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 | E Op T
T ::= (E) | num
Op ::= + | ×

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

   S ::= E $

2. Remove left recursion

   E ::= T (Op T)*
   T ::= (E) | num
   Op ::= + | ×

3. Substitute for Op

   E ::= T ((+ | ×) T)*
   T ::= (E) | 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 α^*
    - While statement that repeats based on current input symbol
  - Example grammar

    S ::= E $

    E ::= T ((+ | ×) T)*

    T ::= (E) | *num*

    num = { 0, 1, … , 9 }

```
void parseS() {
   parseE();
   accept('$');
}
void parseE() {
   parseT();
   while (currChar == '+'
             || currChar == '×') {
         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 ::= α_1 ... A ::= α_m $\Rightarrow$ A ::= $\alpha_1 \mid \ldots \mid \alpha_m$

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

Informal LL(1) condition for EBNF grammars

- Idea
  - For each choice of the form A ::= $\beta\ (\alpha_1 \mid \ldots \mid \alpha_m)\ \gamma$
    - Starters[α_i] and Starters[α_j] must be disjoint for all $1 \leq i, j \leq m$
  - For each repetition of the form A ::= $\beta\ (\alpha)^*\ \gamma$
    - Starters [α] and Starters[γ] are disjoint
    - Nullable[α] is False
- Example
  - Is this EBNF grammar LL(1)?

```
S  ::= A $
A  ::= x z  |  x E (y E)* z
E  ::= a  |  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 ::= α

- Definition
  - ε is the empty sequence of terminals
  - for sets S, T define

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

Nullable nonterminals

- Definition

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

- Computation
  - For each A ::= α

$N_0(A)$ = false

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

This means evaluate Nullable(α) (*defn next slide*) using $N_i(C)$ in place of Nullable (C) for any nonterminal C

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

Nullable[α]

- Definition

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

- Computation
  - inductive over structure of α

1. $\alpha = \varepsilon$ — Nullable(α) = true
2. $\alpha = t, \quad t \in T$ — Nullable(α) = false
3. $\alpha = A, \quad A \in N$ — Nullable(α) = Nullable(A)
4. $\alpha = \alpha_1 \alpha_2 \ldots \alpha_n$ — Nullable(α) = Nullable(α_1) ∧ … ∧ Nullable(α_n)
5. $\alpha = \alpha_1 | \alpha_2 | \ldots | \alpha_n$ — Nullable(α) = Nullable(α_1) ∨ … ∨ Nullable(α_n)
6. $\alpha = \beta^*$ — Nullable(α) = true

Starters(A)

- Definition
  - Let A be a nonterminal in N

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

- Computation
  - For each A ::= α

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

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

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

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

Starters[α]

- Definition

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

- Computation
  - inductive over the structure of α

1.	$\alpha = \varepsilon$	Starters(α) = { ε }
2.	$\alpha = t, \quad t \in T$	Starters(α) = { t }
3.	$\alpha = A, \quad A \in N$	Starters(α) = Starters(A)
4.	$\alpha = \alpha_1 \alpha_2 \ldots \alpha_n$	Starters(α) = Starters(α_1) ⊕ Starters($\alpha_2 \ldots \alpha_n$)
5.	$\alpha = \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$	Starters(α) = Starters(α_1) U … U Starters(α_n)
6.	$\alpha = \beta^*$	Starters(α) = Starters(β) U {ε}

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

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

where

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

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

Example

- Grammar

  S ::= A$

  A ::= BDA | a

  B ::= D | b

  D ::= d | ε

- 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$) = N(A) ∧ N($) = N(A) ∧ false = false

  N(A) = (N(B) ∧ N(D) ∧ N(A)) ∨ N(a) = N(B) ∧ N(D) ∧ N(A)

  N(B) = N(D) ∨ N(b) = N(D) ∨ false = N(D)

  N(D) = N(d) ∨ N(ε) = false ∨ true = true

- Evaluate iteratively till fixpoint

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

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

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

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

$ST(S) = ST(A\$) = ST(A) \oplus ST(\$) = ST(A)$ since A is not nullable

$ST(A) = ST(BDA \mid a) = (ST(B) \oplus ST(D) \oplus ST(A)) \cup ST(a)$
$= (ST(B) \cup ST(D) - \{\varepsilon\}) \cup ST(A) \cup \{a\}$ since both B and D are nullable

