Top-down Parsing
Recursive Descent & LL(1)

Roadmap (Where are we?)

We set out to study parsing

- Specifying syntax
  - Context-free grammars ✓
- Top-down parsers
  - Algorithm & its problem with left recursion ✓
  - Ambiguity ✓
  - Left-recursion removal ✓
- Predictive top-down parsing today
  - The LL(1) Property ✓
  - First sets ✓
  - Follow sets
  - Simple recursive descent parsers
  - Table-driven LL(1) parsers

FOLLOW Sets

FOLLOW(A)

For some $A \in NT$, define FOLLOW(A) as the set of symbols that can occur immediately after A in a valid sentential form

Follow(A) = {t | (t is a terminal and $G \Rightarrow^* \alpha A t \beta$) or (t is eof and $G \Rightarrow^* \alpha A$)}

FOLLOW(S) = {EOF}, where S is the start symbol

To build FOLLOW sets, we will use FIRST sets …

FIRST Sets

Remember FIRST sets from last Monday

FIRST(α) is set of tokens (terminals) that appear as first symbol in some string deriving from α

$x \in \text{FIRST}(\alpha)$ *iff* $\alpha \Rightarrow^* x\,\gamma$, for some γ

FIRST$^+$ sets

Definition of FIRST$^+(A \rightarrow \alpha)$

if $\varepsilon \in$ FIRST(α) then

FIRST$^+(A \rightarrow \alpha)$ = FIRST$(\alpha) \cup$ FOLLOW(A)

else

FIRST$^+(A \rightarrow \alpha)$ = FIRST(α)

Grammar is *LL(1)* iff $A \rightarrow \alpha$ and $A \rightarrow \beta$ implies

$$\text{FIRST}^+(A \rightarrow \alpha) \cap \text{FIRST}^+(A \rightarrow \beta) = \varnothing$$

Computing FOLLOW Sets


```
for each A ∈ NT, FOLLOW(A) ← Ø

FOLLOW(S) ← {EOF}

while (FOLLOW sets are still changing)
    for each p ∈ P, of the form A→B1B2 … Bk
        TRAILER ← FOLLOW(A)
        for i ← k down to 1
            if Bi ∈ NT then
                FOLLOW(Bi) ← FOLLOW(Bi) ∪ TRAILER
                if ε ∈ FIRST(Bi)                    // add right context
                  then TRAILER ← TRAILER ∪ ( FIRST(Bi) - {ε} )
                  else TRAILER ← FIRST(Bi)          // no ε => no right context
            else TRAILER ← {Bi}                     // Bi ∈ T => only 1 symbol
```

Computing FOLLOW Sets

for each A ∈ NT, FOLLOW(A) ← Ø

FOLLOW(S) ← {EOF}

while (FOLLOW sets are still changing)
  for each p ∈ P, of the form A→B_1B_2 … B_k
    TRAILER ← FOLLOW(A)
    for i ← k down to 1
      if B_i ∈ NT then
        FOLLOW(B_i) ← FOLLOW(B_i) ∪ TRAILER
        if ε ∈ FIRST(B_i) *// add right context*
          then TRAILER ← TRAILER ∪ (FIRST(B_i) – {ε})
          else TRAILER ← FIRST(B_i) *// no ε => no right context*
      else TRAILER ← {B_i} *// B_i ∈ T => only 1 symbol*

Computing FOLLOW Sets

for each $A \in NT$, $FOLLOW(A) \leftarrow \emptyset$

$FOLLOW(S) \leftarrow \{EOF\}$

while (FOLLOW sets are still changing)
  for each $p \in P$, *of the form* $A \rightarrow B_1 B_2 \ldots B_k$
    $TRAILER \leftarrow FOLLOW(A)$
    for $i \leftarrow k$ *down to 1*
      if $B_i \in NT$ *then*
        $FOLLOW(B_i) \leftarrow FOLLOW(B_i) \cup TRAILER$
        if $\varepsilon \in FIRST(B_i)$ // add right context
          then $TRAILER \leftarrow TRAILER \cup (FIRST(B_i) - \{\varepsilon\})$
          else $TRAILER \leftarrow FIRST(B_i)$ // no ε => no right context
      else $TRAILER \leftarrow \{B_i\}$ // $B_i \in T$ => only 1 symbol

