EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Formal grammars

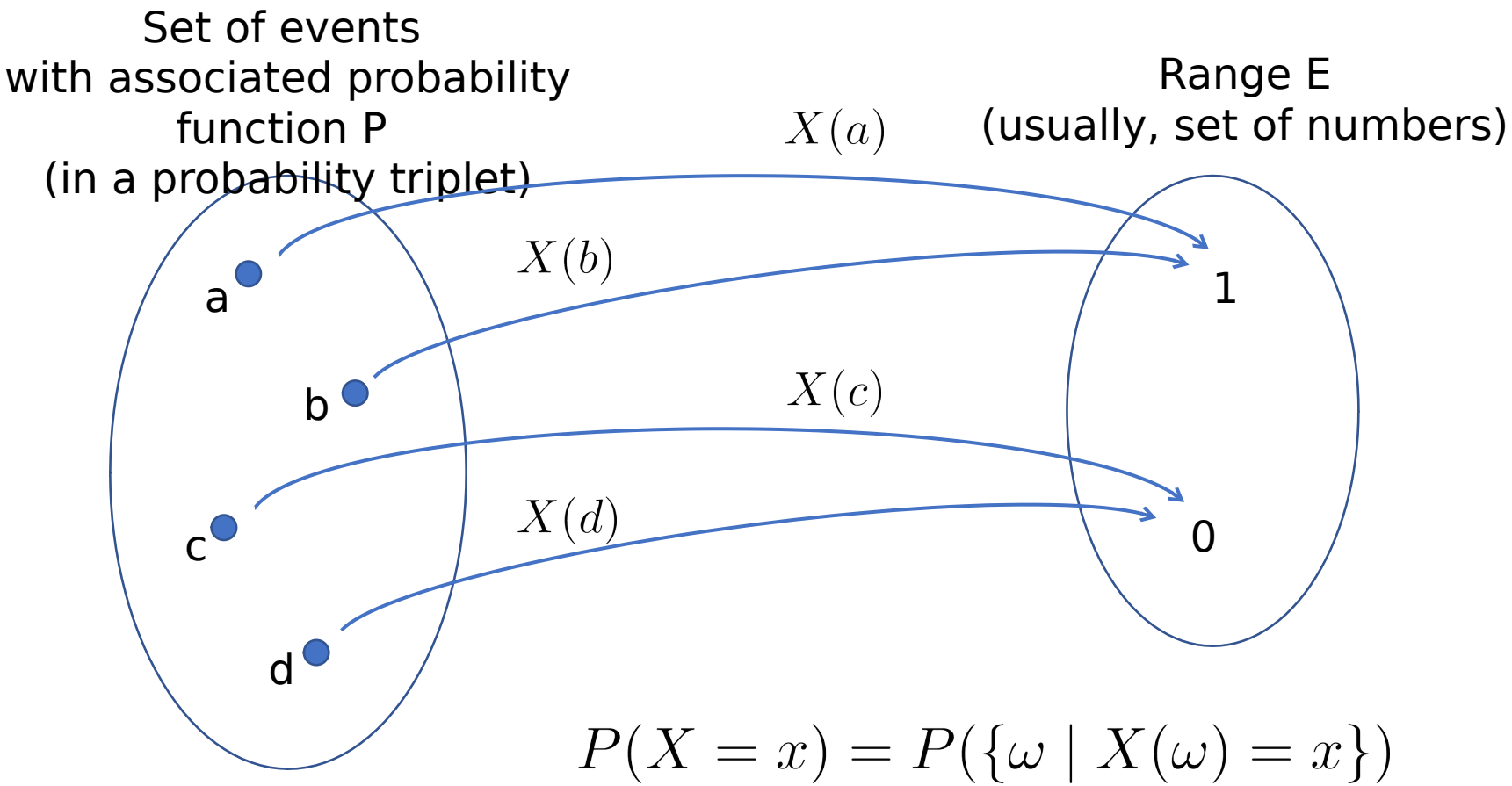
Or: Let’s get formal!