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

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

- Evaluate iteratively till fixpoint

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

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

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

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

$ST_3 = ST_4$

	ST_0	ST_1	ST_2	ST_3
S	{}	{}	{a}	{a,b,d}
A	{}	{a}	{a,b,d}	{a,b,d}
B	{ε}	{b,ε}	{b,d,ε}	{b,d,ε}
D	{ε}	{d,ε}	{d,ε}	{d,ε}

Example: computation of Followers

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

- Simplify FL_0 equations using the defns

$FL_0(S) = \{\}$ — S does not occur in RHS of any rule

$FL_0(A) = (ST(\$) \cup ST(\varepsilon)) - \{\varepsilon\} = \{\$\}$ — A occurs in RHS of 1st and 2nd rules

$FL_0(B) = ST(DA) - \{\varepsilon\} = \{a,b,d\}$ — B occurs in RHS of 2nd rule

$FL_0(D) = (ST(A) \cup ST(\varepsilon)) - \{\varepsilon\} = \{a,b,d\}$ — D occurs in RHS of 2nd and 3rd rules

- Simplify FL_{i+1} equations using defns

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

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

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

$FL_{i+1}(D) = FL_i(D) \cup FL_i(B)$ — $FL_i(B)$ from D in 3rd rule; nothing from 2nd rule since A not nullable

$FL_0 = FL_1$

	FL_0	FL_1
S	{}	{}
A	{$}	{$}
B	{a,b,d}	{a,b,d}
D	{a,b,d}	{a,b,d}

- Iterate till fixpoint. Then add ε 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 ε

LL(1) Condition for EBNF grammars

- EBNF grammar is LL(1) if the following requirements are met
  - For each choice A ::= $\beta\ (\alpha_1 \mid \ldots \mid \alpha_m)\ \gamma$ define

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

    - Predict[α_i] and Predict[α_j] must be disjoint for all $1 \leq i, j \leq m$ and $i \neq j$

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

$$Predict[\alpha] = Starters[\alpha]$$
$$Predict[\gamma] = Starters[\gamma] \oplus Followers[A]$$

    - α must not be Nullable
    - Starters [α] and (Starters[γ]$\oplus$ Followers(A)) must be disjoint

LL(1) Condition for example grammar

- Grammar

S ::= A$ ①

A ::= BDA | a ②

B ::= D | b ③

D ::= d | ε

- Find decision points in grammar

note Nullable(A) = false

  - there are three choice points and no repetition points

1. Predict(BDA) = Starters(B) ⊕ Starters(D) ⊕ Starters(A)
   = {b,d,ε} ⊕ {d,ε} ⊕ {a,b,d} = {a,b,d}
   Predict(a) = {a}

not disjoint!

2. Predict(D) = Starters(D) ⊕ Followers(B) = {d,ε} ⊕ {a,b,d} = {a,b,d}
   Predict(b) = {b}

not disjoint!

3. Predict(d) = {d}
   Predict(ε) = Starters(ε) ⊕ Followers(D) = {ε} ⊕ {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 ::= α in G
  - construct a procedure `parseN()`
  - body of procedure is defined inductively on structure of α

- Operations on current input symbol c within parser
  - c $\in$ T
    - true if c occurs in set T
  - 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 ::= α construct procedure `parseN()`
  - `void parseN(){` *parse α , F* `}` where F = *Followers(N)*
  - F is the *trailing context*, the set of terminals that may be encountered after parsing α

to parse α, F	construct program text	LL(1) condition
$\alpha = t\beta \quad t \in T$	`accept(t);` *parse β,F*	OK
$\alpha = A\beta \quad A \in N$	`parseA();` *parse β,F*	OK
$\alpha = \varepsilon$	(none)	OK
$\alpha = (\alpha_1 \mid \ldots \mid \alpha_n)\,\beta$	`switch c` c $\in$ Starters($\alpha_1\beta$)$\oplus$F: *parse α_1, βF* … c $\in$ Starters($\alpha_n\beta$)$\oplus$F: *parse α_n, βF* *parse β,F*	Starters($\alpha_i\ \beta$)$\oplus$F Starters($\alpha_j\ \beta$)$\oplus$F disjoint for all $1 \le i \ne j \le n$
$\alpha = \gamma^*\ \beta$	`while` c $\in$ Starters(γ) `do` parse γ, (Starters(γ) U βF) *parse β,F*	Starters(γ) disjoint from Starters(β) $\oplus$ F, and γ not nullable