# LL(k): Deterministic top-down parsing

## Parsing
## ISCL-BA-06

Çağrı Çöltekin

ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Winter Semester 2020/21

# So far ...

- Formal languages and automata
- General parsing techniques
    - Top-down – Bottom-up
    - Directional – non-directional
- Chart parsing
    - CKY
    - Early

## So far ...

- Formal languages and automata
- General parsing techniques
  - Top-down – Bottom-up
  - Directional – non-directional
- Chart parsing
  - CKY
  - Early

Coming next:

- Deterministic context-free parsing
- Probabilistic context-free parsing
- Dependency parsing

# Recap: top-down parsing

- General idea: try to generate the input using the grammar rules
  - Initialize with the start symbol
  - Rewrite each non terminal, replacing them with matching RHS in the grammar
  - When there are multiple options, follow one, backtrack and follow others when done
  - Repeat until input sentence is generated (or failed)
- If we always expand the left-most symbol first, the parser is directional, the resulting derivation is the left-most derivation
- Parsing proceeds with two actions:
  predict expanding all RHS of the left-most non-terminal
  match if the left-most item is a terminal, it has to match the next input symbol

# Top-down parsing: an example

$$
\begin{array}{lll}
S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\
VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n
\end{array}
$$

| Matched | Sent. Form | Input | Action |
|---------|------------|-------|--------|
|         | S \$       | d n v a n | init |

# Top-down parsing: an example

$$
\begin{array}{llll}
S & \to NP\ VP & NP \to d\ AN & NP \to AN \\
VP & \to v\ NP & AN \to a\ AN & AN \to n
\end{array}
$$

| MATCHED | SENT. FORM | INPUT | ACTION |
|---------|-----------:|-------|--------|
|         | NP VP \$   | d n v a n | P: S $\to$ NP VP |

# Top-down parsing: an example

$$
\begin{aligned}
S &\rightarrow NP\ VP & NP &\rightarrow d\ AN & NP &\rightarrow AN \\
VP &\rightarrow v\ NP & AN &\rightarrow a\ AN & AN &\rightarrow n
\end{aligned}
$$

| Matched | Sent. Form | Input | Action |
|---------|-----------:|-------|--------|
|         | d AN VP \$ | d n v a n | P: NP $\rightarrow$ d AN |
|         | AN VP \$ | d n v a n | P: NP $\rightarrow$ AN |

# Top-down parsing: an example

$$
\begin{aligned}
S &\rightarrow NP\ VP & NP &\rightarrow d\ AN & NP &\rightarrow AN \\
VP &\rightarrow v\ NP & AN &\rightarrow a\ AN & AN &\rightarrow n
\end{aligned}
$$

| Matched | Sent. Form | Input | Action |
|---------|-----------|-------|--------|
|         | d AN VP \$ | d n v a n | P: NP $\rightarrow$ d AN |
|         | n VP \$ | d n v a n | P: AN $\rightarrow$ n |
|         | a AN VP \$ | d n v a n | P: AN $\rightarrow$ a AN |

# Top-down parsing: an example

$$S \rightarrow NP\ VP \quad NP \rightarrow d\ AN \quad NP \rightarrow AN$$
$$VP \rightarrow v\ NP \quad\quad AN \rightarrow a\ AN \quad AN \rightarrow n$$

| Matched | Sent. Form | Input | Action |
|---|---|---|---|
| | d AN VP \$ | d n v a n | P: NP → d AN |
| | n VP \$ | d n v a n | P: AN → n |
| | a AN VP \$ | d n v a n | P: match ✗ |

# Top-down parsing: an example

$$S \rightarrow NP\ VP \quad NP \rightarrow d\ AN \quad NP \rightarrow AN$$
$$VP \rightarrow v\ NP \quad\quad AN \rightarrow a\ AN \quad AN \rightarrow n$$

| Matched | Sent. Form | Input | Action |
|---------|-----------|-------|--------|
|  | d AN VP \$ | d n v a n | P: NP $\rightarrow$ d AN |
|  | n VP \$ | d n v a n | P: match ✗ |

