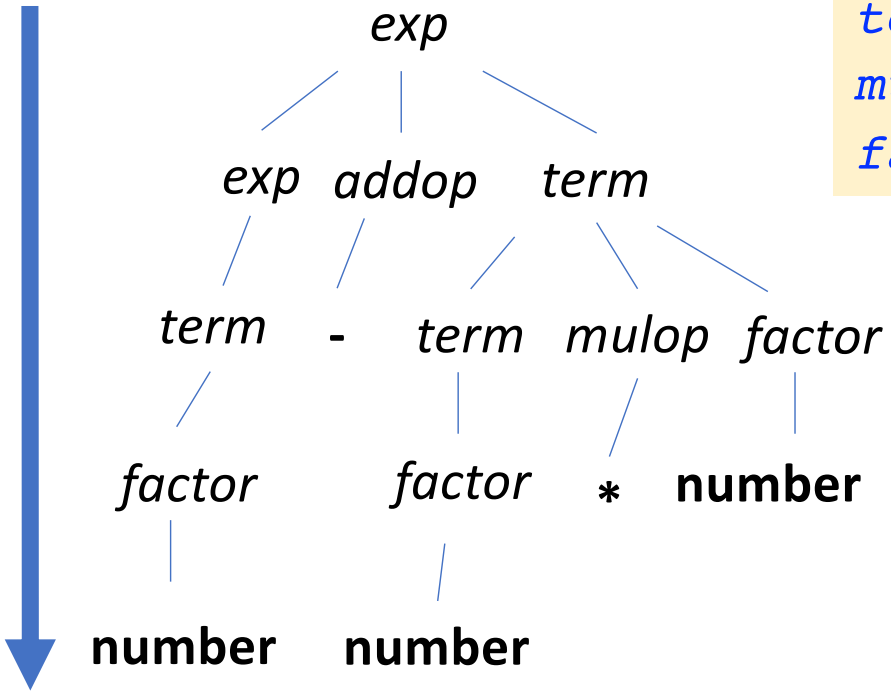
Syntax Analysis
(Chapters 4 & 5)

Top-Down Parsing (Chapter 4)

- Builds a parse tree top down, from the start nonterminal
- and creating tree nodes in **preorder** (a leftmost derivation)

```
34 – 3 * 42
```

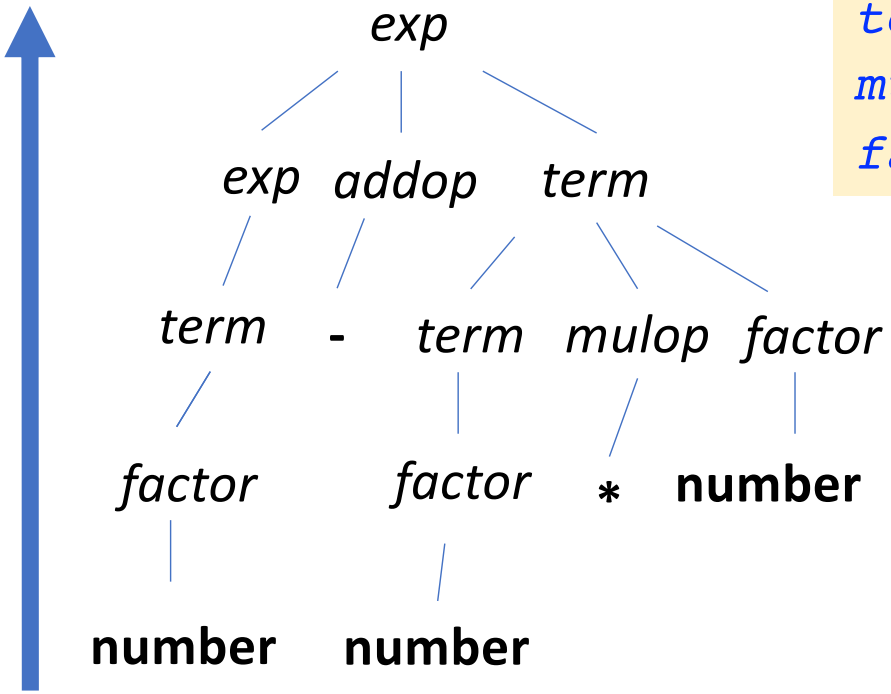



```
exp → exp addop term | term
addop → + | -
term → term mulop factor | factor
mulop → *
factor → ( exp ) | number
```

