THE UNIVERSITY OF MELBOURNE

# Comp90042 Workshop Week 8

9 May

# Table of Contents

1. Language theory

2. Parsing

# Regular language

What are regular grammar and regular language? How are they different?

# Regular language

What are regular grammar and regular language? How are they different?

- Language: set of acceptable strings (e.g., sentences)
- Grammar: a generative description of a language
- Regular language:  a formal language accepted by a regular expression
- Regular grammar: a formal grammar defined by a set of productions rules
  - $A \rightarrow xB$,
  - $A \rightarrow x$
  - $A \rightarrow \varepsilon$

  where A and B are non-terminals, x is a terminal and ε is the empty string.

# Regular grammar example

- Rules:
  - S → A
  - A → a A
  - A → ε

Where S is the start symbol, A is non-terminal, a is terminal and ε is empty

- Example strings?
  - a, aa, aaa, aaaa.
- Regular expression?
  - (a)*

# Properties of regular languages

- L1: regular language
- L2: regular language

- L1∪L2:?

- L1∩L2:?

- Concatenation of L1 and L2:?

# Properties of regular languages

- L1: regular language
- L2: regular language

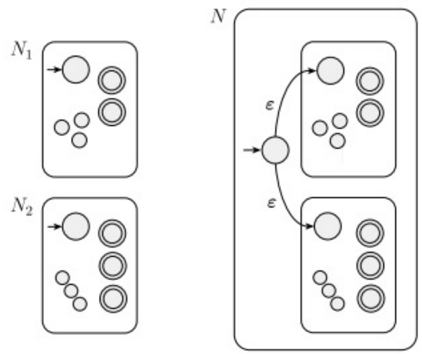
- L1∪L2: regular language
- L1∩L2: regular language
- Concatenation of L1 and L2: regular language

# Properties of regular languages

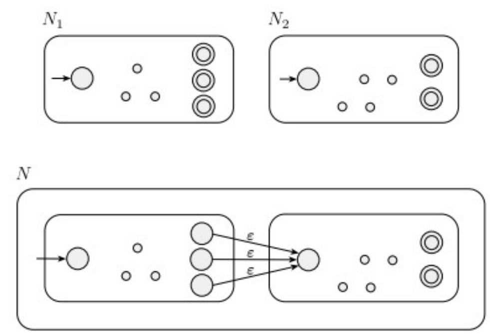
Regular languages are closed under union, intersection and concatenation. What does it mean? Why is it important?

If L1 and L2 are two regular languages, then the union, intersection and concatenation of L1 and L2 are also regular languages

