Computational Psycholinguistics

Lecture 3: Human Sentence Parsing Mechanisms (HSPM)

Afra Alishahi

November 10, 2008

(based on slides by Matthew Crocker)

Sentence Processing

- Combine the words of an utterance to yield the interpretation of a sentence.
- Sentence processing is compositional
  - most of the sentences have never been seen before
  - the interpretation of a sentence depends of the meaning of each of its words, and their order

Human Sentence Processing

- Main sources of information:
  - Grammar: an appropriate formal description of how words of a sentence can be structured into a connected, interpretable representation
  - Empirical evidence concerning people's behaviour when they process language
- Human Sentence Parsing Mechanism (HSPM)
  - Plausible mechanisms used to build syntactic representations using grammatical knowledge

Context Free Grammars

- A simple phrase structure grammar:

S → NP VP
PP → P NP
VP → V NP
VP → V
NP → NP PP
NP → Det N

Det → the
Det → every
N → man, woman
N → book
P → with
V → read, reads

Parsing: rules of grammar must be used to derive (generate) an utterance.

Nonterminal Nodes

Pre-terminal Nodes

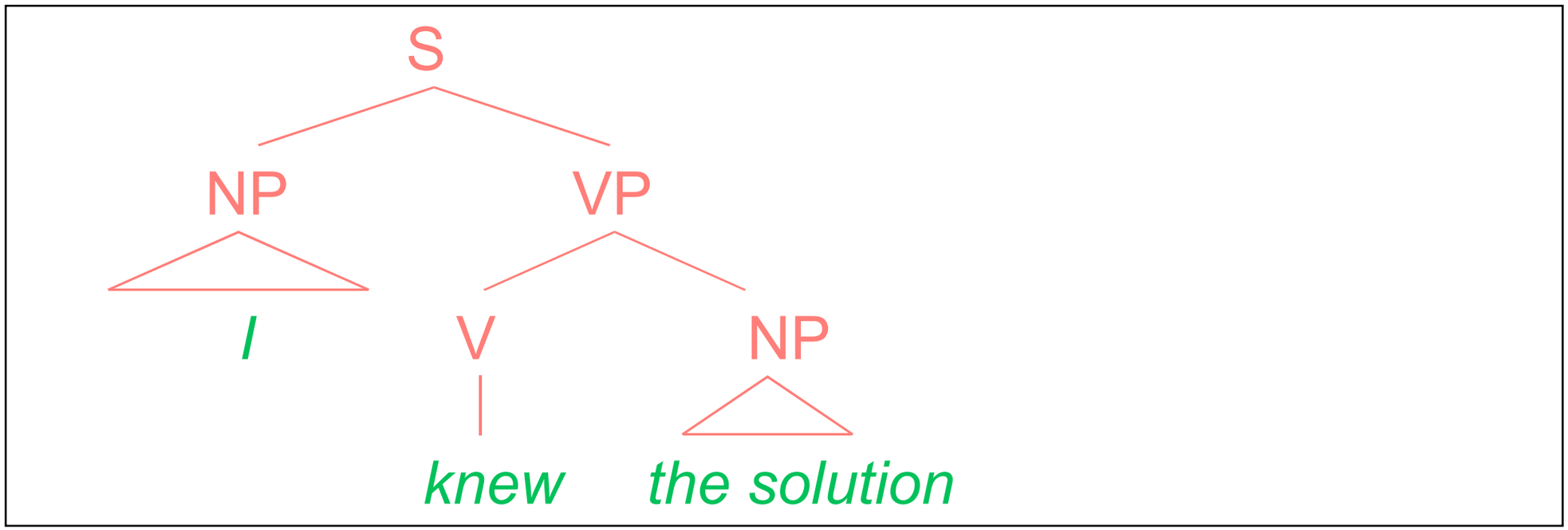
Terminal Nodes

Parsing Issues

- Properties of HPSM
  - Input sentence must be processed and a connected interpretation must be maintained incrementally
  - Parsing decisions must reflect those of human processing in dealing with structural ambiguity
  - Difficulties in human sentence processing must be explained through corresponding increases in processing complexity

Ambiguity

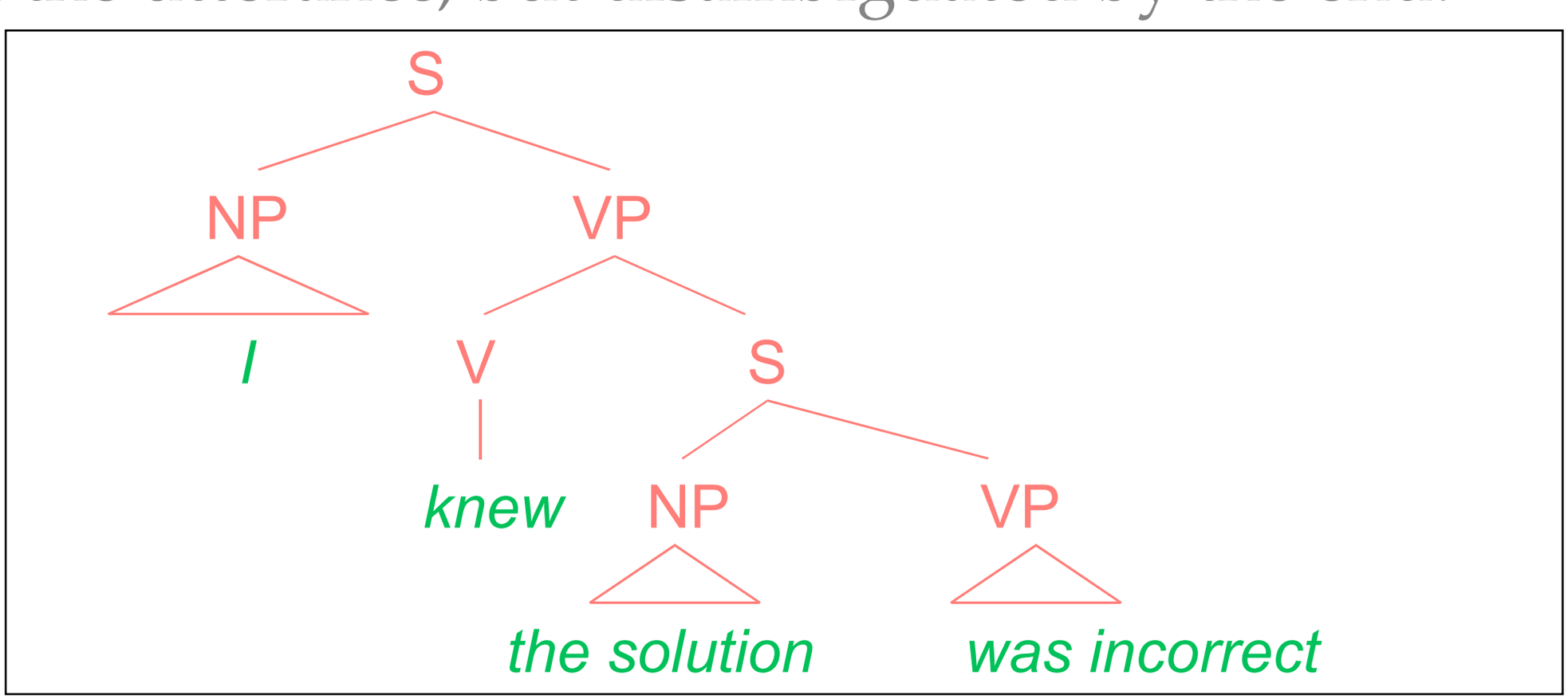
- Local ambiguity
  - more than one possible analyses for the initial sub-string of the utterance, but disambiguated by the end.



I knew the solution ...

