00012

THE OPEN UNIVERSITY OF SRI LANKA
FACULTY OF ENGINEERING TECHNOLOGY
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
BACHELOR OF TECHNOLOGY
ECX6235 – COMPILER DESIGN

Date: 08 September 2015 **Time: 0930 – 1230 hrs.**

Important:

1. This question paper consists of **five** questions.
2. Answer **all** questions in **Part A** (55 marks) and **Three** questions from **Part B** (45 marks).
3. Clearly state your assumptions, if any.

Part A – Answer all questions

Refer the following article in page 3 & 4 to answer the question Q1. Clearly state your assumptions.
Jayeeta Chanda, Sabnam Sengupta, Ananya Kanjilal, and Swapan Bhattacharya. 2010. Formalization of the design phase of software lifecycle: a grammar based approach.

[Q1] A sequence diagram for a part of a design for microwave oven controller is shown in the Figure Q1.

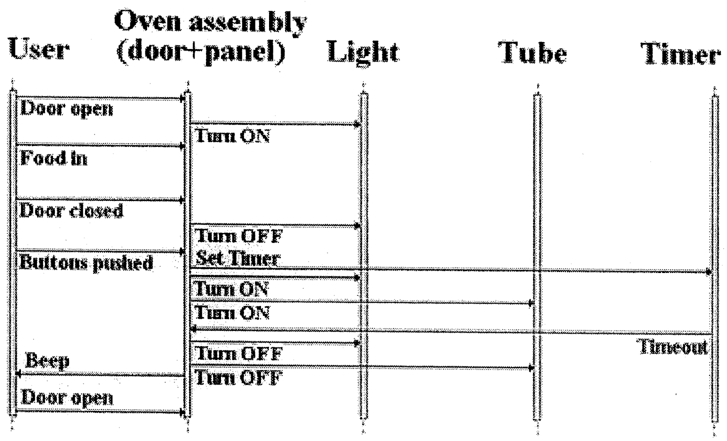

Figure Q1: A sequence diagram for a part of a design for microwave oven controller.

(a) Define the grammar G for the "4.2 Grammar for the Sequence Diagram" on page 3. [05 Marks]

(b) Derive a regular string for the Figure Q1 which is accepted by the grammar G above (a). Clearly show the modifications of the Figure Q1. [25 Marks]

(c) Verify the modified sequence diagram above (b) with "5.1.2.Correctness Rule for Sequence Diagram" on page 4. [18 Marks]

(d) Write LEX implementation syntax for token of the grammar G above (a). [07 Marks]

Part B – Answer only Three questions

[Q2] The FDL grammar rules define as follows (FDL, FDEF, FEXP, FLIST, are non-terminals and others are terminals).

FDL → FDEF FDL | ε
FDEF → #feature #: FEXP
FEXP → #op #(FLIST #)
FLIST → FEXP #, FEXP | #feature

(a) Draw a derivation tree for the following string.
#feature #: #op #(#op #(#feature #)#, #op #(#feature #)#) [03 Marks]

(b) Draw a NFA for the string: {#feature #: #op #(#feature #)}* [04 Marks]

(c) Draw a DFA equivalent to NFA in (b) [08 Marks]

[Q3] Consider the grammar rules given below (DESK, EXPR, CONST, DEFS, DEF are non-terminals and others are terminals).

DESK → print EXPR CONST
EXPR → EXPR + id | id
CONST → where DEFS
DEFS → DEFS DEF | ε
DEF → id = int

(a) Derive the string: print id + id where id = int [02 Marks]

(b) Define the Chomsky Normal Form (CNF) for CFGs. [01 Marks]

(c) Convert the given grammar into CNF. [10 Marks]

(d) Derive the above string in (a) using new grammar in (c) [02 Marks]

[Q4] Consider the grammar rules given below (VEHICLES is a non-terminal and others are terminals).

VEHICLES → car VEHICLES jeep | car jeep

(a) Find the LR(1) sets of items. [06 Marks]

(b) Compute the LR(1) parsing table (Action – Goto) for the corresponding shift-reduce parse engine. [06 Marks]

(c) Show the parsing steps (Input – Action) of the string: car car jeep jeep [03 Marks]

[Q5]

(a) Briefly explains the four types of grammars with applications. [05 Marks]

(b) Draw a diagram and briefly explain the compilation phases by giving examples for each phase. [10 Marks]

attribute→access_specifier data_type attribute_name
| data_type attribute_name

access_specifier→ + | _ | #

data_type→ **void|integer|long|short |date |String |class |double**

attribute_name→ char

method_class→ cname *method_ID* access_specifier data_type method_name (parameter_list)

parameter_list→parameter*

parameter→data_type parameter_name

parameter_name→ **char**

relation→cname multiplicity* cname relationship

relationship→ identifier description type| identifier type

type→ **aggregation** | **association** | **generalization**

identifier →**char**

description → **char**

multiplicity→ **digit .. digit**

char→ [a-z A-Z][a-z A-Z 0-9]+

digit→[0-9 *]

4.2 Grammar for the Sequence Diagram

P: S → sequence_diagram

sequence_diagram → lifeline+

lifeline → object_name focus_of_control *

focus_of_control → focus_ID message+

message → *cname* message_ID time_order message_description source destination

time_order → **digit**$^+$

lifeline_ID → **char**

focus_ID → **char**

message_description → char| method_sequence

source→actor_from | object_from

destination → actor_to | object_to

actor_to → **char**

actor_from → **char**

object_to → object_name

object_from → object_name

object_name → **char :** classname

classname→ **char**

method_sequence → *method_ID* **char**$^+$( )

char → [a-z A-Z 0-9]+

digit → [0 – 9]

