CS4025: Parsing

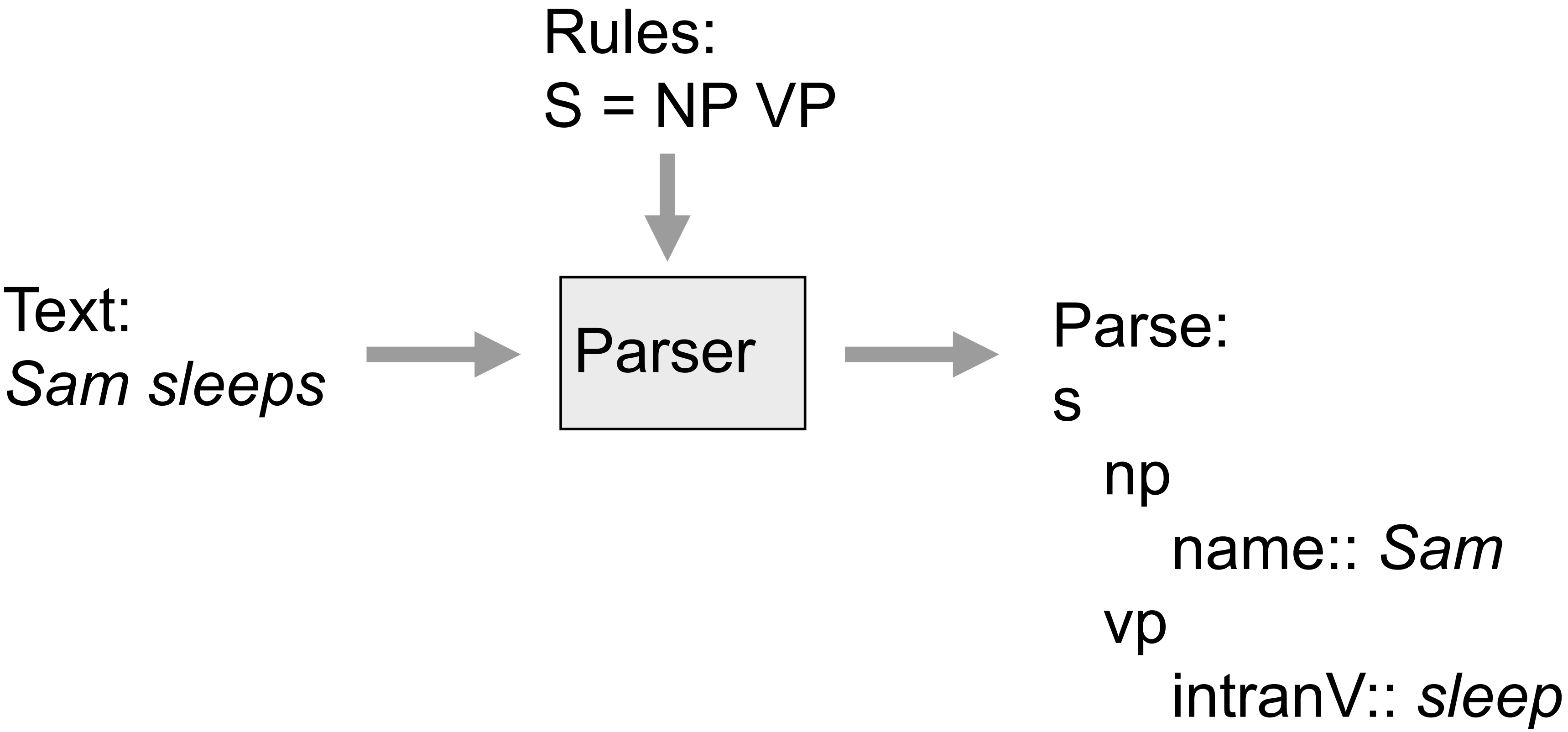
- Top-down parsing
- Bottom-up parsing (shift-reduce/left corner)

See J&M Chapter 10 is 1st ed, 13 in 2nd

Definitions

- A *grammar* is a description of what it takes to be a sentence of a language
- A *parser* is a program that takes a grammar and a string and returns all possible analyses of the string according to the grammar (as trees, nested lists etc.)
- It is possible to build a parser for a single grammar, but in practice they are generic.

