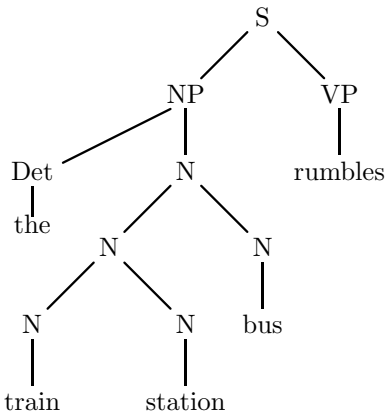
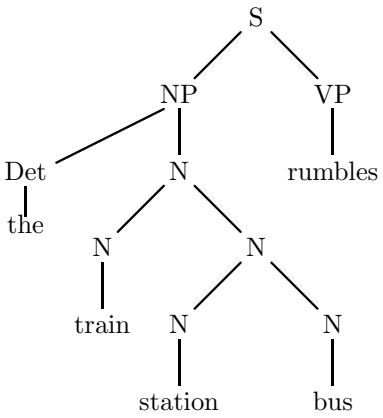
Natural Language Processing 2004 Paper 8 Question 14

Note: This question is taken from lecture 4 of the course.

Part a





Part b A chart records partial results during parsing. A chart is a list of edges. Each edge records a rule application and has the following structure:

[*id*,*left_vertex*, *right_vertex*,*mother_category*, *daughters*]

A vertex is an integer representing a point in the input string. *mother_category* refers to the rule that has been applied to create the edge. *daughters* is a list of the edges that acted as the daughters for this particular rule application.

The following pseudo-code is taken from the lecture notes:

Parse:
Initialize the chart (i.e., clear previous results)
For each word *word* in the input sentence, let *from* be the left vertex, *to* be the right vertex and *daughters* be (*word*)
For each category *category* that is lexically associated with *word*
Add new edge *from*, *to*, *category*, *daughters*
Output results for all spanning edges
(i.e., ones that cover the entire input and which have a mother corresponding to the root category)

Add new edge *from*, *to*, *category*, *daughters*:
Put edge in chart: [*id*,*from*,*to*, *category*,*daughters*]
For each *rule* in the grammar of form *lhs* `->` $cat_1 \ldots cat_{n-1}$,*category*
Find set of lists of contiguous edges $[id_1,from_1,to_1,\ cat_1,daughters_1] \ldots [id_{n-1},from_{n-1},from,\ cat_{n-1},daughters_{n-1}]$
(such that $to_1 = from_2$ etc)
For each list of edges, **Add new edge** $from_1$, *to*, *lhs*, ($id_1 \ldots id$)

Processing (i)

```
. the . train . station . bus . rumbles .
0     1       2         3     4         5
```

The edges are:

```
id  left   right  mother   daughters
1    0       1      Det      (the)
2    1       2      N        (train)
3    0       2      NP       (1,2)
```

```
4    2       3       N       (station)
5    1       3       N       (2,4)
6    0       3       NP      (1,5)
7    3       4       N       (bus)
8    2       4       N       (4,7)
9    1       4       N       (2,8) i.e. (train (station bus))
10   0       4       NP      (1,9)
11   1       4       N       (5,7) i.e. ((train station) bus)
12   0       4       NP      (1,11)
13   4       5       VP      (rumbles)
14   0       5       S       (10,5)
15   0       5       S       (12,5)
```

Part c For packing, the modification to the algorithm is to change the daughters value on an edge to be a set of lists of daughters and to make an equality check before adding an edge so we don't add one that's equivalent to an existing one.

That is, if we are about to add an edge:

[*id*,*left_vertex*, *right_vertex*,*mother_category*, *daughters*]

and there is an existing edge:

[*id-old*,*left_vertex*, *right_vertex*,*mother_category*, *daughters-old*]

we modify the old edge to record the new daughters:

[*id-old*,*left_vertex*, *right_vertex*,*mother_category*, *daughters-old* $\sqcup$ *daughters*]

For (i), everything proceeds as before up to edge 10. However, rather than add edge 11 we match this with edge 9, and add the new daughters to that.

```
9    1       4       N       {(2,8),(5,7)}
```

Edges 12 and 15 are not generated.

For sentence (ii), there are 5 possible bracketings of the 4 Ns, so the basic algorithm would produce a total of 29 edges. 14 of these are saved by packing.

The algorithm given in (b) is exponential in the case where there are an exponential number of parses. Packing means that it runs in cubic time. Producing the unpacked output is still exponential, however.

Unpacking is not necessary for some applications.