## Top-down parsing: an example

$$S \rightarrow NP\ VP \quad NP \rightarrow d\ AN \quad NP \rightarrow AN$$
$$VP \rightarrow v\ NP \quad AN \rightarrow a\ AN \quad AN \rightarrow n$$

| Matched | Sent. Form | Input | Action |
|---------|-----------:|-------|--------|
| d | AN VP \$ | n v a n | match d |

# Top-down parsing: an example

$$
\begin{array}{lll}
S & \to NP\ VP & NP \to d\ AN \quad NP \to AN \\
VP & \to v\ NP & AN \to a\ AN \quad AN \to n
\end{array}
$$

| Matched | Sent. Form | Input | Action |
|---------|-----------:|-------|--------|
| d | d VP \$ | n v a n | P: AN → n |
| n | a AN VP \$ | n v a n | P: AN → a AN |

# Top-down parsing: an example

$$
\begin{array}{lll}
S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\
VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n
\end{array}
$$

| Matched | Sent. Form | Input | Action |
|---------|-----------:|-------|--------|
| d | d VP \$ | n v a n | P: AN → n |
| n | a AN VP \$ | n v a n | P: match ✗ |

# Top-down parsing: an example

| | | |
|---|---|---|
| S  $\to$ NP VP | NP $\to$ d AN | NP $\to$ AN |
| VP $\to$ v NP | AN $\to$ a AN | AN $\to$ n |

| Matched | Sent. Form | Input | Action |
|---------|------------|-------|--------|
| d n | VP \$ | v a n | match n |

# Top-down parsing: an example

$$
\begin{array}{llll}
S & \to \text{NP VP} & \text{NP} \to \text{d AN} & \text{NP} \to \text{AN} \\
\text{VP} & \to \text{v NP} & \text{AN} \to \text{a AN} & \text{AN} \to \text{n}
\end{array}
$$

| Matched | Sent. Form | Input | Action |
|---------|-----------|-------|--------|
| d n | v NP \$ | v a n | P: VP $\to$ v NP |

# Top-down parsing: an example

> S   → NP VP    NP → d AN    NP → AN
> VP → v NP       AN → a AN    AN → n

| Matched | Sent. Form | Input | Action |
|---------|-----------|-------|--------|
| d n v | NP $ | a n | match v |

# Top-down parsing: an example

$$
\begin{array}{llll}
S & \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\
VP & \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n
\end{array}
$$

| Matched | Sent. Form | Input | Action |
|---------|-----------:|-------|--------|
| d n v | AN $ | a n | P: NP → AN |
| d n v | d AN $ | a n | P: NP → d AN |

# Top-down parsing: an example

$$
\begin{array}{lll}
S & \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\
VP & \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n
\end{array}
$$

| Matched | Sent. Form | Input | Action |
|---------|-----------:|-------|--------|
| d n v | AN $ | a n | P: NP → AN |
| d n v | d AN $ | a n | P: match ✗ |

# Top-down parsing: an example

$$
\begin{array}{lll}
S \ \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\
VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n
\end{array}
$$

| Matched | Sent. Form | Input | Action |
|---------|-----------:|-------|--------|
| d n v   | a AN \$   | a n   | P: AN → a AN |
| d n v   | n \$      | a n   | P: AN → n |

# Top-down parsing: an example

$$
\begin{array}{lll}
S & \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\
VP & \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n
\end{array}
$$

| MATCHED | SENT. FORM | INPUT | ACTION |
|---|---:|---|---|
| d n v | a AN $ | a n | P: AN → a AN |
| d n v | n $ | a n | P: match ✗ |

# Top-down parsing: an example

$$
\begin{array}{lll}
S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\
VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n
\end{array}
$$

| Matched | Sent. Form | Input | Action |
|---------|-----------|-------|--------|
| d n v a | AN \$ | n | match a |

