USN | | | | | | | | | |

CMRIT

Internal Assessment Test 2 – May 2023

| Sub: | SYSTEM SOFTWARE AND COMPILERS | | | | | Sub Code: | 18CS61 | Branch: | CSE |
|---|---|---|---|---|---|---|---|---|---|
| Date: | 23.05.2023 | Duration: | 90 mins | Max Marks: | 50 | Sem / Sec: | VI/ A,B,C | | OBE |

| | Answer any FIVE FULL Questions | MARKS | CO | RBT |
|---|---|---|---|---|
| 1 | Write the algorithm to calculate FIRST and FOLLOW. Calculate FIRST and FOLLOW for the non-terminals present in the given grammar.<br>S → aBDh<br>B → cC<br>C → bC / ∈<br>D → EF<br>E → g / ∈<br>F → f / ∈<br><br>**Algorithm for calculation of FIRST-------------------------------------2M**<br>**Algorithm for calculation of FOLLOW--------------------------------2M**<br>**Calculating FIRST for the given grammar------------------------------3M**<br>**Calculating FOLLOW for the given grammar--------------------------3M** | [10] | CO2 | L3 |
| 2 | Construct a predictive parsing table for the following grammar. Show the parsing of the input string: **((a,a)).** Is it LL (1)?<br>**S → (L) / a**<br><br>**L → L , S / S**<br><br>**Construction of Predictive Parsing Table---------------------5 M**<br>**Parsing of the input string----------------------------------------4.5M**<br>**Is it LL(1)-----------------------------------------------------------0.5M** | [10] | CO2 | L3 |
| 3 | i) What is meant by handle and handle processing (handle pruning)? Explain with an example.<br><br>**Handle and handle pruning definition----------------2M**<br>**Example--------------------------------------------------3M**<br>ii) Explain the role of parser and different error recovery strategies.<br><br>**Role of Parser--------------------------2M**<br>**Different error recovery strategies-------------3M** | [5+5] | CO2 | L2 |
| 4 | What are the different types of conflicts in Shift Reduce Parser? Show the parsing of the input string **(id+id*id)/id** by the shift reduce parser and recognize the conflicts while parsing the input string. Consider the following grammar.<br>E→E+T \| E-T \| T<br>T→T*F \| T/F \| F<br>F→ ( E ) \| id<br>**Different types of conflicts in Shift Reduce Parser--------------2M**<br>**Parsing of the input string-----------------------------------------------6M**<br>**Recognize the conflicts while parsing the input string----------2M** | [10] | CO2 | L3 |

| | | | | |
|---|---|---|---|---|
| 5 | Write a grammar and SDD for simple Desk Calculator and show the annotated parser tree for expression **(8+7) / (2+3)**<br><br>**Grammar for desk calculator-----------------------------------2M**<br>**SDD----------------------------------------------------------------3M**<br>**Annotated Parse Tree--------------------------------------------5M** | [10] | CO2 | L3 |
| 6 | i) What is left factoring? Write the following grammar after left factored.<br>S → bSSaaS / bSSaSb / bSb / a<br><br>**Left Factoring definition-----------------------------------2M**<br>**Showing the left factored grammar-----------------------3M**<br>ii) Discuss S-attributed and L-attributed SDD.<br><br>**S-attributed SDD with example----------------------------------2.5M**<br>**L-attributed SDD with example----------------------------------2.5M** | [5+5] | CO2 | L3 |

CI CCI HOD

USN | | | | | | | | | |

Internal Assessment Test 2 – May 2023

| Sub: | SYSTEM SOFTWARE AND COMPILERS | | | | Sub Code: | 18CS61 | Branch: | CSE |
|---|---|---|---|---|---|---|---|---|
| Date: | 23.05.2023 | Duration: | 90 mins | Max Marks: 50 | Sem / Sec: | VI/ A,B,C | | OBE |

Answer any FIVE FULL Questions

| | | MARKS | CO | RBT |
|---|---|---|---|---|
| 1 | Write the algorithm to calculate FIRST and FOLLOW. Calculate FIRST and FOLLOW for the non-terminals present in the given grammar. | [10] | CO2 | L3 |

S → aBDh
B → cC
C → bC / ∈
D → EF
E → g / ∈
F → f / ∈

**Algorithm to calculate FIRST:**

FIRST (α) is defined as the collection of terminal symbols which are the first letters of strings derived from α.

