THEORY OF COMPUTATION AND COMPILERS

Unit - II

CONTEXT FREE GRAMMARS AND PARSING

- Introduction
- Context-Free Grammars - Derivation, Parse trees, Ambiguity
- Types of Parsers
- LL(K) grammars and LL(1) parsing
- Bottom-up Parsing - handle pruning
- **LR Grammar Parsing** 
- LALR parsing
- Parsing ambiguous grammars
- Error Recovery in Parsing
- YACC programming specification

Dr. R. Madana Mohana
Professor, Artificial Intelligence & Data Science | I/c-Head, Artificial Intelligence & Machine Learning
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
Hyderabad - 500 075, Telangana, INDIA
www.cbit.ac.in

Unit-II: Syntax Analysis (or) Parser
Canonical LR-Parser (CLR Parser)

Outline:

- More Powerful LR Parsers
- Canonical LR Parser (CLR Parser)
- Canonical LR(1) Items & DFA

More Powerful LR Parsers

We shall extend the previous **LR** parsing techniques to use one symbol of lookahead on the input. There are *two different methods*:

1. The " **canonical-LR**" or just " **LR**" method, which makes full use of the lookahead symbol(s). This method uses a large set of items, called the **LR(1)** items.
2. The " **lookahead-LR**" or " **LALR**" method, which is based on the **LR(0)** sets of items, and has many fewer states than typical parsers based on the **LR(1)** items.

More Powerful LR Parsers

2. By carefully introducing lookaheads into the **LR(0)** items, we can handle many more grammars with the **LALR** method than with the **SLR** method, and build parsing tables that are no bigger than the **SLR** tables. **LALR** is the method of choice in most situations.

Canonical LR Parser (CLR Parser)

- The **Canonical LR** parser is obtained first by constructing **LR(1)** items.
- Carry extra information in the state so that wrong reductions by **A → α** will be ruled out.
- Redefine **LR** items to include a terminal symbol as a second component (look ahead symbol).
- The general form of the item becomes [**A → α.β,a**] which is called **LR(1)** item.

Canonical LR Parser (CLR Parser)

- Item [**A → α.,a**] calls for reduction only if next input is **a**. The set of symbols.
- **Canonical LR** parsers solve this problem by storing extra information in the state itself. The problem we have with **SLR** parsers is because it does reduction even for those symbols of **Follow(A)** for which it is invalid. So **LR** items are redefined to store **1 terminal** (**look ahead symbol**) along with state and thus, the items now are **LR(1)** items.

Canonical LR(1) Items

Definition:

An item of the form [**A → α.β,a**] where **A → αβ** is a production and '**a**' is a terminal or **$** (the right end marker) is called **LR(1)** item. Here, **1** is lookahead of the item.

Observe the following points:

- In an item of the form [**A → α.β,a**] where **β** is not **ϵ**, the lookahead has no effect.
- The item of the form [**A → α.,a**] calls for the reduction using the production **A → α** if and only if the next input symbol is '**a**'.

Canonical LR(1) Items

CLOSURE(I):

If **I** is a set of items for a grammar **G**, then **CLOSURE(I)** is constructed using the following *two rules*:

Rule 1: Initially, add every item in **I** to **CLOSURE(I)**

Rule 2: If [**A → α.Bβ,a**] is in **CLOSURE(I)** and **B → γ** is a production then, for each terminal '**b**' in **FIRST(βa)**, we add [**B → .γ, b**] to **CLOSURE(I)**.

Rule 3: Apply rule 2 repeatedly until no more new items are added to **CLOSURE(I)**.

Canonical LR(1) Items

CLOSURE(I):

Example: Consider the following augmented grammar and **FIRST** sets

1. S′ → S
2. S → CC
3. C → cC
4. C → d

	S	C
FIRST	c, d	c, d

Find CLOSUR([S′ → .S, $])

Canonical LR(1) Items

CLOSURE(I):

Example: *Solution*

The **CLOSUR**([**S′ → .S, $**]) is computed by adding itself into **CLOSUR** in the beginning as shown below:

CLOSUR([**S′ → .S, $**]) = **{**

S′ → .S, $ **A → α.Bβ,a**	After dot(.), **S** is present. So, add **S → CC** and put dot(.) at the beginning. Here, **β = ϵ, a = $, FIRST(βa) = FIRST($) = $**

Canonical LR(1) Items

CLOSURE(I):

Example: *Solution*

CLOSUR([S′ → .S, $]) = {

S → .CC, $ A → α.Bβ,a	After dot(.), C is present. So, add C → cC | d and put dot(.) at the beginning. Here, β = C, a = $, FIRST(βa) = FIRST(Ca) = {c, d}
C → .cC, c | d C → .d, c | d	After dot(.), terminal is present. So stop at this point.

