Syntax Analysis - LR(1) and LALR(1) Parsing

66.648 Compiler Design Lecture (02/9/98)

Computer Science

Rensselaer Polytechnic

Lecture Outline

- LR(1)Parsing Algorithm
- Examples
- LALR(1) Parsing Algorithm
- Administration

Example

1) S --> P

2) P --> a P a

3) P --> b P b

4) P --> epsilon

As we have seen (did we see) in the last class,

this grammar leads to shift/reduce conflicts in LR(0) grammar.

Yet Another Example

Consider a grammar to generate all nested parentheses

1) S--> P

2) P --> (P)

3) P --> epsilon

In canonical state consisting of items { [S-->.P],[P-->.(P)],[P-->,]} there will be shift reduce conflict.

can you find other canonical states in which shift/reduce conflict occurs.

LR(1) Item

An LR(1) item has the form [A--> alpha . beta, a], where a is the lookahead of the item and it is a terminal symbol (including a $).

LR(1) parser uses the lookahead to improve the precision in invoking the reduce operation.

An LR(1) item [A--> alpha.,a] invokes a reduce action only when the next input symbol is a.

How do we define closure of an item?

Closure of LR(1) item

Let I be the set of LR(1) items. Then, closure(I) is the set of items that can be constructed from I as follows:

1. Every Item in I is also an item in closure(I)
2. If [A--> alpha . B beta,a] is in I and B--> gamma is a production, then add item [B--> .gamma,b] to closure(I),if it is not already a member.

What is b?

b is the first(beta a).

FIRST - revisited

In the example grammar 1,

first(P) = { a,b,epsilon}, First(S)={a,b,epsilon,$)

In the example grammar 2,

first(X) = { (,epsilon} = first(S)

first of terminal symbols can be defined easily. e.g., first(( ) = (

first(X1…Xk) can also be defined easily

Items and Closure Contd

Example 1:

closure of item [S--> .P,$]

[S --> .P, $], [P-->.a P a,$], [P-->.b P b, $], [P-->.,$]

Example 2:

closure of item [X--> (. X), $]

[X --> (. X), $], [X--> .(X),)], [X --> .( ), ()]

GOTO operation

Let I be a set of items, and X be a grammar symbol (terminal/nonterminal). Then

GOTO(I,X) =

closure ({ [A--> alpha X . beta, a] | [A --> alpha . X beta, a] is in I)

Canonical set of LR(1) items

This is similar to LR(0) case.

Enumerate possible states of LR(1) parser. Each state is a canonical set of LR(1) items.

Canonical States

1) Start with the canonical set by performing a closure ([S'--> .S, $]).

2) If I is a canonical set and X is a grammar symbol such that I' = goto(I,X) is nonempty, then make I' a new canonical set. Repeat until no more canonical sets can be added.

Example 1

state 0: { [S--> .P, $], [P--> . a P a, $], [P--> .b P b , $], [P--> .,$] }

state 1: { [S --> P.,$] }

state 2: { [P--> a . P a,$], [P --> . a P a, a], [P --> .b P b, a], [P --> ., a] }

state 3: { [P --> b. P b,$], [P --> .a P a , b], [P--> .b P b, b], [P--> .,b] }

state 4:{ [P --> a P .a, $]}

state 5: { [P --> a . P a, a], [P --> .a P a, a], [P--> .b P b, a], [P--> ., a] }

Example 1 - Contd

Enumerate the rest of the states …

Example 2

S0: { [S--> . X, $], [X --> . (X), $], [X--> ., $]}

S1: { [S--> X.,$]}

S2: { [X --> (. X),$], [X--> . (X),)], [X--> .,)]

S3: { [X --> (X .), $] }

S4: { [X --> (. X),)], [X--> . (X) ,)], [X--> .,)] }

S5:{ [X --> (X). $] }

S6: { [X --> (X .),)] }

S7: { [X --> (X). ,)] }

Finite State Machine

Draw the FSA. The major difference is that transitions can be both terminal and nonterminal symbols.

Actions Associated with LR(1) States

If a state contains an item of the form [A--> beta . a], then state prompts a reduce action provided the next input symbol is a.

If a state contains A--> [alpha . a delta, b] then the state prompts the parser to perform a shift action when the input symbol is a.

If a state contains [S'--> S.,$] and there are no more input symbols left, then the parser is prompted to accept.

Else an error message is prompted.

Parsing Table

state	Input symbol			goto
	(	)	$	X

Parsing Table Contd

si means shift the input symbol and goto state I.

rj means reduce by jth production. Note that we are not storing all the items in the state in our table.

example: ((( ))) $

LR(1) Grammars

A Grammar is said to be LR(1) if there is no conflict present in any of its LR(1) canonical sets I.e. if no state prompts the parser to perform more than one action on some input symbol.

Most programming languages can be described by LR(1), but involves a large number of states.

The number of states can be reduced by appropriately merging certain states. This is what is done in LALR grammar (in YACC)

LALR Grammar

LA LR - Look Ahead LR grammar,

Core of a LR(1) Canonical set

Th core of an LR(1) canonical set is the set of first part of all the items in that canonical set.

e.g. in S2 = {[X--> (.X),$],[X-->.(X),)],[X-->.,)]

has cores ={ X--> (.X),X-->.(X), X-->.}

in S4 = {[X-->(.X),)], [X--> .(X),)],[X-->.,)] have the same cores as above

LALR(1) PARSING TABLE

Basic LALR(1) Parsing: Merge LR(1) states with the same core:

Comments and Feedback

Project 2 is out.

Project 1 is due to-night.

Please keep reading chapter 4 and understand the material. Work out as many exercises as you can.