- *The Language of Thought: computational cognitive science approaches to category learning*
- Who: Fausto Carcassi
- When: Sommer semester 2022

Summary this far

- In the last three weeks, we have looked at a theory in *speculative psychology,* the Language of Thought Hypothesis.
- We have seen what the theory says and arguments for and against it.
- Until now it was all theoretical / philosophical.
- But the real fun part of this is technical!
- At the end, we want to have tools that allow us to build fragments of the LoT in a computer and show how they can learn expressions in the LoT from data.
- Therefore, in the next 4 weeks we will build the technical foundations required to implement these models.
- We saw some basic probability concept last week in the lab. Let’s have a brief look at them again before we move onto this week’s topic.

Random variable X



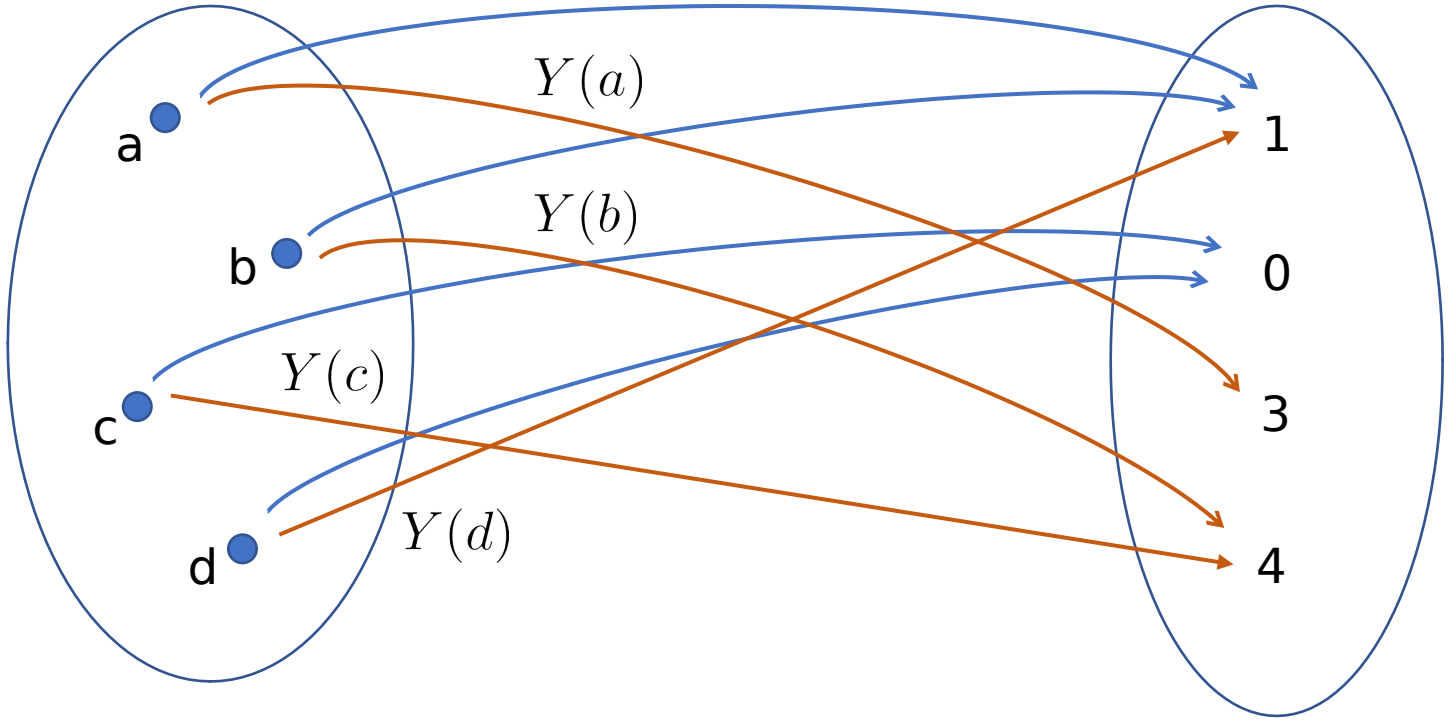
$$P(X = x) = P(\{\omega \mid X(\omega) = x\})$$