Ambiguity

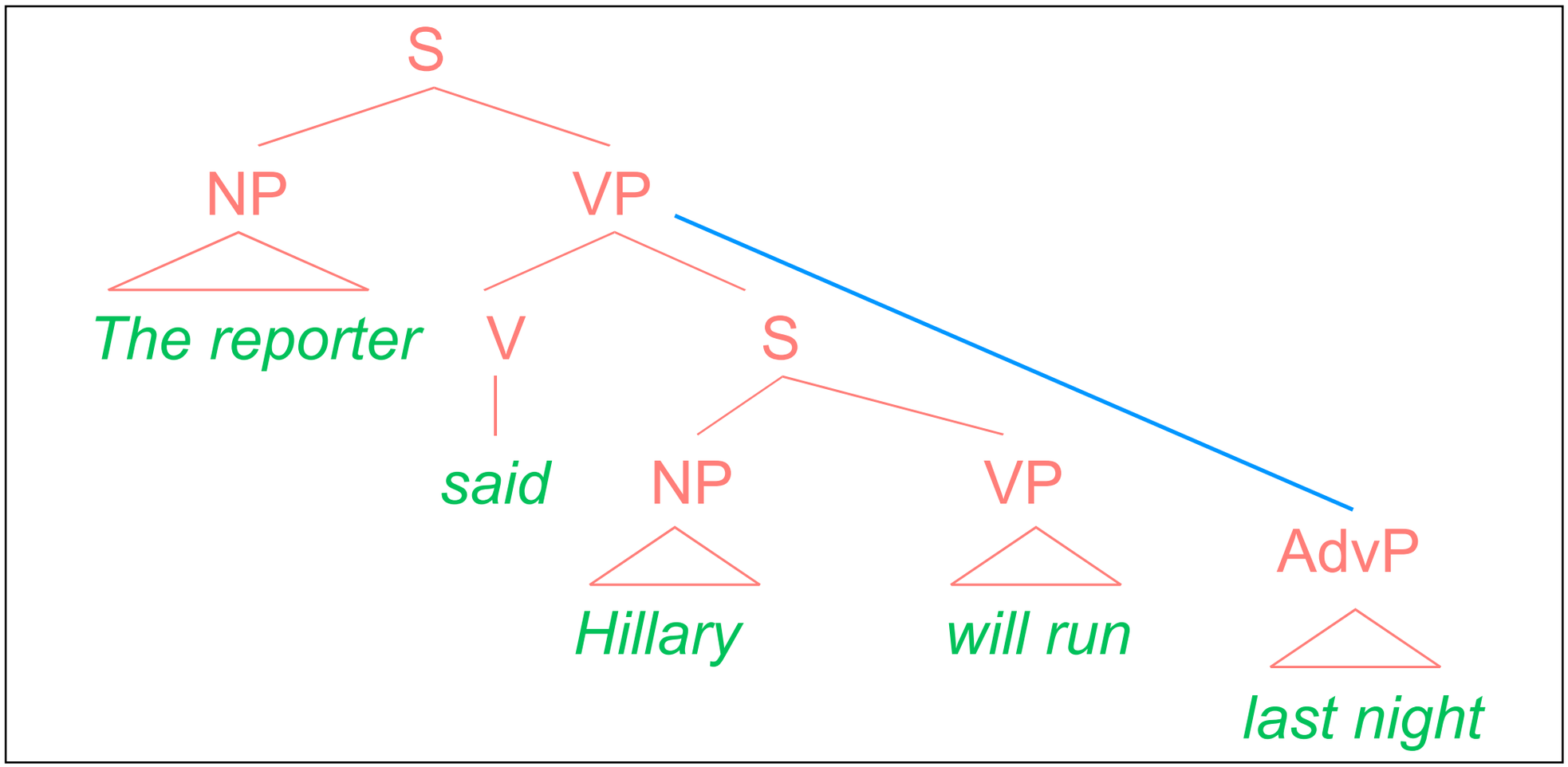
- Local ambiguity
  - more than one possible analyses for the initial sub-string of the utterance, but disambiguated by the end.



I knew the solution was incorrect.

Ambiguity

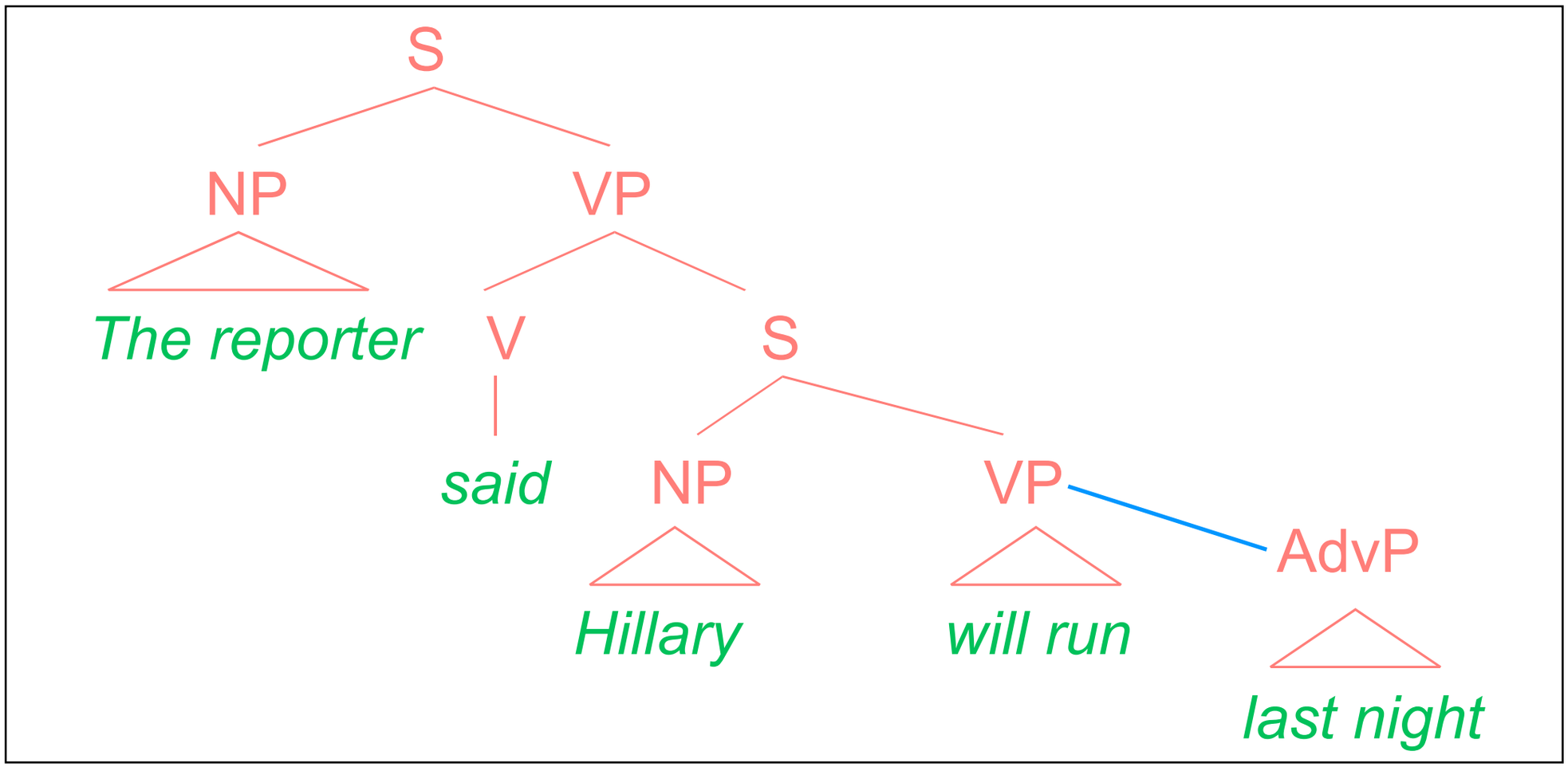
- Global ambiguity: the sentence has two possible interpretations.



"The reporter said Hillary will run last night."

Ambiguity

- Global ambiguity: the sentence has two possible interpretations.



"The reporter said Hillary will run last night."

Processing Complexity

- Garden-path sentences

  "The woman sent the letter was pleased."

  compare to:

  "I know the solution to the problem was wrong."

- Center-embedded constructions:

  "The rat that the cat that the dog chased bit died."

  compare to:

  "John's brother's cat despises rats."

  "This is the dog that chased the cat that bit the rat that died."

Parsing Mechanisms

- Algorithms to build a parse tree for an utterance
  - left-to-right, head-driven, right to left
  - top-down, bottom-up, mixed
- Processing complexity:
  - Time: what time is required to parse a sentence as a function of sentence length, grammar size?
  - Space: how much memory does the parser require?

Bottom-up Parsing

- Driven by the words in the sentence
- Combine words `bottom-up' into higher level constituents

- A simple instance: shift-reduce parser
  - previously seen constituents are stored in a stack
  - shift: move the algorithm to the next word
  - reduce: combine the constituents already found into new constituents

Bottom-up Parsing: An Example

"The …"

stack: [Det]

Det
|
the

Bottom-up Parsing: An Example

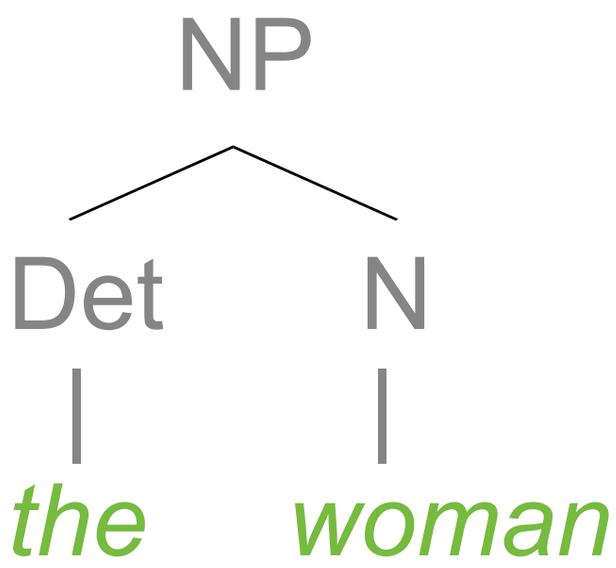
"The woman ..."

stack: [Det,N]

Bottom-up Parsing: An Example

"The woman ..."

