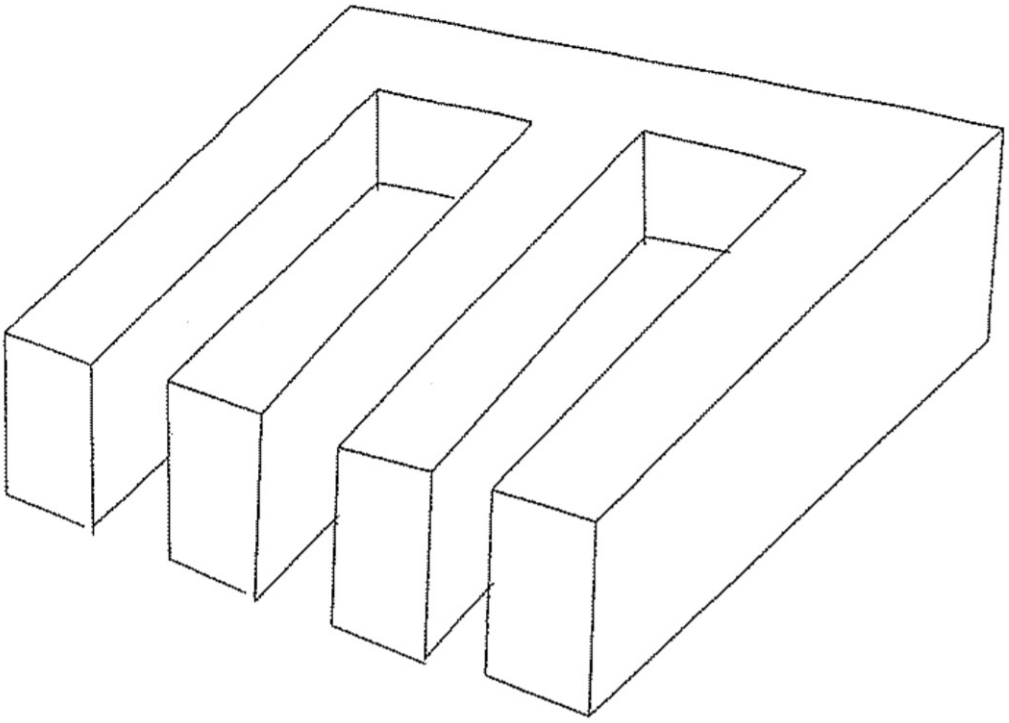
COGS501
Algorithmic Structures in Cognition
Lecture Notes*

Tschomski

September 18, 2020



*These lecture notes are based on the 2019-20 Spring semester classes of Cem Bozşahin.All the lecture videos for this course are *publicly* shared at `https://www.youtube.com/playlist?list=PLfqjcz24ZBbGwcwqUrgnC2BN2taWwuhWC` *(Bozşahin's youtube channel)*

Contents

1 Introducing the Course

Introduction to two foundational concepts in cognitive science: representation of structures and its computation. Mechanistic explanation of structure.

1.1 Description of the Course

The course is an introduction to two foundational concepts in cognitive science: structural representations and computation. As empirical domain, we look at increasingly complex structural representations from morphology and syntax of natural languages. We couple this with an introduction to the theory of computation. We aim to establish that (i) human language capacity is (based on) a computationally describable unconscious system of rules and representations; (ii) that there are mathematically precise ways of talking about different types of structural relations; and (iii) that bringing these two together opens up new avenues in the cognitive scientific investigation of language. Natural language and linguistic knowledge. Language and grammar. Morphology. Syntax and grammatical structure. Semantics: Word meaning and grammatical meaning. Regular expressions and regular grammars. Push-down automata. An introduction to Parsing and derivation.

1.2 Syllabus

Week	Topic
1	Introducing the Course
2	Philosophical Premises and Fundamentals
3	Formal Grammar
4	Context-Free Grammar (CFG)
5	Regular Expressions
6	Finite State Automata (FSA)
7	Removing Non-Determinism
8	Midterm Exam
9	Lambda Calculus
10	Lambda Calculus (Cont'd.)
11	Lambda Calculus (Still Cont'd.)
12	Push Down Automata (PDA)
13	Computational Uncertainty
14	Form, Meaning and Uncertainty in Computation
15	Final Exam

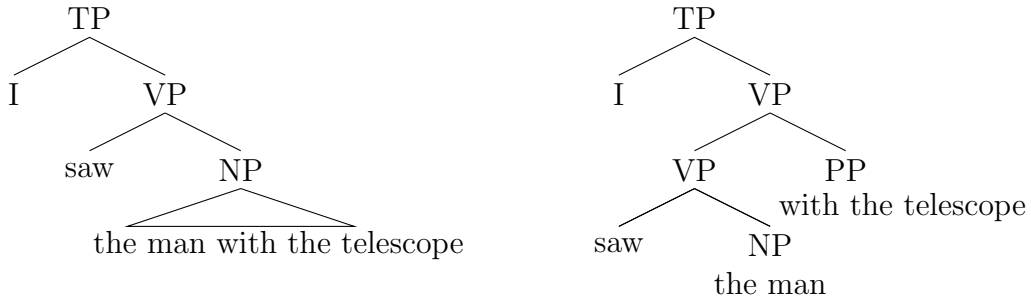
2 Philosophical Premises and Fundamentals

2.1 Ambiguity

Context is ambiguous but structure is not. However, there is structural ambiguity besides contextual ambiguity. The following sentence is ambiguous.

1. I saw the man with the telescope.

The sentence in (1) has two interpretations as shown below.



The different interpretations derive from ***locality***; in other words, sisterhood relations of the phrases. However, it is worth noting that locality is not a physically local relation, yet it is a syntactic dependency relation. See the next sentence.

2. Sen benim adamın kitabını okuduğumu sanıyorsun.

The sentence in (2) shows that the structures with the same color are bound to each other although there is a physical distance between them.

Quick Note: *These explanations seem to be structural, but Structuralism in linguistics means the meticulous investigation of the structures in a language. In this regard, they are philological studies rather than linguistic studies as they don't bring an explanation specific to faculty of language. The process is to set rules and derive the language from these rules. However, in generative (Chomskian) linguistics, the rules (grammar) are directed by the language itself; i.e. language has rules that we cannot discover (we try at least), or can explain partially by revising throughout the time.*

2.2 Formal Studies of Structures

We are trying to come up with mechanisms for problems because rather than focusing on the input and output, we need to see the inside of the black box.

Mechanism	Problem
Simple	Simple
Complex	Complex

Table 1: Marr's Type 1/2

According to Table 1, the following statements are asserted.

- If the problem and mechanism for that are simple, the job is easy.
- If the problem is simple but mechanism is complex, this is not a proper way for science.
- If the problem and mechanism are complex, this is ***Type 2 theory***.
- If the problem is complex and the mechanism is simple, this is ***Type 1 theory***.

Quick Note: *There are two types of sorting: One is comparison sorting such as ordering the number from smaller to larger. The other is non-comparison sorting in which the elements are mapped onto a measure (I use the terms here fuzzily). For example, ordering people according to their height. In this case, height is mapped onto a computable value.*

Marr's Levels[1]

- Computational (goal, constraints, what and understanding the nature of the problem.)
- Algorithmic (algorithms, data structures and representations.)
- Implementation (how it is realized.)

2.3 Formal Definitions

Definition of Natural Numbers
$0 \in \mathbb{N}$
if $n \in \mathbb{N}$ then ${}_s(n) \in \mathbb{N}$
Nothing else is in $\mathbb{N}$

Definition of Even Natural Numbers
$f : 2n \to n$
f is 1-to-1, f is onto, f is total.

Quick Note: *The set of* $\mathbb{N}$ *is an infinite set. Likewise, the set of even natural numbers is also infinite. However, when the sizes of these sets are asked to be compared, it can be intuitively said that one is half of the other, which is not the case, though. Instead, they are both of the same size (infinite).*

- A set is countable if it is equinumerous with some $n \in \mathbb{N}$. Any finite set is countable.
- A set is countably infinite if it is equinumerous with $\mathbb{N}$.
- A set A is equinumerous with a set B if there is a 1-to-1 correspondence from A to B. $g : 2n + 1 \to n$ if $n \in \mathbb{N}$

Complex problem: Instances are too numerous to enumerate (but finite), search algorithm.

[1]Marr, D. (1977). Artificial intelligence—a personal view. *Artificial Intelligence, 9*(1), 37-48.

2.4 Formal Language Theory

Formal study of sets with properties is based on the idea that a well-formed (formal) coverage brings with the amount of computable resources needed.

