

005

THE OPEN UNIVERSITY OF SRI LANKA
FACULTY OF ENGINEERING TECHNOLOGY
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
BACHELOR OF SOFTWARE ENGINEERING
ECX5265 – SOFTWARE CONSTRUCTION



Date: 08 September 2015

Time: 0930 – 1230 hrs.

Important:

1. This question paper consists of **four** questions.
2. Answer **all** questions in **Part A** (60 marks) and **TWO** questions from **Part B** (40 marks).
3. 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 of the user transactions of a Bank ATM is shown in the Figure Q1 as below. Show the assumptions clearly.

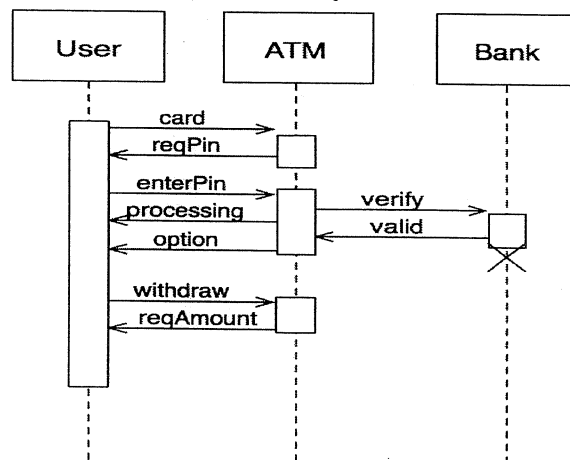


Figure Q1: A Sequence diagram of the user transactions of a Bank ATM

- (a) Define the grammar G for the “4.2 Grammar for the Sequence Diagram” on page 3. [07 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). [10 Marks]

Part B – Answer only Two questions

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

$FDL \rightarrow FDEF \ FDL \mid \epsilon$
 $FDEF \rightarrow \#feature \ \# : FEXP$
 $FEXP \rightarrow \#op \ \#(\ FLIST \ \#)$
 $FLIST \rightarrow FEXP \ \#, FEXP \mid \#feature$

- (a) Draw a derivation tree for the following string.
 $\#feature \ \# : \#op \ \#(\#op \ \#(\#feature \ \#) \#, \#op \ \#(\#feature \ \#) \#)$ [05 Marks]
- (b) Draw a NFA for the string: $\{\#feature \ \# : \#op \ \#(\#feature \ \#) \}^*$ [05 Marks]
- (c) Draw a DFA equivalent to NFA in (b) [10 Marks]
- [Q3] Consider the grammar rules given below (DESK, EXPR, CONST, DEFS, DEF are non-terminals and others are terminals).

$DESK \rightarrow print \ EXPR \ CONST$
 $EXPR \rightarrow EXPR \ + \ id \mid id$
 $CONST \rightarrow where \ DEFS$
 $DEFS \rightarrow DEFS \ DEF \mid \epsilon$
 $DEF \rightarrow id \ = \ int$

- (a) Derive the string: $print \ id \ + \ id \ where \ id \ = \ int$ [02 Marks]
- (b) Define the Chomsky Normal Form (CNF) for CFGs. [02 Marks]
- (c) Convert the given grammar into CNF. [14 Marks]
- (d) Derive the above string in (a) using new grammar in (c) [02 Marks]
- [Q4]
- (a) Briefly explains the four types of grammars with applications. [08 Marks]
- (b) Draw a diagram and briefly explain the compilation phases by giving examples for each phase. [12 Marks]

attribute \rightarrow access_specifier data_type attribute_name
 | data_type attribute_name
 access_specifier \rightarrow + | _ | #
 data_type \rightarrow **void** | **integer** | **long** | **short** | **date** | **String** | **class**
 | **double**
 attribute_name \rightarrow char
 method_class \rightarrow cname method_ID access_specifier data_type
 method_name (parameter_list)
 parameter_list \rightarrow parameter*
 parameter \rightarrow data_type parameter_name
 parameter_name \rightarrow **char**
 relation \rightarrow cname multiplicity* cname relationship
 relationship \rightarrow identifier description type | identifier type
 type \rightarrow **aggregation** | **association** | **generalization**
 identifier \rightarrow **char**
 description \rightarrow **char**
 multiplicity \rightarrow **digit** .. **digit**
 char \rightarrow [a-z A-Z][a-z A-Z 0-9]+
 digit \rightarrow [0-9 *]

