Grammatical inference and language frameworks for LANGSEC

Kerry N. Wood
Richard Harang
U.S. Army Research Laboratory
21 MAY 2015
1. My background is primarily in networking, modeling, and optimization. (I am not an expert in formal language theory.)

2. A natural beginning for our work was a solid literature review. This paper and presentation are a direct result of trying to summarize that work.

3. This is not comprehensive. We focused on those frameworks and techniques we thought might be useful.

We came into this knowing that inferring grammars is difficult. However, rewriting all the existing code to use LANGSEC is probably harder.
• A brief introduction and review of grammars and LANGSEC. (OR) What we think LANGSEC is / is supposed to be.

• Models of learning: definitions are important.

• Models of teaching: is data available / how is it presented?

• Pattern Languages / EFS: our two favorite formalisms.

• Some initial work: where “initial” means “small amount.”

• Conclusions and future work: hope springs eternal.
We all remember Chomsky classes and the recognizer types.

CS101 – formal production rules for ‘strings’ that have an exact mapping to different models of computation

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Power to recognize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>State machine</td>
</tr>
<tr>
<td>Context-free</td>
<td>Stack machine</td>
</tr>
<tr>
<td>Context-sensitive</td>
<td>Bounded automata</td>
</tr>
<tr>
<td>Unrestricted</td>
<td>Turing machine</td>
</tr>
</tbody>
</table>

Grammatical inference:
– You are given the output of the grammar (strings).
– You have to identify the generating structure.
– In essence, it is decompiling. Just harder.
Most codes are imperfect recognizers.

Protocols are (either explicitly or implicitly) grammars.

\[ M \]: All possible messages

\[ A \]: Messages we actually accept

\[ S \]: Messages we want to accept

Trouble begins when we mean to define a recognizer for \( S \), but actually define one for \( A \) (and don’t know it).
“LangSec” – language theoretic security

- Use only simple grammars with provable properties
- Validate all inputs before any processing; reject failures aggressively

\[ A = S \rightarrow A \setminus S = 0 \]

Right now: really only helpful if you “bake it in”

Significant restrictions on message complexity
- Deterministic CFG or weaker (or some non-Chomsky class recognizable in polynomial time)
Learning from positive examples:
- All strings in the grammar are shown to the learner \( \forall s \in \mathbb{LL} \)

Positive and negative examples (aka “complete presentation”):
- Members and non-members are labeled and shown to the learner.
  \( \forall s \in \mathbb{LL} \) and \( \forall s \in \Sigma^* - \mathbb{LL} \)

Learning with an oracle (aka “membership queries”):
- The learner presents strings, the oracle returns membership labels.

Learning with a teacher (aka “equivalence queries”):
- The learner presents guesses as to the hypothesis grammar.
- Teacher verifies correctness, or returns an example string in the language, but not in the learner-suggested grammar.

Compressed / simple examples:
- The minimum set of examples that allow for identification of the grammar.
In general: a learner $A$ takes some data $D$, and outputs a (possible) generating grammar $G$.

Learning in the limit (Gold-style)
- A learner is presented with examples from the entire grammar.
- At each iteration, it guesses a hypothesis grammar.
- At a certain iteration, the guesses never change.

Fixed-time identification
- Learner will see strings in $D$ one at a time; determines beforehand how many steps it will take to produce a correct $A(D)$.

Finite identification
- As above, but learner decides after seeing each string whether to stop

Probably Approximately Correct (PAC) learning
- Given access to examples, a finite-length grammar, and finite-length examples; with probability $(1 - \delta)$, a learner will output a hypothesis that is $\epsilon$ – good.
Inductive inference in the limit.

Ex. 1 → H. 1
Ex. 2 → H. 2
Ex. $M$ → H. $M$
Ex. $M + 1$ → H. $M$
Ex. $M$ → H. $M$
Ex. $M + 1$ → H. $M$
Pattern languages: replace variables with substrings.

- Constants and strings: $\Sigma, \Sigma^*, \Sigma^+$
- Variables: $(x_1, x_2, \ldots) \in X$
- Patterns: $P = \pi_1, \pi_2, \ldots = \Sigma^* \cup X$
- Substitution: $\theta$
  - $\pi \theta = \left[ \frac{s_1}{x_1}, \frac{s_2}{x_2}, \ldots, \frac{s_k}{x_k} \right]$ where $|s_i| \neq 0$

A pattern $\pi_i$ is regular if each variable appears at most once.
Patterns are somewhat natural and somewhat learnable.

The good:

They look just like regular expressions. *Angluin showed they are learnable.*

- Lange and Wiehagen have a simple, but *inconsistent* algorithm.

The bad:

- In general, intractable to determine membership.
  - \( \text{NP} – \text{complete} \)
- In general, they do not map to Chomsky.
  - Regular patterns encode regular languages.
- In general, relationships are hard to determine
  - Not closed under union.
- Lots of “academic” sub-classes and cases.
- PAC learnability is bad.

