Outline

• Motivation
  – Brief History of Grammatical Inference
  – Application of Grammatical Inference
  – Potential Application in Malware Detection

• Automata and Languages
  – Background
  – Chomsky Hierarchy

• GI Algorithms
  – Learning DFA
    • RPNI
    • EDSM
    • Membership Query
  – Learning k-testable Languages

• Open Problems
RPNI Strategy:
If nothing tells me not to generalize, do it.

“The training program of an artificial intelligence can certainly include an informant, whether or not children receive negative instances.” [Gold ‘67]

Heuristic Greedy State Merging/AI Strategy:
If there are good reasons to generalize, then do it.
RPNI Strategy

• RPNI is a deterministic algorithm.
• In RPNI the order of compatibilities are checked right from the start.
• As soon as two states are mergeable, they are merged.
• RPNI is optimistic.
• Are we choosing the best merge?
• **RPNI identifies in the limit.** This may well no longer be the case if we use a heuristic to define the best possible merge.
Greedy State Merging

• The general idea of a greedy state merging (e.g., RPNI) algorithm is as follows:
  – choose two states
  – perform a cascade of forced merges until the automaton is deterministic
  – if this automaton accepts some sentences from $S_-$, backtrack and choose another couple
  – if not, loop until no merging is still possible
Greedy State Merging

• Now how are the moves chosen?
• There are two possibilities:
  – merging a BLUE with a RED
  – promoting a BLUE to RED and changing all its successors that are not RED to BLUE.
• Promotion takes place when a BLUE state can be merged with no RED state.
• Once there are no possible promotions greedily checking in order to find the first admissible merge
Heuristic Greedy State Merging

- Check all possible legal merges between a BLUE state and a RED state
- Compute a score for each merge
- Choose the merge with highest score

Evidence Driven State Merging (EDSM)
Evidence Driven State Merging (EDSM)

- The evidence driven approach consists in computing for every pair of states (one BLUE, the other RED) the score of that merge as the number of strings that end in a same state if that merge is done.
- To do that the strings from S+ and S− have to be parsed.
- If by doing that merge a conflict arises (a negative string is accepted or a positive string is rejected) the score is $-\infty$.
- The merge with the highest score is chosen.
EDSM-Count

Input: $A, S_+, S_-$
Output: the score $sc$ of $A$

for $q \in Q$ do  
  $tp[q] \leftarrow 0$ ; $tn[q] \leftarrow 0$
for $w \in S_+$ do  
  $tp[\delta_A(q, w)] \leftarrow tp[\delta_A(q, w)] + 1$
for $w \in S_-$ do  
  $tn[\delta_A(q, w)] \leftarrow tn[\delta_A(q, w)] + 1$
$sc \leftarrow 0$

for $q \in Q$ do
  if $sc \neq -\infty$ then
    if $tn[q] > 0$ then
      if $tp[q] > 0$ then $sc \leftarrow -\infty$ else $sc \leftarrow sc + tn[q] - 1$
    else
      if $tp[q] > 0$ then $sc \leftarrow sc + tp[q] - 1$
  end
end

return $sc$
EDSM

Input: \( S = \langle S_+, S_- \rangle \), functions COMPATIBLE, CHOOSE

Output: \( \mathcal{A} = \langle \Sigma, Q, q_\lambda, F_\lambda, F_R, \delta \rangle \)

\( \mathcal{A} \leftarrow \text{BUILD-PTA}(S_+); \) \( \text{RED} \leftarrow \{ q_\lambda \}; \) \( \text{BLUE} \leftarrow \{ q_a : a \in \Sigma \text{ and } S_+ \cap a \Sigma^* \neq \emptyset \}; \)

while \( \text{BLUE} \neq \emptyset \) do

\( \text{promotion} \leftarrow \text{false}; \)

for \( q_b \in \text{BLUE} \) do

\hspace{1em} if not promotion then

\hspace{2em} \( bs \leftarrow -\infty; \)

\hspace{2em} atleastonemerge \leftarrow \text{false};

\hspace{2em} for \( q_r \in \text{RED} \) do

\hspace{3em} \( s \leftarrow \text{EDSM-COUNT}(\text{MERGE}(q_r, q_b, A), S_+, S_-); \)

\hspace{3em} \text{if } s > -\infty \text{ then atleastonemerge } \leftarrow \text{true} \)

