Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

- Last time we looked at ways to parse without ever building a grammar
- But what if we want to know what a grammar is for a set of strings?

- Today we will look at grammar induction.

  ...we'll start with an example

CFGs may be inferred using recursive **byte-pair encoding**

The following is a *speech unit* of whale song:

b a a c c d c d e c d c d e c d c d e a a b a a c c d e c d c d e

We are going to infer some rules for this string using the following algorithm:

- count the frequency of all adjacent pairs in the string
- reduce the most frequent pair to a non-terminal
- repeat until there are no pairs left with a frequency > 1

This is used for compression—once we have removed all the repeated strings we have less to transmit or store (we have to keep the grammar to decompress)

CFGs may be inferred using recursive **byte-pair encoding**

b a a c c d c d e c d c d e c d c d e a a b a a c c d e c d c d e

$F \rightarrow c\ d$

b a a c F F e F F e F F e a a b a a c F e F F e

$G \rightarrow F\ e$

b a a c F G F G F G a a b a a c G F G

$H \rightarrow F\ G$

b a a c H H H a a b a a c G H

$I \rightarrow a\ a$

b I c H H H I b I c G H

$J \rightarrow b\ I$

J c H H H I J c G H

$K \rightarrow J\ c$

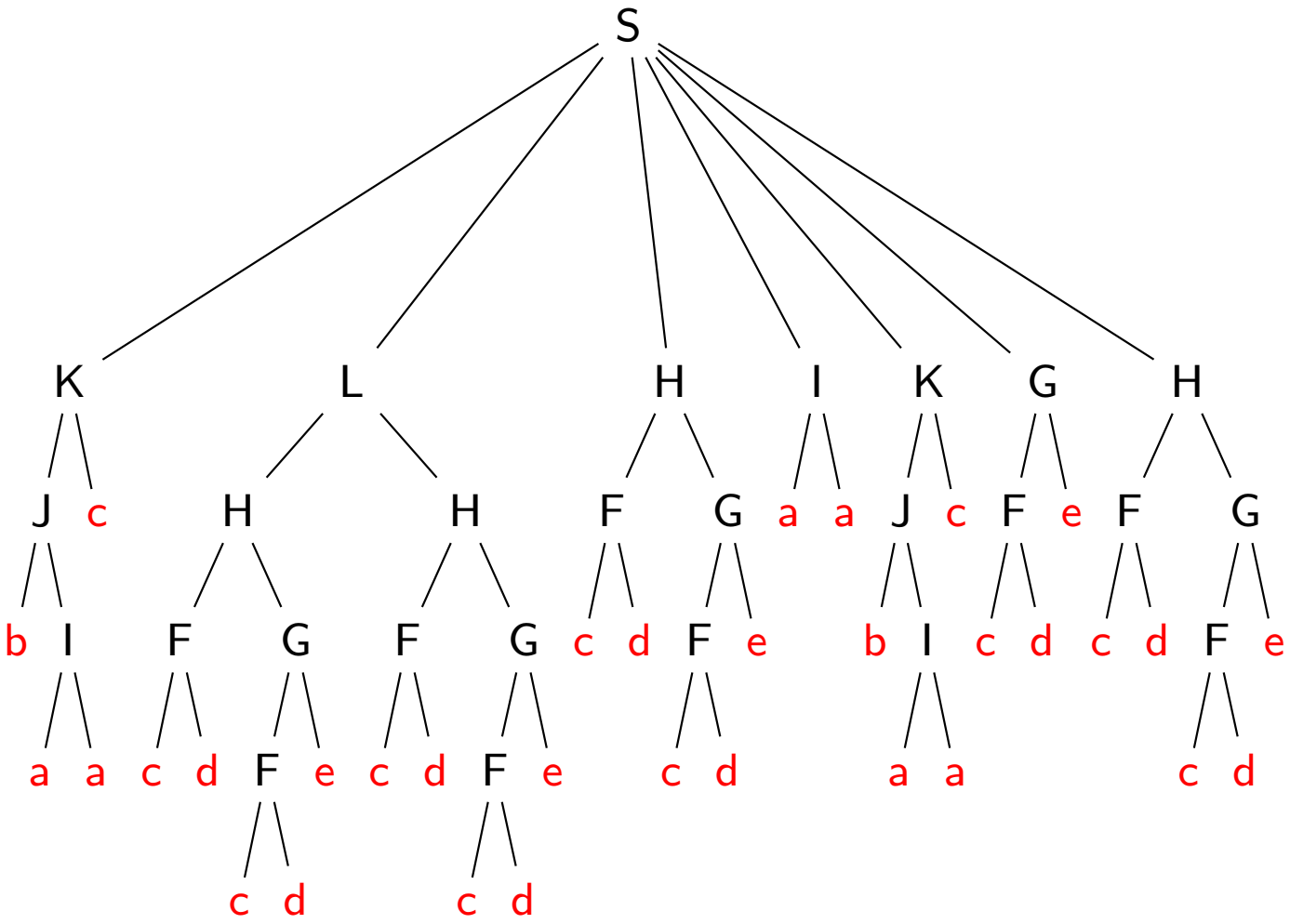
K H H H I K G H

$L \rightarrow H\ H$

K L H I K G H

$S \rightarrow K\ L\ H\ I\ K\ G\ H$

CFGs may be inferred using recursive **byte-pair encoding**



Byte-pair has **shortcomings** for grammar induction

Byte-pair encoding has benefits for encryption but shortcomings when it comes to grammar induction (especially of natural language):

- the algorithm is frequency driven and this might not lead to *appropriate* constituency
- in the circumstance that two pairs have the same frequency we make an arbitrary choice of which to reduce.
- the data is assumed to be non-noisy (all string sequences encountered are treated as valid)
- (for natural language) the algorithm learns from strings alone (a more *appropriate* grammar might be derived by including extra-linguistic information)

We might suggest improvements to the algorithm (such as allowing ternary branching) but in order to compare the algorithms we need a **learning paradigm** in which to study them.

Paradigms are defined over **grammatical systems**

Grammatical system:

- $\mathcal{H}$ a hypothesis space of language descriptions (e.g. all possible grammars)
- Ω a sample space (e.g. all possible strings)
- $\mathcal{L}$ a function that maps from a member of $\mathcal{H}$ to a subset of Ω