Examples:

\( X = \{x_1, x_2, \ldots \} \)

- \( xaybza \)
- \( xx \)
Elementary formal systems (EFS) are a logic programming analog.

\[ \{a^n b^n c^n \mid n \geq 1 \} : \]

\[ \Gamma = \begin{cases} 
\ p(x_1, x_2, x_3) &\leftarrow q(x_1, x_2, x_3) \\
q(ax_1, bx_2, cx_3) &\leftarrow q(x_1, x_2, x_3) \\
q(a, b, c) &\leftarrow 
\end{cases} \]

\[ \{a^n b^n \mid n \geq 1 \} : \]

\[ \Gamma = \begin{cases} 
\ p(ax_1 b) &\leftarrow p(x_1) \\
p(ab) &\ 
\end{cases} \]
A clause / EFS \((A \leftarrow B_1, \ldots, B_m)\) is…

- **Variable bounded**
  - \(\nu(A) \supseteq \nu(B_1) \cup \ldots \cup \nu(B_m)\)

- **Length bounded**
  - \(|A\theta| \geq |B_1\theta| + \ldots + |B_m\theta|\)

- **Regular**
  - \(\pi\) is regular for all \(\pi \in \Gamma\)

- **Right / left linear**
  - pattern of the head is \(xw\) for some \(w \in \Sigma^+\)
  - pattern of the head is \(wx\) for some \(w \in \Sigma^+\)

- **Pattern languages are a single-clause EFS:**
  - \(\Gamma = \{p(\pi) \leftarrow\}\)

  - Recursively enumerable
  - Context-sensitive
  - Context-free
  - Regular
**Hereditary clause:**
- \( p(\pi_1, \ldots, \pi_n) \leftarrow q_1(\tau_1, \ldots, \tau_{t_1}), q_2(\tau_{t_1}, \ldots, \tau_{t_2}), \ldots, q_l(\tau_{t_{l-1}+1}, \ldots, \tau_{t_l}) \)
- If for each \( j = 1, \ldots, t_l \) pattern \( \tau_j \) is a substring of some pattern \( \pi_i \).

**LB-H-EFS(m,k):**
- \(|A\theta| \geq |B_1\theta| + \cdots + |B_m\theta|\) (length bounded) and hereditary
- \( \leq m \) clauses
- \( \leq k \) variable occurrences in the head of each clause

<table>
<thead>
<tr>
<th>Language in Chomsky Hierarchy</th>
<th>EFS</th>
<th>Gold Inferable</th>
<th>(Hereditary)(^b) polynomial-PAC Inferable</th>
</tr>
</thead>
<tbody>
<tr>
<td>recursively enumerable</td>
<td>Variable bounded</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>context-sensitive</td>
<td>Length bounded</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>context-free</td>
<td>regular</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>regular</td>
<td>right/left linear</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>
Many problems involve differentiating these classes.

Q: If we look at these as grammar inference problems, can we develop any intuition / bounds / definition of success?

<table>
<thead>
<tr>
<th>Security problem</th>
<th>Subset</th>
<th>Learning model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised learning</td>
<td>$\mathcal{SA} \setminus S$</td>
<td>PAC with positive and negative examples</td>
</tr>
<tr>
<td>Anomaly detection, type 1</td>
<td>$S$</td>
<td>PAC with positive examples</td>
</tr>
<tr>
<td>Anomaly detection, type 2</td>
<td>$A$</td>
<td>PAC with noisy positive examples</td>
</tr>
<tr>
<td>Fuzzing</td>
<td>$A \setminus S$</td>
<td>Learning with an oracle</td>
</tr>
<tr>
<td>Unsupervised clustering</td>
<td>Subsets of $S$ or $A \setminus S$</td>
<td>PAC with positive examples in a mixture setting</td>
</tr>
</tbody>
</table>
Hypothesis: most data “in the wild” comes from simple grammars.

A protocol or code may claim to accept a large, complex grammar. Really, most messages are members of simpler sub-grammars (or unions of sub-grammars).

- Sometimes classifying messages is a good place to start.
- Are most messages as complex as the overall grammar?
- Use program structure as a guide.
  - Can we apply a grammar to those control flow data?
  - Can we segment a large grammar via sub-grammars?
Lange and Weihagen’s (LW) algorithm was applied to weblog data.

Log entries are batched by endpoint and “inferred.”

A straightforward, initial example.