Computing FOLLOW Sets


```
for each A ∈ NT, FOLLOW(A) ← Ø
FOLLOW(S) ← {EOF}
while (FOLLOW sets are still changing)
    for each p ∈ P, of the form A→B1B2 … Bk
        TRAILER ← FOLLOW(A)
        for i ← k down to 1
            if Bi ∈ NT then
                FOLLOW(Bi) ← FOLLOW(Bi) ∪ TRAILER
                if ε ∈ FIRST(Bi)                    // add right context
                  then TRAILER ← TRAILER ∪ ( FIRST(Bi) - {ε} )
                  else TRAILER ← FIRST(Bi)          // no ε => no right context
            else TRAILER ← {Bi}                     // Bi ∈ T => only 1 symbol
```

Computing FOLLOW Sets


```
for each A ∈ NT, FOLLOW(A) ← Ø
FOLLOW(S) ← {EOF}
while (FOLLOW sets are still changing)
    for each p ∈ P, of the form A→B1B2 ... Bk
        TRAILER ← FOLLOW(A)
        for i ← k down to 1
            if Bi ∈ NT then
                FOLLOW(Bi) ← FOLLOW(Bi) ∪ TRAILER
                if ε ∈ FIRST(Bi)                    // add right context
                  then TRAILER ← TRAILER ∪ ( FIRST(Bi) - {ε} )
                  else TRAILER ← FIRST(Bi)          // no ε => no right context
            else TRAILER ← {Bi}                     // Bi ∈ T => only 1 symbol
```

Computing FOLLOW Sets

```
for each A ∈ NT, FOLLOW(A) ← Ø
FOLLOW(S) ← {EOF}
while (FOLLOW sets are still changing)
    for each p ∈ P, of the form A→B1B2 ... Bk
        TRAILER ← FOLLOW(A)
        for i ← k down to 1
            if Bi ∈ NT then
                FOLLOW(Bi) ← FOLLOW(Bi) ∪ TRAILER
                if ε ∈ FIRST(Bi)                    // add right context
                    then TRAILER ← TRAILER ∪ ( FIRST(Bi) - {ε} )
                    else TRAILER ← FIRST(Bi)        // no ε => no right context
            else TRAILER ← {Bi}                     // Bi ∈ T => only 1 symbol
```

Computing FOLLOW Sets


```
for each A ∈ NT, FOLLOW(A) ← Ø
FOLLOW(S) ← {EOF}
while (FOLLOW sets are still changing)
    for each p ∈ P, of the form A→B1B2 … Bk
        TRAILER ← FOLLOW(A)
        for i ← k down to 1
            if Bi ∈ NT then
                FOLLOW(Bi) ← FOLLOW(Bi) ∪ TRAILER
                if ε ∈ FIRST(Bi )                    // add right context
                  then TRAILER ← TRAILER ∪ ( FIRST(Bi) - {ε} )
                  else TRAILER ← FIRST(Bi)           // no ε => no right context
            else TRAILER ← {Bi}                      // Bi ∈ T => only 1 symbol
```

Computing FOLLOW Sets


```
for each A ∈ NT, FOLLOW(A) ← Ø
FOLLOW(S) ← {EOF}
while (FOLLOW sets are still changing)
    for each p ∈ P, of the form A→B1B2 … Bk
        TRAILER ← FOLLOW(A)
        for i ← k down to 1
            if Bi ∈ NT then
                FOLLOW(Bi) ← FOLLOW(Bi) ∪ TRAILER
                if ε ∈ FIRST(Bi)                    // add right context
                    then TRAILER ← TRAILER ∪ ( FIRST(Bi) - {ε} )
                    else TRAILER ← FIRST(Bi)        // no ε => no right context
            else TRAILER ← {Bi}                     // Bi ∈ T => only 1 symbol
```

Computing FOLLOW Sets

for each $A \in NT$, $FOLLOW(A) \leftarrow \emptyset$

$FOLLOW(S) \leftarrow \{EOF\}$

while (FOLLOW sets are still changing)
  for each $p \in P$, *of the form* $A \rightarrow B_1 B_2 \ldots B_k$
    $TRAILER \leftarrow FOLLOW(A)$
    for $i \leftarrow k$ *down to* 1
      if $B_i \in NT$ *then*
        $FOLLOW(B_i) \leftarrow FOLLOW(B_i) \cup TRAILER$
        if $\varepsilon \in FIRST(B_i)$
          then $TRAILER \leftarrow TRAILER \cup (FIRST(B_i) - \{\varepsilon\})$
          else $TRAILER \leftarrow FIRST(B_i)$ // no ε => no right context
      else $TRAILER \leftarrow \{B_i\}$ // $B_i \in T$ => only 1 symbol

