Compiler

wugouzi

April 1, 2019

Contents

1 Chap5 Bottom-up parsing

1.1 Overview of bottom-up parsing

- A bottom-up parser uses an **explicit stack** to perform a parse
- The parsing stack will contain both tokens and nonterminals

\$	inputstring \$
...	...
\$StartSymbol	\$accept

- **right-most** derivation – backward start with the tokens; end with the start symbol (1+2+(3+4))+5 (E+2+(3+4))+5 (S+2+(3+4))+5 (S+E+(3+4))+5 (S+(3+4))+5 (S+(E+4))+5 (S+(S+4))+5 (S+(S+E))+5 (S+(S))+5 (S+E)+5 (S)+5 E+5 S+5 S+E S
- **parsing actions**: a sequence of **shift** and **reduce** operations **parser state**: a stack of terminals and non-terminals **current derivation step** = always stack + input

derivation	step stack	unconsumed input
(1+2+(3+4))+5		(1+2+(3+4))+5
	(	1+2+(3+4))+5
(E+2+(3+4))+5	(E	+2+(3+4))+5
(S+2+(3+4))+5	(S	+2+(3+4))+5
	(S+	2+(3+4))+5
	(S+2	+(3+4))+5
(S+E+(3+4))+5	(S+E	+(3+4))+5

- 1. **shift**: shift a terminal from the front of the input to the top of the stack

  1. **reduce**: reduce a string at the top of the stack to a nonterminal A, given the BNF choice A

  A bottom-up parser: **shift-reduce parser**

- One further feature of bottom-up parsers grammars are always augmented with a **new start symbol**. if S is the start symbol, a new start symbol S' is added to the grammar : S' S

- example S'->S S ->(S)S|

  S'=>S=>(S)S=>(S)=>()

	Parsing stack	Input	Action
1	$	( ) $	Shift
2	$ (	) $	Reduce S ->
3	$ (S	) $	Shift
4	$ (S)	$	Reduce S ->
5	$ (S) S	$	Reduce S –> (S) S
6	$S	$	Reduce S'–> S
7	$S'	$	Accept

- example E'->E E->E+n|n

  E'=>E=>E+n=>n+n

	Parsing stack	Input	Action
1	$	n+n$	Shift
2	$n	+n$	Reduce E->n
3	$E	+n$	Shift
4	$E+	n$	Shift
5	$E+n	$	Reduce E->E+n
6	$E	$	Reduce E’->E
7	$E’	$	Accept

Right sentential form

- A **sentential** form is any string derivable from the start symbol. Note that this includes the forms with non-terminals at intermediate steps as well.
- A **right-sentential form** is a sentential form that occurs in a step of rightmost derivation (RMD).
- A **sentence** is a sentential form consisting only of terminals

E,E+,E+n are **viable prefixes** of the right sentential form E+n. The sequence of symbols on the parsing stack is called **viable prefix** of the right sentential form

1.2 Finite automata of LR(0) items and LR(0) parsing

- An **LR(0) item** of a context-free grammar: a production choice with a distinguished position in its right-hand side
- If **A** , = , then **A** **ů** is an LR(0) item
- Example S’ S S (S)S |

  S’ ůS S’ Sů S ů(S)S S (ůS)S S (Sů)S S (S)ůS S (S)Sů S ů

1.2.1 Finite automata of items

- The LR(0) items: as the state of a finite automata
- construct the DFA of sets of LR(0) using the subset construction from NFA

$$A \to \alpha \cdot X\eta \xrightarrow{\text{X}} A \to \alpha X \cdot \eta$$

- If X is a token or a nonterminal

- If X is a token, then this transition corresponds to a shift of X from the input to the top of the stack during a parse

$$A \to \alpha \cdot X\eta \xrightarrow{\qquad\epsilon\qquad} X \to$$

- if X is a nonterminal X will never appear as an input symbol
- The **start state** of the NFA the **initial state** of the parser: the stack is empty
- the solution is to augment the grammar by a single production S' -> S
- **S'->ůS** the **start state** of the NFA

1.2.2 The LR(0) parsing algorithm

- the parsing stack to store: **symbols** and **state numbers**
- pushing the new **state number** onto the parsing stack after each push of **a symbol**
- Let s be the current state. Then actions are
  1. if state s contains any item of the form **A -> ůX** (X is a terminal). Then the action is to shift the current input token onto the stack
  2. If state s contains any **complete item** (an item of the form **A->ů**), then the action is to reduce by the rule **A->ů**
     - A **reduction** by the rule **S'->S** where S' is the start state
     - **acceptance** if the input is empty
     - **Error** if the input is not empty
- A grammar is **LR(0)** grammar if the above rules are unambiguous
- A grammar is **LR(0)** iff
  - Each state is a shift state
  - A reduce state containing a single complete item