union



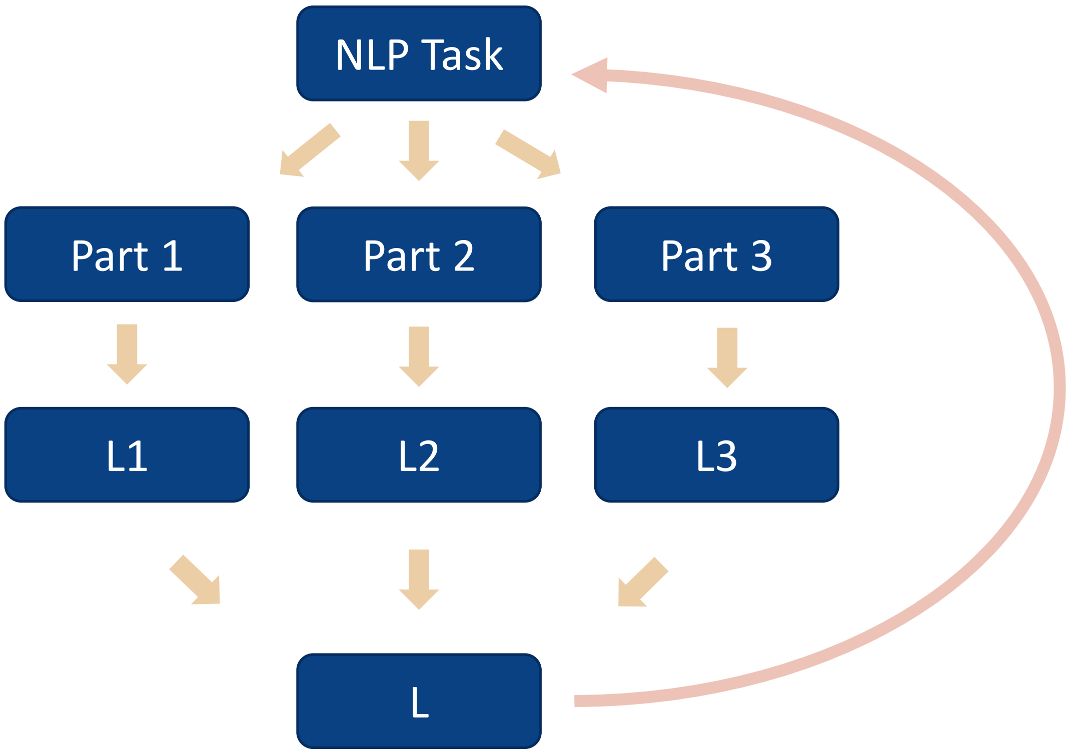

concatenation



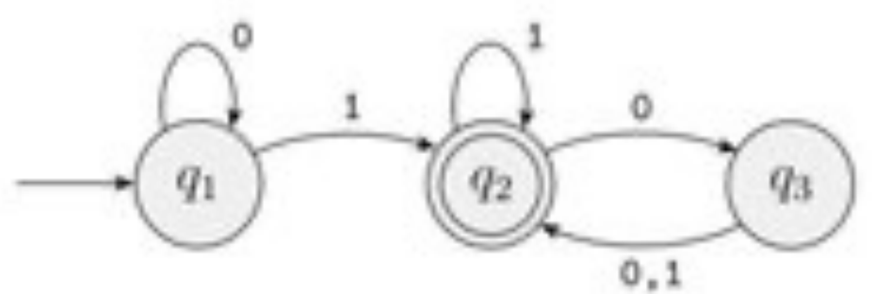

- NLP problems can be factored into small simple parts,
- we can develop regular languages for each part,
- and combine them into a complex system to handle the NLP problems.

# Properties of regular languages

Regular languages are closed under union, intersection and concatenation. What does it mean? Why is it important?



# Finite State Acceptor




An FSA is formally represented as a 5-tuple M1 = (Q, Σ, δ, $q_1$, F)

1. States Q =

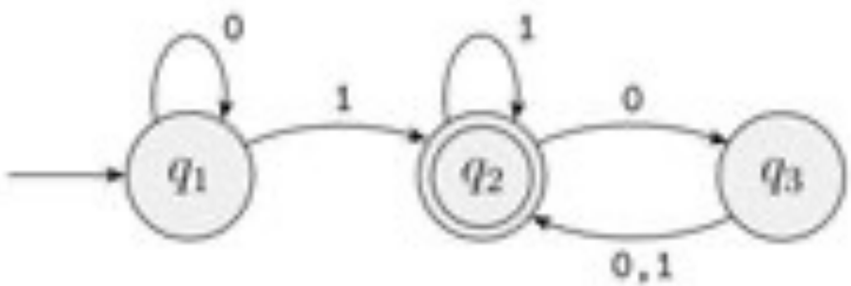
2. Input alphabet  Σ =

3. Transition function δ

4. _ is the start state, and

5. Final state F =

# Finite State Acceptor




An FSA is formally represented as a 5-tuple M1 = (Q, Σ, δ, $q_1$, F)

1. States Q = $\{q_1, q_2, q_3\}$,
2. Input alphabet Σ = {0,1},
3. Transition function δ
4. $q_1$ is the start state, and
5. Final state F = $\{q_2\}$

Transition function

| | 0 | 1 |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_2$ | $q_2$ |

What language does M1 accept?