# Top-down parsing: an example

$$
\begin{array}{lll}
S & \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\
VP & \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n
\end{array}
$$

| Matched | Sent. Form | Input | Action |
|---------|-----------:|-------|--------|
| d n v a | n \$ | n | P: AN → n |

# Top-down parsing: an example

$$
\begin{array}{lll}
S & \rightarrow NP\ VP & NP \rightarrow d\ AN \quad NP \rightarrow AN \\
VP & \rightarrow v\ NP & AN \rightarrow a\ AN \quad AN \rightarrow n
\end{array}
$$

| Matched | Sent. Form | Input | Action |
|---------|-----------:|-------|--------|
| d n v a | n \$ | n | P: AN → n |
| n | a AN \$ | n | P: AN → a AN |

# Top-down parsing: an example

$$
\begin{array}{lll}
S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\
VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n
\end{array}
$$

| Matched | Sent. Form | Input | Action |
|---------|-----------:|-------|--------|
| d n v a | n $ | n | P: AN → n |
| n | a AN $ | n | match ✗ |

# Top-down parsing: an example

$$
\begin{array}{lll}
S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\
VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n
\end{array}
$$

| Matched | Sent. Form | Input | Action |
|---------|:----------:|-------|--------|
| d n v a n | $ | | match n |

# Top-down parsing

- If we follow the predicted productions, we obtain a *leftmost* derivation
- Lots of unnecessary work, backtracking because of useless predictions
- Most of the unnecessary work is done in *predict*
- In this lecture we will look at ways to reduce this
- For some grammars, the unnecessary predictions can be completely avoided, resulting in a *deterministic* parser

# Recursive descent parser

- *Recursive descent* parsers are top-down, recursive parsers where each non-terminal is implemented as a procedure
- For each symbol on a RHS, we either
  - call the sub-procedure (another nonterminal)
  - or match the input symbol

```
1: procedure A( )
2:     select a rule A → X₁, ..., Xₖ
3:     for i = 1 to k do
4:         if Xᵢ is a nonterminal then
5:             call Xᵢ()
6:         else if Xᵢ = current input then
7:             advance the input pointer
8:         else
9:             return error
```

# Recursive descent parser
some remarks

- The interesting idea is that now the parser is a program in a(ny) programming language
- In its general form a recursive descent parser is a backtracking parser
- If we can select a rule deterministically, then we can get a deterministic parser
- Deterministic parsing generally requires a *lookahead* mechanism:
  - Given the non-terminal to expand/rewrite, and the next input symbol(s), for some grammars, we can build a table that can deterministically guide a parser

# Table driven parsing

$$
\begin{aligned}
S &\rightarrow NP\ VP & NP &\rightarrow d\ AN & NP &\rightarrow AN \\
VP &\rightarrow v\ NP & AN &\rightarrow a\ AN & AN &\rightarrow n
\end{aligned}
$$

| non-term. | input (lookahead) | | | | |
|-----------|-----------|-----------|-----------|-----------|-----|
| | d | a | n | v | $ |
| S | S $\rightarrow$ NP VP | S $\rightarrow$ NP VP | S $\rightarrow$ NP VP | S $\rightarrow$ NP VP | |
| NP | NP $\rightarrow$ d AN | NP $\rightarrow$ AN | NP $\rightarrow$ AN | | |
| VP | | | | VP $\rightarrow$ v NP | |
| AN | | AN $\rightarrow$ a AN | AN $\rightarrow$ n | | |

# Table driven parsing: example

| non-term. | input (lookahead) | | | | |
|---|---|---|---|---|---|
| | d | a | n | v | \$ |
| S | S → NP VP | S → NP VP | S → NP VP | S → NP VP | |
| NP | NP → d AN | NP → AN | NP → AN | | |
| VP | | | | VP → v NP | |
| AN | | AN → a AN | AN → n | | |

| Matched | Sent. Form | Input | Action |
|---|---|---|---|
| | S \$ | d n v a n | init |