}

Canonical LR(1) Items

Algorithm to compute CLOSURE(I):

Function **CLOSURE**(**I**)

{

repeat

for each item [**A → α.Bβ, a]** is in **I** and for each production **B → γ** in **G**

for each terminal **b** in **FIRST(βa)**

add [**B → .γ, b]** to **I** if not present earlier

until no more new items added to **I**

return **I**

}

Canonical LR(1) Items

GOTO function:

The **GOTO** (**I, X**) is a function which accepts set of items **I** as the *first argument* and **X** which is a *grammar symbol* (it can be either a terminal or a non-terminal) as the *second argument*. It is formally defined as shown below:

If [**A → α.Xβ, a]** in **I**

GOTO (**I, X**) **= CLOSURE** (**A → αX.β, a**)

The GOTO() function is used to define the transition from one set of items to other set of items.

Canonical LR(1) Items

Algorithm to find GOTO (I, X):

Function **GOTO** (**I, X**)

{

for each item **([A → α.Xβ, a]** is in **I)**

return **CLOSURE** ([**A → αX.β, a]**)

}

Canonical LR(1) Items

GOTO function:

Example: Consider the following augmented grammar and **FIRST** sets

1. S′ → S
2. S → CC
3. C → cC
4. C → d

	S	C
FIRST	c, d	c, d

Find GOTO([S → .CC, $], C)

Canonical LR(1) Items

GOTO function:

Example: *Solution*

It is given that **`I = [S → .CC, $]`**. Observe that immediately after dot (**.**) we have a variable or non-terminal **`C`**. So, to compute **`GOTO (I, C)`** use **`[S → .CC, $]`** and shift dot (**.**) to next position and compute **`CLOSURE`** as shown below:

Canonical LR(1) Items

GOTO function:

Example: *Solution*

CLOSUR([**S → .CC, $**]) = **{**

S → .CC, $ **A → α.Bβ,a**	After dot(.), **C** is present. So, add **C → cC | d** and put dot(.)at the beginning. Here, **β = C**, **a = $**, **FIRST(βa) = FIRST(Ca) = {c, d}**
C → .cC, c | d **C → .d, c | d**	After dot(.), **terminal** is present. So stop at this point.

}

Canonical LR(1) Items

Canonical collection of sets of LR(1) items:

Function **ITEMS** (**G′**)

{

C = CLOSURE ([**S′ → .S, $**])

repeat

for each set of items **I** in **C**

for each grammar symbol **X**

if (**GOTO (I, X)** is not empty and not in **C**

add **GOTO (I, X)** to **C**

until no more sets of items are added to **C**;

return **C**

}

Canonical LR(1) Items

Example:

Consider the following augmented grammar:

1. S′ → S
2. S → CC
3. C → cC
4. C → d

Obtain **LR(1)** items (or) compute canonical collection of sets of **LR(1)** items.

Canonical LR(1) Items

Example: ***Solution***

The **LR(1)** items can be obtained by taking the ([**S′ → .S, $**])

I_0:
S′ → .S, $
S → .CC, $
C → .cC, c | d
C → .d, c | d

I_1: GOTO (I_0, S)
S′ → S., $
I_2: GOTO (I_0, C)
S → C.C, $
C → .cC, $
C → .d, $

Canonical LR(1) Items

Example: ***Solution***

I_3: GOTO (I_0, c)
C → c.C, c | d
C → .cC, c | d
C → .d, c | d

I_4: GOTO (I_0, d)
C → d., c | d

I_5: GOTO (I_2, C)
S → CC., $

I_6: GOTO (I_2, c)
C → c.C, $
C → .cC, $
C → .d, $

I_7: GOTO (I_2, d)
C → d., $

I_8: GOTO (I_3, C)
C → cC., c | d

Canonical LR(1) Items

Example: ***Solution***

GOTO (I_3, c) = I_3

C → c.C, c | d

C → .cC, c | d

C → .d, c | d

GOTO (I_3, d) = I_4

C → d., c | d

I_9: GOTO (I_6, C)

