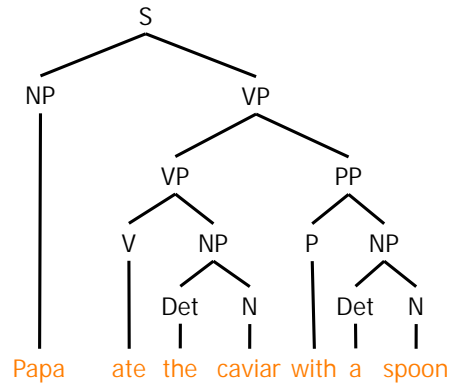


# Parsing

# What is Parsing?

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

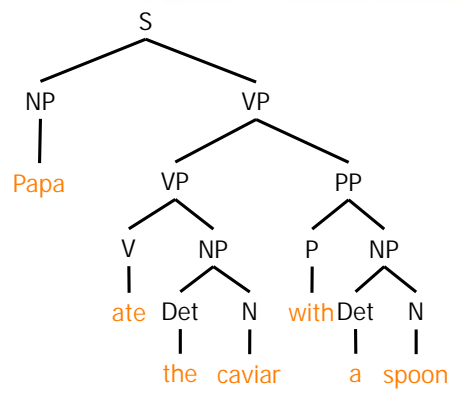
NP → Papa  
 N → caviar  
 N → spoon  
 V → spoon  
 V → ate  
 P → with  
 Det → the  
 Det → a



# What is Parsing?

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

NP → Papa  
 N → caviar  
 N → spoon  
 V → spoon  
 V → ate  
 P → with  
 Det → the  
 Det → a



# Programming languages

```
printf ("/charset [%s",
        (re_opcode_t) *(p - 1) == charset_not ? "^" : "");
assert (p + *p < pend);
for (c = 0; c < 256; c++)
    if (c / 8 < *p && (p[1 + (c/8)] & (1 << (c % 8)))) {
        /* Are we starting a range? */
        if (last + 1 == c && ! inrange) {
            putchar ('-');
            inrange = 1;
        }
        /* Have we broken a range? */
        else if (last + 1 != c && inrange) {
            putchar (last);
            inrange = 0;
        }
        if (! inrange)
            putchar (c);
        last = c;
    }
```

- Easy to parse.
- Designed that way!

# Natural languages

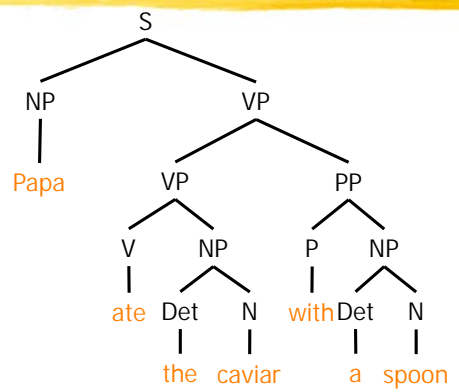
```
printf ("/charset %s", re_opcode_t *p - 1 == charset_not ?
        "^" : ""); assert p + *p < pend; for c = 0; c < 256; c++ if c /
        8 < *p && p[1 + c/8] & 1 << c % 8 Are we starting a range? if
        last + 1 == c && ! inrange putchar '-'; inrange = 1; Have we
        broken a range? else if last + 1 != c && inrange putchar last;
        inrange = 0; if ! inrange putchar c; last = c;
```

- No {} () [] to indicate scope & precedence
- Lots of overloading (arity varies)
- Grammar isn't known in advance!
- Context-free grammar not best formalism

# Ambiguity

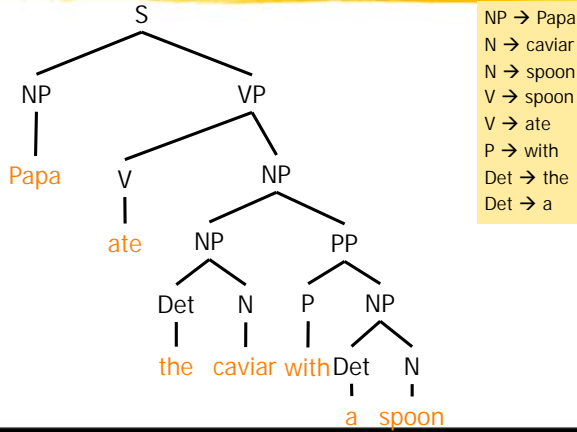
S → NP VP  
 NP → Det N  
 NP → NP PP  
 VP → V NP  
 VP → VP PP  
 PP → P NP

NP → Papa  
 N → caviar  
 N → spoon  
 V → spoon  
 V → ate  
 P → with  
 Det → the  
 Det → a



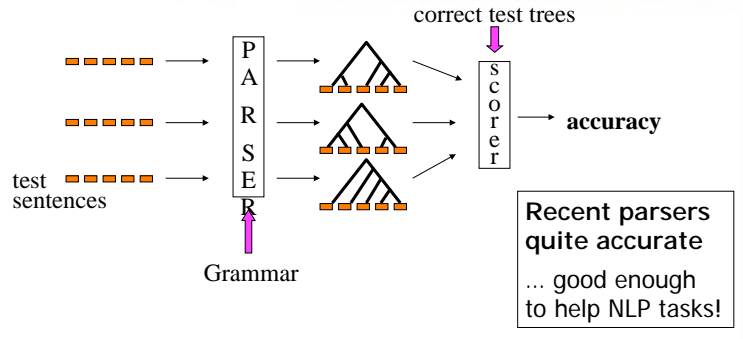
# Ambiguity

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



- NP → Papa
- N → caviar
- N → spoon
- V → ate
- P → with
- Det → the
- Det → a

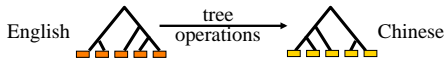
# The parsing problem



Warning: these slides are out of date

# Applications of parsing (1/2)

- Machine translation (Alshawi 1996, Wu 1997, ...)