\hspace{3em} \text{if } s > bs \text{ then } bs \leftarrow s; \quad \overline{q_r} \leftarrow q_r; \quad \overline{q_b} \leftarrow q_b \)

\hspace{2em} end

\hspace{2em} if not atleastonemerge then /* no merge is possible */

\hspace{3em} \| \text{PROMOTE}(q_b, A); \) promotion \leftarrow \text{true};

\hspace{2em} end

end

end

if not promotion then /* we can merge */

\hspace{1em} \text{BLUE} \leftarrow \text{BLUE} \setminus \{ \overline{q_b} \}; \quad \mathcal{A} \leftarrow \text{MERGE}(\overline{q_r}, \overline{q_b}, A) \)

end

for \( x \in S_+ \) do \( F_\lambda \leftarrow F_\lambda \cup \{ \delta(q_\lambda, x) \}; \)

for \( x \in S_- \) do \( F_R \leftarrow F_R \cup \{ \delta(q_\lambda, x) \}; \)

return \( \mathcal{A} \)
EDSM - Example

\[ S^+ = \{a, aaa, aba, bba, abab\} \]
\[ S^- = \{ab, bb\} \]

State \( q_{ab} \) is selected and the counts are computed.
EDSM - Example

\[ S^+ = \{a, aaa, aba, bba, abab\} \]
\[ S^- = \{ab, bb\} \]

- \[ EDSM\text{-Count}(\text{Merge}(q_{\lambda}, q_{ab}, A))= -\infty, \] because this consists in merging \( q_{ab} \) with \( q_{abab} \).
- \[ EDSM\text{-Count}(\text{Merge}(q_{a}, q_{ab}, A))= -\infty, \] because this consists in merging \( q_{a} \) with \( q_{ab} \).
EDSM - Example

\[ S^+ = \{a, aaa, aba, bba, abab\} \]
\[ S^- = \{ab, bb\} \]

Therefore a promotion takes place: Since we have a BLUE which can be merged, the DFA is updated with state \( q_{ab} \) promoted to RED.
EDSM - Example

$S^+ = \{a, aaa, aba, bba, abab\}$
$S^- = \{ab, bb\}$

- $EDSM\text{-Count}(\text{Merge}(q_\lambda, q_a, A)) = 2$
- $EDSM\text{-Count}(\text{Merge}(q_a, q_b, A)) = 3$
EDSM - Exercise

• In Algorithm EDSM, the computation of the scores is very expensive. Can we combine the data driven and the evidence driven approaches in order to not have to compute all the scores but to be able to discover the promotion situations?

• In Algorithm EDSM, once $sc(q,q',A)$ is computed as $-\infty$, does it need to be recomputed? Is there any way to avoid such expensive recomputations?
Dana Angluin’s approach for DFA learning

• In this learning model the learning algorithm has access to a teacher answering the following types of queries:
  – Membership Queries: On a membership query, the learning algorithm selects a string $s$ and ask the teacher if $s$ is in the language that the learning algorithm is attempting to learn.
  – Equivalence Queries: On an equivalence query, the learning algorithm submits a DFA $M$ and asks the teacher if $M$ recognises the language under consideration. If $M$ is not correct, the teacher provides a counterexample.
Dana Angluin’s approach for DFA learning

- References:
  - Balcazar et al.
Outline

• Motivation
  – Brief History of Grammatical Inference
  – Application of Grammatical Inference
  – Potential Application in Malware Detection

• Automata and Languages
  – Background
  – Chomsky Hierarchy

• GI Algorithms
  – Learning DFA
    • RPNI
    • EDSM
    • Membership Query
  – Learning k-testable Languages

• Open Problems
Remember

• Regular languages cannot be identified from positive examples only (Gold 67);
• We have to concentrate on a sub-class of these.
K-testable

- Inference of k-Testable Languages in the Strict Sense and Application to Syntactic Pattern Recognition. García & Vidal et al. 1990
- Concept initially introduced for Pattern Recognition tasks.
K-testable

• Let $k \geq 0$, a $k$-testable machine in the strict sense ($k$-TSS) is a 5-tuple $Z_k = (\Sigma, I, F, T, C)$ with:
  - $\Sigma$: finite alphabet
  - $I, F \subseteq \Sigma^{k-1}$ (allowed prefixes of length less than $k$ and suffixes of length $k-1$) and also all strings of length less than $k$.
  - $T \subseteq \Sigma^k$ (allowed segments)
  - $C \subseteq \Sigma^{<k}$ (short strings);
K-testable