Bottom-Up Parsing (Chapter 5)

- Builds a parse tree bottom up, from the leaf nodes

```
34 – 3 * 42
```




```
exp → exp addop term | term
addop → + | -
term → term mulop factor | factor
mulop → *
factor → ( exp ) | number
```

Top-Down Parsing
(Chapter 4)

Top-Down Parsing (Predictive)

- Recursive-descent parsing
  - versatile
  - better for a hand-written parser
- LL(1) parsing
  - scan from **left to right**, and perform **leftmost** derivation
  - look ahead at most **one** input symbol

```
34 – 3 * 42
```

```
exp → exp addop term | term
addop → + | -
term → term mulop factor | factor
mulop → *
factor → ( exp ) | number
```

LL(1) Parsing

LL(1) Parsing

- Use a **stack** rather than recursive calls to build a tree
- Similar to running some pushdown automaton (PDA)
  - Begin by pushing the start nonterminal to the stack
  - Perform some **actions** based on the stack and next input symbol
  - Accept if both stack and input become empty

E.g.

tokens	**grammar**
( )	$S \rightarrow (\; S \;) \; S \mid \varepsilon$

LL(1) Parsing

- Example

tokens	grammar
( )	$S \rightarrow (S) S \mid \varepsilon$

$: marks stack bottom

stack top: leftmost of RHS

	Parsing Stack	Input	Action
1	$ S	( ) $	$S \rightarrow (S) S$
2	$ S) S (	( ) $	match
3	$ S) S	) $	$S \rightarrow \varepsilon$
4	$ S)	) $	match
5	$ S	$	$S \rightarrow \varepsilon$
6	$	$	match

LL(1) Parsing

- Two Actions:
  - If stack top is a nonterminal A and A → α, replace A with α (**generate**)
  - If stack top is a terminal (token), **match** it with input token
    - If matched, pop stack and advance input
    - Otherwise, throw an error

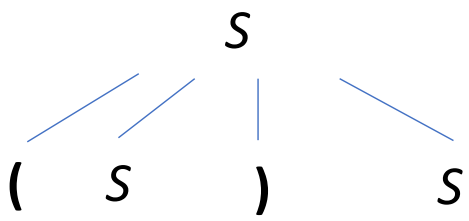
Error Input:
)

	Parsing Stack	Input	Action
1	$ *S*	) $	*S* → (*S*) *S*
2	$ *S*) *S* (	) $	mismatch

	Parsing Stack	Input	Action
1	$ *S*	) $	*S* → ε
2	$	) $	mismatch

LL(1) Parsing

- Parse Tree Construction
  - root node is constructed at the beginning of the parse
  - construct and attach tree nodes in each **generate** action



	Parsing Stack	Input	Action
1	$ *S*	( ) $	*S* → (*S*) *S*
2	$ *S*) *S* (	( ) $	match

First and Follow Sets

Why First ?

$$A \rightarrow a\ B \mid b\ C \mid c\ D$$

Why First ?

$$A \rightarrow a\ B \mid X\ C \mid c\ D$$

$$X \rightarrow Y\ y \mid Z\ z$$

$$Z \rightarrow b\ b \mid z\ z$$

First Set: Definition

- Suppose **α** is a string of terminals and nonterminals, **First(α)** consists of the first terminals that can be derived from **α**.

if $\alpha \Rightarrow^* a\beta$, then $a \in First(\alpha)$

if $\alpha \Rightarrow^* \varepsilon$ (nullable), then $\varepsilon \in First(\alpha)$

E.g.