4.3 Grammar for the State Chart Diagram

P: S → Statechart

Statechart → object event* state$^+$ transition*

object→ cname attribute_list

attribute_list → attribute*

attribute → attr_name attr_value

attr_name→ **char**

attr_value→ **char** | **digit**

event → cname eventname (parameterlist)

state → statename *cname*

eventname→ **char**

statename→ **char**

parameterlist→ **char**

transition → transition_ID *message_ID* prestate event [guard_condition] action poststate
| transition_ID *message_ID* prestate event poststate

action → *cname* char ( )

prestate→ statename

poststate→statename

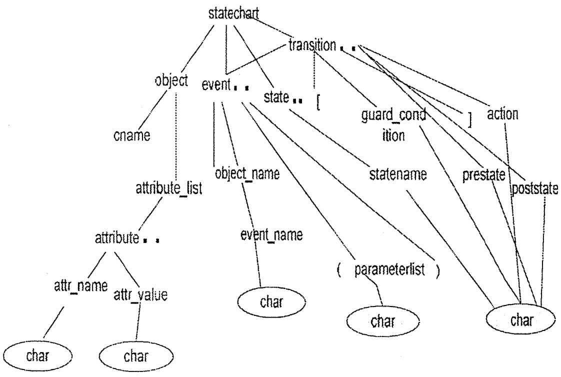

Figure 1: Parse tree for the grammar of state chart diagram

The parse representations for the grammar of statechart are given Figure 1.We can generate the parse tree for class and sequence diagram in the same manner.. From the figures it is clear that all the nonterminals are arriving at the terminal symbols (represented by the leaves of the parse tree) using the set of production rules. The "italicized" non-terminals have been introduced in the production rule to incorporate traceability rules in different elements of UML diagrams used in design phase of SDLC. Even though these non-terminals may not be the intrinsic part of the elements in UML 2.0 specification, they are introduced to ensure requirement traceability and consistency verification of rules using the Proposed Context Free Grammar.

5. VERIFICATION OF PROPOSED RULES

5.1. Verification of Correctness Rule

5.1.1. Correctness Rule for Class Diagram

1. A class diagram must have at least one class.
2. A class diagram may or may not have a relationship (association, generalization, aggregation etc.) between classes.

Rule (1) &(2) can be verified from the following production rules of the grammar

*class_diagram→ class⁺ relation**

class→ name attribute method_ class**

From the above production rule we can derive the regular expression

class_diagram → class1

class_diagram → class1 class2 relation

3. A class must have one and only one name.

class→ name attribute method_class**

From the above production rule we have the regular expression

class → name

4. A class may or may not have one or many attributes and methods.

class→ name attribute method_class**

class→ name

class→ name attribute

class→ name attribute1 ....attributeN

class→ name attribute1 ....attributeN method_class

class→name attribute1.......attributeN

method_class1......method_classN

5. A relation may or may not have multiplicity but it should have relationship.

relation→ multiplicity relationship multiplicity**

relation→relationship

relation→ multiplicity relationship

relation→ multiplicity relationship multiplicity

6. A relationship should have unique id and type and it may or may not have description.

relationship→ identifier description type| identifier type

type→ aggregation | association | generalization

identifier →char

description → char

relationship→ identifier type → id1 aggregation

relationship→ identifier description type → id1 char generalization

Therefore, all the correctness rules for the class diagram can be verified using the proposed UML grammar.

5.1.2.Correctness Rule for Sequence Diagram

1. A sequence diagram must have at least one message.

sequence_diagram → lifeline+

lifeline → object_name focus_of_control +

|object_name *message+*

focus_of_control → focus_ID **message+**

2. A message must have one and only one time order.

message→ *time_order* message_description source destination

3. A message is between one & only source and one &only destination and must have a description. The message description can be a string or a method.

message → time_order *message_description source destination*

message_description→*char|method_sequence*

4.A message must be composed of either one of the following combinations:

- *Two objects*
  source→*object_to*
  destination → |*object_from*
- *One object and one actor*
  source→*actor_to | object_to*
  destination→*actor_from| object_from*
- *Two actors*
  source→*actor_to*
  destination → *actor_from*

5. Sequence diagram have one and only one lifeline and lifeline is uniquely identified by object name.

sequence_diagram → *lifeline*

lifeline → *object_name* focus_of_control +

|*object_name* message+

6. Lifeline have one or many focus of control or atleast one message.

lifeline → object_name focus_of_control +

|object_name *message+*

Therefore, all the correctness rules for the sequence diagram can be verified using the proposed UML grammar

5.1.3. Correctness Rule for State Chart Diagram

1. A state chart diagram consists of one and only one object.
2. A state chart diagram should have at least one state.
3. A state chart diagram consists of zero or more instance of events and transitions.

Rule (1) ,(2) and (3) can be verified by the following production rule of the grammar of the State Chart Diagram.

Statechart → object event* state⁺ transition*

Following regular expressions can be generated from the above production rule

Statechart → object1 state1

Statechart → object2 state1 state2 transition1

Statechart → object3 state1 state2 .....transition 1.... event1.....

Hence rule (1) ,(2) & (3) is verified.

4. An object has a unique identifier (i.e. class name) and a list of attribute.

Rule (4) can be verified by the following production rule