$$P(X = 0) = P(\{c, d\})$$

Random variables

- We say that a random variable *has a certain distribution* or that *it is distributed as a certain distribution* and we write:
  - Random variable its distribution
- Note that we are using *the distribution*, the abstract object
- For instance,
  - Call X the total number of heads from flipping a fair coin four times
  - Then we can write:
  - And we say 'X is distributed as a Binomial with n parameter 4 and p parameter 0.5'

Joint distribution of X and Y



$$P(X = x, Y = y) = P(\{\omega \mid X(\omega) = x \wedge Y(\omega) = y\})$$

$$P(X = 0, Y = 4) = P(\{c\})$$

Today’s plan

- We will briefly go through some basic set theoretical definition
- Then, we will discuss the concept of a *formal grammar* and *formal language.*
- This will give us a rigorous, formal way to model languages, which we need to implement the idea of an LoT into something a computer can understand.

What's the idea behind *grammar*?

- We will discuss material from chapter 16 of Partee et al. (1990)
- (Btw Barbara Partee is an absolute **legend** in linguistics)
- There are two crucial insights to build up to the idea of grammar.
- Suppose we have a *language*.
  - Typically, a natural language, but not necessarily!
- First:
  - There is a notion of well-formedness that is independent of semantics
  - For instance: 'The whiteboard ignores the marathon' is meaningless, but well-formed in some sense.
  - We call this sense *grammatical* well-formedness.
- Second:
  - We can build an 'abstract device' of some kind that tells us which sentences are well-formed in this non-semantic sense.
  - Two types of such devices are popular: automata and formal grammars.
  - There is a correspondence between automata and grammars!
  - In the rest of the course we'll just use grammars.

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Some foundations

- What is a set?
  - Abstract collection of disjoint (non-repeated) objects
- How can we define a set?
  - Listing the elements of the set (but careful: sets are not ordered!)
    - List notation: {table, chair}
  - Stating a property which all and only the objects in the set have
    - Intentional notation: {x | x is an integer greater than 3}
  - Defining rules that generate the elements of the set
    - Recursive definition you have seen in the python lab!
- Two sets are the same iff they contain the same members
- Member, subset, superset, intersection, union, power set

Some foundations

- What is an ordered tuple?
  - <a, b> = {{a}, {a, b}}
  - <a, b, c> = <<a, b>, c>
- Cartesian product of two sets A and B

- What is a relation?
  - A *relation from set A to set B* is any subset of
- What is a function?
  - A *function from A to B* is a relation R from A to B such that:
  - Each element of A appears exactly once in the first elements of R.

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Let’s talk about strings!

- Take a finite set A, called the *vocabulary* or *alphabet*.
- A *string on A* is an **ordered, finite** sequence of occurrences of elements of A
- The *length* of a string is the number of occurrences in the string

- $A = \{a, b, c\}$
- *acbaaabbc*
- Has lengtht 9

- NOTE: Strings are not (plain) sets, cause elements can appear more than once!
- NOTE: Strings with length 1 are still strings, not just elements of A
- NOTE: There is a unique empty string, which we call e.

Some terminology

- *Concatenation* is an operation that takes two strings and returns the first string followed by the second string.
- A^* is the set of all strings that can be built from A.
- The *reversal* of a string a is written a^R
- A *substring* of a string is a section of it
- A *prefix* and a *suffix* are any substring at the beginning or end of a string

- $A = \{a, b, c\}$
- $acba \hat{} aabbc = acbaaabbc$
- $acba \hat{} e = acba$
- $A^* = \{a, aa, ab, ac, ba, bb, bc, aaa, \ldots\}$
- $abc^R = cba$
- acc is a substring of $abaccab$
- a and ab are prefixes of $abac$
- ac and c are suffices of $abac$

