsilon$-optimal strategy can be obtained in polynomial time through Value Iteration
- MDP models are huge
  - mer mutual exclusion protocol
  - $\sim 10\ 000\ 000\ 000\ 000$ states
- Infeasible to perform Value Iteration
- Strategy impossible to parse
Learning as a way out

- Learn what is interesting
Learning as a way out

- Learn what matters
Learning as a way out

- Learn what is interesting
  - Update lower and upper bounds via sampling
  - Let the probabilities guide the search

- Learn what matters
  - Classify State/Action pairs $\mathcal{X} = S \times A$
  - $\mathcal{Y} = \{0, 1\}$
  - $f(s, a) = 1$ if and only if $a$ is the best action in $s$
  - Decision Tree pruning to filter away redundant information
Mer example

- 10,000,000,000,000 states
- Strategy of size 17
- Expert can read off bug

Action graph:

1. $action = \{s1=0 & r1=0 \rightarrow r1'=2\}$
2. $action = \{\text{any action of user1}\}$ with $k \leq 0$
   - $k > 0$: $tt$
   - $k = 0$: $ff$
3. $action = \{r! = 0 & \text{driveUser}=0 \rightarrow r'=0 & gc' = \text{true}\}$
4. $action = \{s2=0 & r2=0 \rightarrow r2'=2\}$ with $tt$
5. $action = \{s2=0 & r2=2 \rightarrow s2'=1\}$ with $tt$
6. $action = \{\text{any synchronized action}\}$ with $tt$
7. $action = \{u1\_request\_comm\}$ with $ff$

Graph transitions:
- $tt$ for actions involving states 1 and 2
- $ff$ for actions involving user 1
Conclusion

- Regular sets can be learnt from examples
  - Passive
  - Active
  - Rich applications

- Decision Trees can help software engineers

- Unused potential of machine learning in verification beyond language learning?
Thank you for your Attention! Questions?
References

- Murphy, Kevin P. "Passively learning finite automata." Santa Fe Institute, 1995.