School of Computing and Information Systems
The University of Melbourne
COMP90042
NATURAL LANGUAGE PROCESSING (Semester 1, 2020)

Sample solutions for discussion exercises: Week 8

Discussion

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

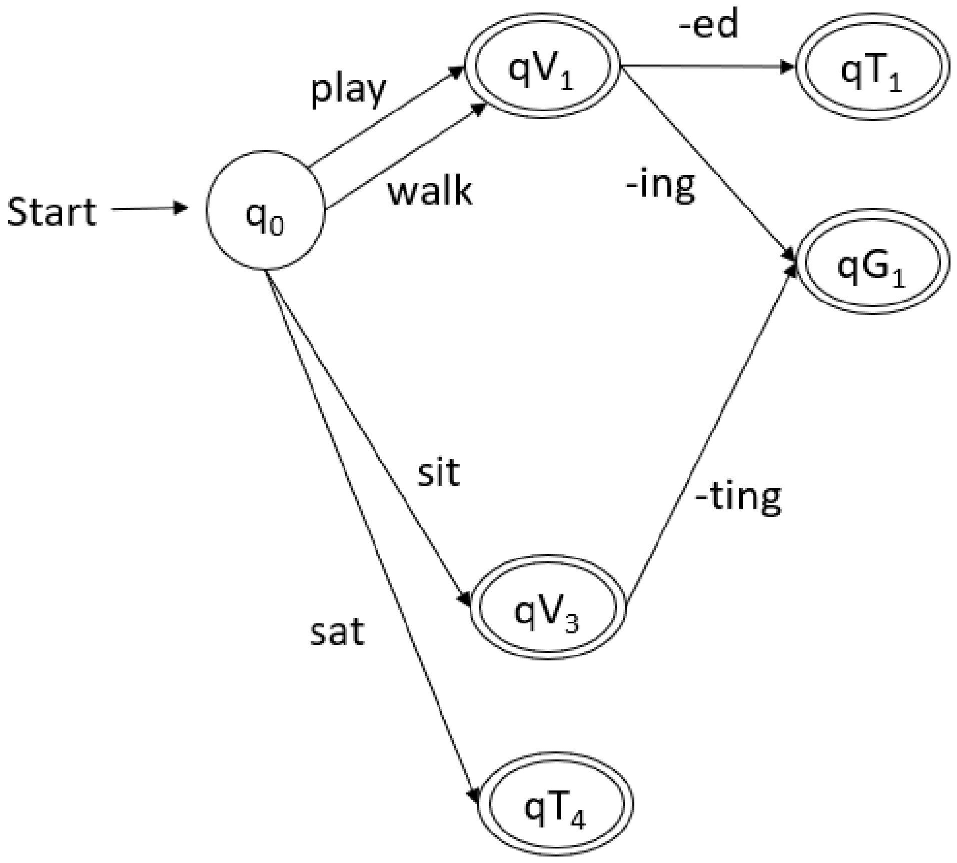
   - A language is a set of acceptable strings and a grammar is a generative description of a language.
   - Regular language is a formal language that can be expressed using a regular expression.
   - Regular grammar is a formal grammar defined by a set of productions rules in the form of A $\rightarrow$ xB, A $\rightarrow$ x and A $\rightarrow$ ϵ, where A and B are non-terminals, x is a terminal and ϵ is the empty string.
   - A language is regular if and only if it can be generated by a regular grammar.
   - For example: A simple regular grammar
     - Rules: S $\rightarrow$ A, A $\rightarrow$ a A, A $\rightarrow$ ϵ
     - S is the start symbol
     - It will generate words such as a, aa, aaa, aaa.
     - The set of words generated by this regular grammar is a regular language.
     - This regular language can also be expressed in regular expression (a)*.

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

   - This means that if L1 and L2 are two regular languages, then L1 $\cup$ L2, L1 $\cap$ L2, and the language strings that are the concatenation of L1 and L2 are also regular languages.
   - This closure property allows us to apply operations on regular languages to produce a regular language.
   - This allows for NLP problems to be factored into small simple parts, such that we can develop regular languages for each part, and combine them into a complex system to handle the NLP problems. This is particularly relevant for transducers and the composition operation, which are used in many NLP pipelines. (Note that FSTs implement "regular relations" rather that regular languages, but the distinction is a bit subtle and is not something we will dive into.)

   (b) 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.

   - There are many correct answers. Below is one:



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

- WFSAs are generalizations of FSAs, with each path assigned a score, computed from the transitions, the initial state, and the final state. The total score for any path is equal to the sum of the scores of the initial state, the transitions, and the final state.
- WFSAs can produce a score for each valid word for sub-word decomposition problems or sentence for words-in-sentence problems, while FSAs have no way to express preferences among technically valid words or sentences.
- For example, WFSAs can assign scores to all strings of characters forming words, so that spelling mistakes, new words, or strange-but-acceptable words can still be handled.
- The same argument holds for sequences of words forming sentences. Clearly some word sequences are gibberish, but being able to provide a numerical score can help in many applications, like how LMs can be used in sentence generation, speech recognition, OCR, translation, etc.

2. What is **parsing**? Why is it important?

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

3. Consider the following simple **context–free grammar**:

```
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"
```

   (a) What changes need to be made to the grammar to make it suitable for **CYK parsing**?

   - For CYK parsing, a grammar needs to be written in Chomsky Normal Form, where each rule consists of either:
     – a (single) non-terminal which re-writes as a single terminal, or
     – a (single) non-terminal which re-writes as exactly two non-terminals
   - Here, we have two rules where a non-terminal re-writes as three non-terminals (`VP -> V NP PP` and `NP -> Det N PP`); we remove these rules and replace them with the following:

```
VP -> V X
X -> NP PP
NP -> Det Y
Y -> N PP
```

   (b) Using the CYK strategy and the above grammar in CNF, parse the following sentences:

      i. “a man saw John”
      ii. “an park by Bob walked an park with Bob”
      iii. “park by the cat with my telescope”
         - This sentence has no parse, because the cell [0,7] doesn’t have an `S`.

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

an	*park*	*by*	*Bob*	*walked*	*an*	*park*	*with*	*Bob*
[0,1] Det	[0,2] NP	[0,3] -	[0,4] NP, X	[0,5] -	[0,6] -	[0,7] S	[0,8] -	[0,9] S, S
	[1,2] N	[1,3] -	[1,4] Y	[1,5] -	[1,6] -	[1,7] -	[1,8] -	[1,9] -
		[2,3] P	[2,4] PP	[2,5] -	[2,6] -	[2,7] -	[2,8] -	[2,9] -
			[3,4] NP	[3,5] -	[3,6] -	[3,7] S	[3,8] -	[3,9] S, S
				[4,5] V	[4,6] -	[4,7] VP	[4,8] -	[4,9] VP, VP
					[5,6] Det	[5,7] NP	[5,8] -	[5,9] NP, X
						[6,7] N	[6,8] -	[6,9] Y
							[7,8] P	[7,9] PP
								[8,9] NP

park	*by*	*the*	*cat*	*with*	*my*	*telescope*
[0,1] N	[0,2] -	[0,3] -	[0,4] Y	[0,5] -	[0,6] -	[0,7] Y
	[1,2] P	[1,3] -	[1,4] PP	[1,5] -	[1,6] -	[1,7] PP
		[2,3] Det	[2,4] NP	[2,5] -	[2,6] -	[2,7] NP, X
			[3,4] N	[3,5] -	[3,6] -	[3,7] Y
				[4,5] P	[4,6] -	[4,7] PP
					[5,6] Det	[5,7] NP
						[6,7] N