# Example: Word Morphology

Draw a Finite State Acceptor (FSA) for word morphology to show the possible derivations from root forms using the words:

play, played, playing;

walk, walked, walking;

sit, sat, sitting.

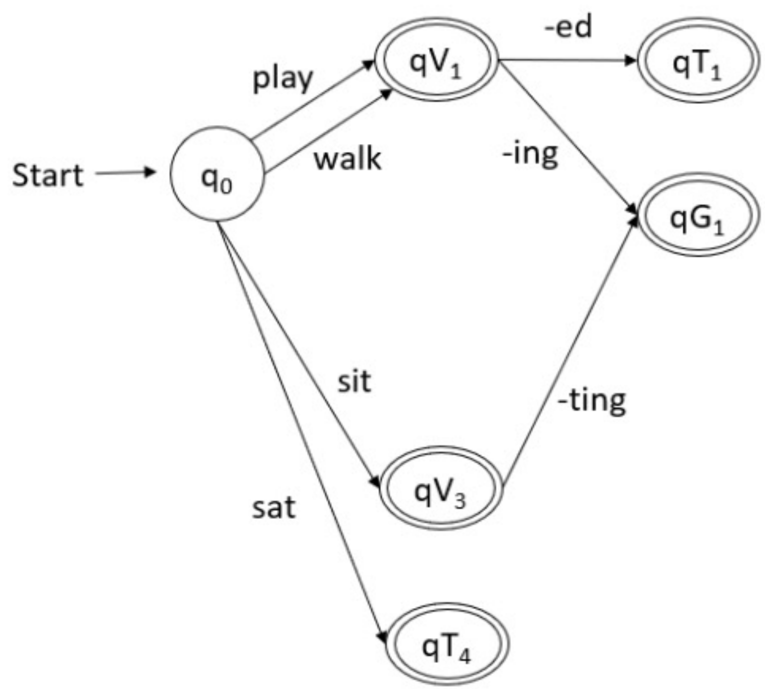
# Example: Word Morphology

Draw a Finite State Acceptor (FSA) for word morphology to show the possible derivations from root forms using the words:

play, played, playing;

walk, walked, walking;

sit, sat, sitting.



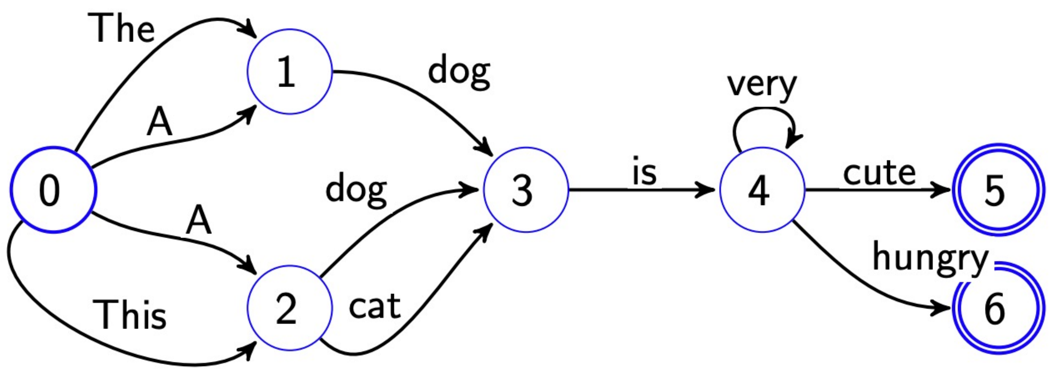
# **FSAs vs** Weighted Finite State Acceptors (WFSAs)

What are Weighted Finite State Acceptors (WFSAs)? When and why are they useful?