# Table driven parsing: example

| non-term. | input (lookahead) | | | | |
| --- | --- | --- | --- | --- | --- |
| | d | a | n | v | \$ |
| S | S → NP VP | S → NP VP | S → NP VP | S → NP VP | |
| NP | NP → d AN | NP → AN | NP → AN | | |
| VP | | | | VP → v NP | |
| AN | | AN → a AN | AN → n | | |

| Matched | Sent. Form | Input | Action |
| --- | --- | --- | --- |
| | NP VP \$ | d n v a n | P: S → NP VP |

# Table driven parsing: example

| non-term. | input (lookahead) | | | |
|---|---|---|---|---|
| | d | a | n | v $ |
| S | S → NP VP | S → NP VP | S → NP VP | S → NP VP |
| NP | NP → d AN | NP → AN | NP → AN | |
| VP | | | | VP → v NP |
| AN | | AN → a AN | AN → n | |

| Matched | Sent. Form | Input | Action |
|---|---|---|---|
| | d AN VP $ | d n v a n | P: NP → d AN |

# Table driven parsing: example

| non-term. | input (lookahead) | | | | |
|---|---|---|---|---|---|
| | d | a | n | v | $ |
| S | S → NP VP | S → NP VP | S → NP VP | S → NP VP | |
| NP | NP → d AN | NP → AN | NP → AN | | |
| VP | | | | VP → v NP | |
| AN | | AN → a AN | AN → n | | |

| Matched | Sent. Form | Input | Action |
|---|---|---|---|
| d | AN VP $ | n v a n | match d |

# Table driven parsing: example

| non-term. | input (lookahead) | | | | |
|---|---|---|---|---|---|
| | d | a | n | v | $ |
| S | S → NP VP | S → NP VP | S → NP VP | S → NP VP | |
| NP | NP → d AN | NP → AN | NP → AN | | |
| VP | | | | VP → v NP | |
| AN | | AN → a AN | AN → n | | |

| MATCHED | SENT. FORM | INPUT | ACTION |
|---|---|---|---|
| d | n VP $ | n v a n | P: AN → n |

# Table driven parsing: example

| non-term. | input (lookahead) | | | | |
|---|---|---|---|---|---|
| | d | a | n | v | $ |
| S | S → NP VP | S → NP VP | S → NP VP | S → NP VP | |
| NP | NP → d AN | NP → AN | NP → AN | | |
| VP | | | | VP → v NP | |
| AN | | AN → a AN | AN → n | | |

| Matched | Sent. Form | Input | Action |
|---|---|---|---|
| d n | VP $ | v a n | match n |

# Table driven parsing: example

| non-term. | input (lookahead) | | | | |
|---|---|---|---|---|---|
| | d | a | n | v | $ |
| S | S → NP VP | S → NP VP | S → NP VP | S → NP VP | |
| NP | NP → d AN | NP → AN | NP → AN | | |
| VP | | | | VP → v NP | |
| AN | | AN → a AN | AN → n | | |

| Matched | Sent. Form | Input | Action |
|---|---|---|---|
| d n | v NP $ | v a n | P: VP → v NP |

# Table driven parsing: example

| non-term. | input (lookahead) | | | | |
|---|---|---|---|---|---|
| | d | a | n | v | $ |
| S | S → NP VP | S → NP VP | S → NP VP | S → NP VP | |
| NP | NP → d AN | NP → AN | NP → AN | | |
| VP | | | | VP → v NP | |
| AN | | AN → a AN | AN → n | | |

| Matched | Sent. Form | Input | Action |
|---|---|---|---|
| d n v | NP $ | a n | match v |

