Parsing

review: CFGs

- Given a start rule, productions tell us how we can *rewrite* non-terminals into other strings
- Some productions rewrite into λ. That just removes the non-terminal

$S \rightarrow A\ B$

$A \rightarrow A\ a$

$A \rightarrow a$

$B \rightarrow B\ b$

$B \rightarrow b$

- To derive the string "a a b b b" we can do the following rewrites:

$S \Rightarrow A\ B \Rightarrow A\ a\ B \Rightarrow a\ a\ B \Rightarrow a\ a\ B\ b \Rightarrow$
$a\ a\ B\ b\ b \Rightarrow a\ a\ b\ b\ b$

the problem of parsing

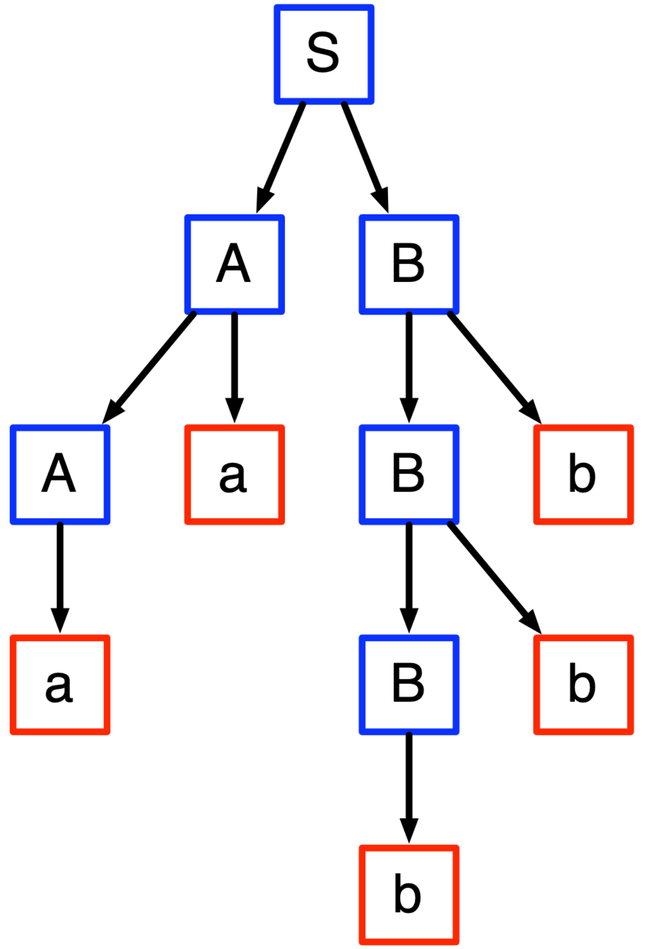
- Using a grammar to generate a string is straightforward

- But **parsing** solves the opposite problem: is a string part of a language?

    - Is there some combination of rewrites that will generate a string?
    - What rewrites were those?

parse trees

- Using a grammar to generate a string is straightforward
- But **parsing** solves the opposite problem: is a string part of a language?
- A **parse tree** shows how a string was generated
  - Interior nodes: non-terminals
  - Leaf nodes: terminals
  - Children of interior nodes: the terminals and non-terminals generated by applying a rule



what does a parser do?

- Parsing is recognizing members in a language specified/defined/generated by a grammar
- Determine the structure of a program (parse tree)
- A parse tree is like a sentence diagram

  The red cat pushed the ball under the table

- When a parser recognizes a rule, it typically takes some action
  - A compiler might generate code or intermediate representation
  - An interpreter might execute the code

next: how do we build a parser?