COMP 550
Assignment 3

Jonathan Pearce
260672004

1) a) i) $(\lambda x.\, x\ x)(\lambda y.\, y\ x)\ z$

$= (\lambda y.\, y\ x)(\lambda y.\, y\ x)\ z$

$= (\lambda y.\, y\ x)\ x\ z$

$= x\ x\ z$

ii) $(\lambda u\, v\, w.\, w\ v\ u)\ a\ a\ (\lambda p\, q.\, q)$

$= (\lambda p\, q.\, q)\ a\ a$

$= a$

iii) $((\lambda v.\, v\ v)(\lambda u.\, u))((\lambda v.\, w)(\lambda v.\, w))$

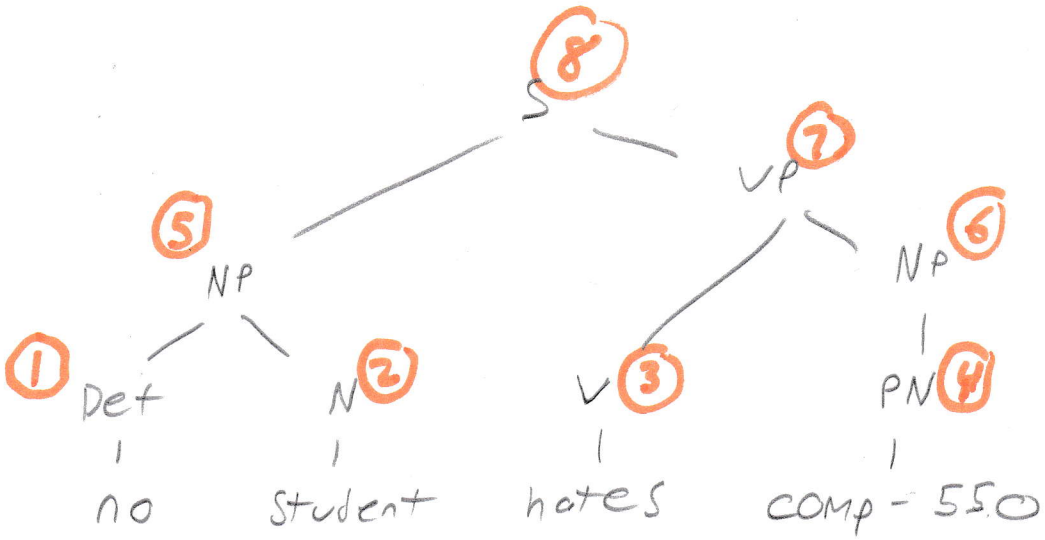
$= ((\lambda u.\, u)(\lambda u.\, u))(\lambda v.\, w)$

$= (\lambda u.\, u)(\lambda v.\, w)$

$= (\lambda u.\, w)$

b) Det → no $\{\lambda P\, \lambda Q\, \forall x \quad P(x) \rightarrow \neg Q(x)\}$

V → hates $\{\lambda w\, \lambda t\, w(\lambda x\, \exists e\ Hates(e) \wedge Hater(e,t) \wedge Hatee(e,x))\}$



1 : $\lambda P\, \lambda Q\, \forall x \quad P(x) \rightarrow \neg Q(x)$

2 : $\lambda x.\ student(x)$

3 : $\lambda w\, \lambda t\ w(\lambda x\, \exists e\ Hates(e) \wedge Hater(e,t) \wedge Hatee(e,x))$

4 : $\lambda x\ x(comp\ 550)$

5 : $(\lambda P\, \lambda Q\, \forall x \quad P(x) \rightarrow \neg Q(x))(\lambda x.\ student(x))$

$= \lambda Q\, \forall x \quad student(x) \rightarrow \neg Q(x)$

6 : $\lambda x.\ x(comp\ 550)$

7 : $(\lambda w\, \lambda t\ w(\lambda x\, \exists e\ Hates(e) \wedge Hater(e,t) \wedge Hatee(e,x)))(\lambda x.\ x\ (comp\ 550))$

$= \lambda t\ \exists e\ Hates(e) \wedge Hater(e,t) \wedge Hatee(e, comp\text{-}550)$