Formal *languages*

- A *formal language* is a subset of A^*
- In order words, it's any (finite or infinite) set of strings over A
  - $A^* = \{a, aa, ab, ac, ba, bb, bc, aaa, \ldots\}$
  - Example: $L=\{a, aa, aaa\}$
- A direct consequence of this setup is that there are uncountably many formal languages for any alphabet, i.e. as many as the real numbers.
- But of course there are only countably many ways we can describe a language!
- Therefore, for any alphabet there are infinitely many languages (i.e. collections of strings) that cannot be described by any device with finite resources).
- In order words, for any strategy that we can come up with to describe languages, there are infinitely many languages that are so unstructured (from the point of view of that description strategy) that they cannot be described in that way.

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Formal *grammars*

- Grammars are one way of describing infinite languages in a finite way.
- A formal grammar consists of four ingredients:
  1. A finite set N of so-called *nonterminal symbols*
  2. A finite set Σ of so-called *terminal symbols*
  3. A finite set P of so-called *production rules*
  4. A symbol S in N designated as the *start symbol*
- I haven’t defined yet what any of this means.
- Let’s just look at a couple examples to get an intuition for how they work.

Formal *grammars*

1. *N* (nonterminal symbols)
2. Σ (terminal symbols)
3. *P* (production rules)
4. *S* (start symbol)

1. {S, x, y}
2. {a, b}
3. 1. S → x
   2. x → xy
   3. y → a
   4. x → b

- Let’s derive a couple sentences in this grammar!

Formal *grammars*

1. *N* (nonterminal symbols)
2. Σ (terminal symbols)
3. *P* (production rules)
4. *S* (start symbol)

1. {S, x, y}
2. {a, b}
3. 1. S → x
   2. x → xy
   3. by → ba
   4. x → b

- Let’s derive a couple sentences in this grammar!

Formal *grammars*

1. *N* (nonterminal symbols)
2. Σ (terminal symbols)
3. *P* (production rules)
4. *S* (start symbol)

1. {S, x, y}
2. {a, b}
3. 1. S → x
   2. x → xy
   3. by → ba
   4. x → b
   5. b → a

WRONG! Why?

- Let’s derive a couple sentences in this grammar!

Formal *grammars*

1. *N* (nonterminal symbols)
2. Σ (terminal symbols)
3. *P* (production rules)
4. *S* (start symbol)

1. {S, x, y}
2. {a, b}
3. 1. S → x
   2. x → xy
   3. by → ba
   4. x → b
   5. y → *e*

- Let’s derive a couple sentences in this grammar!

From language to grammar

- We have been generating sentences in a language from a grammar.
- We can also go the other way: given a language defined in some other way, find a grammar that produces that language.
- Let's try to write a grammar that produces all the palindromes in {a, b}*

1. *N* (nonterminal symbols)
2. Σ (terminal symbols)
3. *P* (production rules)
4. *S* (start symbol)

1. {S}
2. {a, b}
3. 1. S → aSa
   2. S → bSb
   3. S → *e*
   4. S → a
   5. S → b

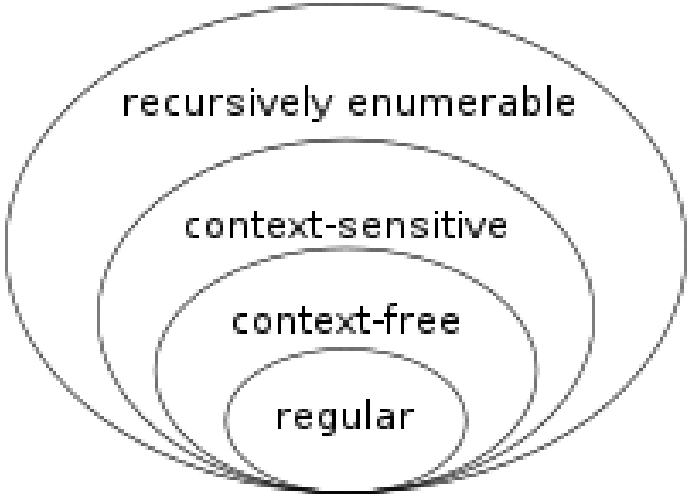
From language to grammar

- All strings with the form: $a^n b^n$

1. N (nonterminal symbols)	1. {S}
2. Σ (terminal symbols)	2. {a, b}
3. P (production rules)	3. 1. S → aSb
4. S (start symbol)	2. S → *e*

