Grammatical inference and language frameworks for LANGSEC

Kerry N. Wood
Richard Harang
U.S. Army Research Laboratory

21 MAY 2015

Important points to remember.

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.

U.S. ARMY RDECOM

ARL

Our contribution is mostly a survey of useful methods.

- 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

Grammar	Power to recognize
Regular	State machine
Context-free	Stack machine
Context-sensitive	Bounded automata
Unrestricted	Turing machine

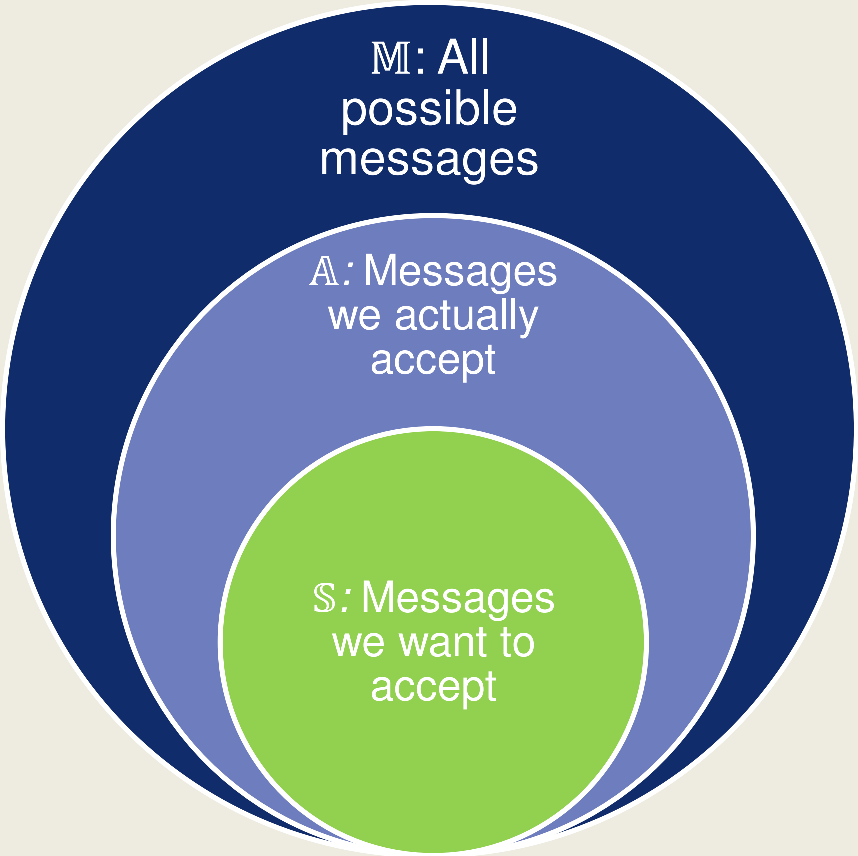
Complexity

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.