First(ABc) = {a, c, d}

First(BC) = {b, c, ε}

Another grammar

$A \rightarrow aB \mid CD$

$B \rightarrow Bb \mid \varepsilon$

$C \rightarrow c \mid \varepsilon$

$D \rightarrow d$

First Set: Properties

1. If **X** is a terminal or **ε**, then **First(X) = {X}**
2. Suppose **X** is a nonterminal and **$X \rightarrow Y_1 Y_2 \ldots Y_k$**
   - if for some **i**, **$Y_1 \ldots Y_{i-1} \Rightarrow^* \varepsilon$** , then **$First(X) \supseteq First(Y_i) - \{\varepsilon\}$**
   - if **$Y_1 \ldots Y_k \Rightarrow^* \varepsilon$**, then **$\varepsilon \in First(X)$**

Why exclude it ?

E.g.

First(A) = {a, c, d}

Another grammar

$A \rightarrow aB \mid CD$
$B \rightarrow Bb \mid \varepsilon$
$C \rightarrow c \mid \varepsilon$
$D \rightarrow d$

First Set: Algorithm

- Compute the First set for each nonterminal iteratively

```
for each nonterminal A
  First(A) = {}
while some First set changed
  for each A → X1X2...Xn
    k = 1
    continue = true
    while continue == true and k<=n
      add First(Xk) – {ε} to First(A)
      if ε ∉ First(Xk)
        continue = false
      k++
    if continue == true
      add ε to First(A)
```

Why iterative ?

First Set: Algorithm

	A	B	C	D
init	{}	{}	{}	{}
round-1	{a}	{b}	{ε}	{d, ε}
round-2	{a, ε, d}	{b}	{ε}	{d, ε}
round-3	{a, ε, d}	{b}	{ε}	{d, ε}

The same results, so iteration stops

$A \rightarrow aB \mid CD$

$B \rightarrow bC$

$C \rightarrow \varepsilon$

$D \rightarrow d \mid \varepsilon$

Why Follow ?

$ ……**b** A

Stack

b……

Tokens

A → a B | ϵ | c D

✓

if b can Follow A

e.g., X → A b C

Follow Set: Definition

- For a nonterminal **A**, if there exists a derivation from the start nonterminal **S ⇒* αAaβ**, then **a ∈ Follow(A)**
- If **S ⇒* αA**, then **$ ∈ Follow(A)**

E.g.

$ always in Follow set of start symbol

grammar

$$A \rightarrow aBD \mid CC$$
$$B \rightarrow a$$
$$C \rightarrow bDe$$
$$D \rightarrow d$$

Follow(A) = {$}

Follow(B) = {d}

Follow(C) = {b, $}

Follow(D) = {e, $}

Follow Set: Definition

- For a nonterminal **A**, if there exists a derivation from the start nonterminal **$S \Rightarrow^* \alpha Aa\beta$ ($a \neq \varepsilon$)**, then **$a \in$ Follow(A)**
- If **$S \Rightarrow^* \alpha A$**, then **$\$ \in$ Follow(A)**

Exercise:

Follow(A) = {$}

Follow(B) = {d, $}

Follow(C) = {d, e, $}

Follow(D) = {e, $}

Another grammar

$A \rightarrow aBD \mid CC$

$B \rightarrow a$

$C \rightarrow De$

$D \rightarrow d \mid \varepsilon$

Follow Set: Properties

1. If **A** is the start symbol, **$ ∈ Follow(A)**
2. For any nonterminal **A**, **ε ∉ Follow(A)**
3. If **A –> αBγ** then **First(γ) – {ε} ⊆ Follow(B)**
4. If **A –> αBγ** and **γ ⇒* ε** then **Follow(A) ⊆ Follow(B)**