Computing FOLLOW Sets

for each $A \in NT$, $FOLLOW(A) \leftarrow \emptyset$

$FOLLOW(S) \leftarrow \{EOF\}$

while (FOLLOW sets are still changing)
  for each $p \in P$, *of the form* $A \rightarrow B_1 B_2 \ldots B_k$
    $TRAILER \leftarrow FOLLOW(A)$
    for $i \leftarrow k$ *down to 1*
      if $B_i \in NT$ *then*
        $FOLLOW(B_i) \leftarrow FOLLOW(B_i) \cup TRAILER$
        if $\varepsilon \in FIRST(B_i)$
          then $TRAILER \leftarrow TRAILER \cup (FIRST(B_i) - \{\varepsilon\})$
          else $TRAILER \leftarrow FIRST(B_i)$ // no ε => no right context
      else $TRAILER \leftarrow \{B_i\}$ // $B_i \in T$ => only 1 symbol

Computing FOLLOW Sets

for each $A \in NT$, $FOLLOW(A) \leftarrow \emptyset$

$FOLLOW(S) \leftarrow \{EOF\}$

while (FOLLOW sets are still changing)
  for each $p \in P$, *of the form* $A \rightarrow B_1 B_2 \ldots B_k$
    $TRAILER \leftarrow FOLLOW(A)$
    for $i \leftarrow k$ *down to* 1
      if $B_i \in NT$ *then*
        $FOLLOW(B_i) \leftarrow FOLLOW(B_i) \cup TRAILER$
        if $\varepsilon \in FIRST(B_i)$
          then $TRAILER \leftarrow TRAILER \cup (FIRST(B_i) - \{\varepsilon\})$
          else $TRAILER \leftarrow FIRST(B_i)$
      else $TRAILER \leftarrow \{B_i\}$ // $B_i \in T$ => only 1 symbol

Computing FOLLOW Sets

for each $A \in NT$, $FOLLOW(A) \leftarrow \emptyset$

$FOLLOW(S) \leftarrow \{EOF\}$

while (FOLLOW sets are still changing)
  for each $p \in P$, *of the form* $A \rightarrow B_1 B_2 \ldots B_k$
    $TRAILER \leftarrow FOLLOW(A)$
    for $i \leftarrow k$ *down to* 1
      if $B_i \in NT$ *then*
        $FOLLOW(B_i) \leftarrow FOLLOW(B_i) \cup TRAILER$
        if $\varepsilon \in FIRST(B_i)$
          then $TRAILER \leftarrow TRAILER \cup (FIRST(B_i) - \{\varepsilon\})$
          else $TRAILER \leftarrow FIRST(B_i)$
        else $TRAILER \leftarrow \{B_i\}$ // $B_i \in T$ => only 1 symbol

Classic Expression Grammar

0	*Goal*	→	*Expr*
1	*Expr*	→	*Term Expr'*
2	*Expr'*	→	+ *Term Expr'*
3		|	- *Term Expr'*
4		|	ε
5	*Term*	→	*Factor Term'*
6	*Term'*	→	* *Factor Term'*
7		|	/ *Factor Term'*
8		|	ε
9	*Factor*	→	number
10		|	id
11		|	(*Expr*)

Symbol	FIRST	FOLLOW
num	num	Ø
id	id	Ø
+	+	Ø
-	-	Ø
*	*	Ø
/	/	Ø
(	(	Ø
)	)	Ø
eof	eof	Ø
ε	ε	Ø
Goal	(,id,num	eof
Expr	(,id,num	), eof
Expr'	+, -, ε	), eof
Term	(,id,num	+, -,), eof
Term'	*, /, ε	+,-,), eof
Factor	(,id,num	+,-,*,/,),eof

Classic Expression Grammar

0	*Goal*	→	*Expr*
1	*Expr*	→	*Term Expr'*
2	*Expr'*	→	+ *Term Expr'*
3		|	- *Term Expr'*
4		|	ε
5	*Term*	→	*Factor Term'*
6	*Term'*	→	* *Factor Term'*
7		|	/ *Factor Term'*
8		|	ε
9	*Factor*	→	number
10		|	id
11		|	(*Expr*)

Goal → *Expr*

TRAILER = FOLLOW(Goal)