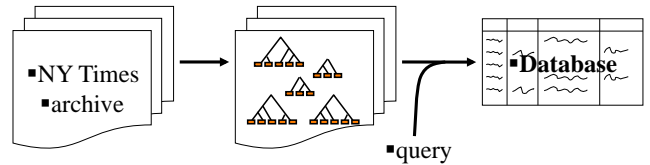


- Speech synthesis from parses (Prevost 1996)  
The government plans to raise income tax.  
The government plans to raise income tax the imagination.
- Speech recognition using parsing (Chelba et al 1998)  
Put the file in the folder.  
Put the file and the folder.

Warning: these slides are out of date

# Applications of parsing (2/2)

- Grammar checking (Microsoft)
- Indexing for information retrieval (Woods 1997)  
... washing a car with a hose ... → vehicle maintenance
- Information extraction (Hobbs 1996)



# Parsing for the Turing Test

- Most linguistic properties are defined over trees.
- One needs to parse to see subtle distinctions. E.g.:

Sara dislikes criticism of her. (*her ≠ Sara*)  
 Sara dislikes criticism of her by anyone. (*her ≠ Sara*)  
 Sara dislikes anyone's criticism of her. (*her = Sara or her ≠ Sara*)

- In rest of lecture (and following two lectures), we'll develop some parsing algorithms on the blackboard.

0 1 2 3 4 5 6 7  
 "Papa ate the caviar with a spoon"

- S → NP VP
- NP → Det N
- NP → NP PP
- VP → V NP
- VP → VP PP
- PP → P NP
- NP → Papa
- N → caviar
- N → spoon
- V → spoon
- V → ate
- P → with
- Det → the
- Det → a

0 1 2 3 4 5 6 7  
 "Papa ate the caviar with a spoon"

First try ... does it work?

- for each constituent on the LIST (Y i j)
  - scan the LIST for an adjacent constituent (Z j k)
  - if grammar has a rule to combine them (X → Y Z)
    - then add the result to the LIST (X i k)

0 1 2 3 4 5 6 7  
 "Papa ate the caviar with a spoon"

Second try ...

- initialize the list using words (T i i+1)  
where T is a preterminal tag like Noun
- for each constituent on the LIST (Y i j)
  - scan the LIST for an adjacent constituent (Z j k)
  - if grammar has a rule to combine them (X → Y Z)
    - then add the result to the LIST (X i k)
- if the above loop added anything, do it again!  
(so that X i k gets a chance to combine or be combined with)

0 1 2 3 4 5 6 7  
 "Papa ate the caviar with a spoon"

Third try ...

- initialize the list using words (T i i+1)  
where T is a preterminal tag like Noun
- for each constituent on the LIST (Y i j)
  - for each adjacent constituent on the list (Z j k)
    - for each rule to combine them (X → Y Z)
      - add the result to the LIST (X i k)  
 if it's not already there
- if the above loop added anything, do it again!  
(so that X i k gets a chance to combine or be combined with)

0 1 2 3 4 5 6 7  
 "Papa ate the caviar with a spoon"

Third try ...

- NP 0 1
- V 1 2
- Det 2 3
- N 3 4
- P 4 5
- Det 5 6
- N 6 7
- V 6 7
- NP 2 4
- NP 5 7
- VP 1 4
- PP 4 7
- ...

Still, that was inefficient when we tried it on the board ...

We kept checking the same pairs that already had failed

## CKY algorithm, recognizer version

- **Input:** string of  $n$  words
- **Output:** yes/no (since it's only a recognizer)
- **Data structure:**  $n \times n$  table
  - rows labeled 0 to  $n-1$
  - columns labeled 1 to  $n$
  - cell  $[i,j]$  lists constituents found between  $i$  and  $j$

## CKY algorithm, recognizer version

- **for**  $i := 1$  to  $n$ 
  - Add to  $[i-1,i]$  all categories for the  $i^{\text{th}}$  word
- **for** width  $:= 2$  to  $n$ 
  - **for** start  $:= 0$  to  $n$ -width
    - Define end  $:=$  start + width
    - **for** mid  $:=$  start+1 to end-1
      - **for** every nonterminal  $Y$  in  $[start, mid]$ 
        - **for** every nonterminal  $Z$  in  $[mid, end]$ 
          - **for** all nonterminals  $X$ 
            - **if**  $X \rightarrow YZ$  is in the grammar
              - **then** add  $X$  to  $[start, end]$

## Alternative version of inner loops

- **for**  $i := 1$  to  $n$ 
  - Add to  $[i-1,i]$  all categories for the  $i^{\text{th}}$  word
- **for** width  $:= 2$  to  $n$ 
  - **for** start  $:= 0$  to  $n$ -width
    - Define end  $:=$  start + width
    - **for** mid  $:=$  start+1 to end-1
      - **for** every rule  $X \rightarrow YZ$  in the grammar
        - **if**  $Y$  in  $[start, mid]$  **and**  $Z$  in  $[mid, end]$ 
          - **then** add  $X$  to  $[start, end]$ .