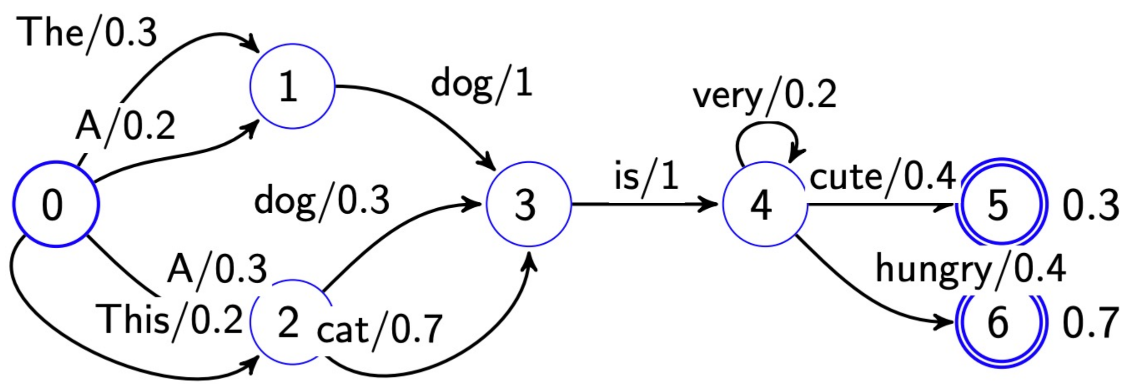
# Weighted FSA (WFSA)

A **WFSA** is an FSA with **weights** on the edges.

FSA




WFSA



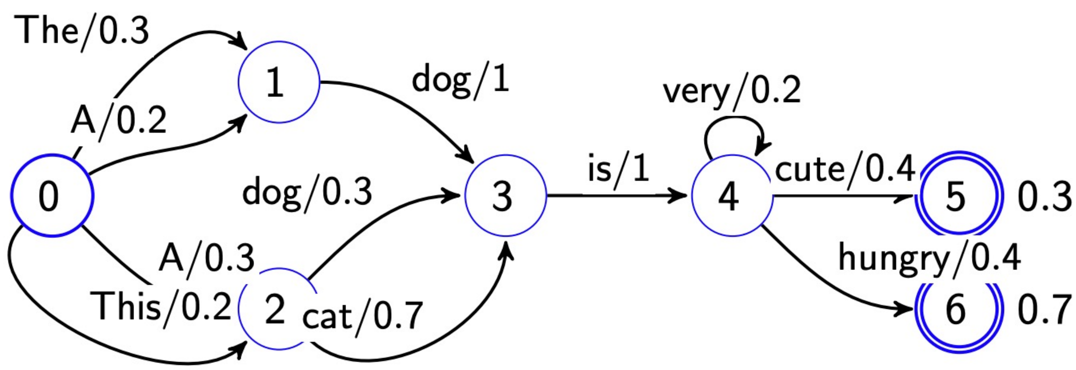



https://courses.engr.illinois.edu/ece417/fa2020/slides/lec15.pdf

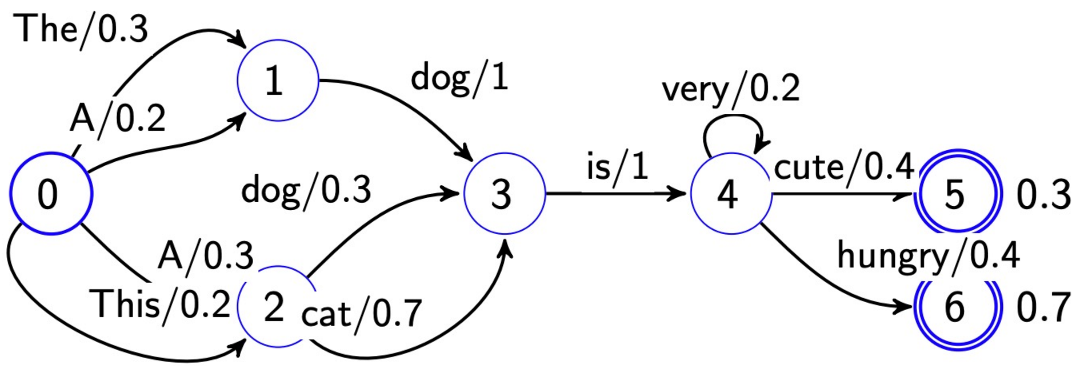
# How does a WFSA work?