# Table driven parsing: example

| non-term. | input (lookahead) | | | | |
|---|---|---|---|---|---|
| | d | a | n | v | $ |
| S | S $\rightarrow$ NP VP | S $\rightarrow$ NP VP | S $\rightarrow$ NP VP | S $\rightarrow$ NP VP | |
| NP | NP $\rightarrow$ d AN | NP $\rightarrow$ AN | NP $\rightarrow$ AN | | |
| VP | | | | VP $\rightarrow$ v NP | |
| AN | | AN $\rightarrow$ a AN | AN $\rightarrow$ n | | |

| Matched | Sent. Form | Input | Action |
|---|---|---|---|
| d n v | AN $ | a n | P: NP $\rightarrow$ AN |

# Table driven parsing: example

| non-term. | input (lookahead) | | | | |
|---|---|---|---|---|---|
| | d | a | n | v | $ |
| S | S → NP VP | S → NP VP | S → NP VP | S → NP VP | |
| NP | NP → d AN | NP → AN | NP → AN | | |
| VP | | | | VP → v NP | |
| AN | | AN → a AN | AN → n | | |

| MATCHED | SENT. FORM | INPUT | ACTION |
|---|---|---|---|
| d n v | a AN $ | a n | P: AN → a AN |

# Table driven parsing: example

| non-term. | input (lookahead) | | | | |
|---|---|---|---|---|---|
| | d | a | n | v | $ |
| S | S → NP VP | S → NP VP | S → NP VP | S → NP VP | |
| NP | NP → d AN | NP → AN | NP → AN | | |
| VP | | | | VP → v NP | |
| AN | | AN → a AN | AN → n | | |

| Matched | Sent. Form | Input | Action |
|---|---|---|---|
| d n v a | AN $ | n | match a |

# Table driven parsing: example

| non-term. | input (lookahead) | | | | |
|---|---|---|---|---|---|
| | d | a | n | v | $ |
| S | S $\to$ NP VP | S $\to$ NP VP | S $\to$ NP VP | S $\to$ NP VP | |
| NP | NP $\to$ d AN | NP $\to$ AN | NP $\to$ AN | | |
| VP | | | | VP $\to$ v NP | |
| AN | | AN $\to$ a AN | AN $\to$ n | | |

| MATCHED | SENT. FORM | INPUT | ACTION |
|---|---|---|---|
| d n v a | n $ | n | P: AN $\to$ n |

# Table driven parsing: example

| non-term. | input (lookahead) | | | | |
|---|---|---|---|---|---|
| | d | a | n | v | \$ |
| S | S → NP VP | S → NP VP | S → NP VP | S → NP VP | |
| NP | NP → d AN | NP → AN | NP → AN | | |
| VP | | | | VP → v NP | |
| AN | | AN → a AN | AN → n | | |

| Matched | Sent. Form | Input | Action |
|---|---|---|---|
| d n v a n | \$ | | match n |

# FIRST and FOLLOW sets

- FIRST and FOLLOW sets are useful for both top-down and bottom-up table driven parsers
- FIRST set of a non-terminal A, FIRST(A), is the set of initial terminal symbols of all strings generated by A
- FOLLOW set of a non-terminal A, FOLLOW(A), is the set of initial terminals that may follow any A according to the grammar
- Both sets generalize to any sentential form
- FIRST and FOLLOW sets are also useful for error recovery during parsing
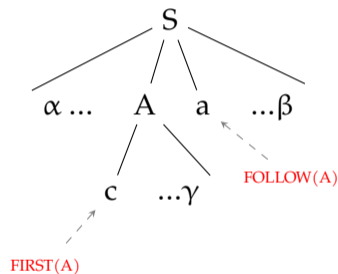
# Computing the FIRST set