Types of grammar – regular

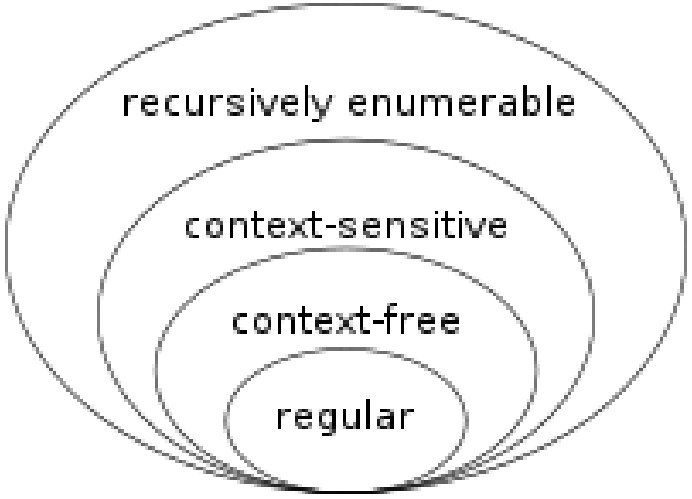
- Note that by putting different restrictions on the production rules, we get different *types* of grammars.
- NOTE: not specific grammar, but TYPES of grammars
- For instance, suppose we introduce the restriction that all rules only take one of three forms (where A and B are any non-terminals and a is a terminal):
  1. A → a
  2. A → aB
  3. A → *e*
- This gives us the set of *right-regular grammars*.
- In the usual categorization of grammar types, these are usually considered the simplest amongst the interesting grammars.
- Question: can we write a right-regular grammar to produce $a^n b^n$?
  - We can't!



Chomsky hierarchy

Types of grammar – context-free

- The second important type of grammar is the *context-free grammars*.
- All their rules are of the form:
  - A □
- Where A is a non-terminal and is any (possibly empty) string of terminals or non-terminals.
- Let’s come up with a context-free grammar that generates ‘The dog barked at the cat’.

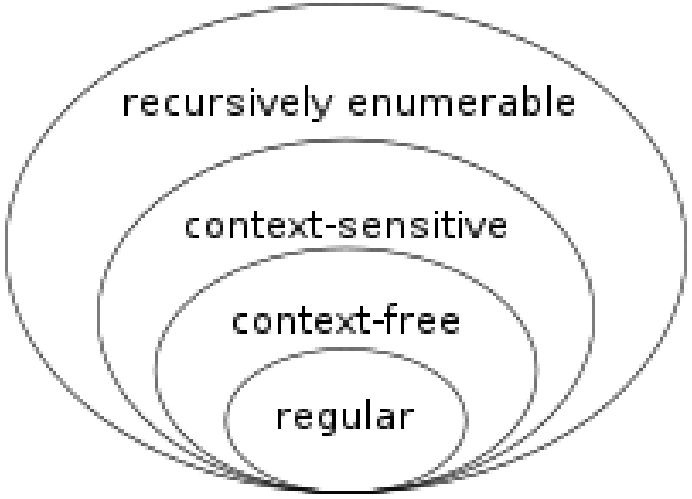


Chomsky hierarchy

Types of grammar – context-sensitive

- Essentially, context-sensitive grammars allow rules with more than one symbol on the left side.
- The rules therefore take this form:

- Where is a single nonterminal
- are (possibly empty) strings that can contain both terminals and non-terminals
- is non-empty and can contain terminals and non-terminals
- Basically, is replaced by , in the context with on the left and on the right.