Add weights to initial/final states of FSA

- λ : assign weights to initial state
- ρ : assign weight to final state
- δ : assign weight to transitions



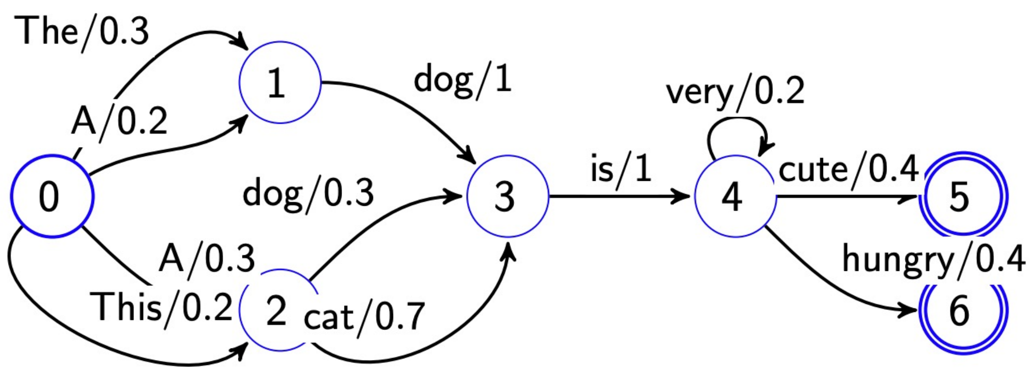
# How does a WFSA work?




A WFSA assign a score for any path π = $t_1$ , $t_2$, … $t_N$

- S(π) = λ($t_0$) + N X δ($t_n$) + ρ($t_N$).
- Dijkstra algorithm for shortest path

# How to accumulate the path score?




An example: there are two matched paths for "A dog is hungry".

The scores are calculated by multiplying the edge scores:

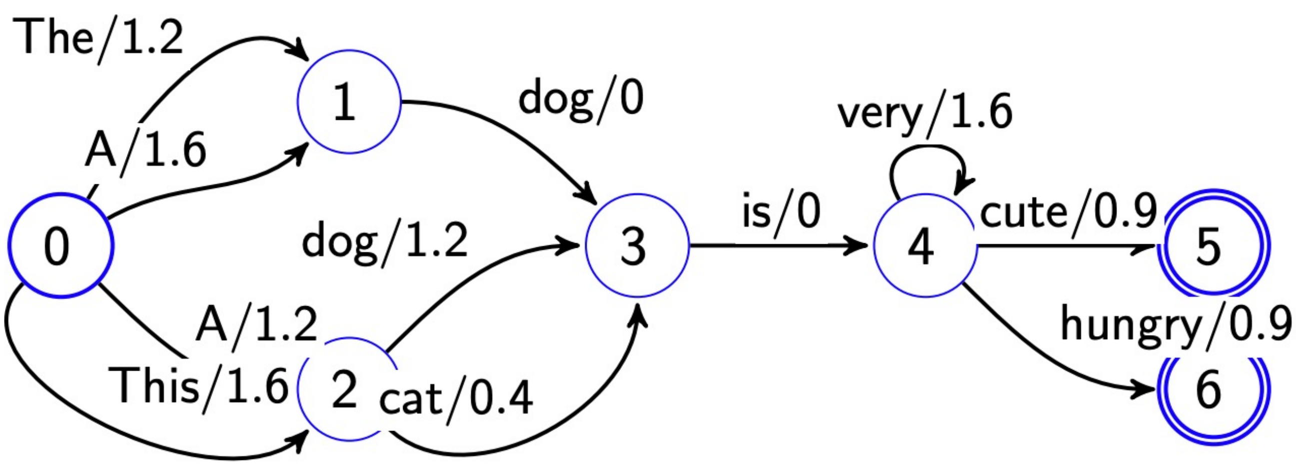
p(Path through state 1) = (0.2)(1)(1)(0.4) = 0.08

p(Path through state 2) = (0.3)(0.3)(1)(0.4) = 0.036

# How to accumulate the path score?

The Issue: WFSAs have floating point underflow problems

The solution: negative log probabilities




"A dog is hungry".