**If X is Grammar Symbol, then First (X) will be −**

- If X is a terminal symbol, then FIRST(X) = {X}
- If X → ε, then FIRST(X) = {ε}
- If X is non-terminal & X → a α, then FIRST (X) = {a}
- If X → $Y_1$, $Y_2$, $Y_3$, then FIRST (X) will be

(a) If Y is terminal, then

FIRST (X) = FIRST ($Y_1$, $Y_2$, $Y_3$) = {$Y_1$}

(b) If $Y_1$ is Non-terminal and

If $Y_1$ does not derive to an empty string i.e., If FIRST ($Y_1$) does not contain ε then, FIRST (X) = FIRST ($Y_1$, $Y_2$, $Y_3$) = FIRST($Y_1$)

(c) If FIRST ($Y_1$) contains ε, then.

FIRST (X) = FIRST ($Y_1$, $Y_2$, $Y_3$) = FIRST($Y_1$) − {ε} ∪ FIRST($Y_2$, $Y_3$)

Similarly, FIRST ($Y_2$, $Y_3$) = {$Y_2$}, If $Y_2$ is terminal otherwise if $Y_2$ is Non-terminal then

- FIRST ($Y_2$, $Y_3$) = FIRST ($Y_2$), if FIRST ($Y_2$) does not contain ε.
- If FIRST ($Y_2$) contain ε, then
- FIRST ($Y_2$, $Y_3$) = FIRST ($Y_2$) − {ε} ∪ FIRST ($Y_3$)

Similarly, this method will be repeated for further Grammar symbols, i.e., for $Y_4$, $Y_5$, $Y_6$ … . $Y_K$

**Algorithm to calculate FOLLOW:**

**Follow (A) is defined as the collection of terminal symbols that occur directly to the right of A.**

FOLLOW(A) = {a|S ⇒* αAaβ where α, β can be any strings}

**Rules to find FOLLOW**

- If S is the start symbol, FOLLOW (S) ={$}
- If production is of form A → α B β, β ≠ ε.

(a) If FIRST (β) does not contain ε then, FOLLOW (B) = {FIRST (β)}

Or

(b) If FIRST (β) contains ε (i. e. , β ⇒* ε), then

FOLLOW (B) = FIRST (β) − {ε} ∪ FOLLOW (A)

∵ when β derives ε, then terminal after A will follow B.

- If production is of form A → αB, then Follow (B) ={FOLLOW (A)}.

**Computation of FIRST and FOLLOW for the following grammar.**

S → aBDh
B → cC
C → bC / ∈
D → EF
E → g / ∈
F → f / ∈

FIRST (S) = {a}
FIRST (B) = {c}
FIRST (C) = {b, ∈}
FIRST (D) = {f, g, ∈}
FIRST (E) = {g, ∈}
FIRST (F) = {f, ∈}
FOLLOW(S) = {$}
FOLLOW(B) = {f, g, h}
FOLLOW(D) = {h}
FOLLOW(C) = {f, g, h}
FOLLOW(E) = {f, h}
FOLLOW(F) = {h}

2 Construct a predictive parsing table for the following grammar. Show the parsing of the input string: **((a,a)).** Is it LL (1)? [10] CO2 L3

**S → (L) / a**

**L → L , S / S**

**Step1: After removing left recursion**

**S → (L) / a**
**L → SL'**
**L' → ,SL' | Ɛ**

**Step2: Calculate FIRST and FOLLOW**

FIRST(S) = {(, a}
FIRST (L)= { ( , a}
FIRST (L')={**,** , Ɛ}
FOLLOW (S) = { $, **,** , ) }
FOLLOW (L) ={ )} = Follow (L')

Step 3: Predictive Parsing Table

| | ( | ) | , | a | $ |
|---|---|---|---|---|---|
| S | **S → (L)** | | | **S → a** | |

| | | | | | |
|---|---|---|---|---|---|
| L | **L → SL'** | | | **L → SL'** | |
| L' | | **L' → Ɛ** | **L'→ ,SL'** | | |

**Yes, it is LL(1).**

Step4: Parsing of the input string **((a,a))**

| Stack | Input String | Action |
|---|---|---|
| $ S | **((a,a))$** | **S → (L)** |
| $ )L( | **((a,a))$** | Match |
| $ )L | **(a,a))$** | **L → SL'** |
| $ )L'S | **(a,a))$** | **S → (L)** |
| $ )L')L( | **(a,a))$** | Match |
| $ )L')L | **a,a))$** | **L → SL'** |
| $ )L')L'S | **a,a))$** | **S → a** |
| $ )L')L'a | **a,a))$** | Match |
| $ )L')L' | **,a))$** | **L'→ ,SL'** |
| $ )L')L'S, | **,a))$** | Match |
| $ )L')L'S | **a))$** | **S → a** |
| $ )L')L'a | **a))$** | Match |
| $ )L')L' | **))$** | **L' → Ɛ** |
| $ )L') | **))$** | **Match** |
| $ )L' | **)$** | **L' → Ɛ** |
| $ ) | **)$** | **Match** |
| $ | **$** | **Accepted** |