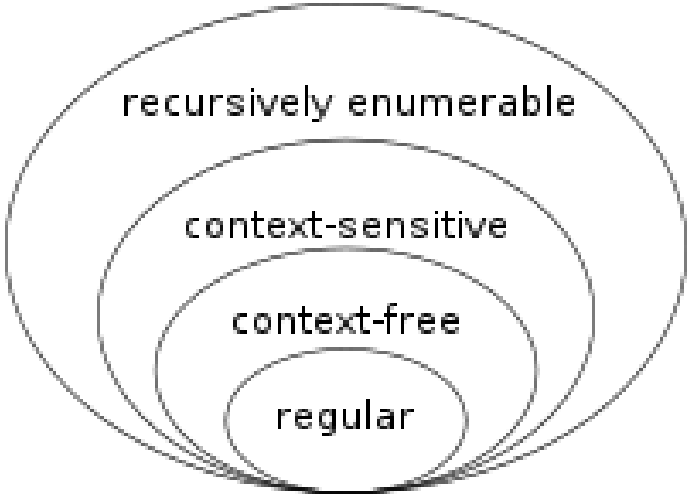


Chomsky hierarchy

Types of grammar – recursively enumerable

- Recursively enumerable grammars are the most expressive ones in the Chomsky hierarchy.
- This means that there are languages (sets of strings) that can be produced by a recursively enumerable grammar but not by a context-sensitive grammar.
- Recursively enumerable grammars also have rules of the form:

- The only different from context-sensitive grammars is that now can be empty.
- Effectively, this means that the only restriction is that the left-hand side of rules is non-empty.



Chomsky hierarchy

Probabilistic context-free grammar (pcfg)

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- $G = (N, \Sigma, P, S,)$
  1. N (nonterminal symbols)
  2. Σ (terminal symbols)
  3. P (production rules)
  4. S (start symbol)
  5. (probabilities on production rules)
- Basically, we just have an old context-free grammar
- But on top of that we also have a function that gives a probability to each production rule, interpreted as:
  - The conditional probability of applying rule , conditional on the left side of the rule being .
- The point of this is that now every derivation (tree) has a probability of being derived, namely the product of the probabilities of the applied rules.
- This gives us a distribution over sentence which crucially *gives higher probability to trees that result from applying fewer and most likely rules.*

Probabilistic context-free grammar (pcfg)

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

1. *N* (nonterminal symbols)
2. Σ (terminal symbols)
3. *P* (production rules)
4. *S* (start symbol)
5. (probabilities on production rules)

Can you guess why they're important to us?

1. {S}
2. {a, b}
3. 1. S → aSa 0.3
   2. S → bSb 0.3
   3. S → e 0.2
   4. S → a 0.1
   5. S → b 0.1

Domain specific languages

- Suppose that we want to define a grammar to talk about relations between people in a family, which can capture every relation we want to talk about (father, mother, etc.)
- What could the primitives be?
- What could the rules be?
- Note that in defining the grammar we have relied on pre-existing knowledge of the meaning of the words.
- In general, when we define a grammar, we also have in mind what the relevant semantics is.

The road ahead

- Eventually, we will encode fragments of the LoT with PCFGs:
  - Each sentence corresponds to a concept
  - Use the probability of each derivation as the prior probability of the concept
- We will get the meaning of that concept by *interpreting* the sentence.
  - Recall from the first lecture that expressions in the LoT have semantic properties, e.g., a meaning
  - (Even though according to Fodor the meaning does not get involved in the mental computations. But this doesn't need to concern us here)
  - However, since pcfgs define infinitely many sentences, we need a way to formalize the interpretation of a sentence that we know works in all cases
  - Next time we'll look at a formal approach to interpreting a grammar (going from a parse tree to a meaning).

The road ahead

- Once we have a piece of LoT formulated as a grammar and a way of knowing what each sentence means (i.e. what it says about the world), we can move onto learning
- In other words, a model of how children go from observations of language use to inferring the concepts (expressions in the LoT) expressed by the language.
- In order to do that, we will look at Bayesian learning for two weeks.
- I hope the grand plan is becoming clear!