Programming Paradigms

Syntax (Part 6)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2020

Overview

- **Specifying syntax**
  - □ Regular expressions
  - □ Context-free grammars
- **Scanning**
- **Parsing**
  - □ Top-down parsing
  - □ Bottom-up parsing ←

Bottom-up Parsing

- **LR(k) parsers**
  - Left-to-right scanning, Right-most derivation, k tokens look-ahead
- **Difficult to do by hand**
- **Mostly based on automatically generated table**

Shift-reduce Algorithm

- **Repeat until all tokens read and all symbols reduced to start symbol:**
  - Shift (i.e., read) input tokens
  - Try to reduce a group of symbols into a single non-terminal

Example: Shift-reduce parsing

$S \rightarrow a\,T\,R\,e$

$T \rightarrow T\,b\,c \mid b$

$R \rightarrow d$

Input: $\underline{a}\ \underline{b}\ \underline{b}\ \underline{c}\ \underline{d}\,\underline{e}$

Steps:

Shift a, shift b

Reduce $T \rightarrow b$

Shift b , shift c

Reduce $T \rightarrow T\,b\,c$

Shift d

Reduce $R \rightarrow d$

Shift e

Reduce $S \rightarrow a\,T\,R\,e$

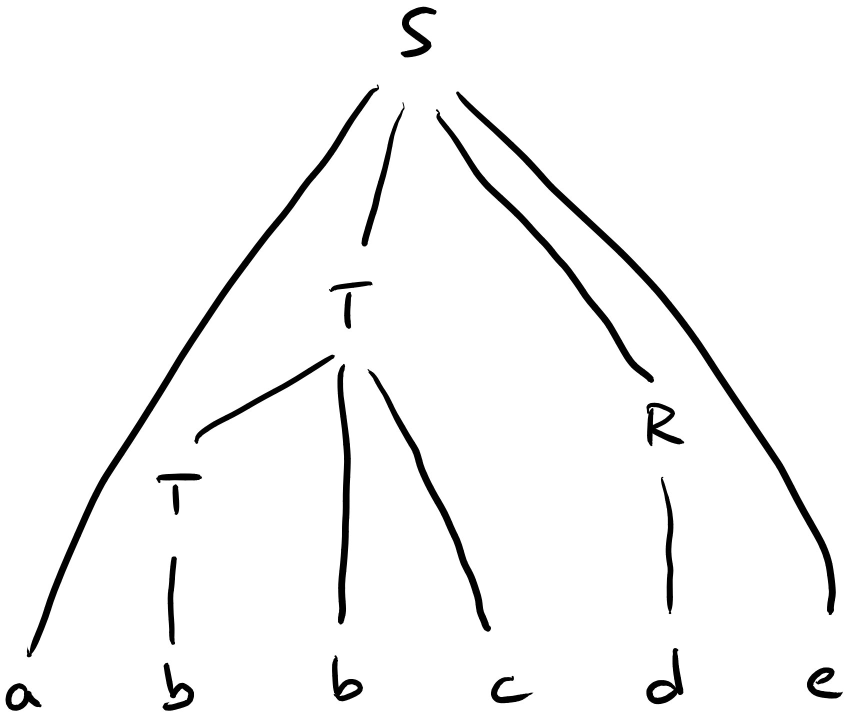

Table-based LR Parsing

- **Two tables**
  - Action table:

    state × T → reduce/shift/accept/error
  - Goto table:

    state × N → state
- **Stack of symbol/state pairs**
  - Record of what has been seen in the past

Example: LR(1) Table

State	a	b	c	d	e	EOF	S	T	R
s0	s1								
s1		s3						2	
s2		s5		s6					4
s3		r3		r3					
s4					s7				
s5			s8						
s6					r4				
s7						acc.			
s8		r2		r2					

Example: LR(1) Table

State	a	b	c	d	e	EOF	S	T	R
s0	s1								
s1		s3						2	
s2		s5		s6					4
s3		r3		r3					
s4					s7				
s5			s8						
s6					r4				
s7						acc.			
s8		r2		r2					

Action table

Example: LR(1) Table

State	a	b	c	d	e	EOF	S	T	R
s0	s1								
s1		s3						2	
s2		s5		s6					4
s3		r3		r3					
s4					s7				
s5			s8						
s6					r4				
s7						acc.			
s8		r2		r2					