If we have $(\mathcal{H}_{cfg}, \Sigma^*, \mathcal{L})$ then for some $G \in \mathcal{H}_{cfg}$ we have:
$\mathcal{L}(G) = \{s_a, s_b, s_c ...\} \subseteq \Sigma*$

Learning function:

The learning function, F, maps from a subset of Ω to a member of $\mathcal{H}$

For $G \in \mathcal{H}_{cfg}$ then $F(\{s_d, s_e, s_f ...\}) = G$ for some $\{s_d, s_e, s_f ...\} \subseteq \Sigma*$

Note that the learning function is an algorithm (referred to as the **learner**) and that **learnability** is a property of a language class (when F surjective).

Learning paradigms specify the nature of **input**

Varieties of input given to the learner:

- **positive evidence**: the learner receives only valid examples from the sample space (i.e. if the underlying grammar is G then the learner receives samples, s_i, such that $s_i \in \mathcal{L}(G)$).
- **negative evidence**: the learner receives samples flagged as not being in the language.
- **exhaustive evidence**: the learner receives every relevant sample from the sample space.
- **non-string evidence**: the learner receives samples that are not strings.

Learning paradigms also specify...

- **assumed knowledge**: the things known to the learner before learning commences (for instance, the hypothesis space, $\mathcal{H}$ might be assumed knowledge).
- **nature of the algorithm**: are samples considered sequentially or as a batch? does the learner generate a hypothesis after every sample received in a sequence? does the learner generate a hypothesis after specific samples only?
- **required computation**: e.g. is the learner constrained to act in polynomial time.
- **learning success**: what are the criteria by which we measure success of the learner?

Gold's learning paradigms have been influential

Gold's best known paradigm modelled language learning as an infinite process in which a learner is presented with an infinite stream of strings of the target language:

- for a grammatical system $(\mathcal{G}, \Sigma^*, \mathcal{L})$
- select one of the languages L in the class defined by $\mathcal{L}$ (this is called the **target language**, $L = \mathcal{L}(G)$ where $G \in \mathcal{G}$)
- samples are presented to the learner one at a time $s_1, s_2, \ldots$ in an infinite sequence
- the learner receives only positive evidence (i.e. only s_i such that $s_i \in L$)
- after each sample the learner produces a hypothesis (i.e. learner produces G_n after having seen the data $s_1, \ldots s_n$
- the evidence is exhaustive, every $s \in L$ will be presented in the sequence.