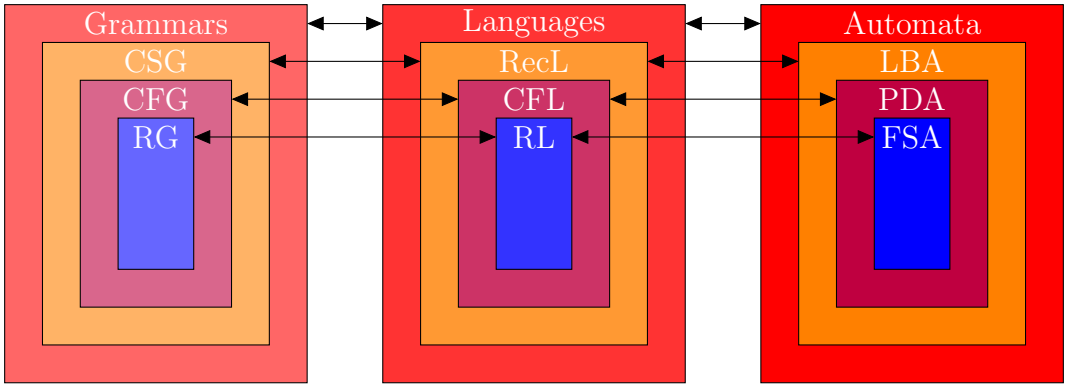


Figure 1: Schema of Grammars, Languages and Automata

Red	Grammars	Languages	Automata
Orange	Context-sensitive Grammar	Recursive Language	Linear-bound Automata
Purple	Context-free Grammar	Context-free Language	Push-down Automata
Blue	Regular Grammar	Regular Languages	Finite-state Automata

Table 2: Languages, Grammars and Automata and their counterparts

Quick Note: *Recursible enumerable languages can be completed by Turing machine. Anything finite is regular. Halting problem (loop) is an unrestricted problem.*

QUIZ 1:

$L = \{I, think, you, like\}$
mini English = Every grammatical sentence in English.
Show that mini English is a countably infinite set.

Answer:

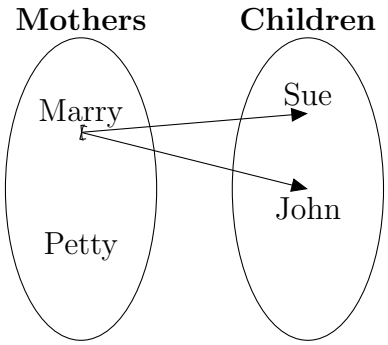
Let $l(n)$ = length of sentence S
$f(2n+3) = n$ where 0 maps onto 3, 1 maps onto 5, 2 maps onto 7 and so on.
This proves that f is 1-1. However, the problem is the sentences which are of the same length such as *"You think I like you"* and *"I think I like you"* whose lengths are 5. The solution is:
Let $g(n)$ = set of all sentences in mini English that are of the length n.
$A = \{I\ think\ I\ like\ you,\ You\ think\ I\ like\ you\ \}$
$g(5)$ maps onto 0, $g(7)$ maps onto 1, $g(9)$ maps onto 2, $g(11)$ maps onto 3 and so on. All those g functions in total constitute mini English.
Let $h(g(2n+3)) = n$, h is 1-1 correspondence. $B = \{$ *"I like you"*, *"You like you"* $\}$

Quick Note: *We need to make a formal definition first, then make the empirical justification.*

3 Formal Grammar

3.1 Formalism and Relation

Formal grammar means two things: Study of formal forms in grammar and the rigorous study of grammar (empirical).
Relation is, different from a function, one to many.



Quick Note: *Not every element in the domain set has to have a correspondence in the range. The mathematical notation of the relation given is:* ⟨ Mary, Sue ⟩ ∈ mother of *or* Mary mother of John.

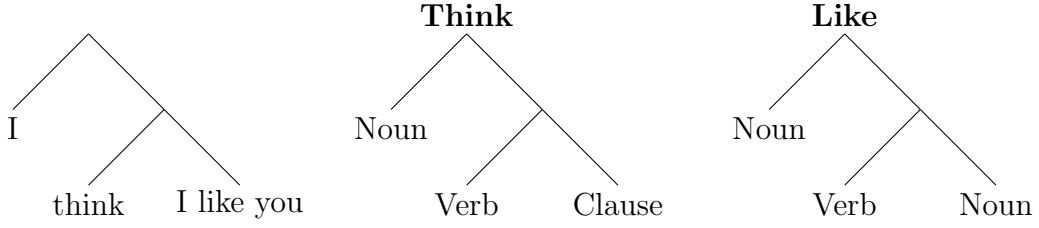
$l(n)$ is an equivalent relation on the length of sentences. An equivalent relation is reflexive, symmetric and transitive.

Reflextive : If $x \in domain(R)$ and $y \in range(R)$, then $x\ R\ y$ and $y\ R\ x$ hold.
Symmetric: If $x \in domain(R)$ then $x\ R\ x$ holds.
Transitive : If $x\ R\ y$ and $y\ R\ z$ then $x\ R\ z$ holds.

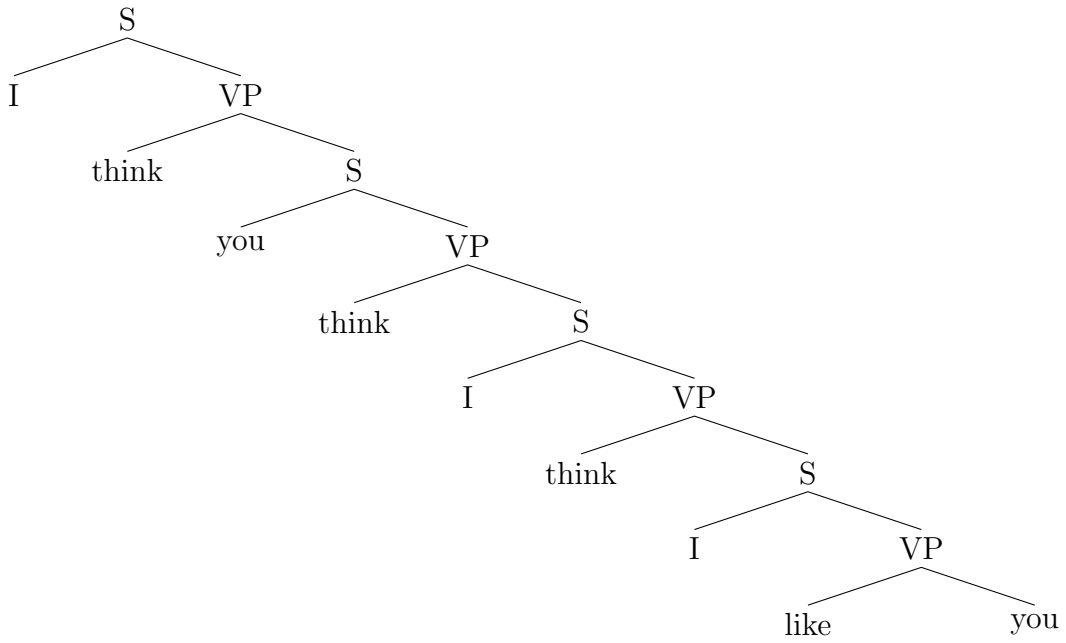
3.2 Formal Study of Structure in mini English

$\Big[$ I think $\big[$ I like you $\big]_{Clause}$ $\Big]_{Clause}$

First Try: All predicates take a subject, but not all predicates take an object.
I think I like you.
*I like I like you.
*I like I think you.



Second Try: I think you think I think I like you.



Third Try: A phrase Structure Grammar of mini English
S → NP VP
NP → I
NP → you
VP → V NP
VP → V S
V → like
V → think

Grammar is a set of rules.

Empirical Problems:

- *"I like I think you"* is bad but grammar says it is OK.
- *"You like I"* is also OK for grammar.

Fourth Try:
V_{tv} → like
V_{psych} → think
VP → V_{tv} NP_{acc}
VP → V_{psych} S
NP_{nom} → You (nominative)
NP_{acc} → You (accusative)
S → NP_{nom} VP
NP_{nom} → I

3.3 Grammars

A ***phrase structure grammar (PSG)*** is the one which all rules can be drawn as trees (phrase markers). Phrase markers are empirical generalizations.

A ***context-free grammar (CFG)*** is a PSG in which we have formal symbols, and siblings are ordered.

Definitions:

Let $G = (V, \Sigma, R, S)$
V: Set of variables (unknowns)
Σ: Set of invariables
R: Set of rules
S: Start symbol ($s \in V$)

a ***CFG*** is where all rules in R are $\alpha \to \beta$;
if $\alpha \in V$ and $\beta \in (V \cup \Sigma)^*$
then α

β is a tree.