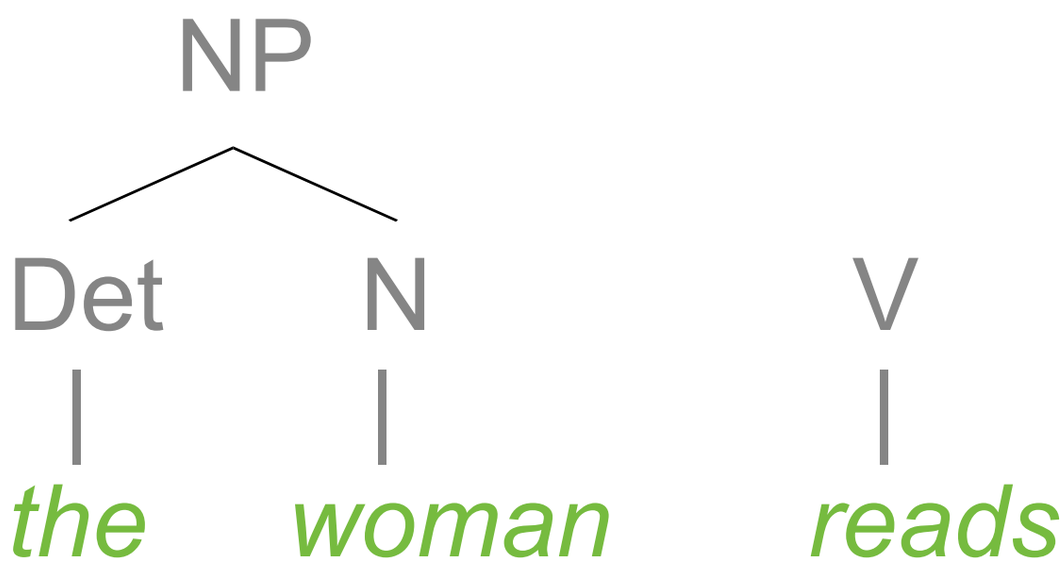
stack: [NP]



Bottom-up Parsing: An Example

"The woman reads."

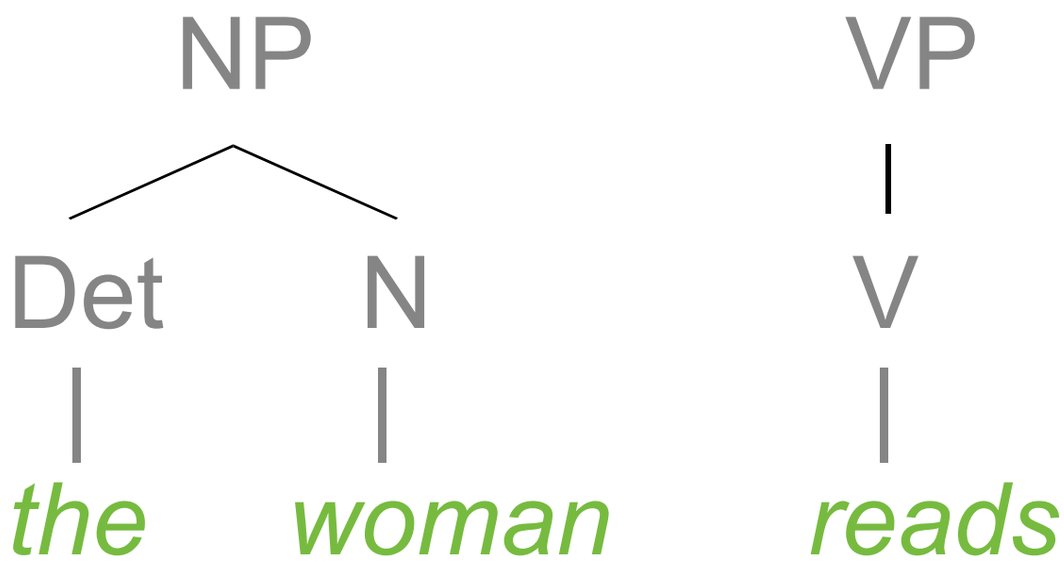
stack: [NP,V]



Bottom-up Parsing: An Example

"The woman reads."

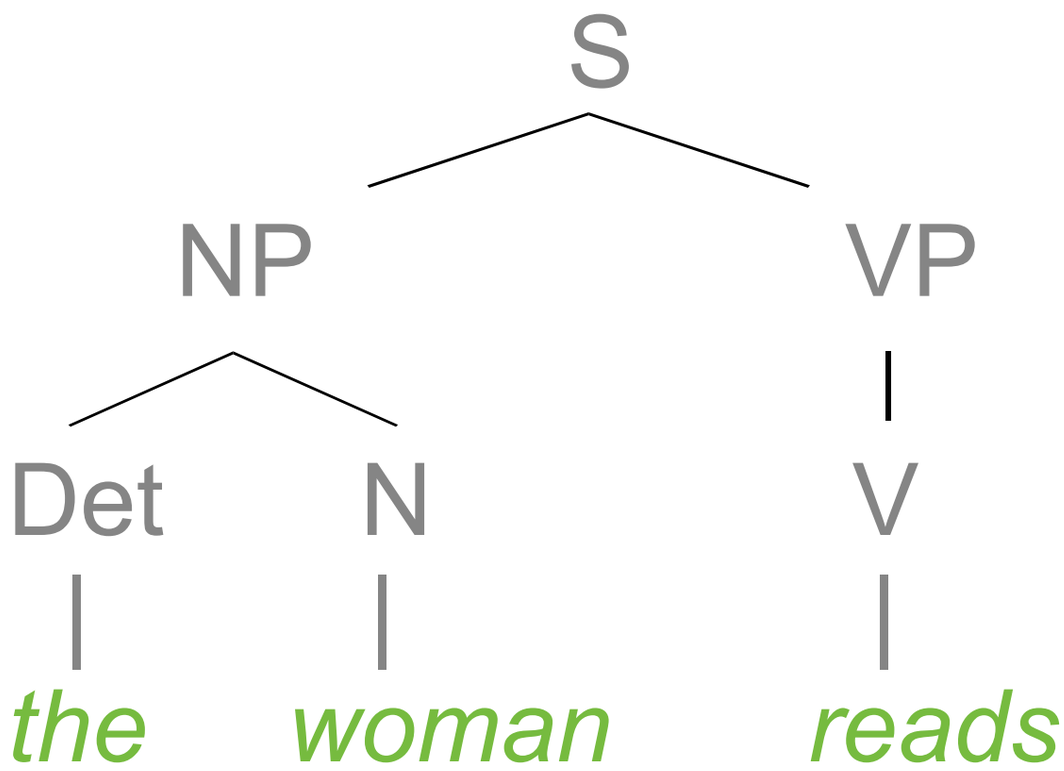
stack: [NP,VP]



Bottom-up Parsing: An Example

"The woman reads."

stack: [S]



Shift-Reduce Parsing Algorithm

1. Initialize *Stack* = []
2. loop: Either *shift*:
    - Determine category, *C*, for next word in sentence;
    - Push *C* onto the stack;
3. Or *reduce*:
    - If categories on the *Stack* match the right-hand side of a rule:
        - Remove those categories from the *Stack*;
        - Push the left-hand side category onto the *Stack*;
4. No more words to process?
    - If *Stack* = [S], then done;
    - else, fail.
5. Goto loop

Choice Points

1. Initialize *Stack* = []
2. loop: Either *shift*:
   - Determine category, *C*, for next word in sentence;
   - Push *C* onto the stack;
3. Or *reduce*:
   - If categories on the *Stack* match the right-hand side of a rule:
     - Remove those categories from the *Stack*;
     - Push the left-hand side category onto the *Stack*;
4. No more words to process?
   - If *Stack* = [S], then done;
   - else, fail.
5. Goto loop

which operation?

which rule?

Top-down Parser

- Assumes that there is a sentence, and works its way down the tree to the words
- Algorithm:
- Expected constituents are stored in a stack
- Each constituent is expanded using a grammar rule
- The predicted constituents are matched against the input words

Bottom-up Parsing: An Example

“ ... ”

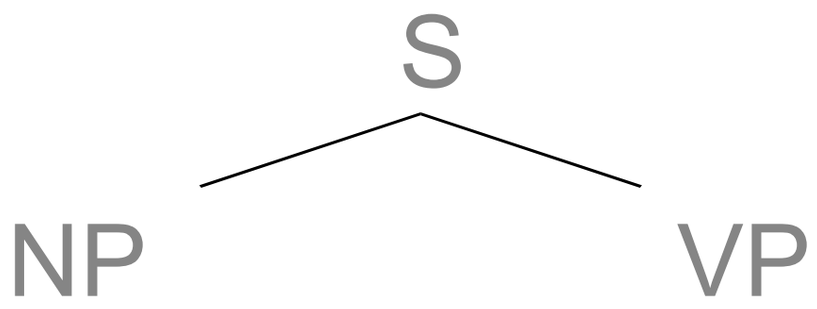
stack: [S]

S

Bottom-up Parsing: An Example

“ ”
…

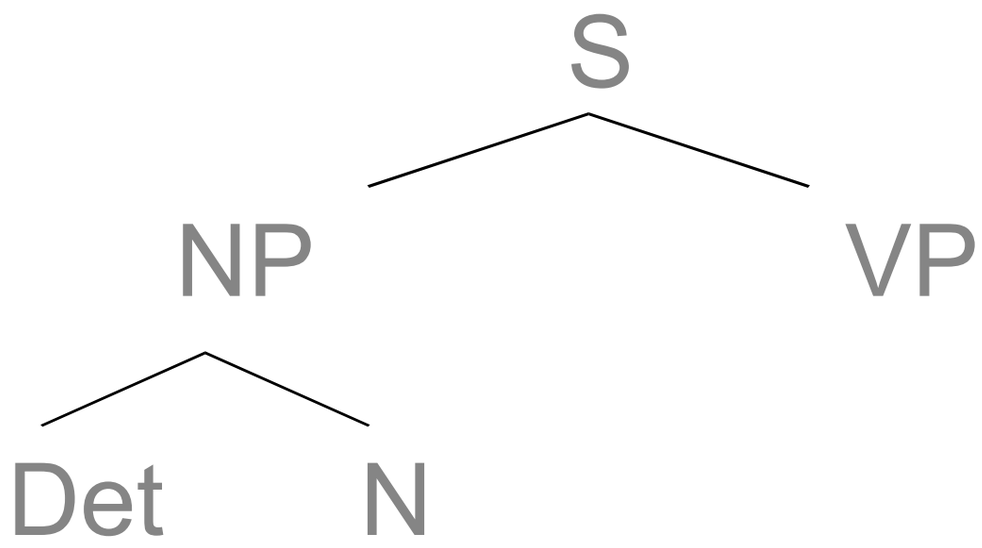
stack: [NP,VP]



