

# COMP 520 - Compilers

Lecture 4 (Tue Jan 25, 2022)

## *Recursive-descent parsing of EBNF grammars and the LL(1) condition*

- **WA2 – parsing written assignment**
  - online on the class web site
  - due date changed to 1/27
- **Additional reading for Thu Jan 27**
  - PLPJ Secn 4.5 Scanner construction (pp 118 – 124)
- **MiniJava project checkpoint PA1**
  - due Tue Feb 1

# Topics

---

- **Recursive descent parser**
  - Constructing a simple recursive descent parser for an EBNF grammar
- **EBNF Grammar analysis**
  - define properties of nonterminals and sentential forms
    - Nullable
    - Starters
    - Followers
  - determine whether a grammar meets the LL(1) condition
- **simpleScannerParser example**
  - walkthrough



# Review: Recursive Descent Parser

---

- **What is it?**
  - a mutually recursive set of procedures to parse a context free grammar meeting the LL(1) condition
- **Context free grammar**
  - one rule per nonterminal
    - EBNF grammars can always be written this way
  - must satisfy the LL(1) condition
    - detailed rules are topic today
    - some non-LL(1) grammars can be transformed to meet the LL(1) condition
      - EBNF is a big help here
      - substitution, left factoring, left recursion elimination
- **Parser**
  - one procedure per nonterminal
    - body of the procedure corresponds to the right hand-side of the rule
    - procedure parses an instance of right hand side, or throws a syntax error
  - helper functions
    - scanner provides the terminals of the grammar, parser accepts them



# Example transformation of a grammar to LL(1)

## Simple grammar to describe arithmetic expressions

$E ::= T \mid E \text{ Op } T$

$T ::= (E) \mid \text{num}$

$\text{Op} ::= + \mid \times$

### 1. Add new start symbol $S$ and terminal $\$$ representing end-of-input

$S ::= E \$$

### 2. Remove left recursion

$E ::= T (\text{Op } T)^*$

$T ::= (E) \mid \text{num}$

$\text{Op} ::= + \mid \times$

### 3. Substitute for $\text{Op}$

$E ::= T ((+ \mid \times) T)^*$

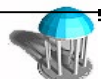
$T ::= (E) \mid \text{num}$



# Recursive descent parsers for EBNF

- How to write a recursive descent parser for EBNF grammars?
  - Choice  $\alpha \mid \beta$ 
    - Conditional or switch construct based on current input symbol
  - Repetition  $\alpha^*$ 
    - While statement that repeats based on current input symbol
  - Example grammar
    - $S ::= E \$$
    - $E ::= T ((+ \mid \times) T)^*$
    - $T ::= ( E ) \mid \textit{num}$
    - $\textit{num} = \{ 0, 1, \dots, 9 \}$

```
void parseS() {
    parseE();
    accept('$');
}
void parseE() {
    parseT();
    while (currChar == '+'
           || currChar == 'x') {
        acceptIt();
        parseT();
    }
}
void parseT() {
    switch (currChar) {
        case '0': ... case '9':
            acceptIt();
            return;
        case '(':
            acceptIt();
            parseE();
            accept(')');
            return;
        default:
            error();
    }
}
```



# Grammar properties: definitions

---

- Given an EBNF grammar

- nonterminal set  $N$ , start symbol  $S$ , terminal set  $T$
- one rule per nonterminal
  - multiple rules with same NT at left can be combined

$$A ::= \alpha_1 \quad \dots \quad A ::= \alpha_m \quad \Rightarrow \quad A ::= \alpha_1 | \dots | \alpha_m$$

- Define

- *Nullable* $[\alpha]$ 
  - Property that is True iff  $\alpha$  can derive the empty string
- *Starters* $[\alpha]$ 
  - Set of terminals that may start derivations from  $\alpha$
  - Includes  $\varepsilon$  if *Nullable* $(\alpha)$
- *Followers* $(A)$  where  $A \in N$ 
  - Set of terminals that may follow  $A$  in a derivation
  - For augmented grammars, only *Followers* $(S)$  includes  $\varepsilon$