3.4 Sets and Concatenations

Let A and B sets, $A \cup B$, $A \cap B$, $A - B$, $AB \to$ set concatenation.
For example, $A = \{0, 1, 2\}$ and $B = \{a, b, c\}$; $AB = \{0a, 0b, 0c, 1a, 1b, 1c, 2a, 2b, 2c\}$

Definition:
$AB = \{x_1, x_2 | x_1 \in A, x_2 \in B\}$
Concatenation of A, $A^* = A^0 \cup A^1 \cup A^2 \cdots \cup A^i$

> *"What is math? Algebra, Calculus. That's it!"*
> (Bozşahin)

$A^0 = \{e\}$ and $\forall A$, $A^0 \neq \emptyset$

Example:
S $\to$ NP VP
VP $\to$ V NP
VP $\to$ V S
NP $\to$ I | You

Variables= $\{S, VP, ZP, V\}$
$\Sigma = \{I, like, you, think\}$
$R = \{S \to NP\ VP, VP \to V\ NP\ \}$

Quick Note: $V \to non - terminals$, $\Sigma \to alphabet\ terminals$

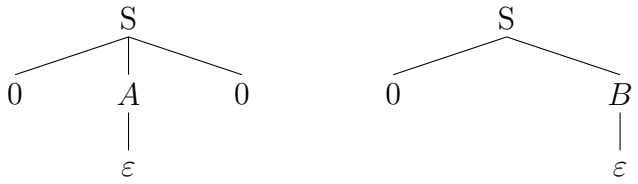
QUIZ 2:

Write a context-free grammar defined over the alphabet $\Sigma = \{0, 1\}$ such that all the strings start with 0 and end with 0. For instance, "010110" $\in L_1$ but "100010" $\notin L_1$.

Answer:

L = set of strings in Σ^* that start and end with 0.
L_1 start and end with with different 0s.
L_1 start and end with with the same 0, which includes the string "0"

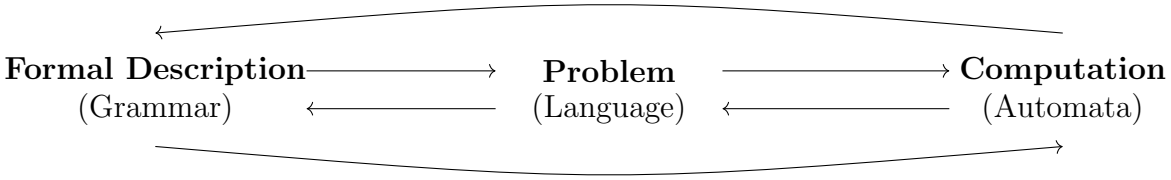
Grammar for L_1: All and only the strings in L_1 (completeness and soundness).
G_1: $S \to 0A0$, $A \to 0A|1A|\varepsilon$
G_2: $S \to 0B$, $B \to C0|\varepsilon$, $C \to C|0C|\varepsilon$



The first tree generated with G_1 does not accept the string which starts and ends with the same 0. However, the second tree generated with G_2 solves the problem.

4 Context-Free Grammars

4.1 Problem of Computation

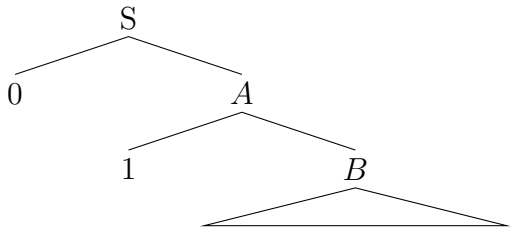


Grammar rules $\alpha \to \beta$ (α is β, α rewrites β). If $|\alpha| = 1$ then we have a CFG (gives us a tree).

$G = \{V, \Sigma, R, S\}$

4.2 Languages and Grammars

Let L_2 has the grammar G_2 which has the phrase markers $S \to 0A$, $A \to 0A|1B|\varepsilon$, $B \to 0A|1B$, show that the string *"011"* is not a string that can be generated with G_2 in parse tree.



This tree will never generate the above-mentioned string. Similarly, the grammar of the language in **Quiz 2** behaves the same. Therefore, although the string is the same, structure is different, which indicates that *if L_1 and L_2 are not the same, then they cannot have equivalent grammars. For a given language, there can be many equivalent grammars.*

Definition:
G_1 and G_2 are weakly equivalent: $f : L(G_1) = L(G_2)$

Example:
Let the language L_3 have the alphabet $\Sigma = \{a, b\}$. That language includes all strings where as do not follow bs.

$\varepsilon, a, b, ab \in L_3$ and $ba, aabba \notin L_3$

$S \to A\ B$**,** $A \to aA|\varepsilon$**,** $B \to bB|\varepsilon$ **(Divide and Conquer!)**

Quick Note: *Only regular languages have unique description.*

Example:
Let the language L_4 have the alphabet $\Sigma = \{a, b\}$. That language includes all strings where the number of a's is equal to that of b's.

1. $G_4' : S \rightarrow aSb|\varepsilon$
2. $G_4'' : S \rightarrow A$, $A \rightarrow aB|\varepsilon$, $B \rightarrow Ab|b$

QUIZ 3:

Let the language L_5 have the alphabet $\Sigma = \{a, b, c\}$. That language includes all strings where there are twice as many b's as a's ($L_5 = \{a^n c b^{2n}\}$). Write the grammar of L_5.

Answer:

$S \rightarrow aSbb \mid c$

S $\longrightarrow$ "$aacbbbb$"
|
$aSbb$
|
$aSbb$
|
c

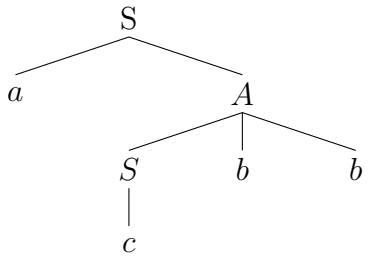
5 Regular Expressions

5.1 Language of a Grammar

Language of a grammar is $L(G) = \{x \mid x \in \Sigma^*, S \overset{*}{\Rightarrow} x\}$

Quick Note: *Double arrow indicates* ***use of a rule****.*

Example:
$acbb \in L$



$S \overset{1}{\Rightarrow} aA \overset{2}{\Rightarrow} aSbb \overset{3}{\Rightarrow} acbb$

$abb \notin L$
$S \overset{1}{\Rightarrow} aA \overset{2}{\Rightarrow} aSbb$, this is not going to lead you to the desired string.

5.2 Different Languages and Grammars

1 counter $L_1 = \{a^n, b^n\}$
2 counters $L_2 = \{a^n, b^m, c^m, b^n\}$
3 counters $L_2 = \{a^n, b^m, c^a, r^a, c^m, b^n\}$

No matter how many counters there are, the problem is the same.

$S_1 \to aS_1b|\epsilon$
$S_2 \to aS_2b|A$, $A \to bAc|\epsilon$
$S_3 \to aS_3b|A$, $A \to bAc|B$, $B \to cBr|\epsilon$

BUT the real problem is something different.

1. $\{a^nb^m|n, m \geq 0\}$
2. $\{a^nb^n|n \geq 0\}$
3. $\{a^nb^nc^n|n \geq 0\}$

1. G_1: $S \to aS|A$, $A \to bA|\epsilon$
2. G_2: $S \to aSb|c$
3. *No context-free grammar for this!*

Some problems *(≡ languages ≡ sets with structure)* are easy enough so that multiple descriptions can be captured by a unique automaton *(Regular expressions)*. Some problems are more resource demanding, so there is no guarantee for unique computation.

$L = \{a^i | \ i \text{ is a prime number.}\} \longrightarrow$ **impossible to write a CFG for this problem.**

Regular languages are a proper subset of context-free languages. There is a regular grammar for every language. A grammar is regular if all rules are of tree forms $A \to aB$ (right regular) or $A \to Ba$ (left regular).

5.3 Regular Expressions (Regexp)

Let $\Sigma = \{a, b\}$

1. For $\forall a \in \Sigma \cup \{\epsilon\}$, α is a regular expression.
2. If α, β are regular expressions, then $\alpha\beta$ (concatenation of two expressions), $\alpha \cup \beta$ (union of two expressions), α^* (k-star) are regular expressions as well.
3. Nothing else generated without *(1)* and *(2)* are regular expression.

Example 1:
$\{a^n b^m | n, m \geq 0\}$
Regexp: a^*b^*

Example 2:
A language where no a follows b.
Regexp: a^*b^*

Example 3:
$\Sigma = \{a, b, c\}$, All and only strings without the substring bb.
Regexp: $(a|c|b(a|c))^*(b|\epsilon)$

> *Math is as useful as art. Math is black art.*
> (Bozşahin)

Quick Note: $\alpha^+ = \alpha\alpha^* \longrightarrow$ ***at least one*** *alpha.*

Example 4:
$\Sigma = \{0, 1\}$, $L =$ All and only strings in Σ^* where every third symbol is 1.
Regexp: $((0|1)(0|1)1)^*(0|1|\epsilon)^2$

Example 5:
$\Sigma = \{a, b, c\}$, $L =$ All and only strings that do not contain the substring ac.
Regexp: $(b|c|a^*b)^*(a|b)^*$

Quick Note: *If $X \subseteq Y$ and $Y \subseteq X$, then $X = Y$.* X and Y are countably infinite.