Bottom-up Parsing: An Example

“ … ”

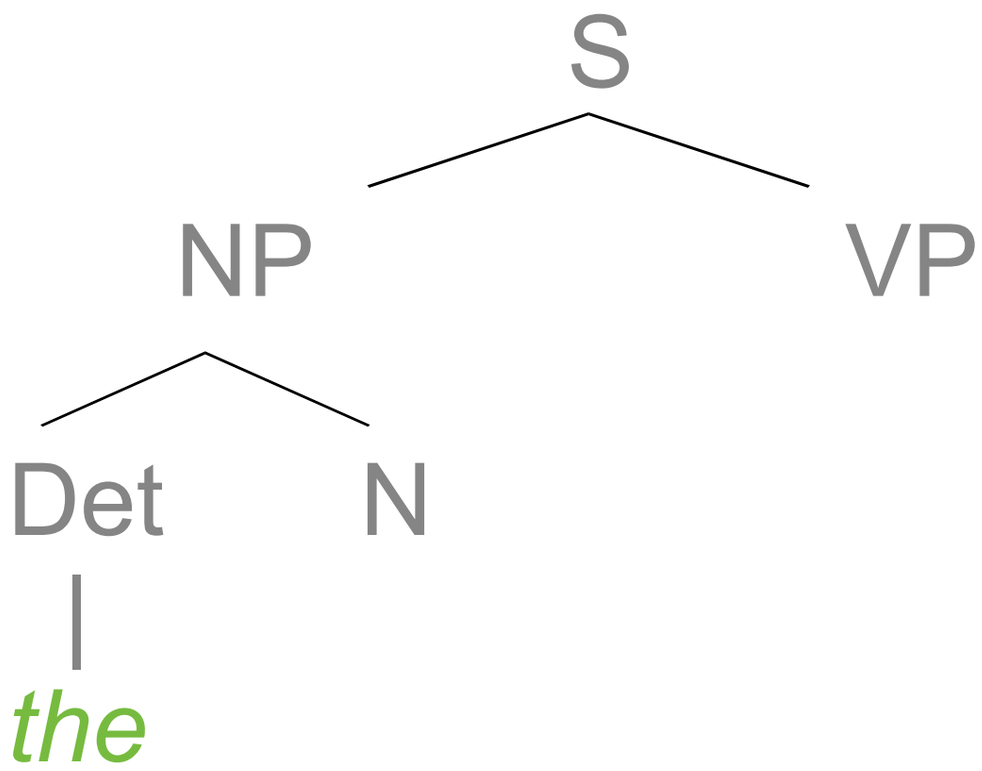
stack: [Det,N,VP]



Bottom-up Parsing: An Example

"The …"

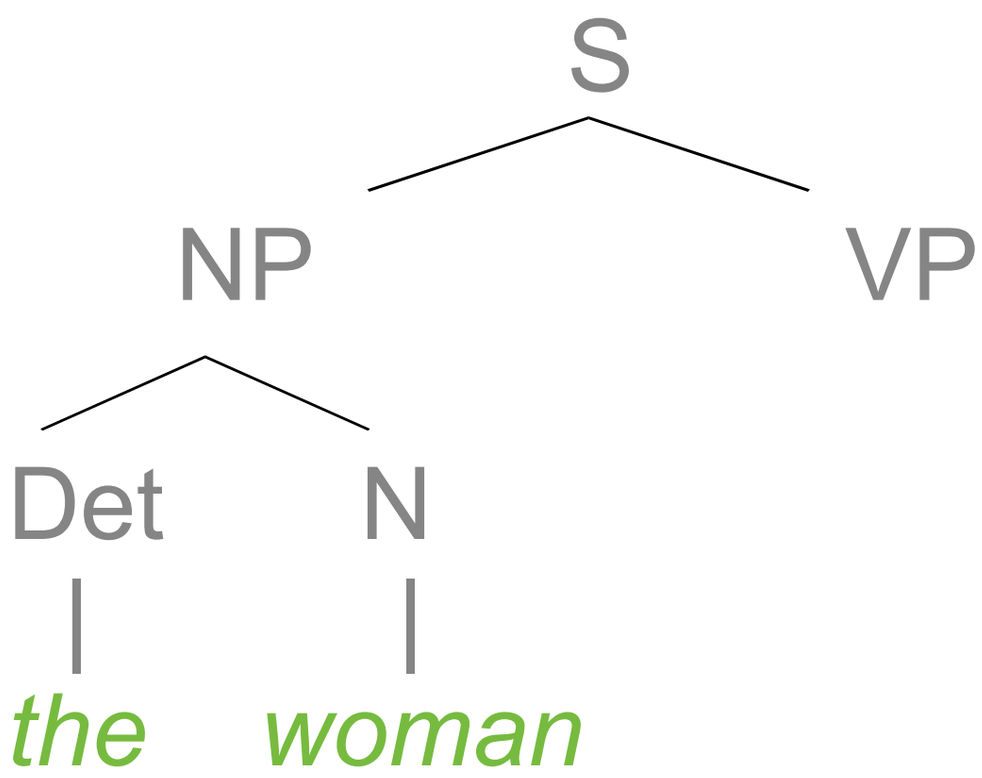
stack: [N,VP]



Bottom-up Parsing: An Example

"The woman …"

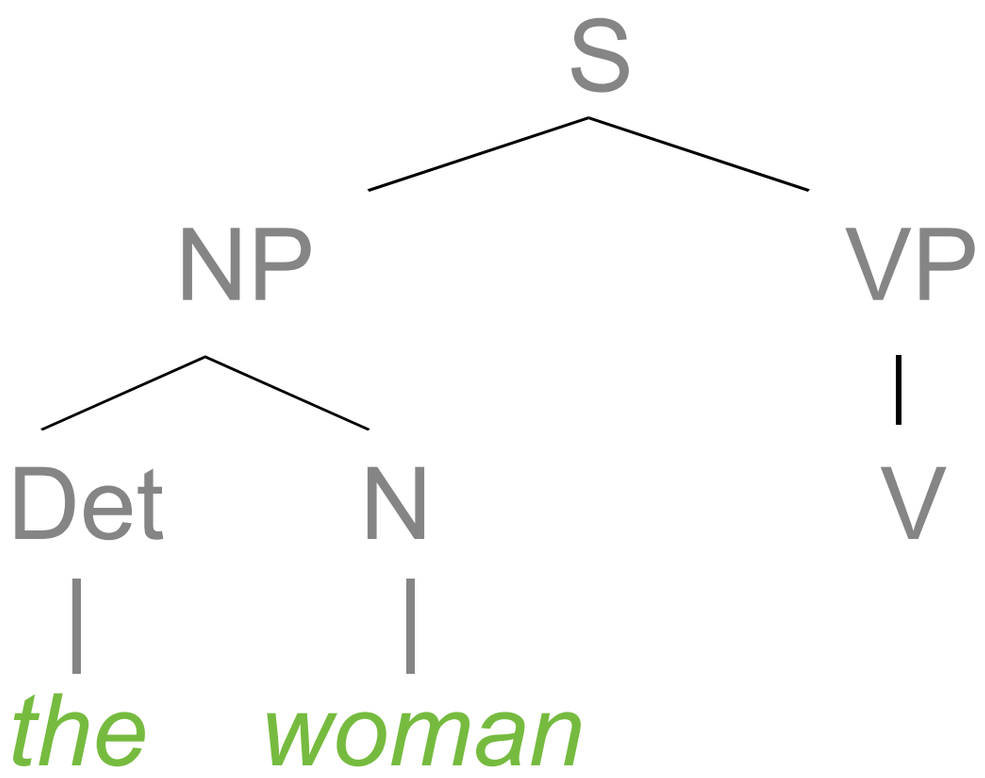
stack: [VP]



Bottom-up Parsing: An Example

"The woman …"