# Informal LL(1) condition for EBNF grammars

- **Idea**

- For each choice of the form  $A ::= \beta (\alpha_1 | \dots | \alpha_m) \gamma$ 
  - $\text{Starters}[\alpha_i]$  and  $\text{Starters}[\alpha_j]$  must be disjoint for all  $1 \leq i, j \leq m$
- For each repetition of the form  $A ::= \beta (\alpha)^* \gamma$ 
  - $\text{Starters}[\alpha]$  and  $\text{Starters}[\gamma]$  are disjoint
  - $\text{Nullable}[\alpha]$  is False

- **Example**

- Is this EBNF grammar LL(1)?

$S ::= A \$$

$A ::= x z \mid x E (y E)^* z$

$E ::= a \mid b$



# How to determine Starters and Followers?

- Constructed by inspection of the grammar in the following order
  1. Jointly determine Nullable(A) for all nonterminals A
  2. Jointly determine Starters(A) for all nonterminals A
  3. Determine Followers(A) for all nonterminals A
- Place the grammar in a form where each nonterminal A has a single rule

$$A ::= \alpha$$

- Definition
  - $\varepsilon$  is the empty sequence of terminals
  - for sets S, T define

$$S \oplus T = \begin{cases} S & \text{if } \varepsilon \notin S \\ (S - \{\varepsilon\}) \cup T & \text{if } \varepsilon \in S \end{cases}$$





# Nullable nonterminals

---

- Definition

$$\text{Nullable}(A) = \begin{cases} \text{true}, & \text{if } A \Rightarrow^* \varepsilon \\ \text{false}, & \text{otherwise} \end{cases}$$

- Computation

- For each  $A ::= \alpha$

$$N_0(A) = \text{false}$$

$$N_{i+1}(A) = N_i(\alpha)$$

This means evaluate  $\text{Nullable}(\alpha)$  (*defn next slide*) using  $N_i(C)$  in place of  $\text{Nullable}(C)$  for any nonterminal  $C$

$$\text{Nullable}(A) = \lim_{i \rightarrow \infty} N_i(A)$$



# Nullable[ $\alpha$ ]

---

- Definition

$$\text{Nullable}(\alpha) = \begin{cases} \text{true}, & \text{if } \alpha \Rightarrow^* \varepsilon \\ \text{false}, & \text{otherwise} \end{cases}$$

- Computation

- inductive over structure of  $\alpha$

1.  $\alpha = \varepsilon$                        $\text{Nullable}(\alpha) = \text{true}$
2.  $\alpha = t$ ,     $t \in T$                $\text{Nullable}(\alpha) = \text{false}$
3.  $\alpha = A$ ,     $A \in N$                $\text{Nullable}(\alpha) = \text{Nullable}(A)$
4.  $\alpha = \alpha_1 \alpha_2 \dots \alpha_n$          $\text{Nullable}(\alpha) = \text{Nullable}(\alpha_1) \wedge \dots \wedge \text{Nullable}(\alpha_n)$
5.  $\alpha = \alpha_1 | \alpha_2 | \dots | \alpha_n$      $\text{Nullable}(\alpha) = \text{Nullable}(\alpha_1) \vee \dots \vee \text{Nullable}(\alpha_n)$
6.  $\alpha = \beta^*$                        $\text{Nullable}(\alpha) = \text{true}$



# Starters(A)

---

- **Definition**

- Let  $A$  be a nonterminal in  $N$

$$\text{Starters}(A) = \{t \mid A \Rightarrow^* t\beta \text{ for some } \beta\} \cup \begin{cases} \{\varepsilon\}, & \text{if Nullable}(A) \\ \{\}, & \text{otherwise} \end{cases}$$

- **Computation**

- For each  $A ::= \alpha$

$$ST_0(A) = \{\varepsilon\}, \text{ if Nullable}(A)$$

$$ST_0(A) = \{\}, \text{ otherwise}$$

$$ST_{i+1}(A) = ST_i(\alpha)$$

$$\text{Starters}(A) = \lim_{i \rightarrow \infty} ST_i(A)$$



# Starters[ $\alpha$ ]

---

- Definition

$$\text{Starters}[\alpha] = \{t \mid \alpha \Rightarrow^* t\beta \text{ for some } \beta\} \cup \begin{cases} \{\varepsilon\}, & \text{if Nullable}(\alpha) \\ \{\}, & \text{otherwise} \end{cases}$$