> *Grammars give you "what", Automata give you "how".*
> (Bozşahin)

Example 5:
$\Sigma = \{\llcorner, \urcorner, c\}$, All and only squares out of Σ.
Regexp: $(\llcorner\urcorner c)^*$

QUIZ 4:

Let $\Sigma = \{a, b, c\}$, given $L =$ All and only strings with even number of a's.

Answer:

$((b|c)^*(a(b|c)^*a)^*(b|c)^*)^*$

6 Finite State Automata

6.1 From Description to Computation

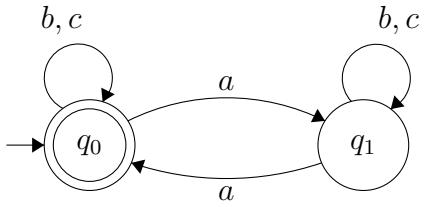
Theorem
For every language, there is a finite state automaton that computes its strings.
For every regular language, there is a regular expression.
For every regular language, there is a regular grammar.

A ***finite state automaton (FSA)*** is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time.[2]

$$M = \langle Q, \Sigma, \Delta, q_0(s), F \rangle$$

Q: Finite set of states
Σ: Alphabet
Δ: Finite set of transitions
q_0: Start state $q_0 \in Q$
F: Set of final (accepting) states $F \subseteq Q$

Example
M_1: All and only strings with even number of a's.



$$M_1 = (\{q_0, q_1\}, \{a, b, c\}, \{\{q_0, a, q_0\}\{q_0, b, q_1\}\{q_0, c, q_1\}\{q_1, a, q_0\}\{q_1, b, q_0\}\{q_1, c, q_0\}\}, q_0, \{q_0\})$$

One-step computation ($\vdash$): how we advance the configuration *(all pieces of information needed)* of computation.
Call *(q,w)* a **configuration** of an fa M, where q is the current state and w is the string on the tape including the current symbol and what lies to the right of it.[3]

A configuration *(q,w)* is such that $p \in Q$ and $w \in (\Sigma \cup (\{\epsilon\})^*$

Initial Configuration: *(s, input)*
Accepting Configuration: *(p, ϵ)* if $p \in F$

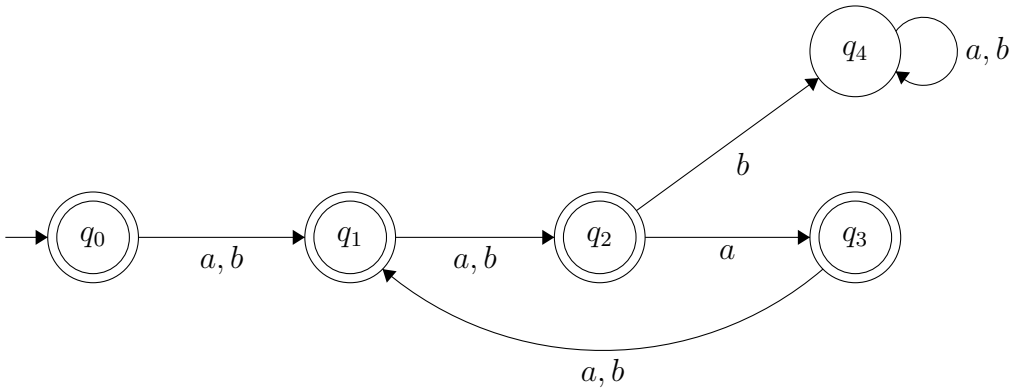
L(M), language of a machine, M=$\{x | x \in \Sigma^*$ and $(s, x) \overset{*}{\vdash} (p, \epsilon)$ $p \in F$

Use of "$\vdash$"
$a \in \Sigma$, $w \in \Sigma^*$ $(r, a, w) \vdash (q, w)$ if $(r, a, q) \in \Delta$

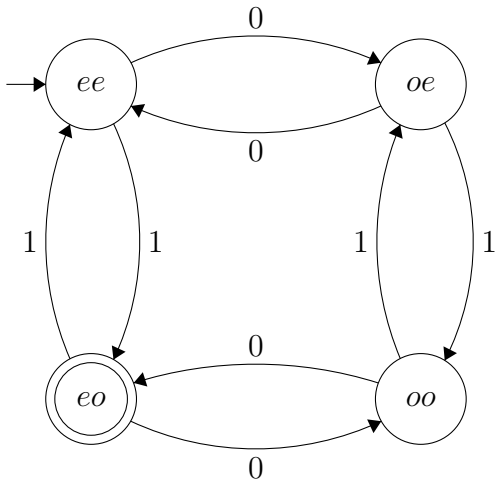
[2] `https://en.wikipedia.org/wiki/Finite-state_machine`
[3] Umut Özge, COGS501 Lecture Notes

Example
L_1 = All and only strings where every third symbol is a.

$(a|b)(a|b)a)^*(a|b|\epsilon)^2$



Example
L_1 = All and only strings with even number of *0*'s and odd number of *1*'s.
$\Sigma = \{0, 1\}$



Quick Note: *In deterministic finite automata, the next configuration is just one, in non-deterministic automata (NFA); on the other hand, there are more than one configuration. NFA give you expressivity but no extra power.*

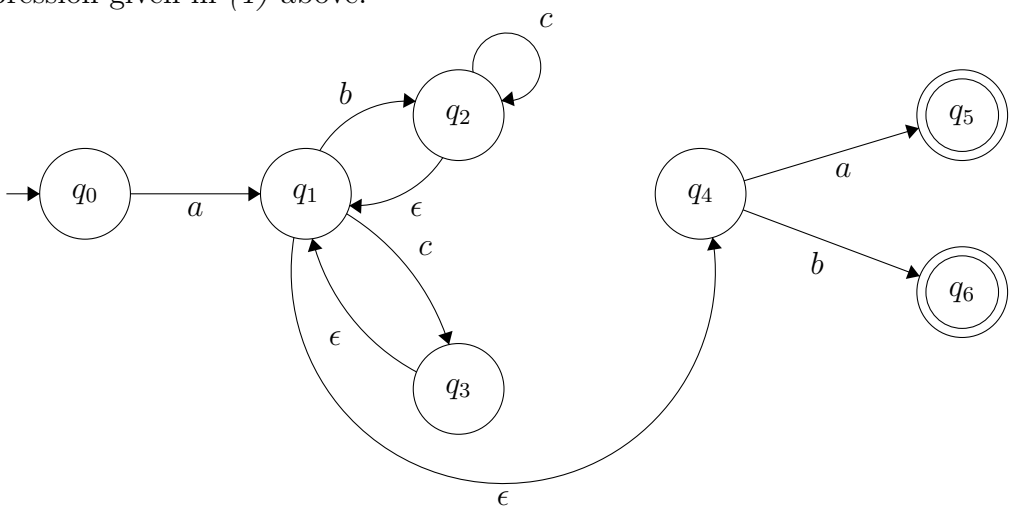
Example

1. $\overline{\overline{a}^{m_1}(b\overline{c^*}^{m_2}|c)^*(\overline{a}^{m_3}|b)}$

- Understand the language and write the machine.
- Write the NFA then turn it into DFA.

Thompson's Construction (Compositional)
First draw the individual machines then combine the final state of one machine to the initial state of the other machine. See the following diagram and observe how the different

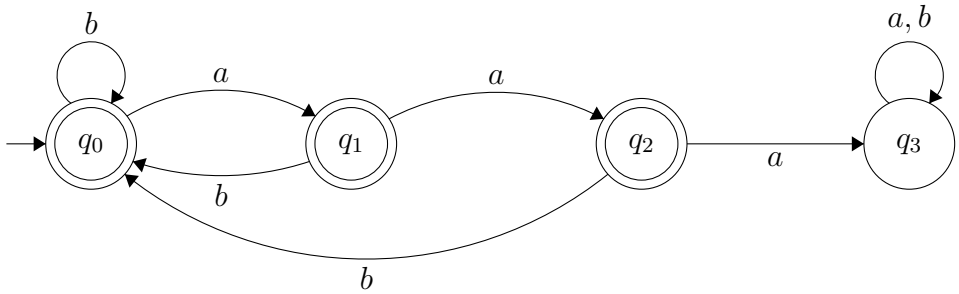
machines are combined with each other to generate the strings acceptable for the regular expression given in *(1)* above.