C → cC., $

GOTO (I_6, c) = I_6

C → c.C, $

C → .cC, $

C → .d, $

GOTO (I_6, d) = I_7

C → d., $

Canonical LR(1) Items

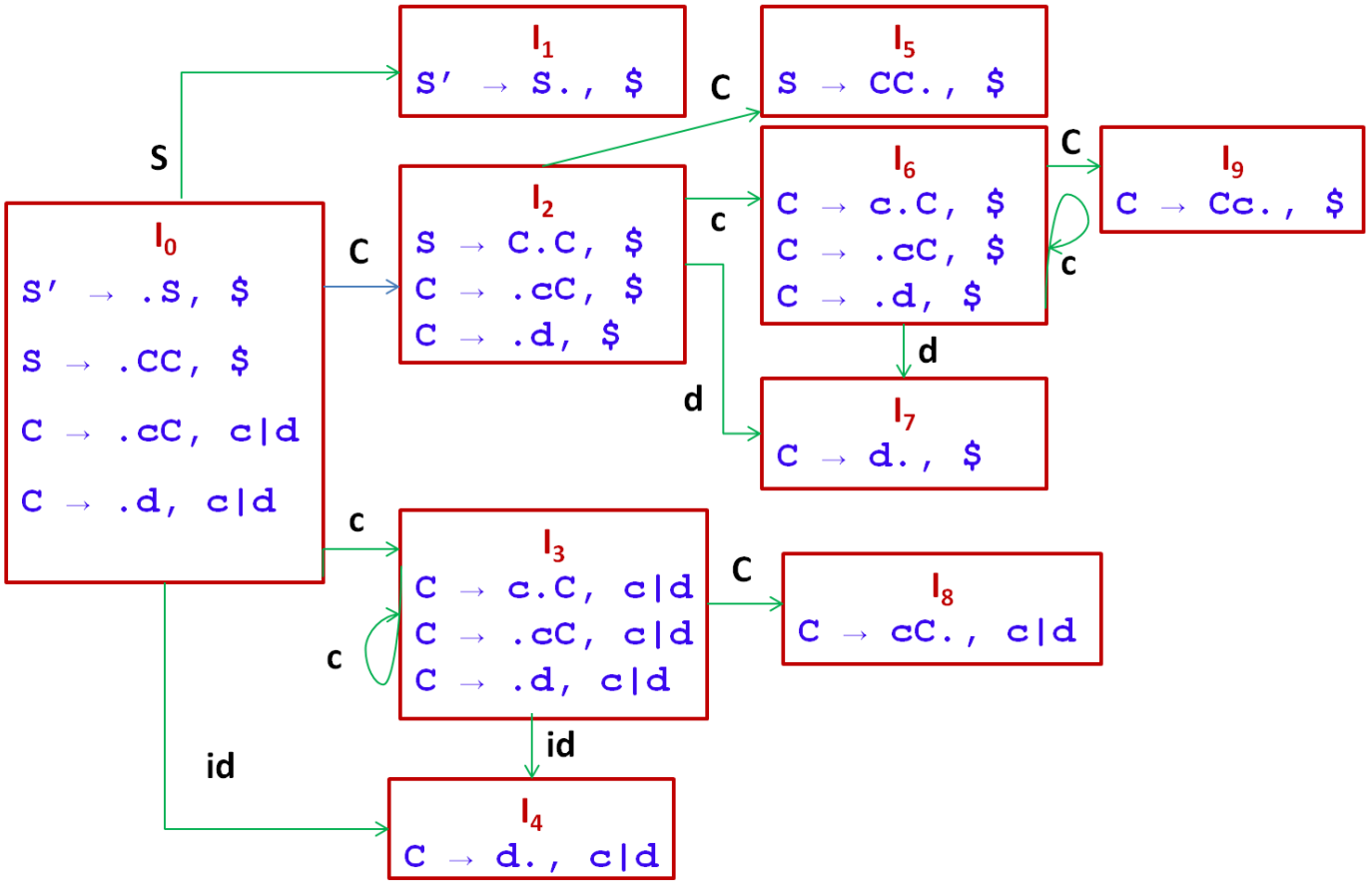

Fig: DFA for LR(1) items

Summary...

Bottom-Up Parsing: CLR-Parser

- More Powerful LR Parsers
- Canonical LR Parser (CLR Parser)
- Canonical LR(1) Items & DFA

Reading: Aho2, Section 4.7.1 & 4.7.2

Next Lecture: Constructing CLR Parsing Table