Natural Language Processing: Second Assignment

Imagine a speech dialogue system that is conversing about a user's travel plans. We will look at how to interpret sentences about when the user wants to arrive or depart. We will ignore the problems of speech processing and concentrate on the analysis once a sequence of words has been recognised.

The deadline for submission of this assignment is **12 noon on Friday 13th December 2013.**

Please submit your assignment by attaching it as a single file to an email to Chris Mellish (c.mellish@abdn.ac.uk). Please note that Chris will not be responding to email on Friday 13th, but he will acknowledge submissions on Monday 16th.

Task 1: Grammar development – 50%

Write a DCG grammar which accepts at least the following sentences:

- i want to arrive before two am
- i want to depart after three am
- please can i depart before three twenty am
- please can i arrive after three pm

Your grammar need only cover the numbers in these examples (this could be improved by, for instance, incorporating the number grammar used in practical 6) and also it need only cover the types of times shown in these examples (e.g. it need not cover "half past two" or "the back of three"). However, the grammar should as far as possible make use of phrase types that will permit easy expansion (see the notes on the Grammatical Analysis of Sentences, also J&M 9-9.2 (first ed) or J&M 12-12.2 (second ed)).

It would make sense to test your grammar using the software of practical 5, but all you need to hand in is the DCG rules and some brief explanation.

Task 2: Semantic interpretation – 30%

For the semantics of these sentences, we should imagine that we have two variables used by the system, "arr" for arrival time and "dep" for departure time, and we wish to make an expression that can be used to test whether the requirement is satisfied for a given possible journey. Absolute times are to be represented as the number of minutes since the start of the day. For instance:

- i want to arrive before two am ===> arr<120
- i want to depart after three pm ===> dep>900

(a) Write down the parse tree for the sentence "i want to arrive before two thirty am" (according to your grammar), and next to each node in the tree write down the semantics that you would like to have associated with the subtree from that node downwards.

CLUE: I suggest that you need to have the semantics of "before" be λy. (λx. x<y) for things to work out properly. You may need to revise your grammar if this does not work with the rules you currently have. Don't worry about the detailed semantics of "i want to" or "please can i", but instead make the semantics of the whole sentence simply the same as the semantics of "arrive before two am" etc.

(b) Now write down semantic attachments for your grammar rules that will produce this effect. Use the notation of lecture 10. If you want to (but this is not required, and you won't get any extra credit for this), you can try this out using the software of practical 6. In that case, you should know the notation for lambda expressions, etc, with the software (see the appendix). For background on how to use semantic attachments and lambda expressions, see lecture 10, practical 6 and J&M 15-15.1 (first ed) or 18-18.2 (second ed).

Task 3: Discussion of the approach – 20%

Lecture 11 discusses approaches to semantic interpretation that do not require an explicit grammar and interpretation rules (see also J&M chapter 17 (first ed) or chapter 20 (second ed) – you will only want to skim this material). Write a very brief discussion (max 100 words) of whether such techniques might be able to handle this imaginary travel application.

Appendix: Semantic interpretation software notation

- lambda abstraction - λ x. y is represented for the software as x:y (a colon between the variable and the body)
- variables - for lambda abstractions, arr and dep. Just use names starting with lower case letters, no quotes
- lambda application - x applied to y is represented by {x, y} (curly brackets)
- arithmetic expressions - you can write things like x<y directly

For instance, the expressions:

- v:and({m1,v},{m2,v}).
- p:f:eats(p,f)).

might be written more normally as:

- λv. (m1(v) & m2(v))
- λp. (λf. eats(p,f))