QUIZ 5:

Let $\Sigma = \{a, b\}$, and given the language $L =$ All and only strings which do not include the substring aaa. Write a machine for L.

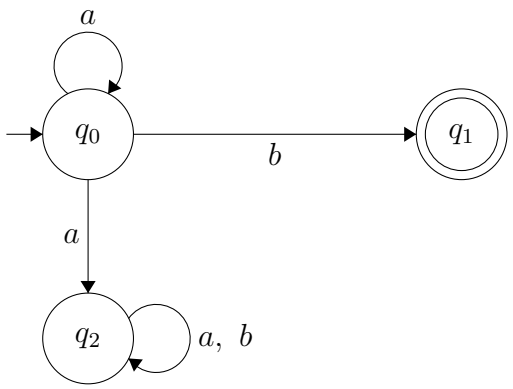
Answer:



7 Removing Non-Determinism

7.1 Non-Deterministic Finite Automata

Let the NFA defined over the alphabet $\Sigma = \{a, b\}$.



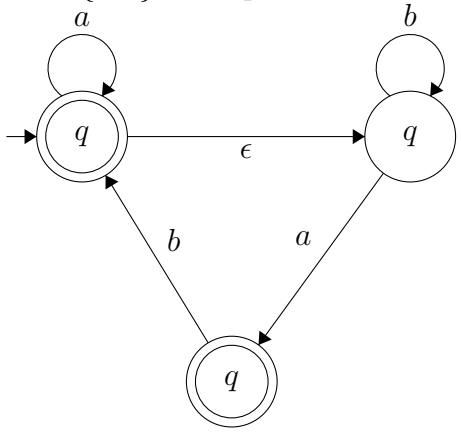
- The language of the NFA, $L(M) = \{ \text{x} \mid (q_0, x) \overset{*}{\vdash} (q_f, \epsilon)\}$
- The grammar of the language, $L(G) = \{ x \mid S \overset{*}{\Rightarrow} x\}$
- If your description is right and your machine is good, the sets of $L(M)$ and $L(G)$ will be equivalent.

7.2 Removing non-determinism from finite-state computation

- Let NFA, $M = \langle Q, \Sigma, \Delta, s, F\rangle$.
- Define DFA, $M' = \langle Q', \Sigma, \delta, s', F'\rangle$.
- Such that $L(M) = L(M')$

Example

Let $\Sigma = \{a, b\}$ the alphabet of the language generated by the NFA, M below.



To eliminate the non-determinism, first what you need is $\epsilon - closure(q) = q\epsilon Q$. The closure of ϵ is all the states we can reach from q by using $\epsilon - transitions$ only.

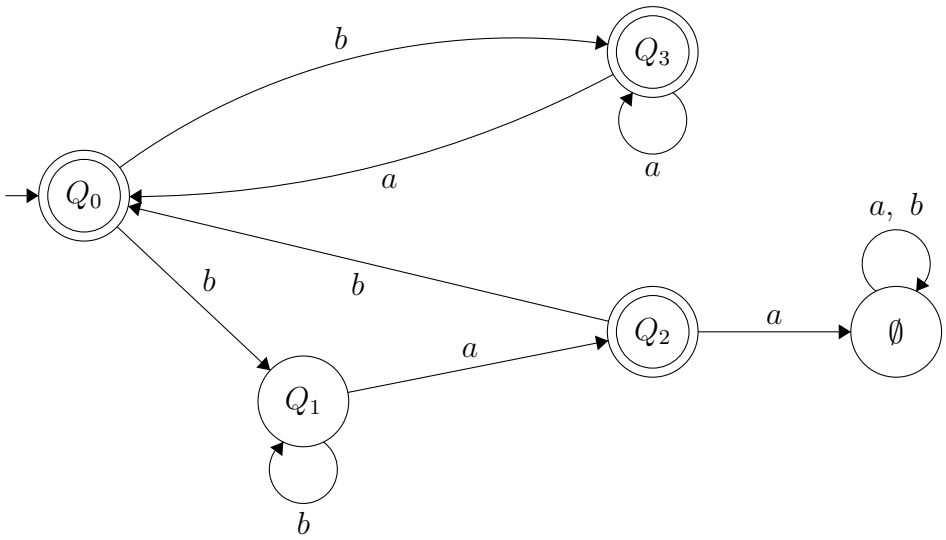
$\epsilon - closure$	
q0	$\{q_0, q_1\}$
q1	$\{q_1\}$
q2	$\{q_2\}$

$s' = \epsilon - closure(s)$

t = input transition function. $t(Q, a)$ = all the states we can go from Q on a before and after ϵ-transition.

t	a	b
q_0	$\{q_0, q_1, q_2\}$	$\{q_1\}$
q_1	$\{q_2\}$	$\{q_1\}$
q_2	$\emptyset$	$\{q_0, q_1\}$

The DFA, M' of the NFA, M above is shown below. The states of the machine are Q_0=$\{q_0, q_1\}$, Q_1=$\{q_1\}$, Q_2=$\{q_2\}$ and Q_3=$\{q_0, q_1, q_2\}$.



$\delta(\{q_0, q_1\}\, a) = t(q_0, a) \cup t(q_1, a)$

Quick Note: *When determining the final states of the DFA, the states which include final states of the NFA are selected. For example, in the NFA, q_0 and q_2 are the final states. Therefore, any state in the DFA which include these states are determined as the final states of the DFA.*

QUIZ*:

*Because of the Covid-19 Pandemic, the lectures started to be conducted via online media. Therefore, the quiz at the end of the class was not given this week.

8 Midterm Questions and Answers[4]

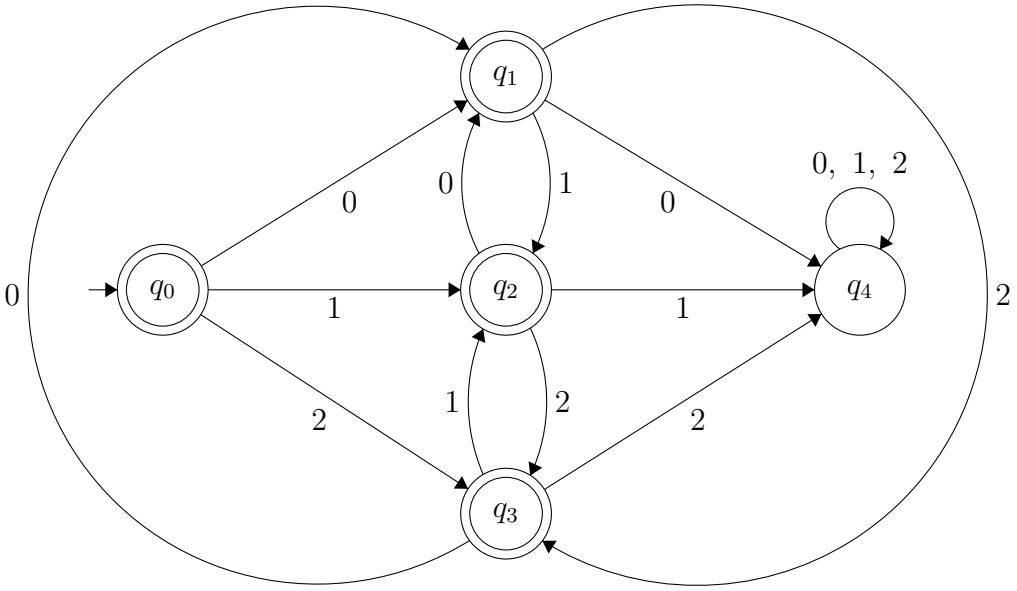
Q1/4. Consider the following finite automaton M1. Is it a DFA? In the simplest English description you can think of, what is the language of this machine?

$M_1 = (Q, \Sigma, \Delta, q_0, \{q_0, q_1, q_2, q_3\})$ where $Q = \{q_0, q_1, q_2, q_3, q_4\}$ $\Sigma = \{0, 1, 2\}$ and

$\Delta = \{(q_0, 0, q_1), (q_0, 1, q_2), (q_0, 2, q_3), (q_1, 0, q_4), (q_1, 1, q_2), (q_1, 2, q_3), (q_2, 0, q_1),$
$(q_2, 1, q_4), (q_2, 2, q_3), (q_3, 0, q_1), (q_3, 1, q_2), (q_3, 2, q_4), (q_4, 0, q_4), (q_4, 1, q_4), (q_4, 2, q_4)\}$

Q1/4. The finite automaton M_1 is a DFA because each possible input determines the following state uniquely. In other words, the current state of the machine is changed by the input, and that input is totally responsible for determining that particular state. For example, if the machine M_1 is in the state q_0 and reads the input 1, it goes to no other state but q_2. The same is valid for the other state-input combinations such as the fact that the combination q_2 and 2 deterministically goes to q_3 and so on. Therefore, the machine M_1 is a DFA.