$= A$

8 $= (\lambda Q\, \forall x \quad student(x) \rightarrow \neg Q(x))(A)$

$= \forall x \quad student(x) \rightarrow \neg \exists e\ (Hates(e) \wedge Hater(e,x) \wedge Hatee(e, comp\ 550))$

c) No student wants an exam

$\exists e \; want(e) \wedge wanter(e, S_1) \wedge wantee(e, S_2)$

$(\lambda Q \; \forall x \; student(x) \longrightarrow \neg Q(x), \; 1),$

$(\lambda Q \; \exists_y \; exam(y) \wedge Q(y), \; 2)$

$1 > 2$ = 2 first then 1

2: $(\lambda Q \; \exists_y \; Exam(y) \wedge Q(y))(\lambda S_2 . \; \exists e \; want(e) \wedge wanter(e, S_1) \wedge wantee(e, S_2))$

$= \exists_y \; exam(y) \wedge \exists e \; want(e) \wedge wanter(e, S_1) \wedge wantee(e, y) = \underline{\underline{W}}$

1: $(\lambda Q \; \forall x \; student(x) \longrightarrow \neg Q(x))(\lambda S_1 . \; W)$

$= \forall x \; student(x) \longrightarrow \neg (\exists_y \; Exam(y) \wedge \exists e \; want(e) \wedge wanter(e, x) \wedge wantee(e, y))$

Interpretation: there is no exam anywhere that any student wants

$2 > 1$: 1 first then 2

1: $(\lambda Q \; \forall x \; student(x) \rightarrow \neg Q(x))(\lambda S_1 . \; \exists e \; want(e) \wedge wanter(e, S_1) \wedge wantee(e, S_2))$

$= \forall x \; student(x) \longrightarrow \neg (\exists e \; want(e) \wedge wanter(e, x) \wedge wantee(e, S_2)) = \underline{\underline{X}}$

2: $(\lambda Q\ \exists_y .\ Exam(y) \wedge Q(y))(\lambda S_2 .\ X)$

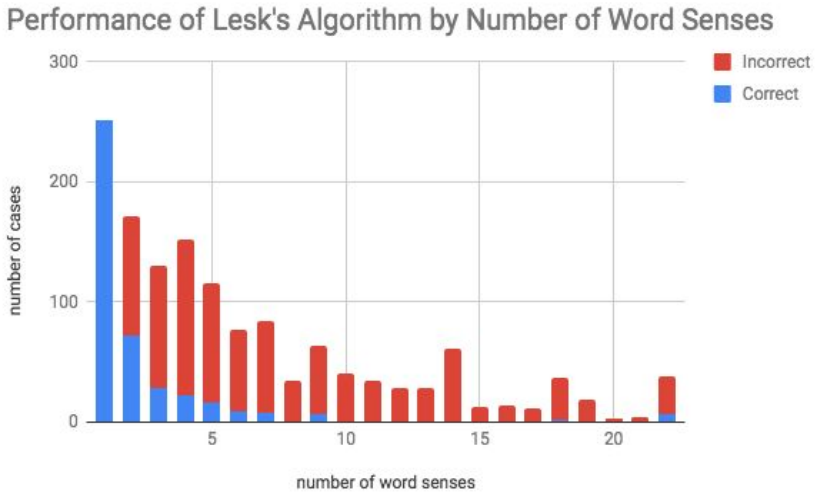
$= \exists_y\ Exam(y) \wedge (\forall x\ student(x) \rightarrow \exists e\ .\ want(e) \wedge wanter(e, x) \wedge wantee(e, y))$

Interpretation: There is a particular exam that no student wants

Question 2 of Assignment 3 focused around the problem of word sense disambiguation. In particular, given a word and the context that surrounds the word, we attempt to find which word sense is expressed. In this assignment we used the dataset from SemEval 2013 Shared Task #12, and utilized WordNet v3.0 as our lexical resource. We divided the dataset into a development set with 194 examples and a testing set with 1450 examples.

To begin we created a baseline algorithm that helped us compare our more advanced models and evaluate their performance more accurately. This baseline model worked by selecting the sense most frequently used for a given word (according to Wordnet), as the sense to resolve the ambiguity. The baseline model scored a 62.3% accuracy on the test set.