Gold's learning paradigms have been influential

Gold defined **identification in the limit** as successful learning:

- There is some number N such that for all $i > N$, $G_i = G_N$ and $\mathcal{L}(G_N) = L$
- N is finite but there are no constraints placed on computation time of the learning function.

In this paradigm a **class** of languages is **learnable** if:

- Every language in the class can be identified in the limit *no matter what order* the samples appear in

Gold's learning paradigms have been influential

Well known results from Gold's paradigm include:

- The class of **suprafinite** languages are not learnable (a suprafinite class of languages is one that contains all finite languages and at least one infinite language)
- This means that e.g. the class of context-free languages are *not learnable* within Gold's paradigm.

We might care about this if we think that Gold's paradigm is a good model for natural language acquisition...(if we don't think this then it is just a fun result!).

Gold: **suprafinite** languages are not learnable

Short proof:

- Let L_∞ be an infinite language $L_\infty = \{s_1, s_2, ...\}$
- Now construct an infinite sequence of finite languages $L_1 = \{s_1\}$, $L_2 = \{s_1, s_2\}$, ...
- Consider a particular presentation order $s_1 ... s_1, s_2 ... s_2, s_3 ...$
- When learning L_1 we repeat s_1 until the learner predicts L_1
- When learning L_2 repeat s_1 until the learner predicts L_1 then repeat s_2 until it predicts L_2
- Continue like this for all L_i: either the learner fails to converge on one of these, or it ultimately fails to converge on L_∞ for finite N.
- We have found an ordering of the samples that makes the learner fail

Many people have investigated what IS learnable in this paradigm. We will look at one example, but to do so we introduce one more grammar.

Categorial grammars are **lexicalized grammars**

In a **classic categorial grammar** all symbols in the alphabet are associated with a finite number of **types**.

- Types are formed from primitive types using two operators, $\backslash$ and $/$.
- If P_r is the set of **primitive types** then the set of all types, T_p, satisfies:
  - $P_r \subset T_p$
  - if $A \in T_p$ and $B \in T_p$ then $A\backslash B \in T_p$
  - if $A \in T_p$ and $B \in T_p$ then $A/B \in T_p$
- Note that it is possible to arrange types in a hierarchy: a type A is a *subtype* of B if A occurs in B (that is, A is a subtype of B iff $A = B$; or ($B = B_1\backslash B_2$ or $B = B_1/B_2$) and A is a subtype of B_1 or B_2).

Categorial grammars are **lexicalized grammars**

- A relation, $\mathcal{R}$, maps symbols in the alphabet Σ to members of T_p.
- A grammar that associates at most one type to each symbol in Σ is called a **rigid grammar**
- A grammar that assigns at most k types to any symbol is a **k-valued grammar**.
- We can define a classic categorial grammar as $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ where:
  - Σ is the alphabet/set of terminals
  - P_r is the set of primitive types
  - S is a distinguished member of the primitive types $S \in P_r$ that will be the root of complete derivations
  - $\mathcal{R}$ is a relation $\Sigma \times T_p$ where T_p is the set of all types as generated from P_r as described above

Categorial grammars are **lexicalized grammars**

A string has a valid parse if the types assigned to its symbols can be combined to produce a derivation tree with root S.

Types may be combined using the two rules of function application:

- Forward application is indicated by the symbol $>$:

$$\frac{A/B \qquad B}{A} >$$

- Backward application is indicated by the symbol $<$:

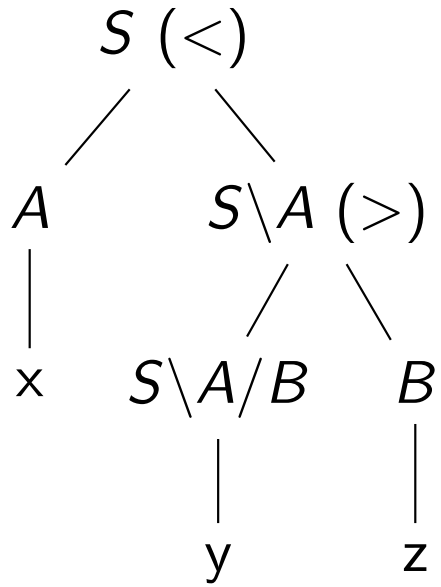
$$\frac{B \qquad A\backslash B}{A} <$$

Categorial grammars are **lexicalized grammars**

Derivation tree for the string *xyz* using the grammar $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ where:

$$
\begin{array}{rcl}
Pr & = & \{S, A, B\} \\
\Sigma & = & \{x, y, z\} \\
S & = & S \\
\mathcal{R} & = & \{(x, A), (y, S\backslash A/B), (z, B)\}
\end{array}
$$

$$
\dfrac{\dfrac{x}{A}\,\mathcal{R} \qquad \dfrac{\dfrac{y}{S\backslash A/B}\,\mathcal{R} \qquad \dfrac{z}{B}\,\mathcal{R}}{S\backslash A}\,>}{S}\,<
$$

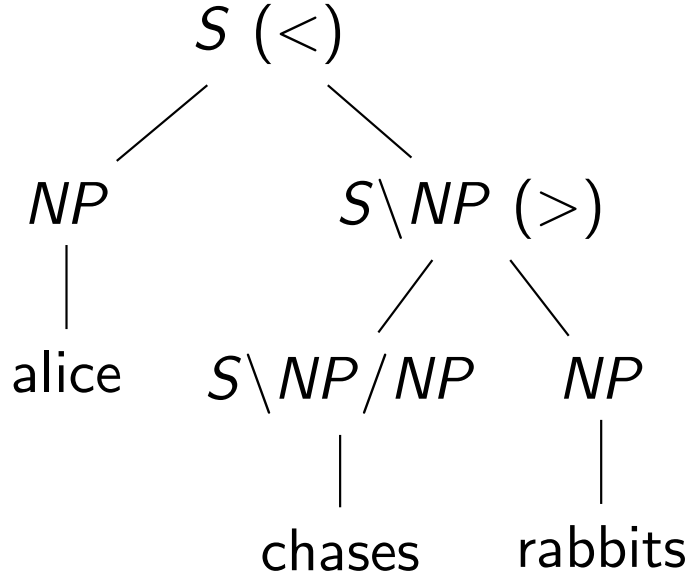


Categorial grammars are **lexicalized grammars**

Derivation tree for the string *Alice chases rabbits* using the grammar $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ where:

$$
\begin{aligned}
Pr &= \{S, NP\} \\
\Sigma &= \{alice, chases, rabbits\} \\
S &= S \\
\mathcal{R} &= \{(alice, NP), (chases, S\backslash NP/NP), \\
&\quad (rabbits, NP)\}
\end{aligned}
$$

$$
\dfrac{\dfrac{alice}{NP}\,\mathcal{R} \qquad \dfrac{\dfrac{chases}{S\backslash NP/NP}\,\mathcal{R} \qquad \dfrac{rabbits}{NP}\,\mathcal{R}}{S\backslash NP}\,>}{S}\,<
$$



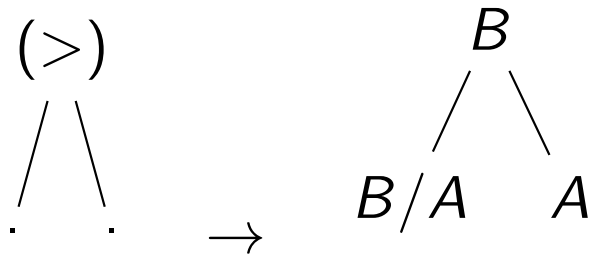
We can construct a **strongly equivalent** CFG

To create a context-free grammar $G_{cfg} = (\mathcal{N}, \Sigma, S, \mathcal{P})$ with strong equivalence to $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ we can define G_{cfg} as:

$$
\begin{array}{rcl}
\mathcal{N} & = & P_r \cup \mathit{range}(\mathcal{R}) \\
\Sigma & = & \Sigma \\
S & = & S \\
P & = & \{A \rightarrow B \;\; A\backslash B \mid A\backslash B \in \mathit{range}(\mathcal{R})\} \\
 & & \cup \{A \rightarrow A/B \;\; B \mid A/B \in \mathit{range}(\mathcal{R})\} \\
 & & \cup \{A \rightarrow a \mid \mathcal{R} : a \rightarrow A\}
\end{array}
$$