The language of this machine $L(M_1)$ is all and only the strings where no symbol occurs twice or more consecutively.



Q2/4. Now try to express the language above formally. In other words, write a regular expression for $L(M_1)$.

Q2/4.

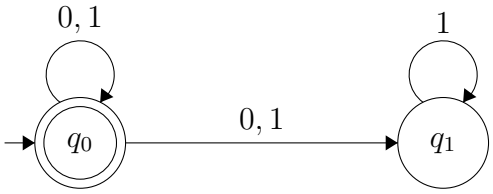
$$\Big(0(1 \mid 2) \mid 0(\ (12)^+(1 \mid \epsilon) \mid (21)^+(2 \mid \epsilon)\ )\Big)^* \Big(0 \mid \epsilon\Big) \mid$$
$$\Big(1(0 \mid 2) \mid 1(\ (02)^+(0 \mid \epsilon) \mid (20)^+(2 \mid \epsilon)\ )\Big)^* \Big(1 \mid \epsilon\Big) \mid$$
$$\Big(2(0 \mid 1) \mid 2(\ (10)^+(1 \mid \epsilon) \mid (01)^+(0 \mid \epsilon)\ )\Big)^* \Big(2 \mid \epsilon\Big)$$

[4]These questions do NOT reflect the real exam questions, and the answers are NOT the correct answers. They are all given just to provide a reference.

Q3/4. Eliminate non-determinism from the following NFA $M_3 = (\{q_0, q_1\}, \{0, 1\}, \Delta_3, q0, \{q_0\})$ where $\Delta_3 = \{(q_0, 0, q_0), (q_0, 0, q_1), (q_0, 1, q_0), (q_0, 1, q_1), (q_1, 1, q_1)\}$.

Q3/4.

The NFA:



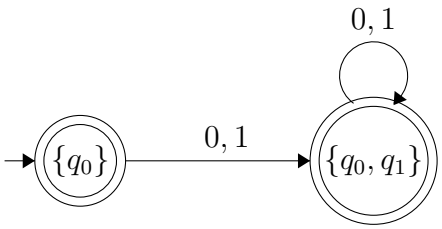
ϵ-closure states: I assumed that by using ϵ we can only stay in the current state, although ϵ-transitions are not explicitly stated; actually there is none. Therefore, I have left the table anyway.

	ϵ
q_0	$\{q_0\}$
q_1	$\{q_1\}$

Input transition function *(t)*:

t	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_1\}$
q_1	$\emptyset$	$\{q_1\}$

The DFA, M_3' of the NFA, M_3:



M_3'=$(\{\{q_0\}, \{q_0, q_1\}\}, \{0, 1\}, \delta_3, \{q_0\}, \{\{q_0\}, \{q_0, q_1\}\})$ where
$\delta_3 = \{(\{q_0\}, 0, \{q_0, q_1\}), (\{q_0\}, 1, \{q_0, q_1\}), (\{q_0, q_1\}, 0, \{q_0, q_1\}), (\{q_0, q_1\}, 1, \{q_0, q_1\})\}$

Q4/4. The grammar below captures a small fragment of our arithmetic, of addition and multiplication of single-digit numbers. For example, 5+(3x2) is well-formed, whereas 5+x(32) is not.

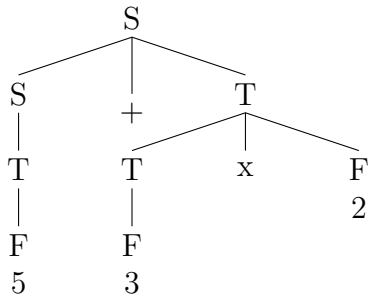
$S \rightarrow S + T \mid T$
$T \rightarrow T + F \mid F$
$F \rightarrow 0|1|\cdots|9|(S)$

First, draw a tree for the well-formed example above using this grammar.
Second, it seems that it captures more than just well-formedness of these simple arithmetic expressions. To see this, compare the following grammar for the same language:

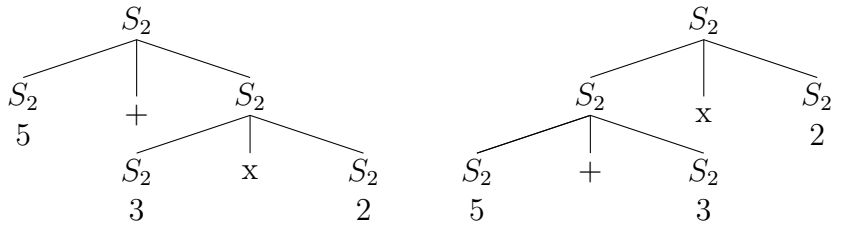
$S_2 \rightarrow S_2 + S_2|S_2 x S_2|0|1|\cdots|9|(S_2)$

What is the property that grammar of (1-3) captures and (4) does not? Would you call this a semantic property, structural, both, or what?

Q4/4.



Unlike the first one, the second grammar cannot differentiate *the order of the operations* (+ and x), which creates an ambiguity in that the grammar accepts not only $5 + (3x2)$ but also $(5 + 3)x2$ without parentheses, seeing the trees below. I would call this as a structural ambiguity which has a semantic implication because this ambiguous structure causes different end results; that is, the meaning of the strings are not the same although they have the same order. Therefore, the property of the first grammar is both structural and semantic.



9 Lambda Calculus

Lambda Calculus, coined by **Alonzo Church** is a glue language for syntax-semantics connection and a theory of substitution. According to **Frege's Principle of Compositionality**, the meaning of the whole is a function of the meaning of the parts and the way they are combined. Emmon Bach's Rule-to-Rule Thesis (1976) stated that a semantic rule accompanies the application of a syntactic rule. Semantic rules are functions and arguments.

9.1 Conversions in λ-Calculus

There is a set of conversions in λ-calculus. Conversion means any abstraction or reduction.

- λ **conversion**
- β **conversion:** $(\lambda x.M]N = \beta M[N/x]$ (substitution of N for free occurrences of x in M) *e.g.:* $(\lambda x. - x\ 2)3 = \beta(-x\ 2)[3/x] = (-3\ 2) = 1$
- α **conversion**
- η **conversion**

9.2 λ-terms

There are some λ-terms.

- **Constants** *(0, 1, John, ...)*
- **Variables** *(x, y, x_1, ...)*
- If M is a λ-term, and x is a variable, then $\lambda x.M$ is a λ-term.
- If M and N are λ-terms, MN is also a λ-term.

Quick Note: *Arguments themselves can be functions as well.*

$\lambda x.x^2$ applies square function. $(\lambda x.x2)sqr = (x\ 2)[sqr/x] = sqr\ 2$

$sqr = \lambda y.x\ yy$, $(\lambda z.zz)(\lambda y.x\ yy) = \beta(z\ z)[(\lambda y.x\ yy)/z] = (\lambda y.x\ yy)z = \beta(x\ yy)[z/y] = (x\ zz)$

Quick Note: *Form-meaning correspondence can be implemented by λ-calculus.*

9.3 Beta-Reducible Expression (β-Redex)

A λ-term is a β-redex if it is of the form $(\lambda x.M)N$ which can be reduced to the form $M[N/x]$.

Here is the problem. What if there is more than one β-redex? Which one is evaluated first?. There are two kinds of order of evaluation:

1. Normal Order of Evaluation (NOE): Evaluate outermost and leftmost β-redex first.
2. Applicative Order of Evaluation (AOE): Evaluate the innermost leftmost β-redex first.

Church-Russer Theorem of Property: If a λ-term has a normal form (a form which is not further reducible), then NOE and AOE give the same normal order.

Take the following λ-term $(\lambda g.\lambda x.gx)((\lambda y.+\ y\ 2)\ 3)$

By AOE, we get $(\lambda x.((\lambda y.+\ y\ 2)\ 3)\ \text{x}) = \lambda x.(+\ 3\ 2)\ x$

By NOE, we get $(\lambda g.gx)(+\ 3\ 2) = \lambda x.(+\ 3\ 2)\ x$

Some λ-terms have no normal form. e.g.: $(\lambda x.xx)(\lambda y.yy) =_\beta (\lambda y.yy)(\lambda y.yy)$ when this expression β-reduced again, the result will be the same, so there is an endless loop here.

