Tutorial 5a
The Left-Corner Algorithm

1 Notation Conventions

Left-corner parsing shares part of the approach with shift-reduce and part with top-down: Both existing elements (a word's category or a constituent we have found) and predictions (based on an existing elements and the rules in the grammar) have a place on the stack. In this parser, we are going to use "slash notation" on the stack to represent both types of elements:

- If an NP has previously been found and the grammar contains S $\rightarrow$ NP VP, we can predict that the NP may actually be the first part of an S whose VP has not been encountered yet. Therefore, on the stack, we can rewrite `NP` as `S/VP`.
- Using the same logic, completed elements are elements in which nothing is missing:
  - Once the sentence's VP is found, the stack will contain `[S/VP VP]`, and the two elements can then be merged to obtain `S/[]`, which in the case of binary grammars is sometimes written simply as `S`.
  - Word categories in the lexicon can be considered as minimal constituents not missing anything: If the next word in the sentence is *cat*, it can be added to the stack as `N/[]`, or more simply as `N`.

2 Operations in Left-Corner Parsing

There are three operations left-corner parsing, of which only one is different in the arc-standard and arc-eager variants.

Shift If there remain words to be processed, lookup the next word in the lexicon and add its category to the stack as `cat`.

Predict Using the rules of the grammar, complete elements can be used to predict their sister node. See the example above: Given the rule S $\rightarrow$ NP VP, `NP` on top of the stack can become `S/VP`.

Merge *(arc-standard variant)* If we predict a constituent, and then happen to find just what is missing to make it complete, then the two can be removed from the stack and replaced with the completed constituent. For example, if the stack contains `[NP/N N]`, both can be replaced with `[NP]`.

Merge *(arc-eager variant)* If we predict a constituent and then find something that could potentially be its next missing daughter, then we can assume we are on the right track to find the predicted constituent.

Example: If the verb in `[S/VP, V]` has just been rewritten as `VP/NP` in a prediction step (state of the stack: `[S/VP, VP/NP]`), then this can be replaced by `[S/NP]`.

Note that the arc-standard variant of "merge" is actually a special case of the arc-eager rule where nothing is missing (in the last example, the arc-standard variant would require the VP to be complete, its NP having already been found (stack before merging: `[S/VP, VP]`).

3 Using an Oracle

Due to the number of rules that can be used for left-corner prediction, there are often many possibilities the left-corner parser must try out at the prediction stage, leading to a lot of backtracking and reprocessing. However, in many cases, we can actually foresee that a given rule will not lead to a correct parse by taking into account that the left-corner relation is transitive: If we are trying to build an S using S → NP VP, than all acceptable left-corners of the NP must also be left-corners of the S.

Thus we can reduce the amount of backtracking in the parser by requiring that a category only be added to the stack if it can in some way lead to the goal we are currently pursuing, i. e. if it is a left-corner of that goal. Whether or not a category is a possible left-corner of the type of constituent we are currently trying to build can be determined either programmatically (e. g. using a COGENT condition definition), or by consulting a precompiled list of possible left-corners, which is always finite for a given grammar.

`lc(det,det)`	all categories are always their own left-corners. This can also be specified as `lc(Cat,Cat)`.
`lc(det,np)`	if NP → Det N
`lc(det,s)`	if, in addition, S → NP VP