FYI: a categorial grammar learner within Gold's paradigm

- Buszkowski developed an algorithm for learning rigid grammars from **functor-argument structures**.
- The algorithm proceeds by inferring types from the available information
- Eg. for Forward Application:



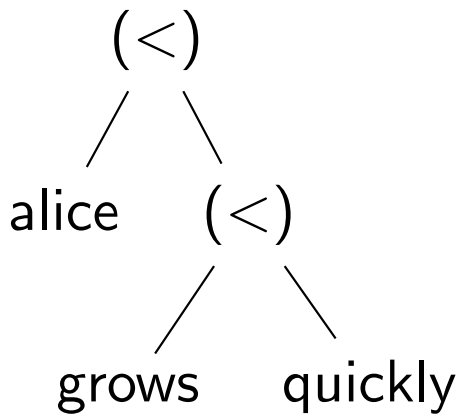
- Variables are unified across all encountered structures.
- Kanazawa constructed a proof to show that the algorithm could learn the class of rigid grammars from an **infinite stream** of functor-argument structures — as required to satisfy Gold's paradigm.

FYI: a categorial grammar learner within Gold's paradigm

Let G_i be the current hypothesis of the learner:

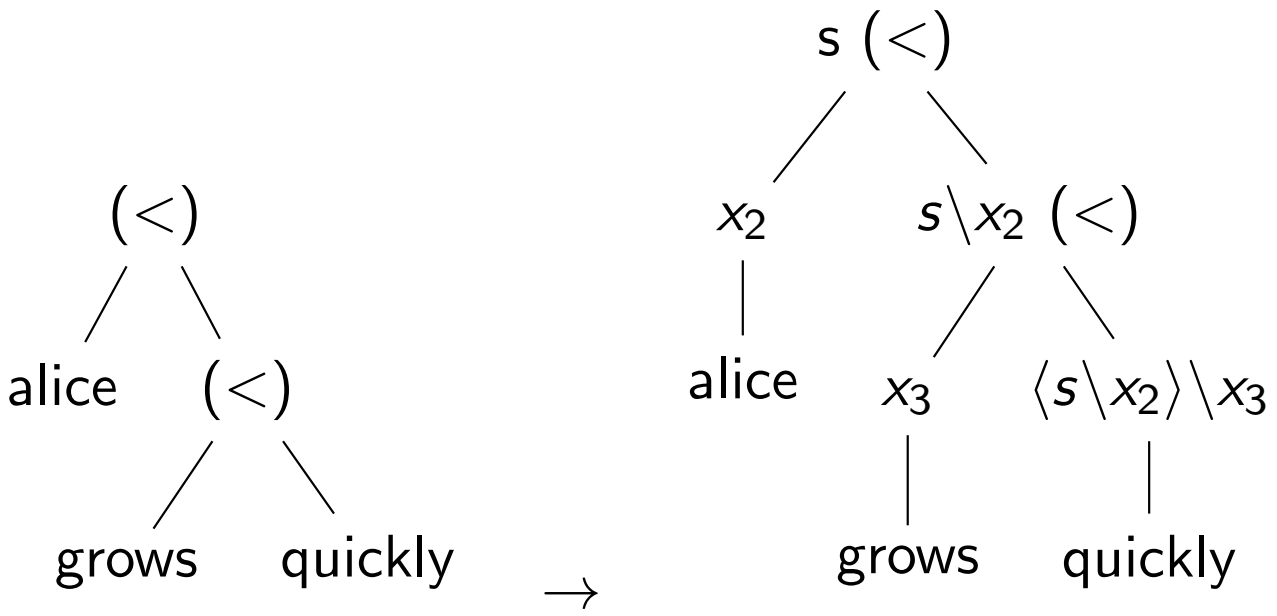
$$\begin{aligned} G_i : \text{alice} &\rightarrow x_1 \\ \text{grows} &\rightarrow s \backslash x_1 \end{aligned}$$