3 i) What is meant by handle and handle processing (handle pruning)? Explain with an example. [5+5] CO2 L2

**Solution:**

A **handle** is a substring that connects a right-hand side of the production rule in the grammar and whose reduction to the non-terminal on the left-hand side of that grammar rule is a step along with the reverse of a rightmost derivation.

Removing the children of the left-hand side non-terminal from the parse tree is called **Handle Pruning**. A rightmost derivation in reverse can be obtained by handle pruning.

| Right Sequential Form | Handle | Reducing Production |
|---|---|---|
| id + id * id | id | E ⇒ id |
| E + id * id | id | E ⇒ id |

| | | |
|---|---|---|
| E + E * id | id | E ⇒ id |
| E + E * E | E + E | E ⇒ E + E |
| E * E | E * E | E ⇒ E * E |
| E (Root) | | |

ii) Explain the role of parser and different error recovery strategies.

**Role of Parser:**

- It **obtains a string of tokens** from the **lexical analyser**
- **verifies that the string** can be generated by the **grammar for the source language**.
- The **parser** returns **any syntax error** for the source language
- It detects and reports any **syntax errors** and produces a **parse tree** from which **intermediate code** can be generated.

**Different error recovery strategies:**

There are mainly five error recovery strategies, which are as follows:

1. Panic mode
2. Phrase level recovery
3. Error production
4. Global correction
5. Symbol table

**Panic Mode:**

This strategy is used by most parsing methods. In this method of discovering the error, the parser **discards input symbols one at a time.** This process is continued until one of the designated sets of synchronizing tokens is found. Synchronizing tokens are delimiters such as **semicolons or ends.** These tokens indicate an end of the input statement.

**Phrase Level Recovery:**

In this strategy, on discovering an error, parser performs local correction on the remaining input. It can replace a prefix of the remaining input with some string. This actually helps the parser to continue its job. The local correction can be replacing the comma with semicolons, omission of semicolons, or, fitting missing semicolons.

**Error Production:**

It requires good knowledge of common errors that might get encountered, then we can augment the grammar for the corresponding language with **error productions**

**that generate the erroneous constructs.** If error production is used during parsing, we can generate an appropriate error message to indicate the error that has been recognized in the input.

**Global Correction:**

We often want such a compiler that makes very few changes in processing an incorrect input string to the correct input string. Given an incorrect input string x and grammar G**,** the algorithm itself can find a parse tree for a related string y (Expected output string); such that a number of insertions, deletions, and changes of token require to transform x into y is as low as possible. Global correction methods increase time & space requirements at parsing time. This is simply a theoretical concept.

**Symbol Table:**

In semantic errors, errors are recovered by using a symbol table for the corresponding identifier and if data types of two operands are not compatible, automatically type conversion is done by the compiler.

4 What are the different types of conflicts in Shift Reduce Parser? Show the parsing of the input string **(id+id*id)/id** by the shift reduce parser and recognize the conflicts while parsing the input string. Consider the following grammar. [10] CO2 L3

E→E+T | E-T | T

T→T*F | T/F | F

F→ ( E ) | id

**Solution:**

Different types of conflicts in shift reduce parser:

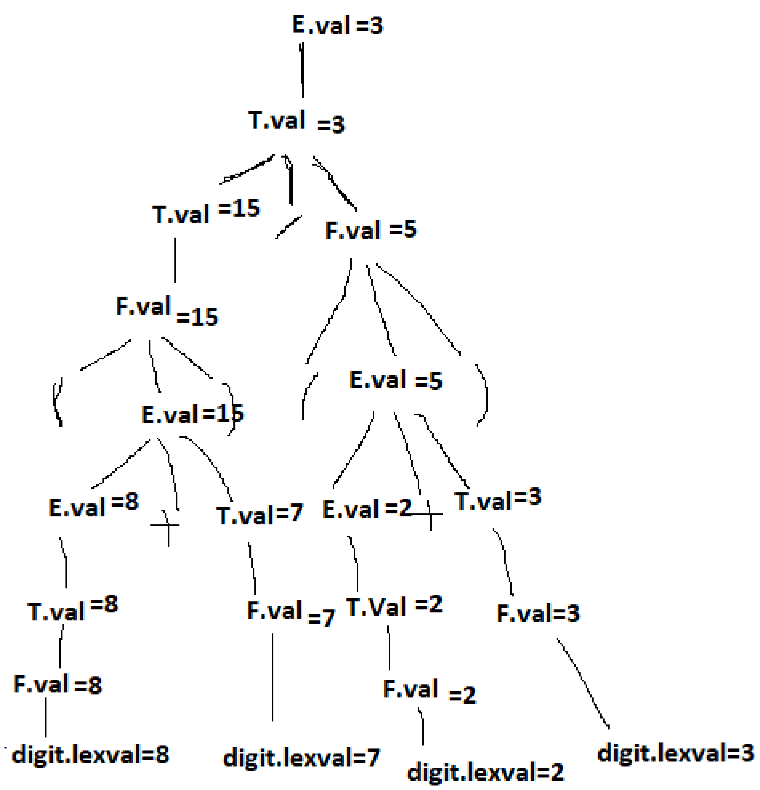
1. Shift Reduce conflicts
2. Reduce Reduce conflicts

| Stack | Input | Action |
|---|---|---|
| $ | **(id+id*id)/id$** | Shift |
| $( | **id+id*id)/id$** | Shift |
| $(id | **+id*id)/id$** | Reduce F→**id** |
| $(F | **+id*id)/id$** | Reduce T→**F** |
| $(T | **+id*id)/id$** | Reduce E→**T** |
| $(E | **+id*id)/id$** | Shift |
| $(E+ | **id*id)/id$** | Shift |
| $(E+id | ***id)/id$** | Reduce F→**id** |
| $(E+F | ***id)/id$** | Reduce T→**F** |
| $(E+T | ***id)/id$** | Shift |
| $(E+T* | **id)/id$** | Shift |
| $(E+T*id | **)/id$** | Reduce F→**id** |
| $(E+T*F | **)/id$** | Reduce T→**T*F** |
| $(E+T | **)/id$** | Reduce E→**E+T** |
| $(E | **)/id$** | Shift |

| | | |
|---|---|---|
| $(E) | **/id$** | Reduce F→**( E )** |
| $F | **/id$** | Reduce T→**F** |
| $T | **/id$** | Shift |
| $T/ | **id$** | Shift |
| $T/id | **$** | Reduce F→**id** |
| $T/F | **$** | Reduce T→**T/F** |
| $T | **$** | Reduce E→**T** |
| $E | **$** | Accepted |

5 Write a grammar and SDD for simple Desk Calculator and show the annotated parser tree for expression **(8+7) / (2+3)** [10] CO2 L3

| Production | Semantic Actions |
|---|---|
| S --> E | Print(E.val) |
| E --> $E_1$ + T | E.val = $E_1$.val + T.val |
| E --> T | E.val = T.val |
| T --> $T_1$ * F | T.val = $T_1$.val * F.val |
| T --> F | T.val = F.val |
| F --> digit | F.val = digit.lexval |




6 i) What is left factoring? Write the following grammar after left factored. [5+5] CO2 L3

S → bSSaaS / bSSaSb / bSb / a

Solution:
Left factoring is a process by which the grammar with common prefixes is transformed to make it useful for Top-down parsers. If more than one grammar production rules has a common prefix string, then the top-down parser cannot make a choice as to which of the production it should take to parse the string in hand.

Step1:

S → bSS' / a
S'→SaaS |SaSb|b

Step2:
S → bSS' / a
S'→SaS'' |b
S''→aS | Sb

ii) Discuss S-attributed and L-attributed SDD.

**S-attributed SDT :**

a. If an SDT uses only synthesized attributes, it is called as S-attributed SDT.
b. S-attributed SDTs are evaluated in bottom-up parsing, as the values of the parent nodes depend upon the values of the child nodes.
c. Semantic actions are placed in rightmost place of RHS.

**L-attributed SDT:**

d. If an SDT uses both synthesized attributes and inherited attributes with a restriction that inherited attribute can inherit values from left siblings only, it is called as L-attributed SDT.
e. Attributes in L-attributed SDTs are evaluated by depth-first and left-to-right parsing manner.
f. Semantic actions are placed anywhere in RHS.
g. Example : S->ABC, Here attribute B can only obtain its value either from the parent – S or its left sibling A but It can't inherit from its right sibling C. Same goes for A & C – A can only get its value from its parent & C can get its value from S, A, & B as well because C is the rightmost attribute in the given production.

CI CCI HOD