The negative log probs for two paths:

$-\ln p(\text{Path through state 1}) = 1.6 + 0 + 0 + 0.9 = 2.5$

$-\ln p(\text{Path through state 2}) = 1.2 + 1.2 + 0 + 0.9 = 3.3$

# FSA vs WFSA

**FSA:**

- can only accept/reject input strings: is it grammatical/valid? (binary)
- returns **VALID** if a string corresponds to a valid path from start to end, and **NOT-VALID** otherwise

**WFSA**:

- decomposes a sequence (a word or a sentence) into sub-sequences
- gives a score: how grammatical/likely a sequence is (eg. N-gram language model; )

# When and why WFSAs are useful?

Words: assign scores to all strings of characters forming words to handle:

- Spelling mistakes
- New words
- Strange but acceptable words

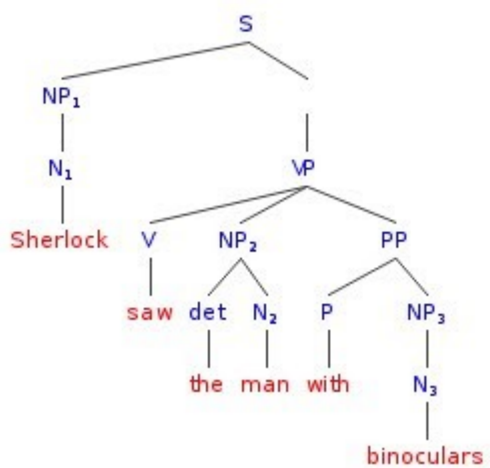
Sentences: assign scores to all strings of words forming sentences to handle:

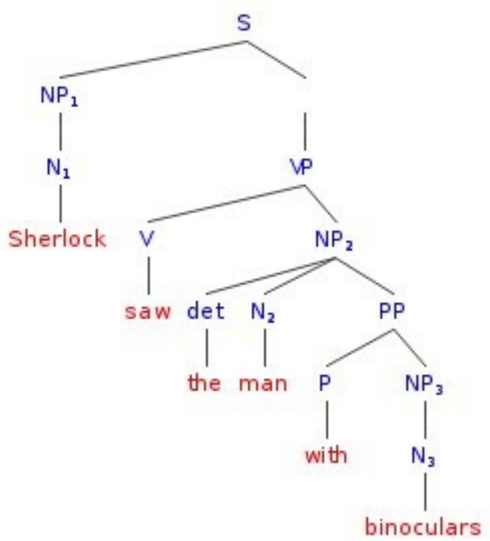
- Sentence generation, speech recognition, OCR, translation

# Parsing

Parsing in general is the process of identifying the structure(s) of a sentence, according to a grammar of the language.

Example parse trees



# Context-free grammar

This is the dog that worried the cat that killed the rat that ate the malt that lay in the house that Jack built

VS

A man that a woman that a child that a bird that I heard saw knows loves

# Context-free grammar

A context-free grammar is a 4-tuple $G = (V, \Sigma, R, S)$

1. $V$, a set of non-terminal symbols (or variables)
2. $\Sigma$, a set of terminal symbols (disjoint from $V$)
3. $R$ a set of rules in the form $A \rightarrow \beta$, where $A$ is a non-terminal, $\beta$ is a string of variables or terminals
4. $S \in V$, the start variable

# Parsing

(a) What changes need to be made to this CFG to make it suitable for CYK parsing?

- S -> NP VP
- VP -> V NP | V NP PP
- PP -> P NP
- V -> "saw" | "walked"
- NP -> "John" | "Bob" | Det N | Det N PP
- Det -> "a" | "an" | "the" | "my"
- N -> "man" | "cat" | "telescope" | "park"
- P -> "on" | "by" | "with"

Step 1: Chomsky Normal Form: Each rule consists of either:

- A -> B C    a (single) non-terminal which re-writes as exactly two non-terminals
- A -> a    a (single) non-terminal which re-writes as a single terminal

# Parsing

(a) What changes need to be made to this CFG to make it suitable for CYK parsing?