9.4 Formal Capture of Structure in Cognition

- Syntax (form) $\rightarrow$ Automata, CFG
- Semantics (meaning) $\rightarrow$ Lambda Calculus
- Uncertainty (inference) $\rightarrow$ Probability, Parsing

The combination of Syntax and Semantics in Mini-English

$S : \lambda x\lambda y.yx \rightarrow NP : np^{'}\ VP : vp^{'}$
$VP : \lambda x\lambda y.xy \rightarrow V_T : v^{'}\ NP : np^{'}$
$VP : \lambda x\lambda y.xy \rightarrow V_C : v^{'}\ S : s^{'}$
$V_T : \lambda x.x \rightarrow like : \lambda x\lambda y\ like^{'}\ xy$
$V_C : \lambda x.x \rightarrow think : \lambda x\lambda y\ think^{'}\ xy$

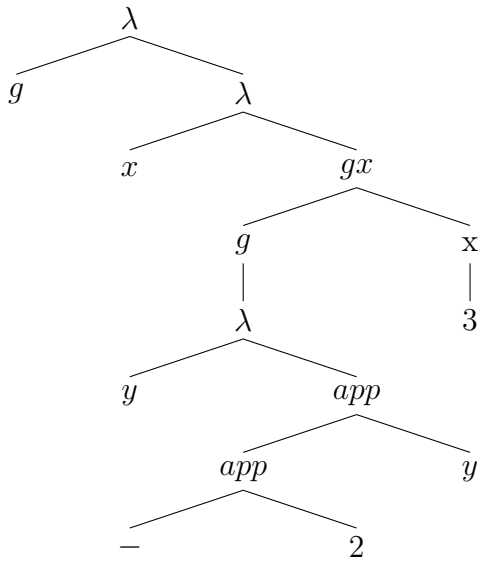
QUIZ 6:

Consider the following lambda term, $(\lambda g.\lambda x.gx)(\lambda y.-\ 2\ y)(3)$

Q1. Write the lambda term in tree notation.
Q2. Evaluate the lambda term using beta-reduction.

Answer:

Question 1:



Question 2:

$(\lambda g.\lambda x.gx)(\lambda y.-\ 2\ y)(3)$
$=_\beta ((\lambda x.gx)[(\lambda y.-\ 2\ y)/g])(3)$
$= (\lambda x.(\lambda y.-\ 2\ y)x)(3)$
$=_\beta ((\lambda y.-\ 2\ y)x)[3/x]$
$= (\lambda y.-\ 2\ y)(3)$
$=_\beta (-\ 2\ y)[3/y]$
$= -\ 2\ 3$
$= (-1)$

QUIZ 7:

Let's take the following lambda term, $((\lambda f.f\ 4)(\lambda h.h\ 2))((\lambda g.g\ 3)\ +)$

1. Reduce the term using Normal Order Evaluation.
2. Reduce the term using Applicative Order Evaluation.

Answer:

Normal Order Evaluation:[5]
$\underline{((\lambda f.f\ 4)(\lambda h.h\ 2))}((\lambda g.g\ 3)\ +)$
$=\underline{((\lambda h.h\ 2)\ 4)((\lambda g.g\ 3)\ +)}$
$=\underline{((((\lambda g.g\ 3)\ +)\ 2)\ 4)}$
$=(((+\ 3)\ 2)\ 4)$

Applicative Order Evaluation:[6]
$\underline{((\lambda f.f\ 4)(\lambda h.h\ 2))}((\lambda g.g\ 3)\ +)$
$=\underline{((\lambda h.h\ 2)\ 4)}\underline{((\lambda g.g\ 3)\ +)}$
$=\underline{((\lambda h.h\ 2)\ 4)}\underline{(+\ 3)}$
$=(((+\ 3)\ 2)\ 4)$

[5] Leftmost outermost β-redexes are underlined.
[6] Leftmost innermost β-redexes are underlined.

10 Push Down Automata (PDA)

PDA is a finite state machine with a stack. PDA, $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ where Γ includes the stack symbols, the rest is the same as those of an FSA.

Configuration of a PDA: (current state, remainder of input, current content of the stack)

Elements of Δ: $((p, a, A), (q, B))$ where p is the current state, a is the current symbol, A is the top of stack, q is the next state and B is the new top of the stack.

In current state p , scan a, pop A, go to q, push B; where $p, q \in Q$, $a \in \Sigma^* \cup \{\epsilon\}$, $A, B \in \Gamma^* \cup \{\epsilon\}$

Example: $L = \{a^n b^n | n \geq 0\}$

$G(L) = S \rightarrow aSb|\epsilon$

Ad hoc PDA construction:

$\Delta = ((p, a, \epsilon), (p, A99, ((p, b, A), (q, \epsilon)), ((q, b, A), (q, \epsilon))$

$L(M) = \{w \in \Sigma^* | (s, w, \epsilon) \vdash^* (p, \epsilon, \epsilon), p \in F\}$

10.1 Algorithm for PDA Construction from CFG

Let CFG $G = (V, \Sigma, R, S)$ where all rules in R are $\alpha \rightarrow \beta$, $\alpha \in V$ and $\beta \in (V \cup \Sigma)^*$

Define PDA, $M = (\{p, g\}, \Sigma, V \cup \Sigma, \Delta, p, \{q\})$

For every $\alpha \rightarrow \beta$ in R,
Define $((p, \epsilon, \beta^R), (p, \alpha))$ in Δ

For every $a \in \Sigma$,
Define $((p, a, \epsilon), (p, a))$ in Δ
Define $((p, \epsilon, S), (q, \epsilon))$, then $L(G) = L(M)$

QUIZ 8:

Let us take a small fragment of arithmetic grammar, this time using semantic rules as well.

$S : \lambda x.\lambda y.\lambda z.yxz \rightarrow S : s^{'} + : + T : t^{'}$
$S : \lambda x.x \rightarrow T : t^{'}$
$T : \lambda x.x \rightarrow F : f^{'}$
$F : \lambda x.x \rightarrow 3{:}3 \mid 5{:}5$

1. Construct the PDA using the algorithm.
2. Compute the result of 3+5 using the PDA. Show use of syntactic rule and semantic action where appropriate.

Answer:

Question 1:

$((p, 3, \epsilon), (p, 3))$
$((p, 5, \epsilon), (p, 5))$
$((p, +, s^{'}), (p, +))$
$((p, \epsilon, 3), (p, F : (\lambda x.x)3 = f^{'}))$
$((p, \epsilon, 5), (p, F : (\lambda x.x)5 = f^{'}))$
$((p, \epsilon, F), (p, T : (\lambda x.x)f^{'}) = t^{'})$
$((p, \epsilon, T), (p, S : (\lambda x.x)t^{'}) = s^{'})$
$((p, \epsilon, t^{'} + s^{'}), (p, S : ((\lambda x.\lambda y.\lambda z.yxz)s^{'} + t^{'})))$
$((p, \epsilon, S), (q, \epsilon))$

Question 2:

$(p, 3 + 5, \epsilon) \vdash$
$(p, +5, 3) \vdash$
$(p, +5, F : (\lambda x.x)3) \vdash$
$(p, +5, T : (\lambda x.x)f^{'}) \vdash$
$(p, +5, S : (\lambda x.x)t^{'}) \vdash$
$(p, +5, s^{'}) \vdash$
$(p, 5, +s^{'}) \vdash$
$(p, \epsilon, (F : (\lambda x.x)5) + s^{'}) \vdash$
$(p, \epsilon, (T : (\lambda x.x)f^{'}) + s^{'}) \vdash$
$(p, \epsilon, t^{'} + s^{'}) \vdash$
$(p, \epsilon, S : ((\lambda x.\lambda y.\lambda z.yxz)t^{'} + s^{'})) \vdash$
$(p, \epsilon, S) \vdash$
(q, ϵ) and accept.

11 Computational Uncertainty

QUIZ 8:

In Part 2 of week 13 video, from 6:24 to 8:44 I show a joint distribution of a sentence and tree as product of probabilities of rule use. If you look at how I conditionalize the probabilities, you will see that it assumes that the tree is built bottom-up.

Question 1: Comment on why this is the case.
Question 2: If the tree is to be built top-down, what would the joint distribution be, in terms of product of conditional probabilities?

Answer:

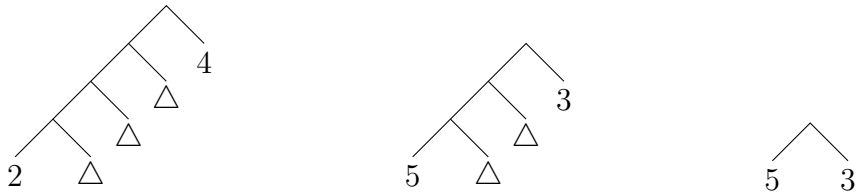
Question 1: The tree is built bottom-up because the units that constitute the sentence such as NP, VP and PP are assumed to form the sentence, and they are assumed to be formed by other units such as V and P_{with} etc. Therefore, if the tree was built in top-down fashion, the joint probability of the tree would be determined by the surface items such as *I, saw, telescope, the etc.* given the sentence S and the phrase they constitute. Therefore, the joint probability of the sentence would be calculated from these surface items if the tree were built in top-down assumption. Although the build-architectures are different, this will not affect the joint probability (but I cannot be sure as we do not have any value indicating the probability values of the phrases and surface items).