Operation of a parser



Rules:
S = NP VP

Text:
Sam sleeps

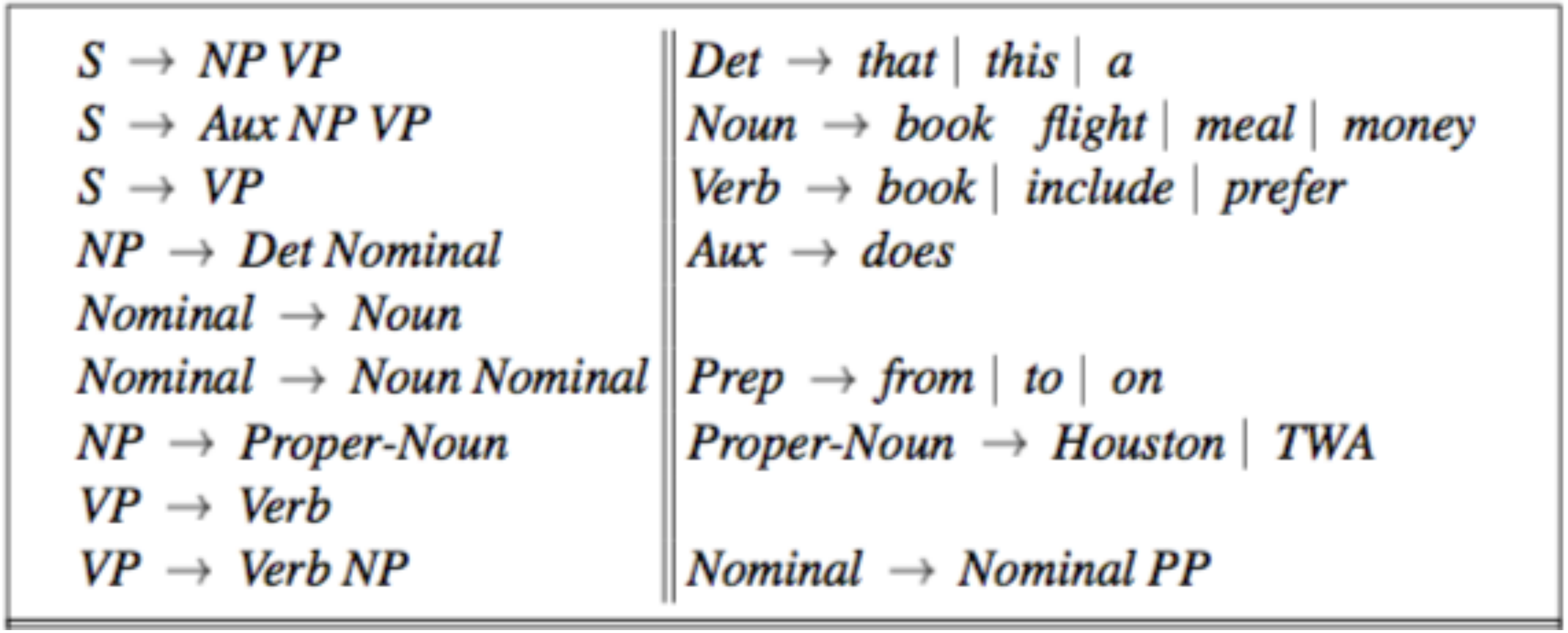
Parser

Parse:
s
  np
    name:: *Sam*
  vp
    intranV:: *sleep*

Top down parser

- » Start with initial symbol (eg, S), guess at an appropriate rewrite rule
- » If first symbol is a non-terminal, rewrite it with another guessed rule. Keep doing until first symbol is a terminal.
  - – leads to infinite loop in some cases!
- » If terminal matches first word of real sentence, continue with second word.
- » (As with any approach to recognition) need to be able to recover if the wrong choice is made (search)