- S -> NP VP
- VP -> V NP | V NP PP
- PP -> P NP
- V -> "saw" | "walked"
- NP -> "John" | "Bob" | Det N | Det N PP
- Det -> "a" | "an" | "the" | "my"
- N -> "man" | "cat" | "telescope" | "park"
- P -> "on" | "by" | "with"

Step 1: Chomsky Normal Form: Each rule consists of either:

- A -> B C a (single) non-terminal which re-writes as exactly two non-terminals
- A -> a a (single) non-terminal which re-writes as a single terminal

# Parsing

(a) What changes need to be made to this CFG to make it suitable for CYK parsing?

- S -> NP VP
- VP -> V NP | ~~V NP PP~~
- PP -> P NP
- V -> "saw" | "walked"
- NP -> "John" | "Bob" | Det N | ~~Det N PP~~
- Det -> "a" | "an" | "the" | "my"
- N -> "man" | "cat" | "telescope" | "park"
- P -> "on" | "by" | "with"

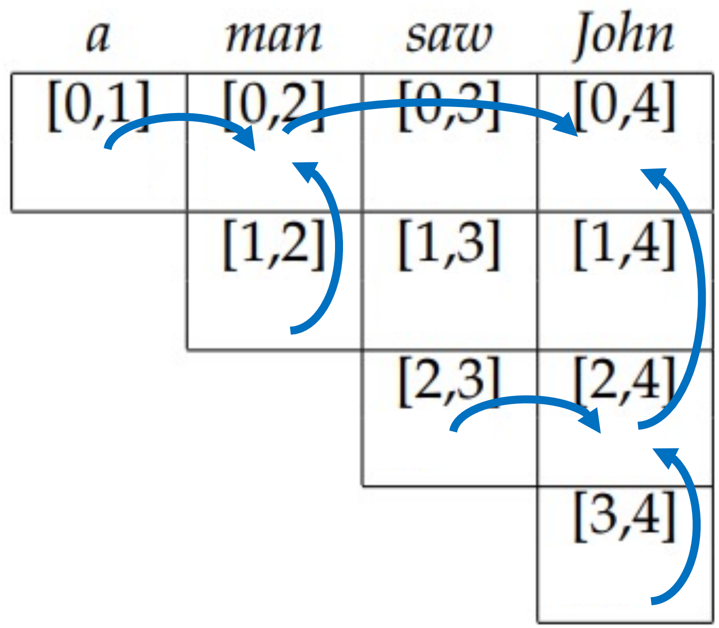
- VP -> V X
- X -> NP PP

- NP -> Det Y
- Y -> N PP

Step 1: Chomsky Normal Form: Each rule consists of either:

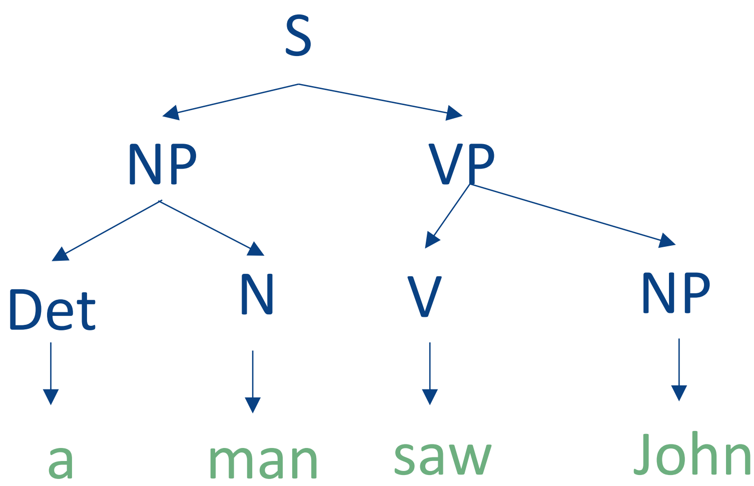
- A -> B C  a (single) non-terminal which re-writes as exactly two non-terminals
- A -> a  a (single) non-terminal which re-writes as a single terminal