- Computation

- inductive over the structure of  $\alpha$

1.  $\alpha = \varepsilon$                        $\text{Starters}(\alpha) = \{ \varepsilon \}$
2.  $\alpha = t, \quad t \in T$                $\text{Starters}(\alpha) = \{ t \}$
3.  $\alpha = A, \quad A \in N$                $\text{Starters}(\alpha) = \text{Starters}(A)$
4.  $\alpha = \alpha_1 \alpha_2 \dots \alpha_n$            $\text{Starters}(\alpha) = \text{Starters}(\alpha_1) \oplus \text{Starters}(\alpha_2 \dots \alpha_n)$
5.  $\alpha = \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$        $\text{Starters}(\alpha) = \text{Starters}(\alpha_1) \cup \dots \cup \text{Starters}(\alpha_n)$
6.  $\alpha = \beta^*$                            $\text{Starters}(\alpha) = \text{Starters}(\beta) \cup \{\varepsilon\}$



# Followers(A)

---

- Definition

$$Followers(A) = \{t \mid S \Rightarrow^* \alpha A t \beta\} \cup \begin{cases} \{\varepsilon\}, & \text{if } S \Rightarrow^* \alpha A \\ \{\}, & \text{otherwise} \end{cases}$$

- Computation

Define  $Followers(A)$ , the followers for nonterminal  $A$ , as

$$Followers(A) = \left( \lim_{i \rightarrow \infty} FL_i(A) \right) \cup (\{\varepsilon\} \text{ if } S \Rightarrow \alpha A)$$

where

$$FL_0(A) = \left( \bigcup_{C \Rightarrow \alpha A \beta} Starters[\beta] \right) - \{\varepsilon\}$$

$$FL_{i+1}(A) = FL_i(A) \cup \bigcup_{C \Rightarrow \alpha A \beta \text{ and } Nullable(\beta)} FL_i(C)$$



# Example

---

- Grammar

$S ::= A\$$

$A ::= BDA \mid a$

$B ::= D \mid b$

$D ::= d \mid \varepsilon$

- Compute sets Nullable, Starters, Followers

- is grammar LL(1)?



# Example: computation of Nullable

```
S ::= A$
A ::= BDA | a
B ::= D | b
D ::= d | ε
```

- Simplify using the inductive defns on RHS of rules

$$N(S) = N(A\$) = N(A) \wedge N(\$) = N(A) \wedge \text{false} = \text{false}$$

$$N(A) = (N(B) \wedge N(D) \wedge N(A)) \vee N(a) = N(B) \wedge N(D) \wedge N(A)$$

$$N(B) = N(D) \vee N(b) = N(D) \vee \text{false} = N(D)$$

$$N(D) = N(d) \vee N(\epsilon) = \text{false} \vee \text{true} = \text{true}$$

- Evaluate iteratively till fixpoint

$$N_{i+1}(S) = \text{false}$$

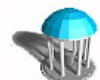
$$N_{i+1}(A) = N_i(B) \wedge N_i(D) \wedge N_i(A)$$

$$N_{i+1}(B) = N_i(D)$$

$$N_{i+1}(D) = \text{true}$$

fixpoint:  
 $N_2 = N_3$

	$N_0$	$N_1$	$N_2$	$N_3$
S	F	F	F	F
A	F	F	F	F
B	F	F	T	T
D	F	T	T	T



# Example: computation of Starters

```

S ::= A$
A ::= BDA | a
B ::= D | b
D ::= d | ε
    
```

- Simplify equations using the inductive defns

$$ST(S) = ST(A\$) = ST(A) \oplus ST(\$) = ST(A) \quad \text{since A is not nullable}$$