Let the next functor-argument structor encountered in the steam be:



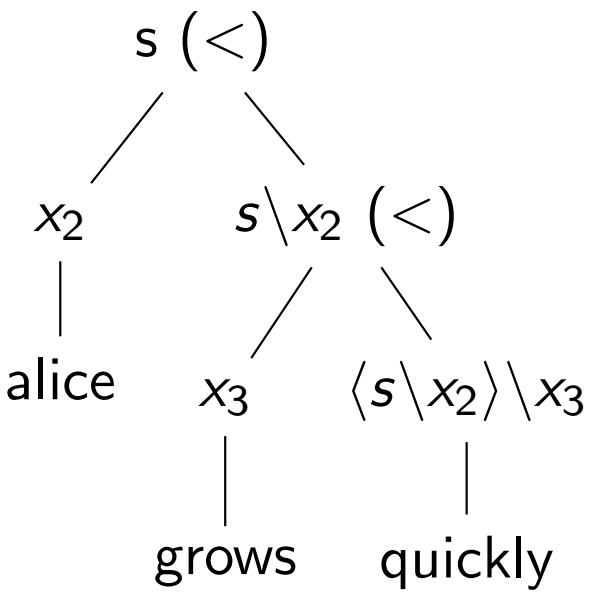
FYI: a categorial grammar learner within Gold's paradigm

Infer types to the new functor-argument structure:



FYI: a categorial grammar learner within Gold's paradigm

- Look up words at the leaf nodes of the new structure in G_i
- If the word exists in G_i, add types inferred at leaf nodes to the existing set of types for that word; else create new word entry.



$$G_i : \text{alice} \rightarrow x_1$$
$$\text{grows} \rightarrow s\backslash x_1$$

$$G_{i+1} : \text{alice} \rightarrow x_1, x_2$$
$$\text{grows} \rightarrow s\backslash x_1, x_3$$
$$\text{quickly} \rightarrow \langle s\backslash x_2\rangle\backslash x_3$$

FYI: a categorial grammar learner within Gold's paradigm

$$\begin{aligned} G_{i+1} : \text{alice} &\rightarrow x_1, x_2 \\ \text{grows} &\rightarrow s \backslash x_1, x_3 \\ \text{quickly} &\rightarrow \langle s \backslash x_2 \rangle \backslash x_3 \end{aligned}$$

- Unify the set of types. If unification fails then fail.

$$\begin{aligned} x_2 &\mapsto x_1 \\ x_3 &\mapsto s \backslash x_1 \end{aligned}$$

- Output the lexicon.

$$\begin{aligned} G_{i+1} : \text{alice} &\rightarrow x_1 \\ \text{grows} &\rightarrow s \backslash x_1 \\ \text{quickly} &\rightarrow \langle s \backslash x_2 \rangle \backslash \langle s \backslash x_1 \rangle \end{aligned}$$

FYI: a categorial grammar learner within Gold's paradigm

Using this learner within Gold's paradigm over various sample spaces it is possible to prove:

- Rigid grammars are learnable from functor-argument structor and strings
- k-valued grammars (for a specific k) are learnable from functor-argument structor and strings

Note that the above mentioned grammars are subsets of the CFGs

Gold's paradigm **is not** much like human acquisition

- Gold's paradigm requires convergence in a finite number of steps (hypotheses of G) the amount of data it sees is unbounded.
- Gold's learner can use unbounded amounts of computation.
- A child only sees a limited amount of data, and has limited computational resources
- Success in this paradigm tells you absolutely nothing about the learner's state at any finite time.
- Children learn progressively
- The learner has to learn for every possible presentation of the samples (including presentations that have been chosen by an adversary with knowledge of the internal state of the learner).
- It is arguable that the distributions are in some way helpful: **parentese**

Gold's paradigm **is not** much like human acquisition

- Gold's learner is required to exactly identify the target language.
- We do not observe this in humans

  We observe agreement on *grammaticality* between adults and children approaching adult competence but we also observe differences in word choices and grammaticality judgments between adults speakers.
- Gold's learner requires a hypothesis to be selected after every step.
- In fact there is evidence that children only attend to selective evidence (Goldilocks effect)