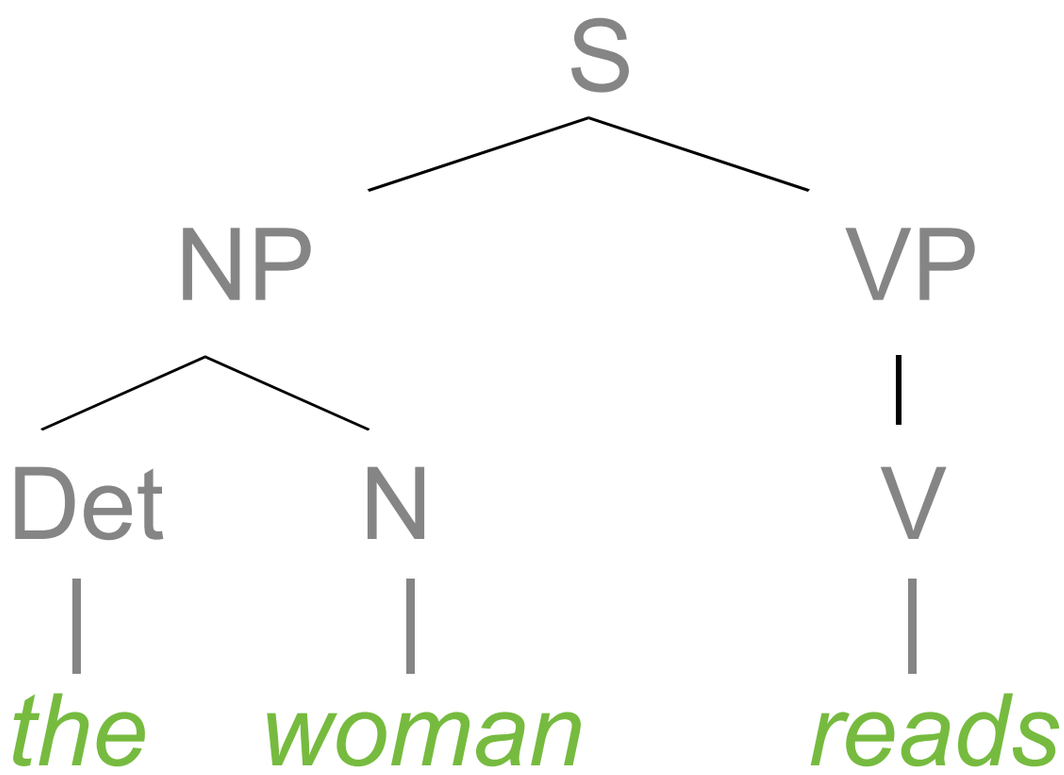
stack: [V]



Bottom-up Parsing: An Example

"The woman reads."

stack: []



Top-down Parsing Algorithm

1. Initialise Stack = [S]
2. If top(Stack) is a non-terminal, N:
   - Select rule N ➜ RHS;
   - pop(N) off the stack and push(RHS) on the stack;
3. If top(Stack) is a pre-terminal, P:
   - Get next word, W, from the input;
   - If P ➜ W, then pop(P) from the stack;
   - Else fail;
4. No more words to process?
   - If Stack = [], then done;
   - else, fail.
5. Goto ②

Choice Points

1. Initialise Stack = [S]
2. If top(Stack) is a non-terminal, N:
   - Select rule N ➜ RHS;
   - pop(N) off the stack and push(RHS) on the stack;
3. If top(Stack) is a pre-terminal, P:
   - Get next word, W, from the input;
   - If P ➜ W, then pop(P) from the stack;
   - Else fail;
4. No more words to process?
   - If Stack = [], then done;
   - else, fail.
5. Goto ②

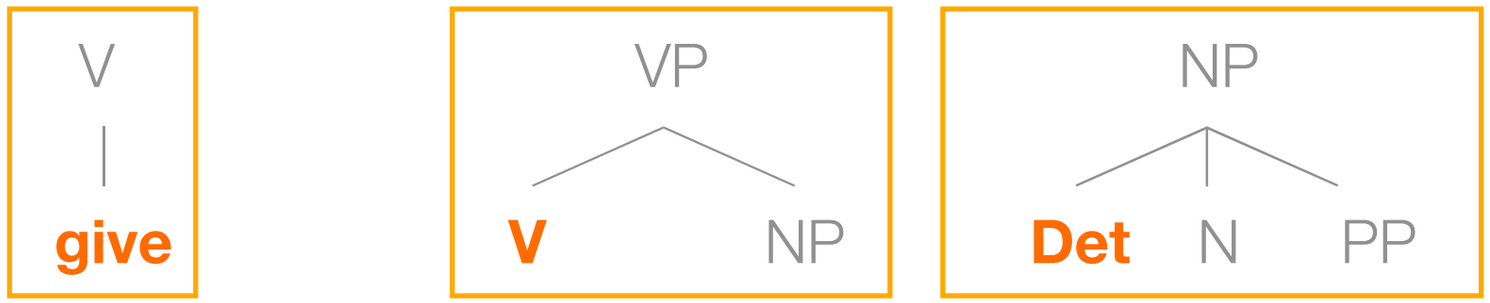
which rule?

Cognitive Plausibility

- Incrementality
  - Both process the input words incrementally
  - Bottom-up does not maintain a connected interpretation incrementally
- Input-driven
  - Bottom-up is input-driven
  - Top-down is not, and has problem with left-recursion

Left Corner Parser

- Combines bottom-up and top-down strategies
- Main intuition:
  - Match the "left-corner" of a rule to the input (bottom-up) to project its mother category
  - Predict the remaining categories on the right (top-down)



Left Corner Parser: an Example

"The …"

stack: [S]

S

Det

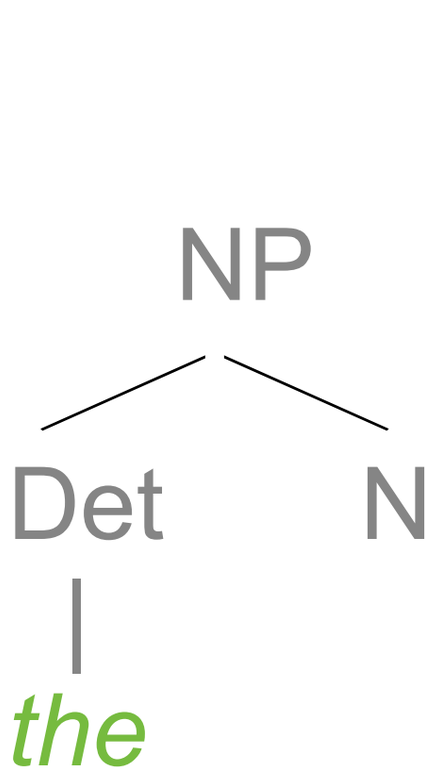
|

the

Left Corner Parser: an Example

"The …"

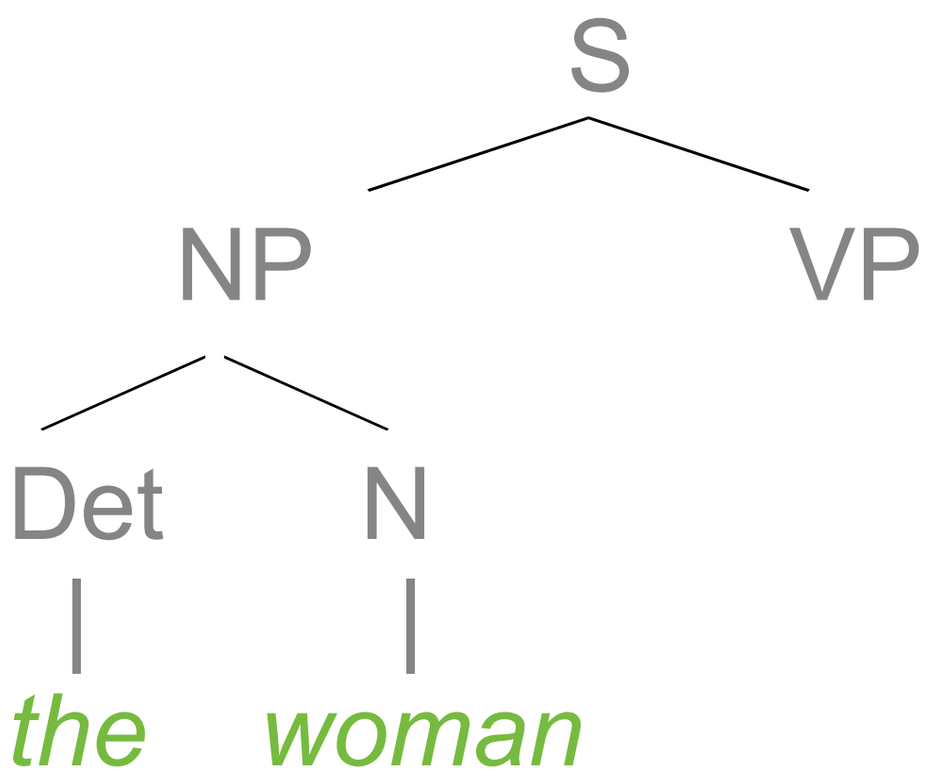
stack: [N,S]



Left Corner Parser: an Example

"The woman …"

