Assignment #5

CSE 447: Natural Language Processing
Eric Boris: 1976637
Collaborators: Allyson Ely, Ardi Madadi

Written Problems

Problem 1

1. $\text{MAKES-NOISE(ABIGAIL)} \implies \forall x\, \neg\text{CAN-SLEEP}(x)$
2. $\text{MAKES-NOISE(ABIGAIL)} \implies \exists x\, \neg\text{CAN-SLEEP}(x)$
3. $\forall x \text{BROTHER(ABIGAIL}, x) \implies \neg\text{CAN-SLEEP}(x)$
4. $\exists x \text{BROTHER(ABIGAIL}, x) \wedge \text{MAKES-NOISE}(x) \implies \neg\text{CAN-SLEEP(ABIGAIL)}$

Problem 3
Given a CNF PCFG, a sentence x of length n, and an unlabeled phrase-structure tree, describe a linear time dynamic programming algorithm that produces the most probable labeled tree consistent with the input. We assume that the phrase-structure tree is ordered, i.e. that children spans occur before parent spans. We discover the tree labels by using a dynamic programming approach. And we do this by iterating over the phrase-structure tree, starting with children spans. We know that all single-spans, i.e. $< i_k, j_k >$ s.t. $k = k$, are the words of the sentence. We assign to each single-span cell in the dynamic programming table the most probable label of that word using the PCFG and a backpointer to that word. Having visited every single-span entry, we encounter the multiple-span entries, $< i_k, j_l >$ s.t. $k \neq l$ and s.t. $1 \leq k$, $k < l$, and $l \leq n.$ Each cell at this level is a possible parent to two children since the tree is CNF. We consider the possible labels to this cell by considering that cell's possible children and choose that with the highest probability as given by the PCFG or assign no label if one is not found in the PCFG. We assign to each cell the most probable label and a backpointer to the cells from which it was generated. Upon reaching the last entry in the phrase-structure tree, we've reached the root of the tree. We now follow the backpointers along the path through the dynamic programming table to reconstruct the most probable labeling of the tree. Because we iterate over the phrase-structure tree once and only consider a limited number of children for any cell, the algorithm is upper bounded and runs in linear time.

Problem 4

1.

Interpreting “duck” as a noun

	Stack	Buffer	Action
0		He | saw | her | duck | .	NT(S)
1	(S	He | saw | her | duck | .	NT(NP)
2	(S | (NP	He | saw | her | duck | .	SHIFT
3	(S | (NP | He	saw | her | duck | .	REDUCE
4	(S | (NP He)	saw | her | duck | .	NT(VP)
5	(S | (NP He) | (VP	saw | her | duck | .	SHIFT
6	(S | (NP He) | (VP | saw	her | duck | .	NT(NP)
7	(S | (NP He) | (VP | saw | (NP	her | duck | .	SHIFT
8	(S | (NP He) | (VP | saw | (NP | her	duck | .	SHIFT
9	(S | (NP He) | (VP | saw | (NP | her | duck	.	REDUCE
10	(S | (NP He) | (VP | saw | (NP her duck)	.	REDUCE
11	(S | (NP He) | (VP saw (NP her duck))	.	SHIFT
12	(S | (NP He) | (VP saw (NP her duck)) | .		REDUCE
13	(S (NP He) (VP saw (NP her duck)).)		

Interpreting “duck” as a verb

	Stack	Buffer	Action
0		He | saw | her | duck | .	NT(S)
1	(S	He | saw | her | duck | .	NT(NP)
2	(S | (NP	He | saw | her | duck | .	SHIFT
3	(S | (NP | He	saw | her | duck | .	REDUCE
4	(S | (NP He)	saw | her | duck | .	NT(VP)
5	(S | (NP He) | (VP	saw | her | duck | .	SHIFT
6	(S | (NP He) | (VP | saw	her | duck | .	NT(VP)
7	(S | (NP He) | (VP | saw | (VP	her | duck | .	SHIFT
8	(S | (NP He) | (VP | saw | (VP | her	duck | .	SHIFT
9	(S | (NP He) | (VP | saw | (VP | her | duck	.	REDUCE
10	(S | (NP He) | (VP | saw | (VP her duck)	.	REDUCE
11	(S | (NP He) | (VP saw (VP her duck))	.	SHIFT
12	(S | (NP He) | (VP saw (VP her duck)) | .		REDUCE
13	(S (NP He) (VP saw (VP her duck)).)		

2. A feature we could use to inform the next action could be to see if the last action was a REDUCE. If it was, we likely want to perform a NT, with some smaller probability of performing another REDUCE, and a very small to zero probablity of performing a SHIFT. Based on the tables above, as well as Figure 1 on the spec, this feature will give us increased performance in classifying the next action.

3. We can use a variant of Dijkstra's algorithm to maintain a frontier of possible actions and progress by choosing the next best action at each step. We represent the search space as a graph with nodes representing actions and edge weights between nodes representing probabilities. We follow the hightest probability path. Invalid paths are assigned negative infinite weight and infinite loops are prevented by limiting recursion depth to the number of tokens remaining in the buffer. The highest probability tree then is that wih the best path through the search space as found by Dijkstra's.

4. Our first constraint is to prevent left recursion, i.e. no $X \to X \dots$. Next, we restrict cycles with the constraints that 1) if $X \to Y$ and $X \to Z$ then $\text{First}(Y) \cap \text{First}(Z) = \emptyset$ and 2) if $X \to Y$ and $X \to Z$ and $\text{First}(Z)$ contains ϵ then $\text{First}(Y) \cap \text{Follow}(Z) = \emptyset$. Thus, by preventing left recursion and cycles, we prevent infinite loops.