Goto table

Example: LR(1) Table

State	a	b	c	d	e	EOF	S	T	R
s0	s1								
s1		s3						2	
s2		s5		s6					4
s3		r3		r3					
s4					s7				
s5			s8						
s6					r4				
s7						acc.			
s8		r2		r2					

s means shift to some state

Example: LR(1) Table

State	a	b	c	d	e	EOF	S	T	R
s0	s1								
s1		s3						2	
s2		s5		s6					4
s3		r3		r3					
s4					s7				
s5			s8						
s6					r4				
s7						acc.			
s8		r2		r2					

r means reduce using some production

Example: LR(1) Table

State	a	b	c	d	e	EOF	S	T	R
s0	s1								
s1		s3						2	
s2		s5		s6					4
s3		r3		r3					
s4					s7				
s5			s8						
s6					r4				
s7						acc.			
s8		r2		r2					

Accept input (i.e., done with parsing)

Example: LR(1) Table

State	a	b	c	d	e	EOF	S	T	R
s0	s1								
s1		s3						2	
s2		s5		s6					4
s3		r3		r3					
s4					s7				
s5			s8						
s6					r4				
s7						acc.			
s8		r2		r2					

No entry means error

Table-based LR(1) Parsing

```
stack.push(EOF, 0);
nextToken = lookAhead();
repeat
  s = state on top of stack
  if action[s, nextToken] = shift s′
    stack.push(nextToken, s′);
    nextToken = lookAhead();
  else if action[s, nextToken] = reduce x -> y1 .. ym
    pop m pairs from stack
    s′ = state on top of stack
    push(x, goto[s′, x])
  else if action[s, nextToken] = accept
    accept and return
  else error()
```

Table-based LR(1) Parsing

Stack hold roots of partial trees found so far

```
stack.push(EOF, 0);
nextToken = lookAhead();
repeat
  s = state on top of stack
  if action[s, nextToken] = shift s′
    stack.push(nextToken, s′);
    nextToken = lookAhead();
  else if action[s, nextToken] = reduce x -> y1 .. ym
    pop m pairs from stack
    s′ = state on top of stack
    push(x, goto[s′, x])
  else if action[s, nextToken] = accept
    accept and return
  else error()
```

Table-based LR(1) Parsing

Reduce partial trees into a non-terminal by applying a rule

```
stack.push(EOF, 0);
nextToken = lookAhead();
repeat
  s = state on top of stack
  if action[s, nextToken] = shift s′
    stack.push(nextToken, s′);
    nextToken = lookAhead();
  else if action[s, nextToken] = reduce x -> y1 .. ym
    pop m pairs from stack
    s′ = state on top of stack
    push(x, goto[s′, x])
  else if action[s, nextToken] = accept
    accept and return
  else error()
```

Table-based LR(1) Parsing

```
stack.push(EOF, 0);
nextToken = lookAhead();
repeat
  s = state on top of stack
  if action[s, nextToken] = shift s′
    stack.push(nextToken, s′);
    nextToken = lookAhead();
  else if action[s, nextToken] = reduce x -> y1 .. ym
    pop m pairs from stack
    s′ = state on top of stack
    push(x, goto[s′, x])
  else if action[s, nextToken] = accept
    accept and return
  else error()
```

Read another token

Table-based LR(1) Parsing

```
stack.push(EOF, 0);
nextToken = lookAhead();
repeat
  s = state on top of stack
  if action[s, nextToken] = shift s′
    stack.push(nextToken, s′);
    nextToken = lookAhead();
  else if action[s, nextToken] = reduce x -> y1 .. ym
    pop m pairs from stack
    s′ = state on top of stack
    push(x, goto[s′, x])
  else if action[s, nextToken] = accept
    accept and return
  else error()
```

All subtrees reduced to start symbol

How to Get the Table?

- **Using a "characteristic finite-state machine" computed from the grammar**
- **Details differ for different kinds of LR parsers**
  - □ SLR (simple LR)
  - □ LALR (look-ahead LR)
  - □ Full-LR
- **Beyond the scope of this course**

Quiz: Parsing

Which of these statements is true?

- Recursive descent builds a parse tree from the bottom up.
- The k in LR(k) stands for k tokens look-ahead.
- PREDICT sets are used to compute FIRST and FOLLOW sets.
- The stack of a top-down parser contains the symbols expected in the future.

Please vote via Ilias.

Quiz: Parsing

Which of these statements is true?

- ~~Recursive descent builds a parse tree from the bottom up~~.
- The k in LR(k) stands for k tokens look-ahead.
- ~~PREDICT sets are used to compute FIRST and FOLLOW sets~~.
- The stack of a top-down parser contains the symbols expected in the future.

Please vote via Ilias.

Overview

- **Specifying syntax**
  - □ Regular expressions
  - □ Context-free grammars
- **Scanning**
- **Parsing**
  - □ Top-down parsing
  - □ Bottom-up parsing ✓