# CYK Example 1




1. S -> NP VP
2. VP -> V NP | V X
3. PP -> P NP
4. X -> NP PP
5. Y -> N PP
6. NP -> "John" | "Bob" | Det N | Det Y
7. V -> "saw" | "walked"
8. Det -> "a" | "an" | "the" | "my"
9. N -> "man" | "cat" | "telescope" | "park"
10. P -> "on" | "by" | "with"

# CYK Example 1




| *a* | *man* | *saw* | *John* |
|---|---|---|---|
| [0,1] Det | [0,2] NP | [0,3] - | [0,4] S |
| | [1,2] N | [1,3] - | [1,4] - |
| | | [2,3] V | [2,4] VP |
| | | | [3,4] NP |

1. S -> NP VP
2. VP -> V NP | V X
3. PP -> P NP
4. X -> NP PP
5. Y -> N PP
6. NP -> "John" | "Bob" | Det N | Det Y
7. V -> "saw" | "walked"
8. Det -> "a" | "an" | "the" | "my"
9. N -> "man" | "cat" | "telescope" | "park"
10. P -> "on" | "by" | "with"

# CYK example 2

| *an* | *park* | *by* | *Bob* | *walked* | *an* | *park* | *with* | *Bob* |
|---|---|---|---|---|---|---|---|---|
| [0,1] | [0,2] | [0,3] | [0,4] | [0,5] | [0,6] | [0,7] | [0,8] | [0,9] |
| | [1,2] | [1,3] | [1,4] | [1,5] | [1,6] | [1,7] | [1,8] | [1,9] |
| | | [2,3] | [2,4] | [2,5] | [2,6] | [2,7] | [2,8] | [2,9] |
| | | | [3,4] | [3,5] | [3,6] | [3,7] | [3,8] | [3,9] |
| | | | | [4,5] | [4,6] | [4,7] | [4,8] | [4,9] |
| | | | | | [5,6] | [5,7] | [5,8] | [5,9] |
| | | | | | | [6,7] | [6,8] | [6,9] |
| | | | | | | | [7,8] | [7,9] |
| | | | | | | | | [8,9] |

1. S -> NP VP
2. VP -> V NP | V X
3. PP -> P NP
4. X -> NP PP
5. Y -> N PP
6. NP -> "John" | "Bob" | Det N | Det Y
7. V -> "saw" | "walked"
8. Det -> "a" | "an" | "the" | "my"
9. N -> "man" | "cat" | "telescope" | "park"
10. P -> "on" | "by" | "with"

# CYK example 3

| *park* | *by* | *the* | *cat* | *with* | *my* | *telescope* |
|---|---|---|---|---|---|---|
| [0,1] | [0,2] | [0,3] | [0,4] | [0,5] | [0,6] | [0,7] |
| | [1,2] | [1,3] | [1,4] | [1,5] | [1,6] | [1,7] |
| | | [2,3] | [2,4] | [2,5] | [2,6] | [2,7] |
| | | | [3,4] | [3,5] | [3,6] | [3,7] |
| | | | | [4,5] | [4,6] | [4,7] |
| | | | | | [5,6] | [5,7] |
| | | | | | | [6,7] |

1. S -> NP VP
2. VP -> V NP | V X
3. PP -> P NP
4. X -> NP PP
5. Y -> N PP
6. NP -> "John" | "Bob" | Det N | Det Y
7. V -> "saw" | "walked"
8. Det -> "a" | "an" | "the" | "my"
9. N -> "man" | "cat" | "telescope" | "park"
10. P -> "on" | "by" | "with"

# Takeaways

1. Regular languages:
   1. Regular Grammar
   2. Finite state acceptors (FSA)
   3. Weighted finite state acceptors (WFSA)

2. Context Free Languages
   1. Context Free Grammar
   2. CYK algorithm for parsing