- The FIRST set of a terminal symbol contains only itself
- To compute the FIRST sets of nonterminals, repeat the following until no new symbols are added to any of the sets
    1. For each rule $X \rightarrow Y_1 Y_2 \ldots Y_k$ in the grammar,
        - place all terminals in FIRST$(Y_i)$ if $Y_1 Y_2 \ldots Y_{i-1} \stackrel{*}{\Rightarrow} \epsilon$
        - if $\epsilon$ is in all FIRST$(Y_i)$ for all i = 1, ..., k, add $\epsilon$ to FIRST$(X)$
    2. if the rule processed is $X \rightarrow \epsilon$, add $\epsilon$ to FIRST$(X)$
- Then, FIRST set of any sentential form, FIRST$(X_1 X_2 \ldots X_k)$ can be computed:
    – For $i = 1, \ldots, k$
        1. Add all non-$\epsilon$ symbols from $X_i$ to FIRST$(X_1 X_2 \ldots X_k)$
        2. If $\epsilon \notin$ FIRST$(X_i)$, stop
    – if $\epsilon \in$ FIRST$(X_i)$ for all $i = 1, \ldots, k$, add $\epsilon$ to FIRST$(X_1 X_2 \ldots X_k)$

# Computing the FOLLOW set

- Calculate the FIRST sets
    1. Place $ in the FOLLOW(S)
    2. For a production $A \rightarrow \alpha B \beta$, add everything in FIRST($\beta$) except $\epsilon$ to FOLLOW(B)
    3. For a production $A \rightarrow \alpha B$, or $A \rightarrow \alpha B \beta$ where FIRST($\beta$) contains $\epsilon$, add all items in FOLLOW(A) to FOLLOW(B)
    4. Repeat 3 until no more items are added to any of the FOLLOW sets

# LL(1) grammars

- A grammar is called and LL(1) grammar, if we can find a table similar to our example:
    - If there is only a single prediction for each (non-terminal, lookahead) pair, then the grammar is an LL(1) grammar
- L's stand for *Left-to-right* and *Leftmost derivation*, (1) indicates the number of lookahead symbols needed
- If we increase the number of lookahead symbols, we get LL(k) grammars
- LL(k) grammar can be parsed with a top-down parser without backtracking
- Not every context free grammar is LL(k)
- But, programming language grammars are mostly LL(1)

# LL(1) grammars
formal definition

- If a grammar is LL(1) then whenever $A \rightarrow \alpha$ and $A \rightarrow \beta$ are two rules in the grammar, then
    - The sets of non-terminals of strings derived from $\alpha$ and $\beta$ are disjoint
    - Only one (or none) of $\alpha$ and $\beta$ can derive the empty string
    - If $\beta \overset{*}{\Rightarrow} \epsilon$, $\alpha$ cannot start with a terminal that may follow $A$
- In other words:
    - FIRST($\alpha$) and FIRST($\beta$) are disjoint
    - if $\epsilon$ is in FIRST($\alpha$), then FIRST($\beta$) and FOLLOW($A$) are disjoint sets

# Construction of LL(1) table

- If there are no $\epsilon$ productions, the table can be easily constructed from the FIRST sets
- Otherwise, after computing FIRST and FOLLOW sets, the following procedure fills the LL(1) table
    - For each rule $A \rightarrow \alpha$ in the grammar
        1. For each terminal $a$ in FIRST($\alpha$), add $A \rightarrow \alpha$ to table cell [A, a]
        2. If $\epsilon$ is in FIRST($\alpha$), then for each terminal b in FOLLOW(A) add $A \rightarrow \alpha$ to table cell [A, b]

# Example
calculating FIRST sets

$$S \to BA \quad A \to aBA \mid \epsilon \quad B \to CD \quad D \to bCD \mid \epsilon \quad C \to cSc \mid d$$

- Repeat until no additions
  1. For each $X \to Y_1 Y_2 \dots Y_k$
     - place all terminals in $FIRST(Y_i)$ if $Y_1 Y_2 \dots Y_{i-1} \overset{*}{\Rightarrow} \epsilon$
     - if $\epsilon$ is in all $FIRST(Y_i)$ for all i = 1, ..., k, add $\epsilon$ to $FIRST(X)$
  2. if the rule processed is $X \to \epsilon$, add $\epsilon$ to $FIRST(X)$