$$\begin{aligned} ST(A) &= ST(BDA \mid a) = (ST(B) \oplus ST(D) \oplus ST(A)) \cup ST(a) \\ &= (ST(B) \cup ST(D) - \{\epsilon\}) \cup ST(A) \cup \{a\} \quad \text{since both B and D are nullable} \end{aligned}$$

$$ST(B) = ST(D) \cup ST(b) = ST(D) \cup \{b\}$$

$$ST(D) = ST(d) \cup ST(\epsilon) = \{d, \epsilon\}$$

- Evaluate iteratively till fixpoint

$$ST_{i+1}(S) = ST_i(A)$$

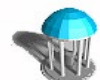
$$ST_{i+1}(A) = (ST_i(B) \cup ST_i(D) - \{\epsilon\}) \cup ST_i(A) \cup \{a\}$$

$$ST_{i+1}(B) = ST_i(D) \cup \{b\}$$

$$ST_{i+1}(D) = \{d, \epsilon\}$$

$ST_3 = ST_4$

	$ST_0$	$ST_1$	$ST_2$	$ST_3$
<b>S</b>	$\{\}$	$\{\}$	$\{a\}$	$\{a,b,d\}$
<b>A</b>	$\{\}$	$\{a\}$	$\{a,b,d\}$	$\{a,b,d\}$
<b>B</b>	$\{\epsilon\}$	$\{b,\epsilon\}$	$\{b,d,\epsilon\}$	$\{b,d,\epsilon\}$
<b>D</b>	$\{\epsilon\}$	$\{d,\epsilon\}$	$\{d,\epsilon\}$	$\{d,\epsilon\}$





# Example: computation of Followers

```

S ::= A$
A ::= BDA | a
B ::= D | b
D ::= d | ε
    
```

- Simplify  $FL_0$  equations using the defs

$$FL_0(S) = \{\}$$

$$FL_0(A) = (ST(\$) \cup ST(\epsilon)) - \{\epsilon\} = \{\$\}$$

$$FL_0(B) = ST(DA) - \{\epsilon\} = \{a, b, d\}$$

$$FL_0(D) = (ST(A) \cup ST(\epsilon)) - \{\epsilon\} = \{a, b, d\}$$

S does not occur in RHS of any rule

A occurs in RHS of 1<sup>st</sup> and 2<sup>nd</sup> rules

B occurs in RHS of 2<sup>nd</sup> rule

D occurs in RHS of 2<sup>nd</sup> and 3<sup>rd</sup> rules

- Simplify  $FL_{i+1}$  equations using defs

$$FL_{i+1}(S) = FL_i(S)$$

$$FL_{i+1}(A) = FL_i(A) \cup FL_i(A) = FL_i(A)$$

$$FL_{i+1}(B) = FL_i(B) \quad \text{since DA not nullable}$$

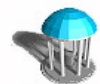
$$FL_{i+1}(D) = FL_i(D) \cup FL_i(B) \quad \begin{array}{l} FL_i(B) \text{ from D in 3}^{\text{rd}} \text{ rule} \\ \text{nothing from 2}^{\text{nd}} \text{ rule} \\ \text{since A not nullable} \end{array}$$

$FL_0 = FL_1$

	$FL_0$	$FL_1$
<b>S</b>	{}	{}
<b>A</b>	{\$}	{\$}
<b>B</b>	{a,b,d}	{a,b,d}
<b>D</b>	{a,b,d}	{a,b,d}

- Iterate till fixpoint. Then add  $\epsilon$  for any NT X where  $S \Rightarrow^* \beta X$

– in augmented grammars, this can only happen when  $X = S$ , so Followers(S) is the only set that includes  $\epsilon$



# LL(1) Condition for EBNF grammars

- EBNF grammar is LL(1) if the following requirements are met

- For each choice  $A ::= \beta (\alpha_1 | \dots | \alpha_m) \gamma$  define

$$\text{Predict}[\alpha_i] = \text{Starters}[\alpha_i \gamma] \oplus \text{Followers}[A]$$

- $\text{Predict}[\alpha_i]$  and  $\text{Predict}[\alpha_j]$  must be disjoint for all  $1 \leq i, j \leq m$  and  $i \neq j$