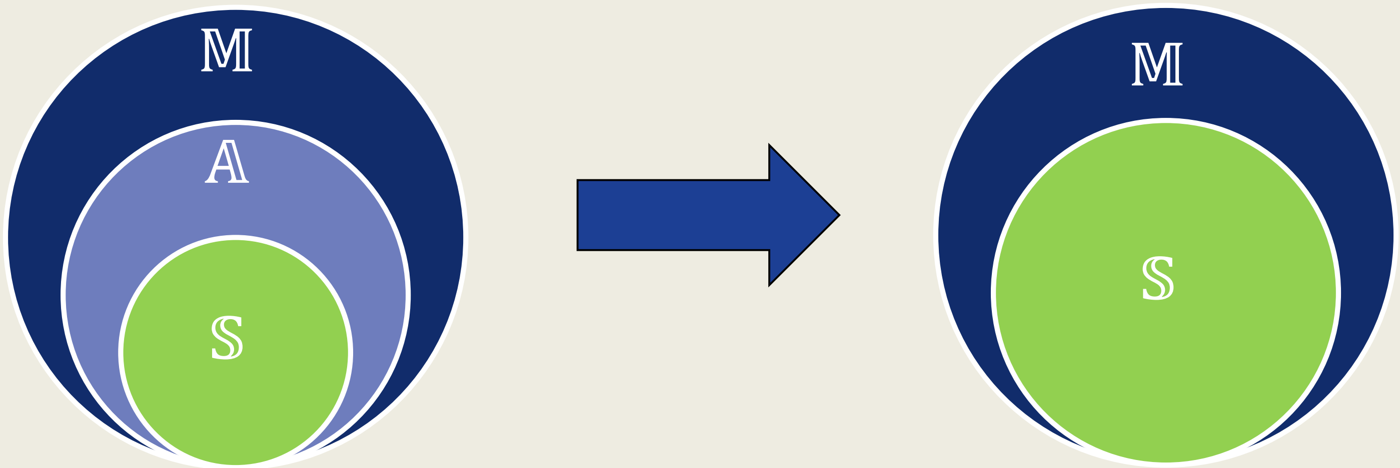


Trouble begins when we mean to define a recognizer for $\mathbb{S}$, but actually define one for $\mathbb{A}$ (and don't know it).

Why not just parse?

"LangSec" – language theoretic security

- Use only simple grammars with provable properties
- Validate all inputs before any processing; reject failures aggressively
- $\mathbb{A} = \mathbb{S} \rightarrow \mathbb{A} \setminus \mathbb{S} = 0$



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: access to examples.

Learning from *positive examples:*

- **All strings in the grammar are shown to the learner** $(S_+ = \forall s \in \mathbb{L})$

Positive and negative examples (aka "complete presentation"):

- **Members and non-members are labeled and shown to the learner.**
- $(S_+ : \forall s \in \mathbb{L})$ **and** $(S_- : \forall\, s \in \Sigma^* - \mathbb{L})$

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.**

U.S. ARMY RDECOM®

Models of learning.

ARL

In general: a learner $\mathcal{A}$ takes some data D, and outputs a (possible) generating grammar $\mathbb{G}$.

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 $\mathcal{A}(D)$.