$FIRST(S) =$
$FIRST(A) =$
$FIRST(B) =$
$FIRST(C) =$
$FIRST(D) =$

# Example
calculating FIRST sets

$$S \rightarrow BA \quad A \rightarrow aBA \,|\, \epsilon \quad B \rightarrow CD \quad D \rightarrow bCD \,|\, \epsilon \quad C \rightarrow cSc \,|\, d$$

- Repeat until no additions
    1. For each $X \rightarrow Y_1 Y_2 \ldots Y_k$
        - place all terminals in FIRST($Y_i$) if $Y_1 Y_2 \ldots Y_{i-1} \stackrel{*}{\Rightarrow} \epsilon$
        - if $\epsilon$ is in all FIRST($Y_i$) for all i = 1, ..., k, add $\epsilon$ to FIRST($X$)
    2. if the rule processed is $X \rightarrow \epsilon$, add $\epsilon$ to FIRST($X$)

FIRST(S) = {c,d}
FIRST(A) = {a,$\epsilon$}
FIRST(B) = {c,d}
FIRST(C) = {c,d}
FIRST(D) = {b,$\epsilon$}

# Example
calculating FOLLOW sets

$$S \rightarrow BA \quad A \rightarrow aBA \mid \epsilon \quad B \rightarrow CD \quad D \rightarrow bCD \mid \epsilon \quad C \rightarrow cSc \mid d$$

1. Place $ in the FOLLOW(S)
2. For a production $A \rightarrow \alpha B \beta$, add everything in FIRST($\beta$) except $\epsilon$ to FOLLOW(B)
3. For $A \rightarrow \alpha B$, or $A \rightarrow \alpha B \beta$ where FIRST($\beta$) contains $\epsilon$, add items in FOLLOW(A) to FOLLOW(B)

|  | S | A | B | C | D |
|---|---|---|---|---|---|
| FIRST | {c,d} | {a,$\epsilon$} | {c,d} | {c,d} | {b,$\epsilon$} |

FIRST(S) =
FIRST(A) =
FIRST(B) =
FIRST(C) =
FIRST(D) =

# Example
calculating FOLLOW sets

$$S \rightarrow BA \quad A \rightarrow aBA \mid \epsilon \quad B \rightarrow CD \quad D \rightarrow bCD \mid \epsilon \quad C \rightarrow cSc \mid d$$

1. Place \$ in the FOLLOW(S)
2. For a production $A \rightarrow \alpha B\beta$, add everything in FIRST($\beta$) except $\epsilon$ to FOLLOW(B)
3. For $A \rightarrow \alpha B$, or $A \rightarrow \alpha B\beta$ where FIRST($\beta$) contains $\epsilon$, add items in FOLLOW(A) to FOLLOW(B)

|       | S     | A              | B     | C     | D              |
|-------|-------|----------------|-------|-------|----------------|
| FIRST | {c,d} | {a,$\epsilon$} | {c,d} | {c,d} | {b,$\epsilon$} |

FIRST(S) = {c,\$}
FIRST(A) = {c,\$}
FIRST(B) = {a,c,\$}
FIRST(C) = {a,b,c,\$}
FIRST(D) = {a,c,\$}

## Example
constructing the LL(1) table

$$S \rightarrow BA \quad A \rightarrow aBA \mid \epsilon \quad B \rightarrow CD \quad D \rightarrow bCD \mid \epsilon \quad C \rightarrow cSc \mid d$$

|  | S | A | B | C | D |
|---|---|---|---|---|---|
| FIRST | {c,d} | {a,$\epsilon$} | {c,d} | {c,d} | {b,$\epsilon$} |
| FOLLOW | {c,\$} | {c,\$} | {a,c,\$} | {a,b,c,\$} | {a,c,\$} |

|  | a | b | c | d | \$ |
|---|---|---|---|---|---|
| S |  |  |  |  |  |
| A |  |  |  |  |  |
| B |  |  |  |  |  |
| C |  |  |  |  |  |
| D |  |  |  |  |  |