- Only admissible strings are those either corresponding exactly to strings either:
  - in C
  - or those whose prefix of length $k - 1$ in I, suffix of length $k - 1$ is in F and where all substrings of length $k$ belong to T.
K-testable - Example

- A DFA corresponding to the 3−TSS machine
  \[ Z_3 = (\{a, b\}, I = \{aa, ab\}, F = \{ab\}, C = \{a\}, T = \{aaa, aab, aba, bab\}) \]
K-testable Machine to DFA

Input: a $k-$TSS machine $\langle \Sigma, I, F, T, C \rangle$
Output: a DFA $\langle \Sigma, Q, q_\lambda, F_A, \delta \rangle$

$Q \leftarrow \emptyset$;
$F_A \leftarrow \emptyset$;

for $pu \in I \cup C$, $p, u \in \Sigma^*$ do $Q \leftarrow Q \cup \{q_p\}$;

for $au \in T$, $a \in \Sigma$, $u \in \Sigma^*$ do $Q \leftarrow Q \cup \{q_u\}$;

for $ua \in T$, $a \in \Sigma$, $u \in \Sigma^*$ do $Q \leftarrow Q \cup \{q_u\}$;

for $pau \in I \cup C$, $a \in \Sigma$, $p, u \in \Sigma^*$ do $\delta(q_p, a) = q_{pa}$;

for $aub \in T$, $a, b \in \Sigma$, $u \in \Sigma^*$ do $\delta(q_{au}, b) = q_{ub}$;

for $u \in F \cup C$ do $F_A \leftarrow F_A \cup \{q_u\}$;

return $\langle \Sigma, Q, q_\lambda, F_A, \delta \rangle$
How to learn a K-testable Machine from $S$

- Learning k-testable languages is really only about finding the prefixes, substrings and suffixes that occur in the data.
- It can be learnt from one class of sample.
K-testable Machine Learning Algorithm

Input: a sample $S$
Output: a $k-$TSS machine $\langle \Sigma, I(S), F(S), T(S), C(S) \rangle$

$\Sigma$ is the alphabet used in $S$;
$I(S) \leftarrow \Sigma^{k-1} \cap\text{PREF}(S)$;
$C(S) \leftarrow \Sigma^{<k} \cap S$;
$F(S) \leftarrow \Sigma^{k-1} \cap\text{SUFF}(S)$;
$T(S) \leftarrow \Sigma^{k} \cap \{ v : uvw \in S \}$;
return $\langle \Sigma, I(S), F(S), T(S), C(S) \rangle$
Example

• Let $S = \{a, aa, abba, abbbba\}$ be our learning sample and suppose we choose $k = 3$.
• It follows by construction that:
  • $\Sigma = \{a, b\}$
  • $I(S) = \{aa, ab\}$
  • $F(S) = \{aa, ba\}$
  • $T(S) = \{abb, bbb, bba\}$
  • $C(S) = \{a, aa\}$
Example

Automaton learnt from sample $S = \{a, \text{aa}, \text{abba, abbbba}\}$. 
Exercise

• Run $A_{k-TSS}$ for $k=1, 2, 3$ and $15$, with $S = \{ab, abab, abababab\}$.

• Compute the complexity in time and space of $A_{k-TSS}$.

• Prove that identifying the entire class of testable languages (i.e. the union for all $k$) is impossible from positive examples only.
Outline

• Motivation
  – Brief History of Grammatical Inference
  – Application of Grammatical Inference
  – Potential Application in Malware Detection

• Automata and Languages
  – Background
  – Chomsky Hierarchy

• GI Algorithms
  – Learning DFA
    • RPNI
    • EDSM
    • Membership Query
  – Learning k-testable Languages

• Open Problems
Open Problems in GI

- Dealing with noise
- Classes of languages that adequately mix Chomsky’s hierarchy with edit distance compacity
- Stochastic context-free grammars
- Polynomial learning from text
- Fast algorithms
Open Problems in GI

• Colin de la Higuera, Ten open problems in grammatical inference, 2006, Proceedings of ICGI 2006, Tokyo LNCS 4201 32-44