Example grammar



S → *NP VP*
S → *Aux NP VP*
S → *VP*
NP → *Det Nominal*
Nominal → *Noun*
Nominal → *Noun Nominal*
NP → *Proper-Noun*
VP → *Verb*
VP → *Verb NP*

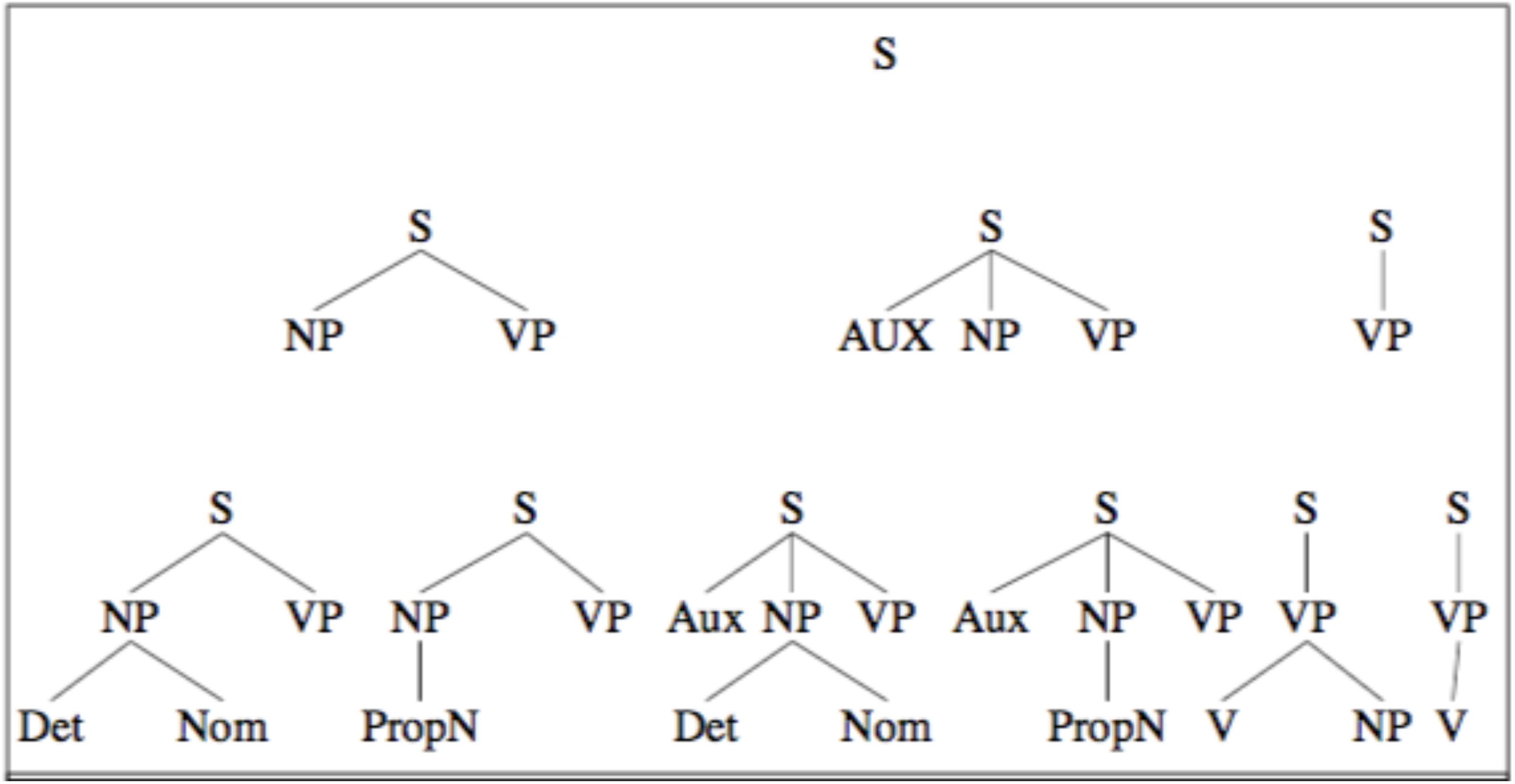
Det → *that* | *this* | *a*
Noun → *book* *flight* | *meal* | *money*
Verb → *book* | *include* | *prefer*
Aux → *does*