- For each rule $A \rightarrow \alpha$ in the grammar
  1. For each terminal $a$ in FIRST$(\alpha)$, add $A \rightarrow \alpha$ to table cell $[A, a]$
  2. If $\epsilon$ is in FIRST$(\alpha)$, then for each terminal $b$ in FOLLOW$(A)$ add $A \rightarrow \alpha$ to table cell $[A, b]$

# Example
constructing the LL(1) table

$$S \rightarrow BA \quad A \rightarrow aBA \,|\, \epsilon \quad B \rightarrow CD \quad D \rightarrow bCD \,|\, \epsilon \quad C \rightarrow cSc \,|\, d$$

|  | S | A | B | C | D |
|---|---|---|---|---|---|
| FIRST | {c,d} | {a,$\epsilon$} | {c,d} | {c,d} | {b,$\epsilon$} |
| FOLLOW | {c,\$} | {c,\$} | {a,c,\$} | {a,b,c,\$} | {a,c,\$} |

- For each rule $A \rightarrow \alpha$ in the grammar
  1. For each terminal $a$ in FIRST($\alpha$), add
     $A \rightarrow \alpha$ to table cell $[A, a]$
  2. If $\epsilon$ is in FIRST($\alpha$), then for each terminal $b$
     in FOLLOW($A$) add $A \rightarrow \alpha$ to table cell
     $[A, b]$

|  | a | b | c | d | \$ |
|---|---|---|---|---|---|
| S |  |  | BA | BA |  |
| A | aBA |  | $\epsilon$ |  | $\epsilon$ |
| B |  |  | CD | CD |  |
| C |  |  | cSc | d |  |
| D | $\epsilon$ | bCD | $\epsilon$ |  | $\epsilon$ |

# Summary

- LL(1) grammars can be parsed deterministically (without backtracking) using top-down parsers
- Like any top-down parser, left-recursion needs additional care
- Not every context free grammar is LL(k), but programming language grammars are mostly LL(1)
- LL(k) parsing is intuitive and relatively easy to construct by hand, but LR(k) grammars (bottom-up, deterministic) are more powerful (next lecture)
- Suggested reading: Grune and Jacobs (2007, ch.8), Aho et al. (2007, Section 4.4)

# Summary

- LL(1) grammars can be parsed deterministically (without backtracking) using top-down parsers
- Like any top-down parser, left-recursion needs additional care
- Not every context free grammar is LL(k), but programming language grammars are mostly LL(1)
- LL(k) parsing is intuitive and relatively easy to construct by hand, but LR(k) grammars (bottom-up, deterministic) are more powerful (next lecture)
- Suggested reading: Grune and Jacobs (2007, ch.8), Aho et al. (2007, Section 4.4)

Next:

- Deterministic bottom-up parsing
- Suggested reading: Grune and Jacobs (2007, ch.9), Aho et al. (2007, Section 4.5–4.7)

# Acknowledgments, references, additional reading material

📄 Aho, Alfred V., Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman (2007). *Compilers: Principles, Techniques, & Tools*. Pearson/Addison Wesley. ISBN: 9780321486813.

📄 Grune, Dick and Ceriel J.H. Jacobs (2007). *Parsing Techniques: A Practical Guide*. second. Monographs in Computer Science. The first edition is available at `http://dickgrune.com/Books/PTAPG_1st_Edition/BookBody.pdf`. Springer New York. ISBN: 9780387689548.

# Exercise

compute the FIRST and FOLLOW sets, and LL(1) table for $S \rightarrow iEtSQ \mid a \quad Q \rightarrow eS \mid \epsilon \quad E \rightarrow b$