FOLLOW(Expr) ← *FOLLOW(Expr)* ∪ *TRAILER*

Symbol	FIRST	FOLLOW
num	num	Ø
id	id	Ø
+	+	Ø
-	-	Ø
*	*	Ø
/	/	Ø
(	(	Ø
)	)	Ø
eof	eof	Ø
ε	ε	Ø
Goal	(,id,num	eof
Expr	(,id,num	), eof
Expr'	+, -, ε	), eof
Term	(,id,num	+, -,), eof
Term'	*, /, ε	+,-,), eof
Factor	(,id,num	+,-,*,/,),eof

Classic Expression Grammar

0	*Goal*	→	*Expr*
1	*Expr*	→	*Term Expr'*
2	*Expr'*	→	+ *Term Expr'*
3		|	- *Term Expr'*
4		|	ε
5	*Term*	→	*Factor Term'*
6	*Term'*	→	* *Factor Term'*
7		|	/ *Factor Term'*
8		|	ε
9	*Factor*	→	number
10		|	id
11		|	(*Expr*)

Expr → *Term Expr'*

TRAILER = FOLLOW(Expr)

FOLLOW(Expr') ← *FOLLOW(Expr')* ∪ *TRAILER*

Symbol	FIRST	FOLLOW
num	num	Ø
id	id	Ø
+	+	Ø
-	-	Ø
*	*	Ø
/	/	Ø
(	(	Ø
)	)	Ø
eof	eof	Ø
ε	ε	Ø
Goal	(,id,num	eof
Expr	(,id,num	), eof
Expr'	+, -, ε	), eof
Term	(,id,num	+, -,), eof
Term'	*, /, ε	+,-,), eof
Factor	(,id,num	+,-,*,/,),eof

Classic Expression Grammar

0	*Goal*	→	*Expr*
1	*Expr*	→	*Term Expr'*
2	*Expr'*	→	+ *Term Expr'*
3		|	- *Term Expr'*
4		|	ε
5	*Term*	→	*Factor Term'*
6	*Term'*	→	* *Factor Term'*
7		|	/ *Factor Term'*
8		|	ε
9	*Factor*	→	(*Expr*)
10		|	number
11		|	id

Prod'n	FIRST+
0	(,id,num
1	(,id,num
2	+
3	-
4	ε,), eof
5	(,id,num
6	*
7	/
8	ε,+,-,), eof
9	(
10	number
11	id

FIRST+(A→β) is identical to FIRST(β) except for productions 4 and 8

FIRST+(Expr'→ ε) is {ε,), eof}

FIRST+(Term'→ ε) is {ε,+,-,), eof}

Building Top-down Parsers for LL(1) Grammars

Given an *LL(1)* grammar, and its FIRST & FOLLOW sets …

- Emit a routine for each non-terminal
  - Nest of if-then-else statements to check alternate rhs's
  - Each returns true on success and throws an error on false
  - Simple, working (*perhaps inefficent*) code
- This automatically constructs a recursive-descent parser

Improving matters

- Nest of if-then-else statements may be slow
  - Good case statement implementation would be better
- What about a table to encode the options?
  - Interpret the table with a skeleton, as we did in scanning

Building Top-down Parsers

Strategy

- Encode knowledge in a table
- Use a standard "skeleton" parser to interpret the table

Example

- The non-terminal *Factor* has 3 expansions
  (*Expr*) or Identifier or Number
- Table might look like:

0	*Goal*	→	*Expr*	
1	*Expr*	→	*Term Expr'*	
2	*Expr'*	→	+ *Term Expr'*	
3		|	- *Term Expr'*	
4		|	ε	
5	*Term*	→	*Factor Term'*	
6	*Term'*	→	* *Factor Term'*	
7		|	/ *Factor Term'*	
8		|	ε	
9	*Factor*	→	(*Expr*)	
10		|	number	
11		|	id	

Terminal Symbols

Non-terminal Symbols	+	-	*	/	Id.	Num.	(	)	EOF
Factor	—	—	—	—	11	10	9	—	—

Cannot expand *Factor* into an operator ⇒ error

Expand *Factor* by rule 10 with input "number"

Building Top-down Parsers

Building the complete table

- Need a row for every *NT* & a column for every *T*

LL(1) Expression Parsing Table

	+	-	*	/	Id	Num	(	)	EOF
Goal	—	—	—	—	0	0	0	—	—
Expr	—	—	—	—	1	1	1	—	—
Expr'	2	3	—	—	—	—	—	4	4
Term	—	—	—	—	5	5	5	—	—
Term'	8	8	6	7	—	—	—	8	8
Factor	—	—	—	—	11	10	9	—	—