Follow Set: Algorithm

- Compute the Follow set for each nonterminal iteratively

```
for each nonterminal A
  if A is start-symbol
    Follow(A)={$}
  else
    Follow(A)={}
while some Follow set changed
  for each A → X1X2...Xn
    for each Xi that is a nontermianl
      add First(Xi+1Xi+2...Xn) – {ε} to Follow(Xi)
      if ε ∈ First(Xi+1Xi+2...Xn)
        add Follow(A) to Follow(Xi)
```

3rd property

4th property

Follow Set: Algorithm

First	S	A	B	C
	{e}	{e}	{b}	{e}

Follow	S	A	B	C
init	{$}	{}	{}	{}
round-1	{$}	{b, a}	{$}	{b, a}
round-2	{$}	{b, a}	{$}	{b, a}

The same results, so iteration stops

$S \rightarrow AB$
$A \rightarrow eC$
$B \rightarrow bAa$
$C \rightarrow e$

Example: Compute First/Follow Set

$E \rightarrow T\ E'$

$E' \rightarrow +\ T\ E' \mid \varepsilon$

$T \rightarrow F\ T'$

$T' \rightarrow *\ F\ T' \mid \varepsilon$

$F \rightarrow (\ E\ ) \mid \mathbf{id}$

First(E) = { **(**, **id** }

First(E') = { **+**, **ε** }

First(T) = { **(**, **id** }

First(T') = { *, **ε** }

First(F) = { **(**, **id** }

Follow(E) = { **$**, **)** }

Follow(E') = { **$**, **)** }

Follow(T) = { **$**, **)**, **+** }

Follow(T') = { **$**, **)**, **+** }

Follow(F) = { **$**, **)**, **+**, * }

Example:

E → T E’
E’ → + T E’ | ε
T → F T’
T’ → * F T’ | ε
F → (E) | id

	FIRST
E	{ (, id }
E’	{ +, ε }
T	{ (, id }
T’	{ *, ε }
F	{ (, id }

F → (E) | id

FIRST(F) = { (, id }

T’ → * F T’ | ε

FIRST(T’) = { *, ε }

E’ → + T E’ | ε

FIRST(E’) = { +, ε }

T → F T’

FIRST(T) = FIRST(F) = { (, id }

E → T E’

FIRST(E) = FIRST(T) = { (, id }

Example:

E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

FOLLOW(E)
E is the start symbol
$ ε FOLLOW(E)
F → (**E**)
) ε FOLLOW(E)
FOLLOW(E) = { $,) }

FOLLOW(E')
E → T **E'** & E' → + T **E'**
.....E■..... → .....TE'■.....
- FOLLOW(E) is contained in FOLLOW(E')
- { $,) } is contained in FOLLOW(E')

FOLLOW(E') = { $,) }

Example:

E → T E’
E’ → + T E’ | ε
T → F T’
T’ → * F T’ | ε
F → (E) | id

	FIRST
E	{ (, id }
E’	{ +, ε }
T	{ (, id }
T’	{ *, ε }
F	{ (, id }

FOLLOW(T)

E → T **E’** & E’ → + T **E’**

FIRST(E’) – { ε } is contained in FOLLOW(T)

→ { + } is contained in FOLLOW(T)

ε belongs to FIRST(E’)

→ FOLLOW(E) is contained in FOLLOW(T)

→ { $,) } is contained in FOLLOW(T)

FOLLOW(T) = { +, $,) }

Example:

E → T E’
E’ → + T E’ | ε
T → F T’
T’ → * F T’ | ε
F → (E) | id

	FIRST
E	{ (, id }
E’	{ +, ε }
T	{ (, id }
T’	{ *, ε }
F	{ (, id }

FOLLOW(T’)

T → F T’ & T’ → * F T’

FOLLOW(T) is contained in FOLLOW(T’)

→ { +, $,) } is contained in FOLLOW(T’)

FOLLOW(T’) = { +, $,) }

Example:

E → T E’
E’ → + T E’ | ε
T → F T’
T’ → * F T’ | ε
F → (E) | id

	FIRST
E	{ (, id }
E’	{ +, ε }
T	{ (, id }
T’	{ *, ε }
F	{ (, id }

FOLLOW(F)