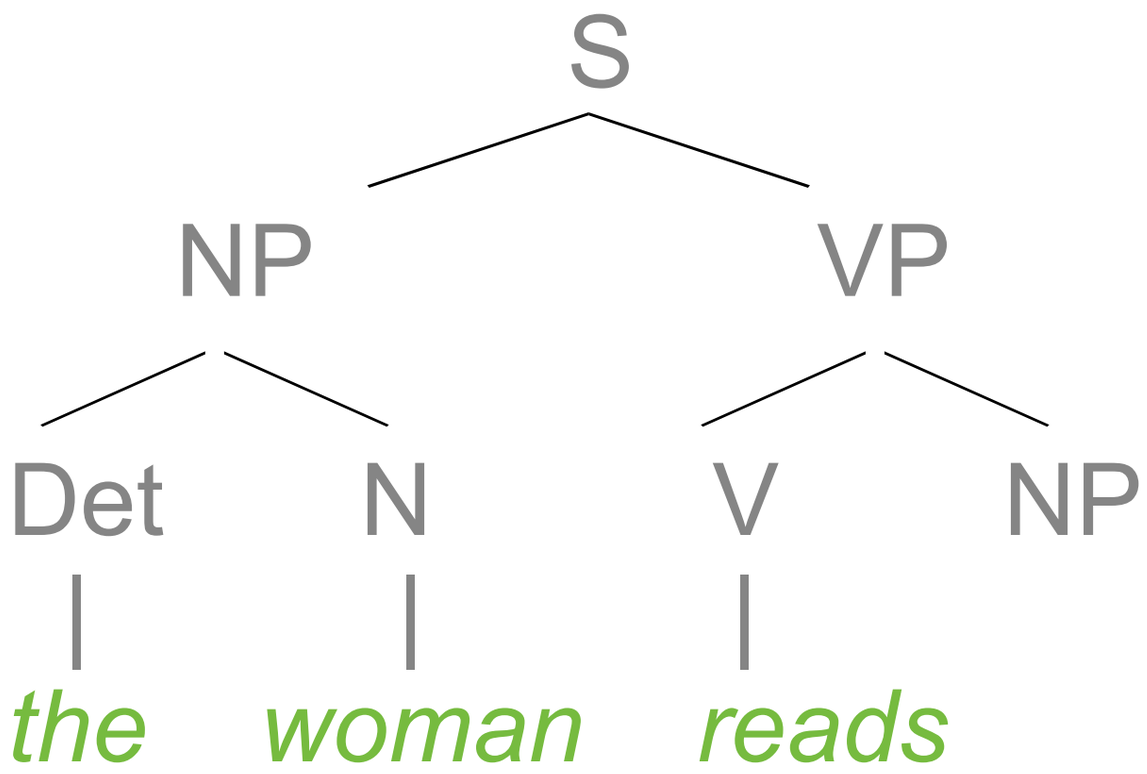
stack: [VP]



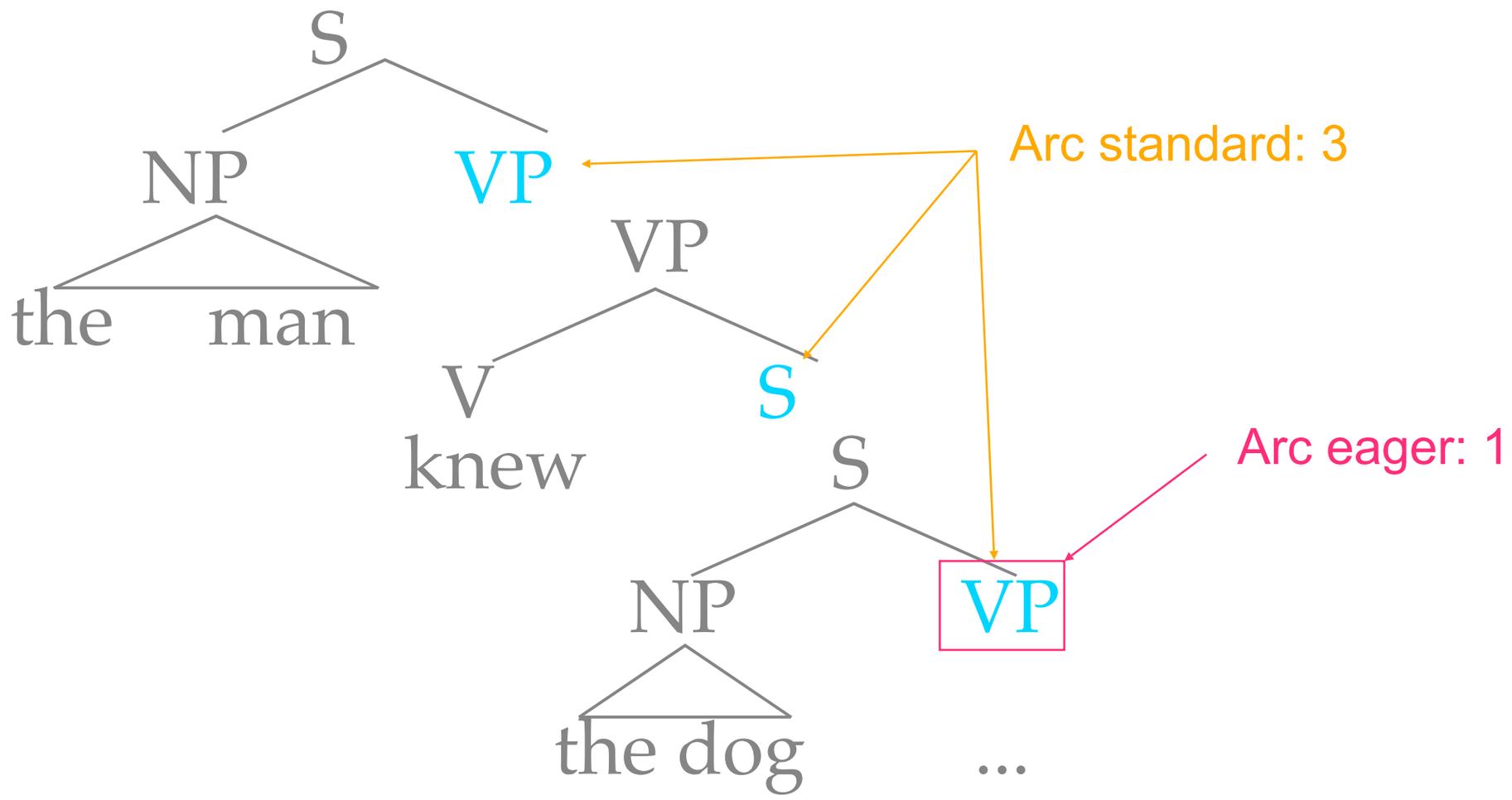
Left Corner Parser: an Example

"The woman reads …"

stack: [NP]

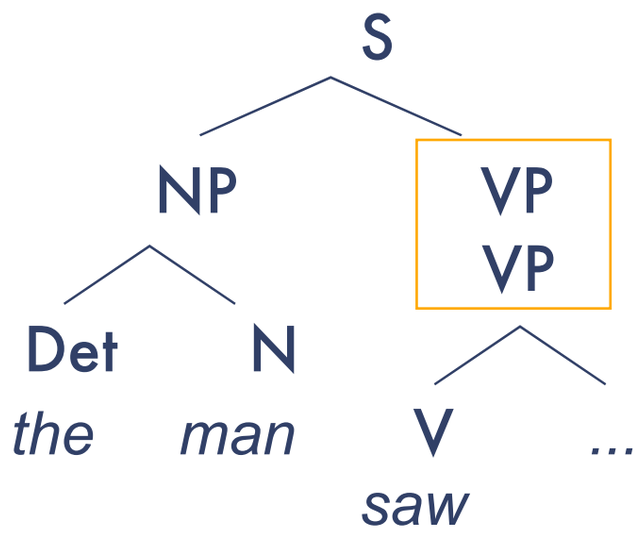


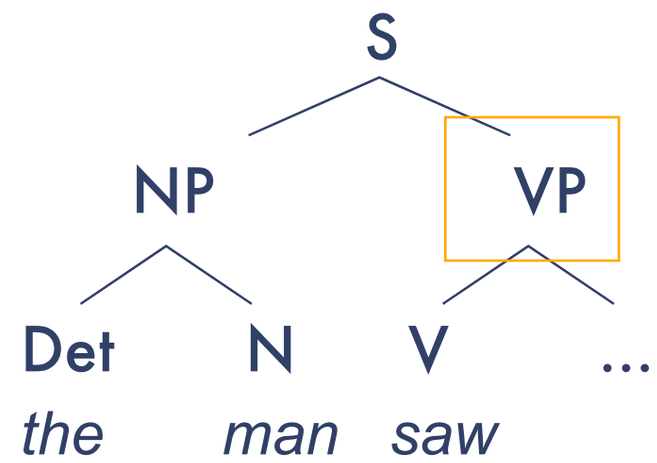
Is LC Parser Incremental?