Prep → *from* | *to* | *on*
Proper-Noun → *Houston* | *TWA*

Nominal → *Nominal PP*

Book that flight.

Top-down parsing



Left-recursive

- A grammar is is left-recursive if it contains a non-terminal category that has a derivation that include s itself anywhere along its leftmost branch.
- Left recursive grammar, e.g.

```
np ---> det, ap, n.
ap ---> ap, a.            (left recursion)
ap ---> a.
det ---> [the].
a ---> [little].
a ---> [old].
n ---> [lady].
```

- Indirect left recursion, e.g.

```
np ---> det, n.
det ---> np, [’s].
```

Left Recursion

Goals	**Words**
NP	the little old lady
Det AP N	the little old lady
the AP N	the little old lady
AP N	little old lady
AP A N	little old lady
AP A A N	little old lady
AP A A A N	little old lady
AP A A A A N	little old lady
…	

Handling left-recursion

- Don't write left-recursive grammars. E.g. instead of:

  NP ---> Det Noun

  NP ---> NP PP

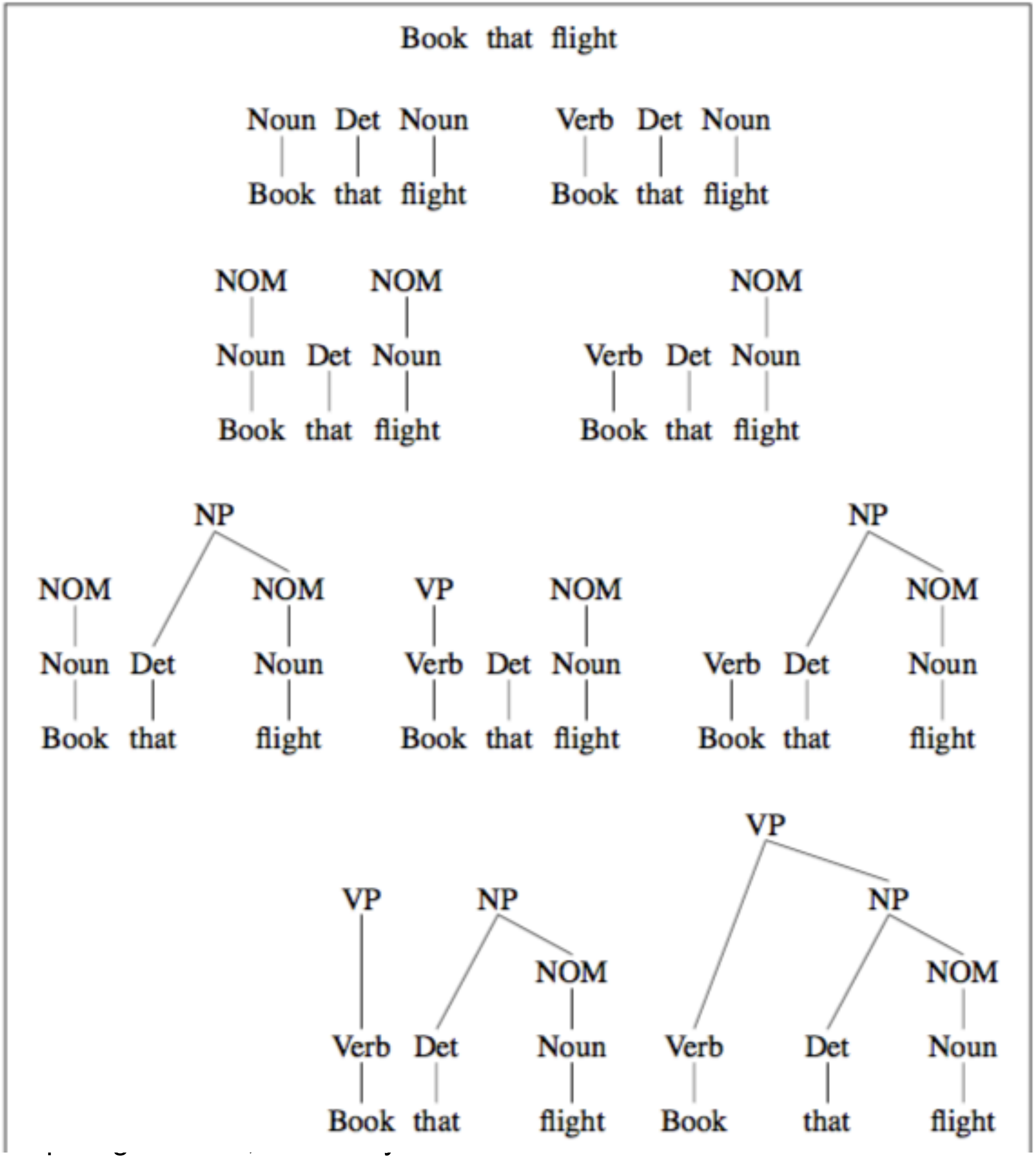
  write:

  NP1 ---> Det Noun

  NP ---> NP1 PPs
- Impose a limit on the depth of recursion attempted by the parser(number of goals allowed in the list).
- Use a different type of parser (e.g. bottom-up)

Bottom up parser

- Look for combinations of items that correspond to the right hand side of a rule, and replace them with the left hand side of the rule
- Keep doing this
- As for top-down, there may be alternatives, and picking the wrong one may lead to a blind alley



- The parser extends one level to the next by looking for places in the parse-in-progress
- The right-hand -side of some rule might fit.

Problem: Empty rules

If we allow “empty” rules, where there are no symbols

on the RHS (permitted in some variants of Context Free Grammar) then pure bottom-up parsing is unsuitable.

NP -> NP1 PPs

PPs -> PP PPs

PPs ->

When to “reduce” using the last rule?

Shift-reduce parsing only makes sense if all rules have at least one symbol on the RHS.

Solutions

- Don't write rules with empty RHS's.
- Grammars with empty rules can be automatically translated into grammars without them (accepting the same strings, but, of course, the structures returned will be different).
- Use a different type of parser (e.g. top-down)

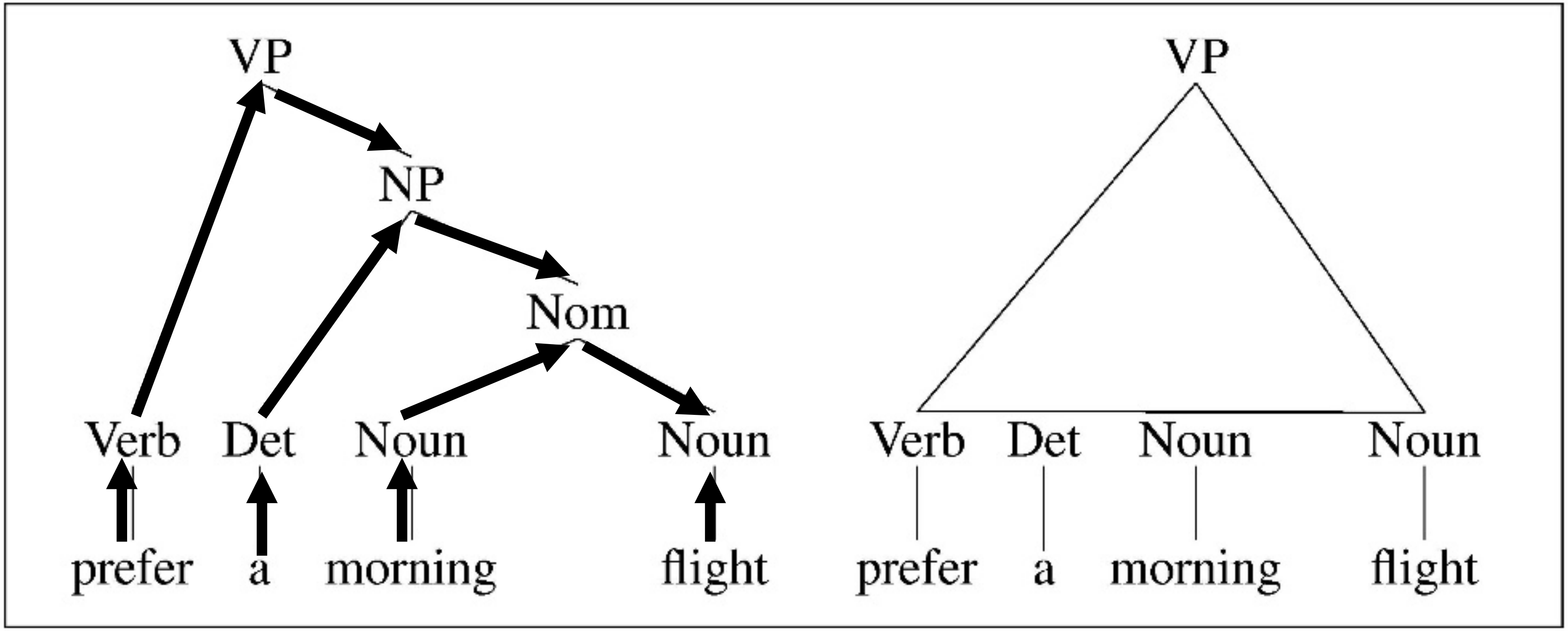
Top down vs. bottom up

- The top-down strategy never wastes time exploring trees that cannot result in an S, but does spend considerable effort on *S* trees that are not consistent with the input
- Bottom-up parsers, on the other hand, never suggest trees that are not at lea