T → F T’ & T’ → * F T’

FIRST(T’) – { ε } is contained in FOLLOW(F)

→ { * } is contained in FOLLOW(F)

ε belongs to FIRST(T’)

→ FOLLOW(T) is contained in FOLLOW(F)

→ { +, $,) } is contained in FOLLOW(F)

FOLLOW(F) = { *, +, $,) }

Example:

E → T E’
E’ → + T E’ | ε
T → F T’
T’ → * F T’ | ε
F → (E) | id

	FIRST
E	{ (, id }
E’	{ +, ε }
T	{ (, id }
T’	{ *, ε }
F	{ (, id }

	FOLLOW
E	{ $,) }
E’	{ $,) }
T	{ +, $,) }
T’	{ +, $,) }
F	{ *, +, $,) }

Back to LL(1) Parsing

LL(1) Parsing

- Parsing Table
  - if the stack top is **N**, and the lookahead token is **T**, then entry **[N, T]** in the table is the production rule to use

	Parsing Stack	Input	Action
1	…	…	…
2	…	…	…
3	**$** … *S*	**(** …**$**	*S* → **(** *S* **)** *S*
4	…	…	…
5	…	…	…

next token *(terminal)*

	(	)	$
S	S → (S) S	…	…

stack top

(nonterminal)

LL(1) Parsing

- Parsing Table Construction

  Given A → α

  - for each token a in First(α), add A → α to the entry [A, a]
  - if ε ∈ First(α), for each a in Follow(A), add A → α to entry [A, a]

S → (S) S
S → ε

First((S) S) = { (}

First(ε) = { ε } Follow(S) = {), $ }

	(	)	$
S	S → (S) S	S → ε	S → ε

LL(1) Parsing

- Parsing Table Construction

  Given A → α

  - for each token a in First(α), add A → α to the entry [A, a]
  - if ε ∈ First(α), for each a in Follow(A), add A → α to entry [A, a]

Exercise:

$$S \rightarrow A$$
$$A \rightarrow (A) A$$
$$A \rightarrow \varepsilon$$

	(	)	$
S	S → A		S → A
A	A → (A) A	A → ε	A → ε

LL(1) Parsing

- Parsing Table Construction: Example

r1 *E* → *T E′*
r2 *E′* → *addop T E′*
r3 *E′* → *ε*
r4 *addop* → **+**
r5 *addop* → **-**
r6 *T* → *F T′*
r7 *T′* → *mulop F T′*
r8 *T′* → *ε*
r9 *mulop* → *****
r10 *F* → **(** *E* **)**
r11 *F* → **id**

	(	id	)	+	-	*	$
E	r1	r1					
E’			r3	r2	r2		r3
addop				r4	r5		
T	r6	r6					
T’			r8	r8	r8	r7	r8
mulop						r9	
F	r10	r11					

LL(1) Parsing

- LL(1) Grammar
  - A grammar is an **LL(1) grammar** if the associated LL(1) parsing table has at most one production in each table entry
  - Cannot be ambiguous
  - A subset of CFG

$$S \rightarrow A$$
$$A \rightarrow (A) A$$
$$A \rightarrow \varepsilon$$

	(	)	$
S	S → A		S → A
A	A → (A) A	A → ε	A → ε

LL(1) Parsing: Algorithm

```
push start symbol S onto stack
while stack top ≠ $ and next token ≠ $
  if stack top is a and a == next token
    pop stack
    advance input
  else if stack top is A and next token is a
          and [A,a] has rule A → X1X2...Xn
    pop stack
    for i from n to 1
      push Xi onto stack
  else
    error
if stack top == next token == $
  accept
else
  error
```

Issues Related to LL(1)

A Grammar is LL(1) iff

$A \rightarrow \alpha \mid \beta$ ⇨ $First(\alpha) \cap First(\beta) = \Phi$

$A \rightarrow \alpha \mid \beta$

$St \quad \beta \Rightarrow^* \varepsilon$ ⇨ $First(\alpha) \cap Follow(A) = \Phi$