4.2 Grammar for the Sequence Diagram

P: S \rightarrow sequence_diagram
 sequence_diagram \rightarrow lifeline+
 lifeline \rightarrow object_name focus_of_control *
 focus_of_control \rightarrow focus_ID message+
 message \rightarrow cname message_ID time_order message_description
 source destination
 time_order \rightarrow **digit**+
 lifeline_ID \rightarrow **char**
 focus_ID \rightarrow **char**
 message_description \rightarrow char | method_sequence
 source \rightarrow actor_from | object_from
 destination \rightarrow actor_to | object_to
 actor_to \rightarrow **char**
 actor_from \rightarrow **char**
 object_to \rightarrow object_name
 object_from \rightarrow object_name
 object_name \rightarrow **char** : classname
 classname \rightarrow **char**
 method_sequence \rightarrow method_ID **char**+ ()
char \rightarrow [a-z A-Z 0-9]+
digit \rightarrow [0-9]

4.3 Grammar for the State Chart Diagram

P: S \rightarrow Statechart

Statechart \rightarrow object event* state* transition*
 object \rightarrow cname attribute_list
 attribute_list \rightarrow attribute*
 attribute \rightarrow attr_name attr_value
 attr_name \rightarrow **char**
 attr_value \rightarrow **char** | **digit**
 event \rightarrow cname eventname (parameterlist)
 state \rightarrow statename cname
 eventname \rightarrow **char**
 statename \rightarrow **char**
 parameterlist \rightarrow **char**
 transition \rightarrow transition_ID message_ID prestate event
 [guard_condition] action poststate
 | transition_ID message_ID prestate event
 poststate
 action \rightarrow cname char ()
 prestate \rightarrow statename
 poststate \rightarrow statename

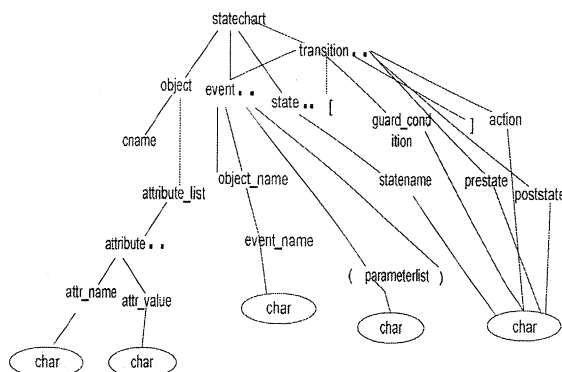


Figure 1: Parse tree for the grammar of state chart diagram

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

$class_diagram \rightarrow class^+ relation^*$

$class \rightarrow name attribute^* method_class^*$

From the above production rule we can derive the regular expression

$class_diagram \rightarrow class1$

$class_diagram \rightarrow class1 class2 relation$

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

$class \rightarrow name attribute^* method_class^*$

From the above production rule we have the regular expression

$class \rightarrow name$

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

$class \rightarrow name attribute^* method_class^*$

$class \rightarrow name$

$class \rightarrow name attribute$

$class \rightarrow name attribute1 attributeN$

$class \rightarrow name attribute1 attributeN method_class$

$class \rightarrow name attribute1 attributeN$

$method_class1 method_classN$

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

$relation \rightarrow multiplicity^* relationship multiplicity^*$

$relation \rightarrow relationship$

$relation \rightarrow multiplicity relationship$

$relation \rightarrow multiplicity relationship multiplicity$

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

$relationship \rightarrow identifier description type | identifier type$

$type \rightarrow aggregation | association | generalization$

$identifier \rightarrow char$

$description \rightarrow char$

$relationship \rightarrow identifier type \rightarrow id1 aggregation$

$relationship \rightarrow identifier description type \rightarrow id1 char generalization$

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.

$sequence_diagram \rightarrow lifeline^+$

$lifeline \rightarrow object_name focus_of_control +$

$| object_name message^+$

$focus_of_control \rightarrow focus_ID message^+$

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

$message \rightarrow time_order message_description source destination$

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.

$message \rightarrow time_order message_description source destination$

$message_description \rightarrow char | method_sequence$

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

- o Two objects

$source \rightarrow object_to$

$destination \rightarrow | object_from$

- o One object and one actor

$source \rightarrow actor_to | object_to$

$destination \rightarrow actor_from | object_from$

- o Two actors

$source \rightarrow actor_to$

$destination \rightarrow actor_from$

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

$sequence_diagram \rightarrow lifeline$

$lifeline \rightarrow object_name focus_of_control +$

$| object_name message^+$

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

$lifeline \rightarrow object_name focus_of_control +$

$| object_name message^+$

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.

$Statechart \rightarrow object event^* state^+ transition^*$

Following regular expressions can be generated from the above production rule

$Statechart \rightarrow object1 state1$

$Statechart \rightarrow object2 state1 state2 transition1$

$Statechart \rightarrow object3 state1 state2 transition1 event1$

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