<table>
<thead>
<tr>
<th>POST Requests (Anonymized URIs)</th>
<th># of Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>/7NWS/XBPD07F/HXZRT/6BR/PR9M/<em>x0</em>.Q7OJ</td>
<td>83199</td>
</tr>
<tr>
<td>/0YDO8J33LKSC/DKWPZJ/QCSY1BWNRIIG65;R9LZ64B6GI=<em>x0__x1__x2</em></td>
<td>2366</td>
</tr>
<tr>
<td>/5HEZP8EKVK3R9RGF9D_x0__x1__x2__x3</td>
<td>204950</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GET Requests (Anonymized URIs)</th>
<th># of Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>/KPGF/7OH0EHB1.HE9?id=_%F95X4L2%<em>x1__x2__x3__x4__x5__x6__x7</em>%x8__x2__x3__x11__x12__x6__x14__x15='x16'</td>
<td>6464</td>
</tr>
<tr>
<td>/89TM/8J7MNN1/3H6WH/H751AQ57?<em>t=cd&amp;33XKQ5I6=</em>%x0__IMK6XIR3=x1_1&amp;<em>AELAN=%x2</em>_&amp;COV=en&amp;89S8A2JUFSVXE=%x3_</td>
<td>3096</td>
</tr>
</tbody>
</table>
LW applied as in previous slide.

Zalewski’s AFL does the test case generation / organizing.
Future work.

- Explore ongoing work on binary analysis / malware auto-generation to help us identify grammars.
  - Instrument binaries via fuzzing.
  - Analyze binaries via lifting and decompiling tools.
  - Learn “low hanging fruit” grammars.
  - If most of the grammar is ignored most of the time, maybe the “simple stuff” is a good place to start.

- Input normalizer / firewall for known applications.
  - Once you know (a) the grammar, enforce the grammar.

- EFS seems intuitive. Can we use it to help define protocols and message structures that adhere to specific classes?
Backup Slides
For example…

- **Can we program provably interoperable stacks?**
  - Does your protocol need to be written in BNF? Then the answer is **NO**
    - BNF is context-free
    - Equivalence between two context-free grammars is *undecideable* (unless you know your grammar is deterministic)
    - In other words: if you have two *different* implementations of the same (context-free) protocol, then you can *never* prove that they accept the same set of strings (unless you prove that both are deterministic)
    - See Sassaman et al. 2013: x.509 certificate forgeries

- **Can we write a behavioral malware detector?**
  - Are you trying to detect the behavior *before* it actually happens? Then the answer is **NO**
  - Program X: “Halt if this embedded program does Y, otherwise loop forever”; does program X halt?

- **If I have a bunch of good traffic and a bunch of bad traffic, can I learn a classifier that will generalize?**
  - **NO** (as long as factoring integers is as hard as we think it is)
  - Valiant and Kearns: learning even a regular grammar in polynomial time that (with high probability) will have bounded error is not possible
  - If you can learn DFAs in the PAC framework, you can factor Blum integers

- **Can I at least validate that incoming messages are well-formed first?**
  - **PROBABLY NOT FAST ENOUGH** (since most network protocols are stronger than context-free (see Davidson et al. 2009)
  - Parsing CSGs can be anywhere up to NP (Satta, 1992)
“Finite Thickness” and “finite elasticity”
- Finite Thickness: No (non-empty) set of strings occurs in infinitely many languages in the class under consideration
- Finite Elasticity: There is no infinite set of strings that is consistent with every infinite sequence of languages in the class

The existence of “tell-tales” in the class
- A string or set of strings unique to a single member of the class
- In a security context: “signature”

Finite language, positive presentation, no noise:
- Memorize everything you’ve ever seen
- Doesn’t scale
Having an oracle that knows the grammar and can provide complex responses helps

- To learn DFAs in polynomial time with arbitrary data requires membership and equivalence queries (Angluin, ’88)
  - Extends to a small subset of CFGs (those that can be written compactly in CNF; algorithm is polynomial in size of set of symbols)

A training set designed to teach the learner your grammar helps

- CFGs are PAC-learnable from “simple” presentations

Some non-Chomsky grammars are identifiable under various conditions

- Notable example: Pattern languages can be identified from positive examples…
  - …but are NP-complete to recognize
Finding minimal DFAs from positive and negative examples is NP-hard
- Angluin ’78; Gold ’78

Approximating a minimal target DFA from positive and negative examples to within any finite polynomial factor is NP-hard
- Pitt and Warmuth, ’93

Representation-independent result: DFAs are cryptographically hard to learn from labeled examples
- Existence of a weak learner for a DFA implies a polynomial advantage in recovering the LSB of the plaintext of an RSA-encrypted message
- Kearns and Valiant, ’94

To learn DFAs in polynomial time with arbitrary data requires membership and equivalence queries
- Extends to a small subset of CFGs (those that can be written compactly in CNF; algorithm is polynomial in size of set of symbols)
- Angluin, ’88

DFAs are learnable from “simple” examples, but those examples must be selected based on the DFA and learning algorithm

Very, very simple CFGs appear to be empirically learnable from data…
- …With a lot of effort, a little luck, and no theoretical guarantees
- See, e.g., Omphalos competition