Left Recursion

- **Rewriting**: break it into two rules: (i) generate base case first and (ii) generate the repetition using right recursion

$A \rightarrow A\ \alpha \mid \beta$

$A \rightarrow \beta\ A'$

$A' \rightarrow \alpha\ A' \mid \varepsilon$

exp → *exp addop term* | *term*

exp → *term exp'*

exp' → *addop term exp'* | ε

Left Factoring

- **Issue**: when grammar rule choices share a common prefix, look ahead one symbol may not be sufficient to determine the rule
- **Rewriting**: take the common part out and add a new nonterminal

$A \rightarrow \alpha\ \beta \mid \alpha\ \gamma$

$A \rightarrow \alpha\ A'$

$A' \rightarrow \beta \mid \gamma$

if-stmt → **if** (*exp*) *stmt*
 | **if** (*exp*) *stmt* **else** *stmt*

if-stmt → **if** (*exp*) *stmt else-part*

else-part → **else** *stmt* | ε

SAMPLE PROBLEMS

Example

S → A | B C
A → a A | ε
B → b B | ε
C → c C | d C | ε

	FIRST	FOLLOW
S	a,b,c,d,ε	$
A	a,ε	$
B	b,ε	c,d,$
C	c,d,ε	$

	a	b	c	d	$
S	S → A	S → B C	S → B C	S → B C	S → A S → B C
A	A → a A				A → ε
B		B → b B	B → ε	B → ε	B → ε
C			C → c C	C → d C	C → ε

Example

LEXP → ATOM | LIST
ATOM → num | id
LIST → (LSEQ)
LSEQ → LSEQ LEXP | LEXP

LEXP → ATOM | LIST
ATOM → num | id
LIST → (LSEQ)
LSEQ → LEXP LSEQ'
LSEQ' → LEXP LSEQ' | ε

	FIRST	FOLLOW
LEXP	num, id, (	$, num, id, (,)
ATOM	num, id	$, num, id, (,)
LIST	(	$, num, id, (,)
LSEQ	num, id, (	)
LSEQ'	num, id, (, ε	)

Example

LEXP → ATOM | LIST
ATOM → num | id
LIST → (LSEQ)
LSEQ → LEXP LSEQ'
LSEQ' → LEXP LSEQ' | ε

	FIRST	FOLLOW
LEXP	num, id, (	$, num, id, (,)
ATOM	num, id	$, num, id, (,)
LIST	(	$, num, id, (,)
LSEQ	num, id, (	)
LSEQ'	num, id, (, ε	)

LEXP → ATOM | LIST FIRST(ATOM) ∩ FIRST(LIST) = ∅

ATOM → num | id FIRST(num) ∩ FIRST(id) = ∅

LSEQ' → LEXP LSEQ' | ε FIRST(LEXP) ∩ FOLLOW(LSEQ') = ∅

⇒ Grammar is LL(1)

Example

LEXP → ATOM | LIST
ATOM → num | id
LIST → (LSEQ)
LSEQ → LEXP LSEQ'
LSEQ' → LEXP LSEQ' | ε

	FIRST	FOLLOW
LEXP	num, id, (	$, num, id, (,)
ATOM	num, id	$, num, id, (,)
LIST	(	$, num, id, (,)
LSEQ	num, id, (	)
LSEQ'	num, id, (, ε	)

	num	id	(	)	$
LEXP	LEXP → ATOM	LEXP → ATOM	LEXP → LIST		
ATOM	ATOM → num	ATOM → id			
LIST			LIST → (LSEQ)		
LSEQ	LSEQ→LEXP LSEQ'	LSEQ→LEXP LSEQ'	LSEQ→LEXP LSEQ'		
LSEQ'	LSEQ'→LEXP LSEQ'	LSEQ'→LEXP LSEQ'	LSEQ'→LEXP LSEQ'	LSEQ'→ε	