Variations of LC Parser

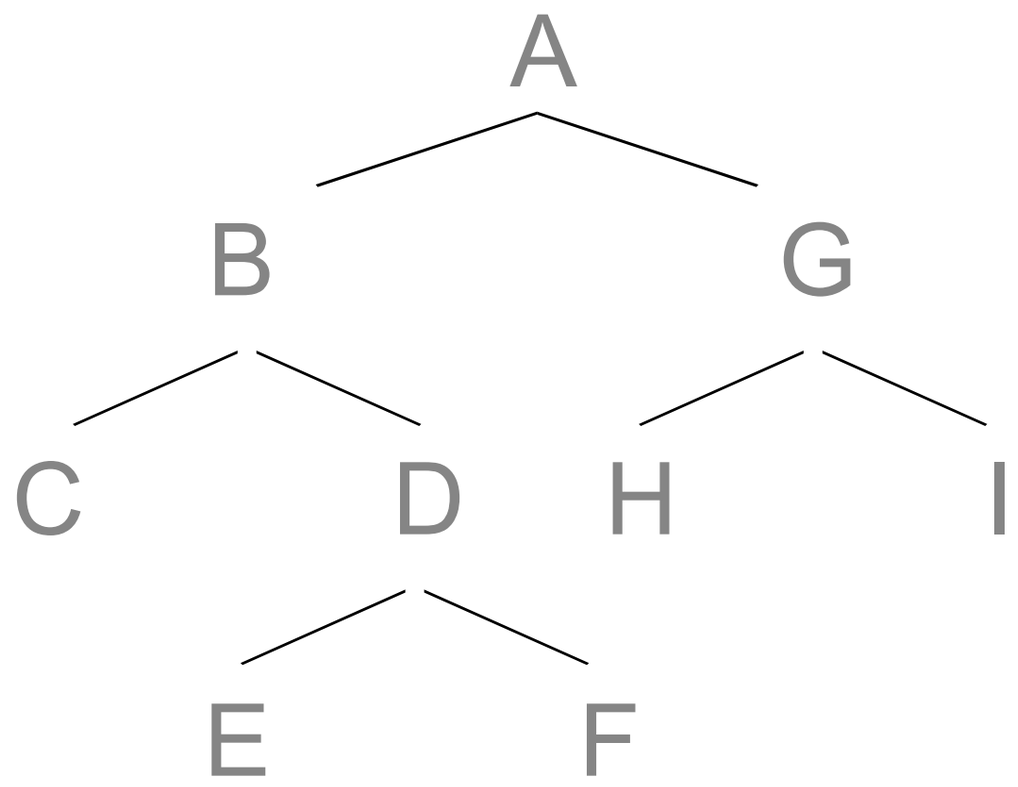
- Arc-standard vs. arc-eager





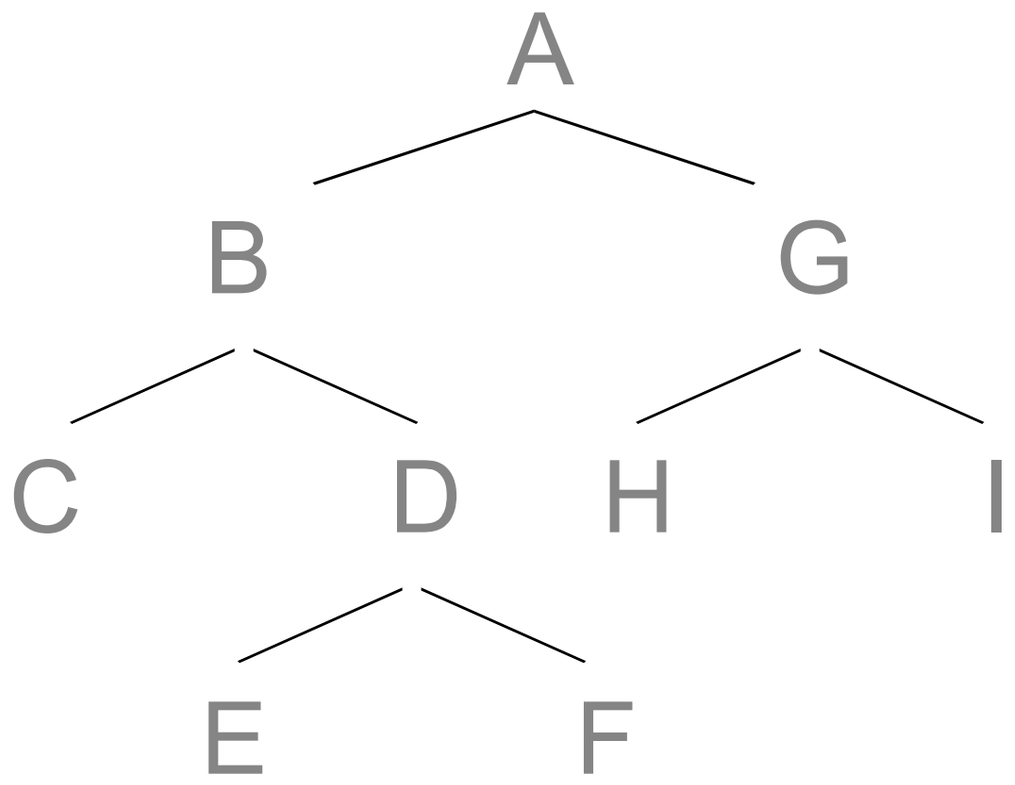
- Arc-standard is safer in ambiguity resolution
- Arc-eager is incremental, needs less memory

Parsers Reviewed



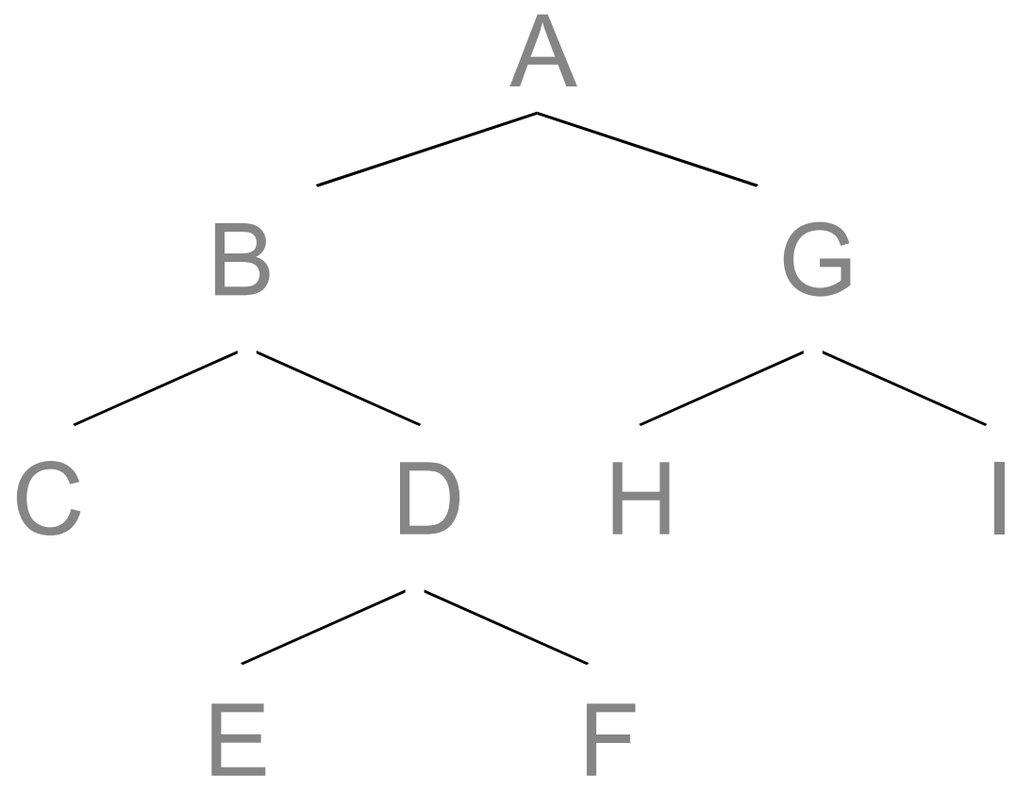
- Top-down: [A B C D E F G H I]

Parsers Reviewed



- Bottom-up: [C E F D B H I G A]

Parsers Reviewed

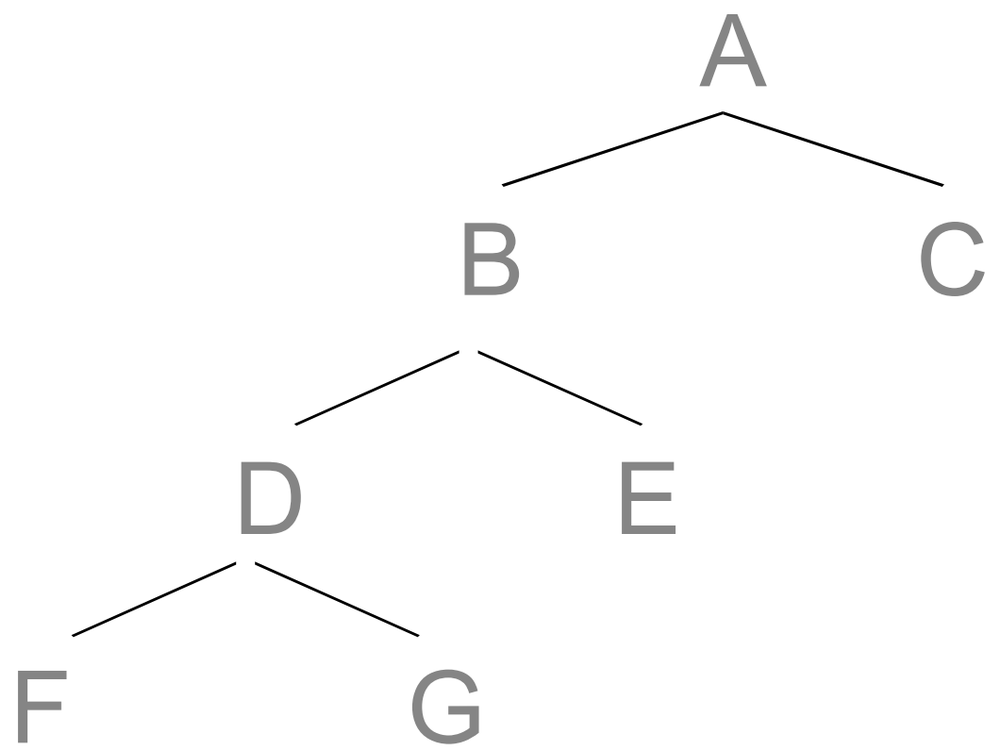


- Left-corner (arc-eager): [C B E D F A H G I]

Memory Load

- Left-embedding:

[[[*John's brother*]*'s car door*]*'s handle*] *broke off.*

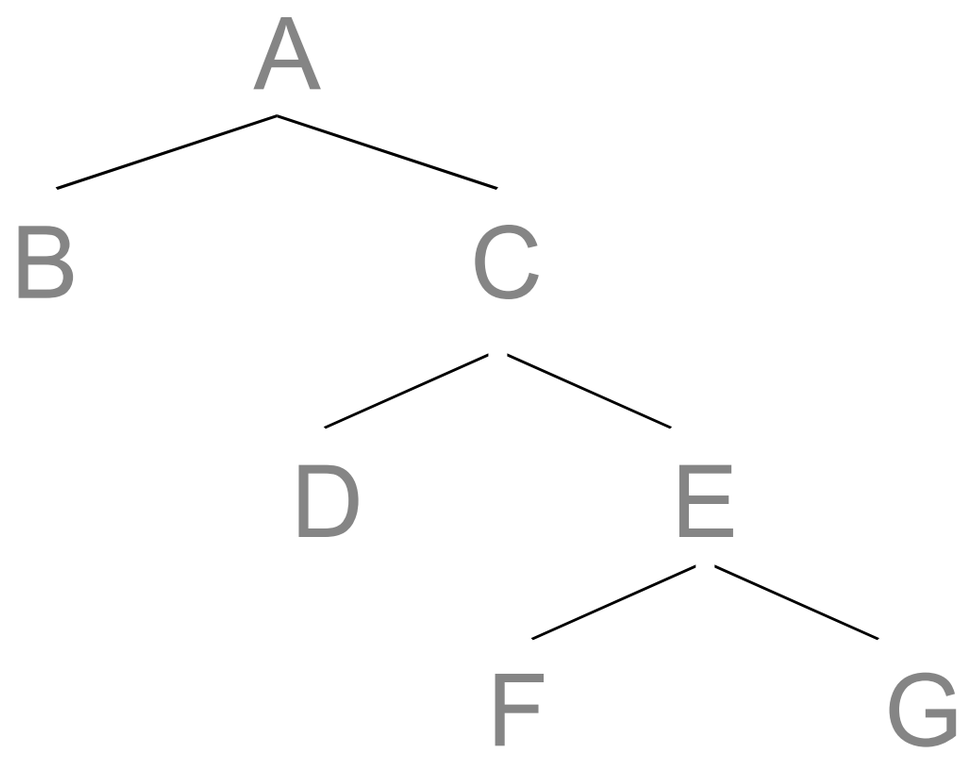


Bottom-up: easy Top-down: hard Left-corner: easy

Memory Load

- Right-embedding:

[*Bill knows* [*Mary said* [*she likes cats.*]]]

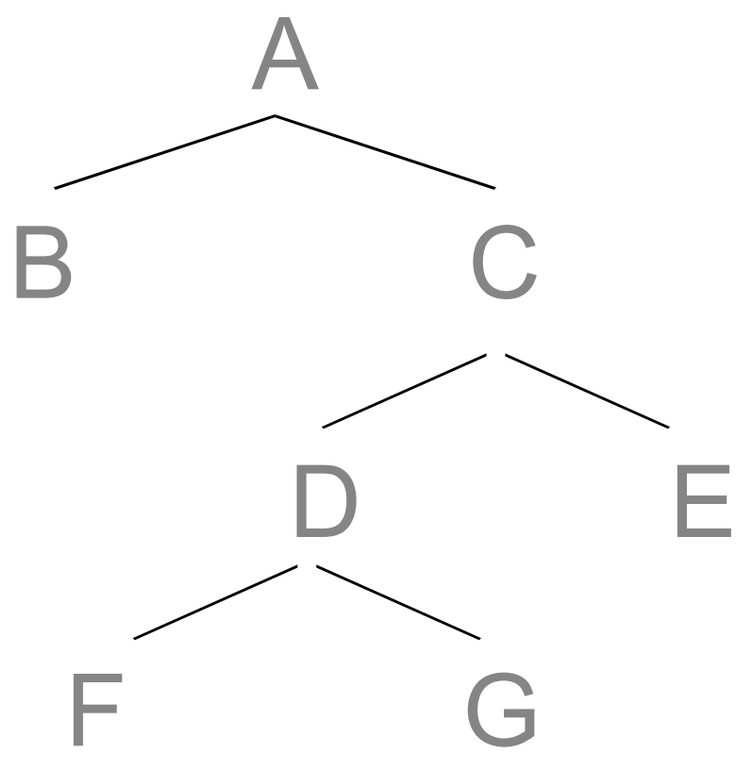


Bottom-up: hard Top-down: easy Left-corner: easy

Memory Load

- Center-embedding:

[*The mouse* [*the cat* [*the dog chased*] *bit*] *died.*]



Bottom-up: hard Top-down: hard Left-corner: hard

Summary of Behaviour

Node	Arcs	Left	Centre	Right
Top-down	Either	O(n)	O(n)	O(1)
Shift-reduce	Either	O(1)	O(n)	O(n)
Left-corner	Standard	O(1)	O(n)	O(n)
Left-corner	Eager	O(1)	O(n)	O(1)
People		O(1)	O(n)	O(1)

Ambiguity in Parsing

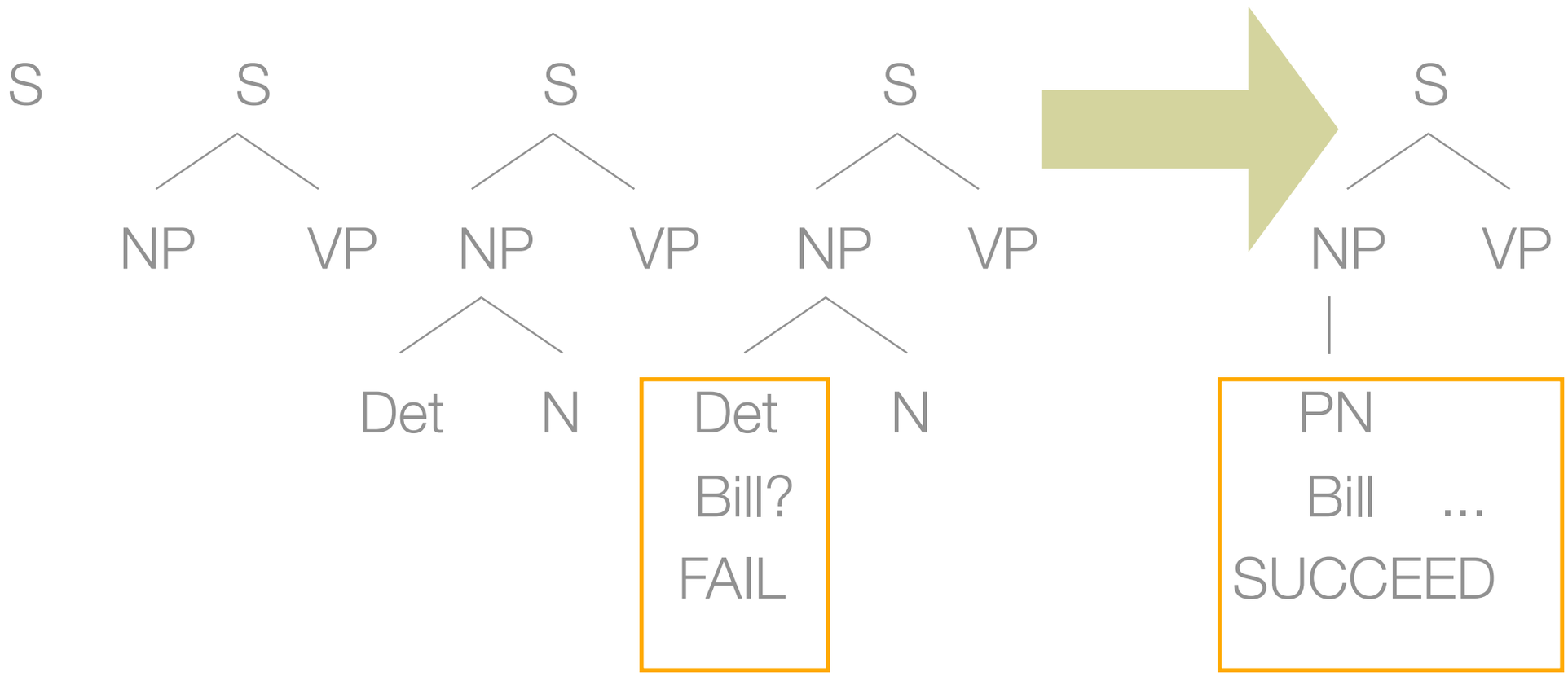
- What if more than one rule can be selected?
  - Local ambiguity: a parse derivation may fail later
  - Global ambiguity: multiple parses can succeed
- Mechanisms of handling ambiguity during parsing:
  - Backtracking
  - Parallelism
  - Determinism
  - Underspecification

Backtracking Parsers

- Parsing is a sequence of rule selections
- If at one point, more than one rule can be applied, this is called a choice point
  - Make a decision, based on some selection rule
  - If subsequently parsing ‘blocks’, return to a choice point and re-parse from there
- Which choice point to return to?
  - Usually the last, why?
  - What other choice point selection rules could be used?

Backtracking: an example

Bill reads



Parallel Parsers

- Build parse trees through successive rule selections
  - If more than one rule may be applied, create a new parse derivation for each possibility
  - Pursue all parses in parallel
  - If any of the parses ‘blocks’, discard it
- Multiple local ambiguities => number of parallel derivations grows exponentially
  - Bounded parallelism: pursue a fixed number
  - How do we choose which ones to keep?

Parallel: an example