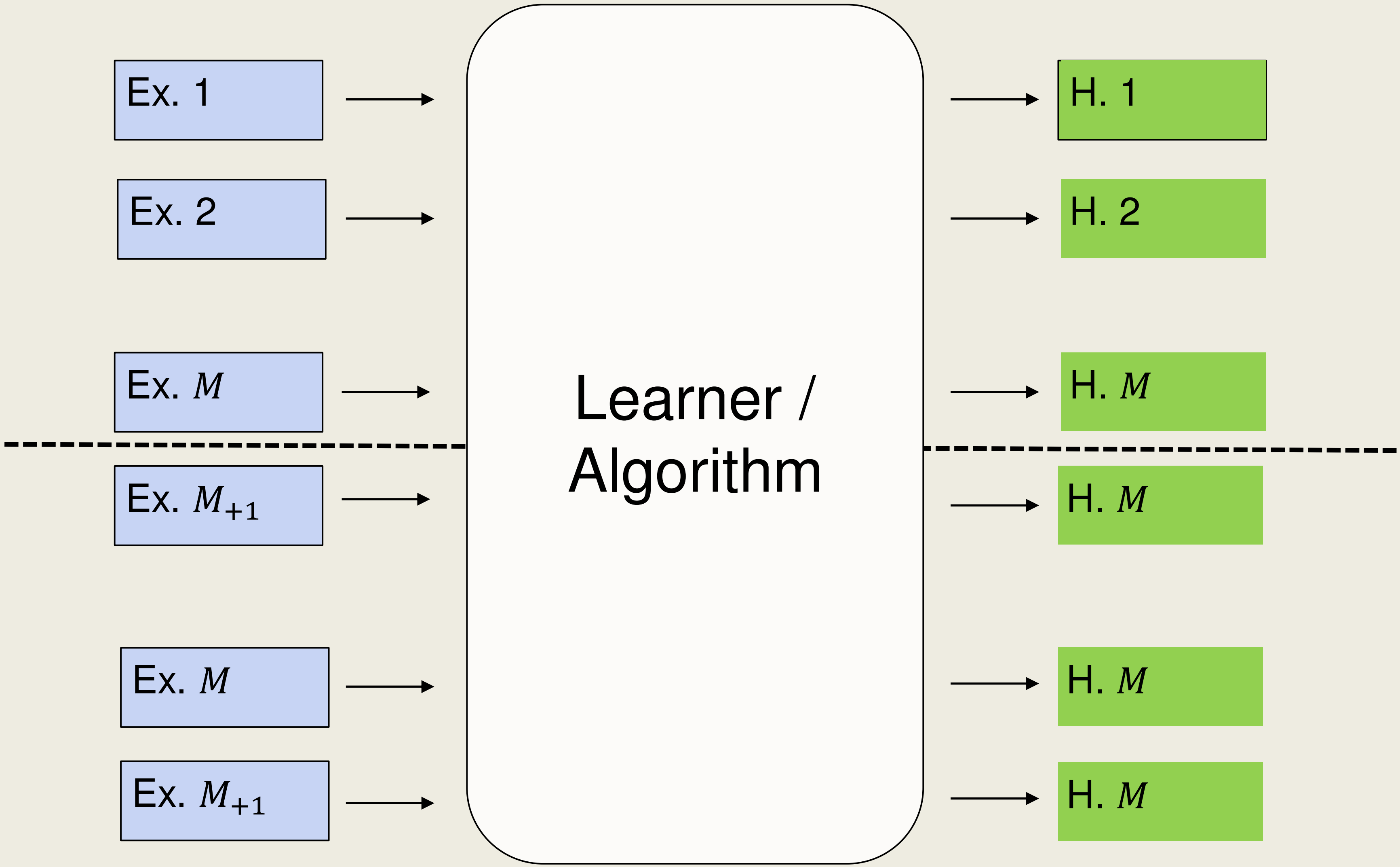
Finite identification

- 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 ϵ – good.

Inductive inference in the limit.



Pattern languages: replace variables with substrings.

- Constants and strings: $\Sigma, \Sigma^*, \Sigma^+$
- Variables: $(\mathrm{x}_1, \mathrm{x}_2, \ldots) \in \mathcal{X}$
- Patterns : $\mathcal{P} = \pi_1, \pi_2, \ldots = \Sigma^* \cup \mathcal{X}$
- Substitution: θ
  - $\pi\theta = \left[\frac{s_1}{x_1}, \frac{s_2}{x_2}, \ldots, \frac{s_k}{x_k}\right]$ **where** $|\boldsymbol{s_i}| \,!= \mathbf{0}$

A pattern π_i is <u>regular</u> if each variable appears at most once.

U.S. ARMY RDECOM®

Patterns are somewhat natural and somewhat learnable.

ARL

The good:

They look just like regular expressions.

<u>Angluin showed they are learnable.</u>

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

The bad:

- **In general, intractable to determine membership.**
  - $NP - complete$
- **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:

$\mathcal{X} = \{x_1, x_2, \ldots\}$

$xaybza$

xx

Elementary formal systems (EFS) are a logic programming analog.

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

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

$\{a^n b^n \mid n \geq 1\}$:

$$\Gamma = \begin{cases} p(ax_1 b) \leftarrow p(x_1) \\ p(ab) \end{cases}$$

EFS' with specific properties map nicely to Chomsky.

A clause / EFS ($A \leftarrow B_1, \ldots, B_m$) is…

- **Variable bounded**
  - $v(A) \supseteq v(B_1) \cup \ldots \cup v(B_m)$ ← Recursively enumerable
- **Length bounded**
  - $|A\theta| \geq |B_1\theta| + \cdots + |B_m\theta|$ ← Context-sensitive
- **Regular**
  - π is regular for all $\pi \in \Gamma$ ← Context-free
- **Right / left linear**
  - pattern of the head is xw for some $w \in \Sigma^+$
  - pattern of the head is wx for some $w \in \Sigma^+$

  ← Regular

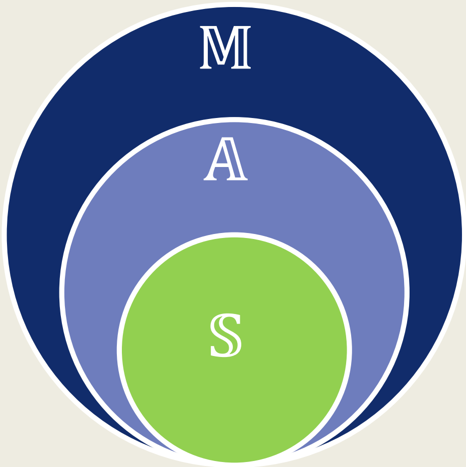
- **Pattern languages are a single-clause EFS:**
  - $\Gamma = \{p(\pi) \leftarrow\}$

Inferability results are *slightly* more encouraging.

- **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 τ_j is a substring of some pattern π_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

Language in Chomsky Hierarchy	EFS	Gold Inferable		(Hereditary)[b] polynomial-PAC Inferable	
	# of clauses →	∞[a]	N[a]	∞	(m,k)
recursively enumerable	Variable bounded	NO	NO	NO	--
context-sensitive	Length bounded	NO	YES	NO	NO[c]
context-free	regular	NO	YES	NO	YES
regular	right/left linear	NO	YES	NO	YES

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?