Question 2: $P(T|S) = P(S|I) \times P(NP|I) \times P(S|saw) \times P(VP_1|saw) \times P(VP_2|saw) \times P(V|saw) \times P(S|the\ man) \times P(VP_1|the\ man) \times P(VP_2|the\ man) \times P(NP|the\ man) \times P(P_{with}|with) \times P(PP|with) \times P(VP_1|with) \times P(S|with) \times P(NP|the\ telescope) \times P(PP|the\ telescope) \times P(VP_1|the\ telescope) \times P(S|the\ telescope)$

12 Final Exam Questions and Answers

QUESTIONS:

We are given a language of the sequence of the symbol '△' separating two single-digit natural numbers. In the general form the input is n is $n\triangle\cdots\triangle m$. It means $(((n^2)^2)\cdots^2)-m$, where the number of squaring is the number of occurrences of △ in the sequence. For example, $5\triangle\triangle 3$ means 625-3 because $((5^2)^2)-3=625-3$, and $2\triangle\triangle\triangle 4$ means $256-4$. The sequence 5 3 means $5-3$. We would expect the following trees for these examples:



The following grammar can generate these trees once we treat '△' as the square function:

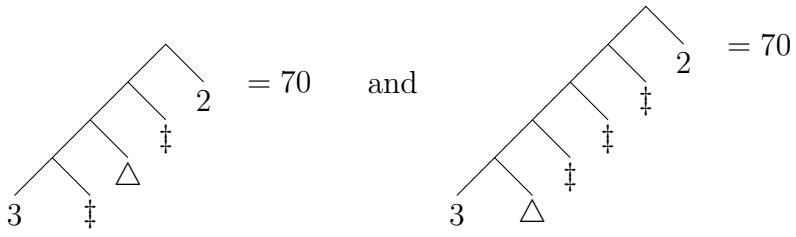
$S \to AN$ where N is a natural number between 0 and 9.
$A \to A\triangle$
$A \to N$

1. (10 points) If we were only interested in the form, this would be a regular language. Write a regular expression for it.

2. (20 points) Since we are not only interested in form in this course, we should be able to interpret the trees above from grammar. Show how the first tree above can be evaluated step by step, using lambda terms in the process.

3. (30 points) Show a PDA that does form-meaning computation for this language in locked-step with rule use.

4. (40 points) Since we are not only interested in this course about the form, its meaning and locked-step computation of its structure, but also about how they behave altogether under uncertainty, we might want to look into some models of this phenomenon in extended use.
   Suppose that the owner/processor of this system notices that what she can do with squaring she can also do with self-addition (symbolized by ‡) to get the same result, or a mixture of the two. For example:



ANSWERS[7]**:**
Question 1: (0|1|2|3|4|5|6|7|8|9) $\triangle^*$ (0|1|2|3|4|5|6|7|8|9)
where $\triangle$ corresponds to 2 that squares its argument.

Question 2:
$= (\lambda f.\lambda g(-\ f\ g))((\lambda x.\lambda y.x^y)((\lambda z.\lambda w.z^w)(5 \triangle \triangle 3)))$
$=_\beta (\lambda f.\lambda g(-\ f\ g))((\lambda x.\lambda y.x^y)((\lambda z.\lambda w.z^w)[5/z](\triangle \triangle 3)))$
$= (\lambda f.\lambda g(-\ f\ g))((\lambda x.\lambda y.x^y)((\lambda w.5^w)(\triangle \triangle 3)))$
$=_\beta (\lambda f.\lambda g(-\ f\ g))((\lambda x.\lambda y.x^y)((\lambda w.5^w[\triangle/w](\triangle 3)))$
$= (\lambda f.\lambda g(-\ f\ g))((\lambda x.\lambda y.x^y)(5^\triangle)(\triangle 3)))$
$=_\beta (\lambda f.\lambda g(-\ f\ g))((\lambda x.\lambda y.x^y)[25/x](\triangle 3)))$
$= (\lambda f.\lambda g(-\ f\ g))((\lambda y.25^y)(\triangle 3)))$
$=_\beta (\lambda f.\lambda g(-\ f\ g))((\lambda y.25^y)[\triangle/y](3)))$
$= (\lambda f.\lambda g(-\ f\ g))(25^\triangle)(3)$
$=_\beta (\lambda f.\lambda g(-\ f\ g))[625/f](3)$
$= (\lambda g(-\ 625\ g))(3)$
$=_\beta (\lambda g(-\ 625\ g)[3/g])$
$= (-\ 625\ 3)$
$= 622$

[7]These questions do NOT reflect the real exam questions, and the answers are NOT the correct answers. They are all given just to provide a reference.

Question 3:

Revising the grammar rules with lambda terms:
$S : \lambda x.\lambda y.xy \rightarrow A : a'\ N : n'$
$A : \lambda x.\lambda y.xy \rightarrow A : a'\ \triangle : \triangle$
$A : \lambda x.x \rightarrow N : n'$
$N : \lambda x.x \rightarrow 0 : 0|1 : 1| \cdots |8 : 8|9 : 9$

PDA:
Rule 1:[8] $((p, 5, \epsilon), (p, 5))$
Rule 2: $((p, \epsilon, 5), (p, N : (\lambda x.x)5 = n'))$
Rule 3: $((p, \epsilon, N), (p, A : (\lambda x.x)n' = a'))$
Rule 4: $((p, \triangle, \epsilon), (p, \triangle))$
Rule 5: $((p, \epsilon, \triangle A), (p, A : (\lambda x.\lambda y.xy)\ \triangle\ a'))$
Rule 6: $((p, \epsilon, n'\ a'), (p, S : ((\lambda x.\lambda y.xy)n'\ a')))$
Rule 7: $((p, \epsilon, S), (q, \epsilon))$

Rule Use:
$(p, 5 \triangle \triangle 3, \epsilon) \vdash$
$(p, \triangle \triangle 3, N : (\lambda x.x)5) \vdash$
$(p, \triangle \triangle 3, A : (\lambda x.x)n') \vdash$
$(p, \triangle \triangle 3, a') \vdash$
$(p, \triangle 3, \triangle a') \vdash$
$(p, \triangle 3, A : (\lambda x.\lambda y.xy)\ \triangle\ a') \vdash$
$(p, 3, \triangle a') \vdash$
$(p, 3, A : (\lambda x.\lambda y.xy)\ \triangle\ a') \vdash$
$(p, \epsilon, (N : (\lambda x.x)3)a') \vdash$
$(p, \epsilon, n'\ a') \vdash$
$(p, \epsilon, S : (\lambda x.\lambda y.xy)n'\ a') \vdash$
(q, ϵ, ϵ) and accept.

[8]Here 5 is representative of all natural numbers from 0 to 9; to cut it short, I did not include all the rules dedicated to all numbers, so this rule also includes rules such as $((p, 3, \epsilon), (p, 3))$

Question 4:

In designing this system, it is assumed that the meaning (result) of the input pair (3, 2) is 70 in the sentence and a possible world. Although there are two different sequences, the meaning is the same. To find out the most likely sequence, the rule uses of the functors (‡ △) can be evaluated. Therefore, the joint probabilities of each tree can be considered. The likelihood of most likely sequence given the input pair and the result is the product of probabilities of the joint probabilities of both tree given their rule uses.

ArgMax $P(M|sentence,\ world)$ = ArgMax $\Sigma P(M, Tree?|sentence,\ world)$ =
ArgMax $P \prod(M, \alpha_i|\beta_i, sentence, world)$ =
Joint probability of the trees given their structures and rules by which they are derived.

In doing this computation, both structures are taken into consideration.

ArgMax $P(70|3 \ddagger \triangle \ddagger 2,\ world)$ = ArgMax $\Sigma P(70, Tree_1|3 \ddagger \triangle \ddagger 2,\ world)$ =
ArgMax $P \prod(M, \alpha_i|\beta_i, sentence, world) = P(Tree_1|S) = P(S|A\ N) \times P(A|A \ddagger) \times \cdots \times P(N|2)$*(Joint Probability of the first tree given its structure)*

ArgMax $P(70|3 \triangle \ddagger \ddagger \ddagger 2,\ world)$ = ArgMax $\Sigma P(70, Tree_2|3 \triangle \ddagger \ddagger \ddagger 2,\ world)$ =
ArgMax $P \prod(M, \alpha_i|\beta_i, sentence, world) = P(Tree_2|S) = P(S|A\ N) \times P(A|A \ddagger) \times \cdots \times P(N|2)$*(Joint Probability of the first tree given its structure)*

Note: World is the world where the input pair (3,2) corresponds to 70.