Parsing

- During parsing, we take the stream of tokens that the scanner produced and attempt to determine if they represent a valid syntactic sequence for the language
- Ideally, we also either generate a structure (such as a parse tree) that represents the valid derived program structure, or provide suitable error messages for invalid program structures
- Parsers are generally described as either top-down or bottom-up, based on the order in which they deduce the tree structure

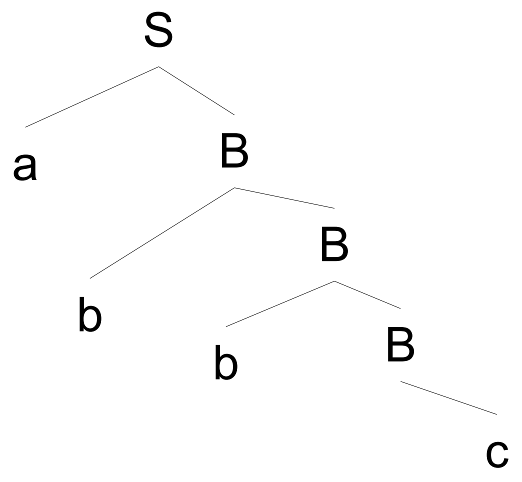
Grammars,derivations, parse trees

- As per our earlier discussions on formal descriptions of languages, we will often describe the syntax of a language using a set of context free grammar rules
- The structure of a valid program under the grammar can then be represented either as a derivation sequence or a parse tree

S-->aB
B-->bB
B-->c

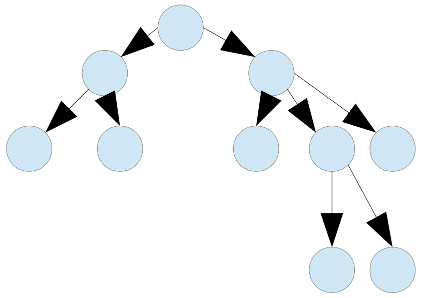
Target string abbc

S-->aB (rule 1 on S)
  -->abB (rule 2 on B)
  -->abbB (rule 2 on B)
  -->abbc (rule 3 on B)



Top-down vs bottom-up

- visualize the results as a parse tree
- top-down parsers attempt to start from the root and apply rules that expand/build the tree towards the leaves
- bottom-up parsers attempt to start from the leaves and apply rules that group/build from those towards the root



Top-down parsers

- Top-down parsers start from the top level non-terminal for a grammar and repeatedly try to pick which non-terminal derivation rule to apply next, until eventually the result is a collection of tokens matching the input sequence
- If an incorrect choice of rule is made at some point in the derivation process then the tokens produced won't match with the input sequence, and we'll back up our derivation to the most recent decision point, pick a different rule, and try again

Bottom-up parsers

- Bottom-up parsers will read the stream of input tokens, looking for sequences that can be grouped together to form the right hand side of a grammar rule, e.g. In a string abcdefg realizing the bcd matches X-->bcd
- We then replace the grouped tokens with the nonterminal from the left hand side of the rule (e.g. X) giving us a new string to try to parse (aXefg)
- Note the new string is a mix of tokens and nonterminals
- We continue until we are left with simply the root nonterminal for the grammar