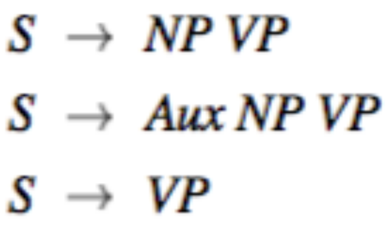
Left corner recognition

- A variety of bottom-up recognition that has some elements of top-down
- Find something that is the first element (only) of the RHS of a rule
- Hypothesise that this rule applies
- Recognise following constituents bottom-up
- Notice when they correspond to what is expected from the hypothesised rule
- The hypothesised rule allows some reduction in the bottom-up possibilities considered

Example


VP
NP
Nom
Verb
Det
Noun
Noun
prefer
a
morning
flight
VP
Verb
Det
Noun
Noun
prefer
a
morning
flight

Example (left corner)



$S \rightarrow NP\ VP$

$S \rightarrow Aux\ NP\ VP$

$S \rightarrow VP$

Using the left-corner notion, it is easy to see that only the *second* rule is a viable candidate since the word *Does* can not serve as the left-corner of either the *NP* or the *VP* required by the other two *S* rules.

Book the flight.

Search: Choosing between td and bu

All approaches to recognition involve search:

- Top down – when there is more than one rule with the same LHS
- Bottom up – when there is more than one rule with the same RHS (includes lexical rules)

Sometimes one can choose one or other approach according to the characteristics of one’s grammar (also left-recursion, empty rules)

Edinburgh Package Parsing strategy

- Default parsing strategy is top-down recursive descent, but left corner is also provided
  - » Use set_parser_type(lc). to switch to “left corner” parsing
- Search strategy is depth-first backtracking search (inherited from Prolog)