The next model we implemented was the NLTK implementation of Lesk's algorithm. Preprocessing the data was the only significant task required for this model. We ensured that words were tokenized and lemmatized and removed stop words. NLTK's implementation of Lesk's algorithm scored 29.6% accuray on the test set. Surprisingly, Lesk's algorithm performed much worse than the baseline model, this suggests that utilizing the context of the words we are trying to disambiguate in this dataset is not as effective as simply selecting the most frequently used word sense for a particular word. This can be explained by analyzing the number of senses for the average word as well as the average number of context words. The average number of senses for a word being disambiguated in the test set is 7.1 and each of those words has on average 16.0 context words associated with it. Therefore it is reasonable to see that Lesk's algorithm would struggle given so little context and so many word sense options. For a given word, the more word sense options that are available, the greater the probability that an incorrect sense will happen to have a higher overlap score than the correct sense. The histogram below shows the number of senses for correct and incorrect instances of Lesk's algorithm on the test set.



Note: words with more than 22 senses were not included in the histogram as Lesk's algorithm did not correctly disambiguate any word with more than 22 senses

It is clear that Lesk's algorithm does better with words that have a lower number of word sense options. Further, ignoring examples with only one word sense to choose from, Lesk's algorithm's accuracy is only 12.2%. It is clear that using only the context words and word sense definitions is not nearly enough to accurately disambiguate word senses, more information is needed such as the frequency of word sense information used in the baseline model.

This was the motivation behind the next model we developed. For this model we combined the overlap feature from Lesk's algorithm and the distributional information about the frequency of word senses from the baseline algorithm. We attempted to boost the baseline model performance using the Lesk signal to partially reorder the senses. To evaluate word senses, a linear combination of word sense frequency rank and overlap between context and sense definition was computed, with the highest scoring sense being chosen. The formula of the linear model for an ambiguous word w and a particular word sense s of w is as follows,

$$senseScore(s, w) = \alpha * senseRank(s) + overlap(context(w), def(s))$$

Where senseRank was simply the reverse order of the frequency, i.e. the most frequently used sense of a group of n senses would have senseRank = n and the least frequently used sense had senseRank = 1. The overlap score was simply a count of the number of words in the set of context words that were found in the definition of the sense as well. We trained the linear function on the development set with a few different values of α. The results are below,

α	8	6	4	2	1	0.9	0.45	0.225
Accuracy (%)	67.6	67.6	59.8	56.7	50.5	43.3	41.8	38.7

From this training it was clear that as the distributional information about the frequency of word senses is weighted more heavily the accuracy increases. Choosing α = 6, the accuracy on the test set was 62.3% (same as the baseline model). This implies that the overlap feature was contributing nearly nothing to the model during testing. Therefore we can conclude that combining Lesk's algorithm overlap information with the frequency of word senses (in a linear combination) does not create a more powerful model in terms of word sense disambiguation. It is possible that a more complex function would improve the accuracy of this model.

With adding an overlap feature to the frequency of word senses proving to be ineffective at improving disambiguation accuracy, this motivated my idea for the last model. My goal was to find a way to make the overlap calculation used in Lesk's algorithm more useful. For this final model I made use of an external lexical resource, in particular a list of unigrams ordered by frequency of use on the internet (https://www.kaggle.com/rtatman/english-word-frequency). The binary calculation used in Lesk's algorithm's overlap score seems too simplistic (if there is an overlap add 1, else add 0). I choose to incorporate word frequency rank into my new overlap calculation. The idea was that if a frequently used word was found within the overlap then it would contribute little to the overlap score (since it is more likely to find this word in text), but if a less commonly used word was in the overlap it would contribute to the score significantly. The formula for word w and sense s was the following,

$$senseScore(s, w) = \sum_{q \in overlap(context(w), def(s))} (1 - (log(freqRank(q)))^{-1})$$

Where freqRank(q) was simply the frequency rank of word q in the list of unigrams (e.g. freqRank('the') = 1). If a word was not found in the list of unigrams it contributed a score of 1 to the senseScore as it was assumed this word was very uncommon. With no parameters there was no training necessary on this model. The test set accuracy was 45.9%. This method was able to outperform Lesk's algorithm by over 15% using minimal external information. This model demonstrates that further lexical resources can make Lesk's algorithm a much more powerful tool. It would be interesting to go back and experiment with combining the baseline model and this modified Lesk's algorithm to see if we can create a more accurate model than the baseline.