- For each repetition  $A ::= \beta (\alpha)^* \gamma$  define

$$\text{Predict}[\alpha] = \text{Starters}[\alpha]$$

$$\text{Predict}[\gamma] = \text{Starters}[\gamma] \oplus \text{Followers}[A]$$

- $\alpha$  must not be Nullable
- $\text{Starters}[\alpha]$  and  $(\text{Starters}[\gamma] \oplus \text{Followers}[A])$  must be disjoint



# LL(1) Condition for example grammar

- Grammar

$S ::= A\$$  ①  
 $A ::= BDA \mid a$  ②  
 $B ::= D \mid b$   
 $D ::= d \mid \epsilon$  ③

- Find decision points in grammar

note Nullable(A) = false

– there are three choice points and no repetition points

1.  $\text{Predict}(BDA) = \text{Starters}(B) \oplus \text{Starters}(D) \oplus \text{Starters}(A)$   
 $= \{b, d, \epsilon\} \oplus \{d, \epsilon\} \oplus \{a, b, d\} = \{a, b, d\}$

$\text{Predict}(a) = \{a\}$  ← not disjoint! →

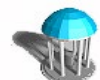
2.  $\text{Predict}(D) = \text{Starters}(D) \oplus \text{Followers}(B) = \{d, \epsilon\} \oplus \{a, b, d\} = \{a, b, d\}$

$\text{Predict}(b) = \{b\}$  ← not disjoint! →

3.  $\text{Predict}(d) = \{d\}$   
 $\text{Predict}(\epsilon) = \text{Starters}(\epsilon) \oplus \text{Followers}(D) = \{\epsilon\} \oplus \{a, b, d\} = \{a, b, d\}$

← not disjoint! →

- Grammar is not LL(1)!



# Constructing a recursive descent parser for EBNF grammar $G$

---

- For each rule  $N ::= \alpha$  in  $G$ 
  - construct a procedure  $\text{parseN}()$
  - body of procedure is defined inductively on structure of  $\alpha$
- Operations on current input symbol  $c$  within parser
  - $c \in T$ 
    - true if  $c$  occurs in set  $T$
  - $\text{accept}(t)$ 
    - if  $c == t$  then advance  $c$  to next input symbol else parse error



# Construction of recursive descent EBNF parser

- For each rule  $N ::= \alpha$  construct procedure `parseN()`
  - `void parseN() { parse  $\alpha$ , F }` where  $F = \text{Followers}(N)$
  - $F$  is the *trailing context*, the set of terminals that may be encountered after parsing  $\alpha$

to parse $\alpha$ , $F$	construct program text	LL(1) condition
$\alpha = t\beta \quad t \in T$	<code>accept(t); <code>parse <math>\beta</math>, F</code></code>	OK
$\alpha = A\beta \quad A \in N$	<code>parseA(); <code>parse <math>\beta</math>, F</code></code>	OK
$\alpha = \varepsilon$	(none)	OK
$\alpha = (\alpha_1   \dots   \alpha_n) \beta$	<code>switch c</code> <code>  <math>c \in \text{Starters}(\alpha_1\beta) \oplus F</math>: <code>parse <math>\alpha_1</math>, <math>\beta F</math></code></code> <code>  ...</code> <code>  <math>c \in \text{Starters}(\alpha_n\beta) \oplus F</math>: <code>parse <math>\alpha_n</math>, <math>\beta F</math></code></code> <code>  <code>parse <math>\beta</math>, F</code></code>	$\text{Starters}(\alpha_i \beta) \oplus F$ $\text{Starters}(\alpha_j \beta) \oplus F$ disjoint for all $1 \leq i \neq j \leq n$
$\alpha = \gamma^* \beta$	<code>while c <math>\in \text{Starters}(\gamma)</math> do</code> <code>  <code>parse <math>\gamma</math>, (<math>\text{Starters}(\gamma) \cup \beta F</math>)</code></code> <code>  <code>parse <math>\beta</math>, F</code></code>	$\text{Starters}(\gamma)$ disjoint from $\text{Starters}(\beta) \oplus F$ , and $\gamma$ not nullable