Security problem	Subset	Learning model
Supervised learning	$\mathbb{S} \cup \mathbb{A} \setminus \mathbb{S}$	PAC with positive and negative examples
Anomaly detection, type 1	$\mathbb{S}$	PAC with positive examples
Anomaly detection, type 2	$\mathbb{A}$	PAC with noisy positive examples
Fuzzing	$\mathbb{A} \setminus \mathbb{S}$	Learning with an oracle
Unsupervised clustering	Subsets of $\mathbb{S}$ or $\mathbb{A} \setminus \mathbb{S}$	PAC with positive examples in a mixture setting

Simple → inferable.

<u>Hypothesis</u>: *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?

U.S. ARMY RDECOM®

Weblog data from live system.

ARL

POST Requests (Anonymized URIs)	# of Samples
`/7NWS/XBPD07F/HXZRT/6BR/PR9M/_x0_.Q7OJ`	83199
`/0YDO8J33LKSC/DKWPZJ/QCSY1BWNRIG65;R9LZ64B6GI=_x0_!_x1_!_x2_`	2366
`/5HEZP8EKVK3R9RGF9D_x0__x1__x2__x3_`	204950

GET Requests (Anonymized URIs)	# of Samples
`/KPGF/7OH0EHB1.HE9?id=_x0_%F95X4L2%_x1__x2__x3__x4_'_x5__x6_%_x7_%_x8__x2__x3__x11_'_x12__x6_%_x14_%_x15_='_x16_'`	6464
`/89TM/8J7MNN1/3H6WH/H751AQ57?_t=cd&33XKQ5I6=_x0_&IMK6XIR3=_x1_&AELAN=_x2_&COV=en&89S8A2JUFSVXE=_x3_`	3096

- Lange and Weihagen's (LW) algorithm was applied to weblog data.
- Log entries are batched by endpoint and "inferred."
- A straightforward, initial example.

Fuzzing $\overset{?}{\leftrightarrow}$ Grammar Inference

<corpus>/jpeg/full/images/*src:000619*		
ffd8ffe000104a46494600010102001c001c0000ffdb004300281c1e231e19282321232d2b 28303c64413c37373c7b585d4964918099968f808c8aa0b4e6c3a0aadaa_x0_8a8cc8ffcbd aeef5ffffff9bc1ffffffaffe6fdfff8ffdb0043012b2d2d3c353c76_x1_4176f8a58ca5f 8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f 8f8f_x2_f8f8f8f8f8f8f8f8f8f8ffc10011080020002003012200021101031101ffc40018 00010100030000000000000000000000_x3_03000104ffc4002510000_x4_0_x5_01040 103050000000000000000_x6_0_x7_3_x8_0_x9_41221312241_x10_71_x11_143361a1ffc 40016010101010000000000000000000000000000102ffc4001a1101000203010000000000 0000000000000_x12_122241ffda000c03010002110311000f00567_x13_c_x14_	25	(504,117)
/jpeg/edges-only/images/*src:003554*		
ffd8ffdd0004_x0_00b400e8ffdb00438028101efffaffe6fdfff8ffdb0043012b2d2d3c35 3c64414176f8a58ca5f8f8d500010000f8f8f8f0f8f8f84a64653d79906fc317d3caf8f8f8 f8f8f8f8f801001c0000ffdb004300281c1e231e19282336232d2bef303c64413c37373c 7b585d4964918099960018c8aa0b4ad8a8cc8ffe4daee097affff9bc1ffffffaffd1fdff f8ffdb0043012b2d2d3cf8f8f8f8e3f8f0e9f900f8f84a6a653d79906fdf17d3b7f8f8f8f8 f8f8f8f8f8ffca001108_x2_0_x3__x4__x4_2003012200021101031101ffda00060001180 1220002110103 1101ffda000600010300030001000000000004000000001ffda000801010 10101010101 0_x1	12	(260,0)
/jpeg/edges-only/images/*src:000512*	12	(170,80)
ffd8ffe000104a_x0__x3__x4__x3__x1_0_x2_		

- **LW applied as in previous slide.**
- **Zalewski's AFL does the test case generation / organizing.**

U.S. ARMY RDECOM® ARL

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?

Questions?

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 <u>requires</u> 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