Row we built earlier

Building Top-down Parsers

Building the complete table

- Need a row for every *NT* & a column for every *T*
- Need an interpreter for the table (*skeleton parser*)

LL(1) Skeleton Parser

```
word ← NextWord()                  // Initial conditions, including
push EOF onto Stack                // a stack to track local goals
push the start symbol, S, onto Stack
TOS ← top of Stack

loop forever
  if TOS = EOF and word = EOF then
     break & report success        // exit on success
  else if TOS is a terminal then
     if TOS matches word then
        pop Stack                  // recognized TOS
        word ← NextWord()
     else report error looking for TOS  // error exit
  else                             // TOS is a non-terminal
     if TABLE[TOS,word] is A→ B1B2…Bk then
        pop Stack                  // get rid of A
        push Bk, Bk-1, …, B1       // in that order
     else break & report error expanding TOS
  TOS ← top of Stack
```

LL(1) Skeleton Parser

```
word ← NextWord()              // Initial conditions, including
push EOF onto Stack            // a stack to track local goals
push the start symbol, S, onto Stack
TOS ← top of Stack
loop forever
  if TOS = EOF and word = EOF then
     break & report success    // exit on success
  else if TOS is a terminal then
     if TOS matches word then
        pop Stack              // recognized TOS
        word ← NextWord()
     else report error looking for TOS  // error exit
  else                         // TOS is a non-terminal
     if TABLE[TOS,word] is A→ B1B2…Bk then
        pop Stack              // get rid of A
        push Bk, Bk-1, …, B1   // in that order
     else break & report error expanding TOS
  TOS ← top of Stack
```

LL(1) Skeleton Parser

```
word ← NextWord()              // Initial conditions, including
push EOF onto Stack            // a stack to track local goals
push the start symbol, S, onto Stack
TOS ← top of Stack

loop forever
  if TOS = EOF and word = EOF then
     break & report success    // exit on success
  else if TOS is a terminal then
     if TOS matches word then
        pop Stack              // recognized TOS
        word ← NextWord()
     else report error looking for TOS  // error exit
  else                         // TOS is a non-terminal
     if TABLE[TOS,word] is A→ B1B2…Bk then
        pop Stack              // get rid of A
        push Bk, Bk-1, …, B1   // in that order
     else break & report error expanding TOS
  TOS ← top of Stack
```

LL(1) Skeleton Parser

```
word ← NextWord()              // Initial conditions, including
push EOF onto Stack            // a stack to track local goals
push the start symbol, S, onto Stack
TOS ← top of Stack

loop forever
  if TOS = EOF and word = EOF then
     break & report success    // exit on success
  else if TOS is a terminal then
     if TOS matches word then
        pop Stack              // recognized TOS
        word ← NextWord()
     else report error looking for TOS  // error exit
  else                         // TOS is a non-terminal
     if TABLE[TOS,word] is A→ B1B2…Bk then
        pop Stack              // get rid of A
        push Bk, Bk-1, …, B1   // in that order
     else break & report error expanding TOS
  TOS ← top of Stack
```

LL(1) Skeleton Parser

```
word ← NextWord()              // Initial conditions, including
push EOF onto Stack            // a stack to track local goals
push the start symbol, S, onto Stack
TOS ← top of Stack

loop forever
  if TOS = EOF and word = EOF then
     break & report success    // exit on success
  else if TOS is a terminal then
     if TOS matches word then
        pop Stack              // recognized TOS
        word ← NextWord()
     else report error looking for TOS  // error exit
  else                         // TOS is a non-terminal
     if TABLE[TOS,word] is A→ B1B2…Bk then
        pop Stack              // get rid of A
        push Bk, Bk-1, …, B1   // in that order
     else break & report error expanding TOS
  TOS ← top of Stack
```

Building Top-down Parsers

Building the complete table

- Need a row for every *NT* & a column for every *T*
- Need a table-driven interpreter for the table
- Need an algorithm to build the table

Filling in TABLE[X,y], $X \in NT$, $y \in T$

1. entry is the rule $X \rightarrow \beta$, if $y \in \text{FIRST}^{+}(X \rightarrow \beta)$
2. entry is *error* if rule 1 does not define

If any entry has more than one rule, *G* is not *LL(1)*

We call this algorithm the *LL(1)* table construction algorithm