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

Date: August 13, 2013 **Time: 0930 – 1230 hrs**

Important:

1. This question paper consists of **five** 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 attached article of a case study (*CASE-STUDY: A VIDEO STORE*) in page 3 & 4 to answer the question Q1.

[Q1] The attached article explains a grammar-based approach to validating class diagrams and illustrates this technique using a simple case-study. The article presents a brief description of a video store case study. A video store consists of a video and customer database.

(a) Identify the grammar rules and briefly explain with examples how a cyclical nonterminating relationship in the grammar cause to fail the use case strings to be parsed.

[10 Marks]

(b) Identify the grammar rules and briefly explain with examples how to remove a cyclical nonterminating relationship in the grammar.

[10 Marks]

(c) Define the grammar **G** after removing the cyclical nonterminating relationship.

[10 Marks]

(d) What are the terminals and the non-terminals in this grammar? [05 Marks]

(e) Write LEX implementation syntax for token of the compiler. [10 Marks]

(f) Write an example-string, if you wish to rent a cartoon film and two action films for ten days from the video store.

[05 Marks]

(g) Validate your string using the grammar **G**. [10 Marks]

Part B – Answer only TWO questions

[Q2]

(a) Draw a NFA for the regular expression $(ab)^*(ba \mid a^*)$. [05 Marks]

(b) Convert the NFA obtained in (a) to a DFA. [15 Marks]

[Q3] Consider the grammar.

$$E \rightarrow BA$$

$$A \rightarrow \&BA \mid \varepsilon$$

$$B \rightarrow 1 \mid 0$$

; Where *E, A, B* are non-terminals and others are terminals.

(a) Derive the string: *1 & 0 & 1* [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 your new grammar in (c). [02 Marks]

[04]

(a) Briefly explain the four types of grammars with applications. [10 Marks]

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

[05] A Turing Machine accepts only the strings of the form $0^n1^n2^n$ for (n > 0) and the blank symbol B.

(c) Draw the transition graph. [14 Marks]

(d) List the moves made for the input " 001122 " using instantaneous descriptions. [06 Marks]

End.

the system. The user-supplied strings are tested against the parser generated for the DSL by LISA. Ideally, only all the positive strings should be parsed successfully. If a positive string is not parsed, it indicates that a use case desired by the user can not be represented by the current class diagram. As additional feedback (Stage 5), the approach provides a list of strings that are most similar to the string provided. This metric is calculated using the Levenshtein distance [6]. The Levenshtein distance, also known as the *Edit Distance*, measures the similarity between two strings. The distance is the number of deletions, insertions, or substitutions required to transform a *source* string into a *target* string. The greater the Levenshtein distance, the more dissimilar the strings are. In the next section, we illustrate the validation process using a Video Store case study.

4. CASE-STUDY: A VIDEO STORE

This section presents a brief description of a video store case study. A video store consists of a video and customer database. The video database contains information on all video titles currently on file, and the user (customer) database contains information on all current members of the video store, as well as all videos currently rented by each customer. A customer can walk into the video store and either rent a movie, or become a member of the store. The customer is served by the owner of the store (or an employee). The store owner can also add new titles to the video database. Figure 2 gives the use case diagram for this example.

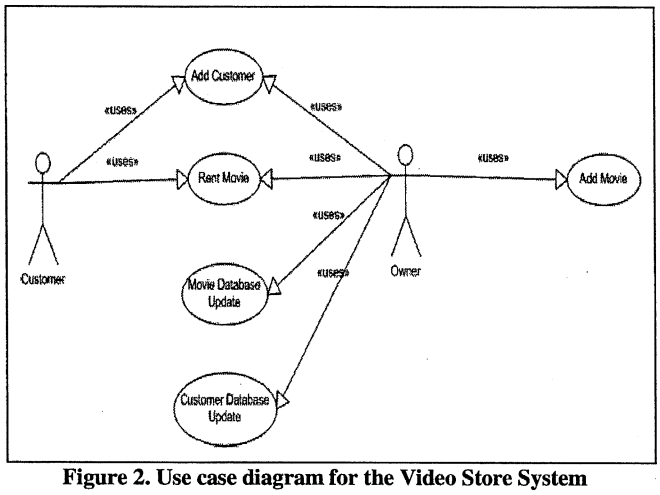

Figure 2. Use case diagram for the Video Store System

In UML, the functionality of the system is represented by use cases that interact with the system actors. In the video store example, the actors are the customer and owner. Each use case can be refined to an activity diagram. Activity diagrams focus on work performed during the activities in a use case instance or in an object. Due to space limitations, we only present the activity diagram for the *rent movie* use case (see Figure 3). The use case and activity diagrams are used by the user in forming the input test cases for the feedback component of the process. The activity diagrams are not used by the automated component of the validation process, but they are used by the user when constructing appropriate use cases while analyzing the CFG. The activity diagram also helps the user better formulate the desired functionality of the system.

4.1 Validating Static Behavior

In UML, class diagrams model the static structure of a system – the classes and their relationships. However, class diagrams do not explain how these structures cooperate to manage their tasks and provide the functionality of the system. The video store system is composed of the classes *VideoStore*, *User* and *Movies* (see Figure 4). The association *Rentals* describes the videos rented by a customer.

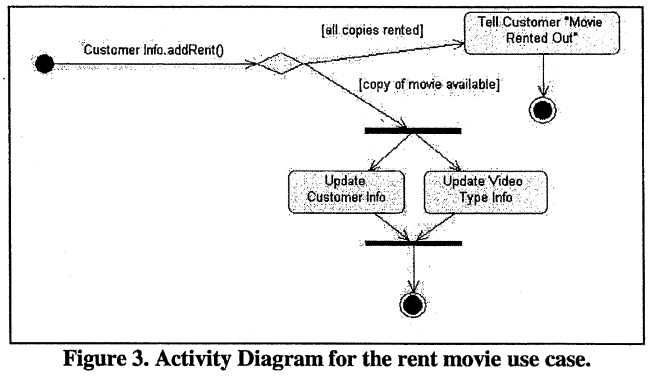

Figure 3. Activity Diagram for the rent movie use case.

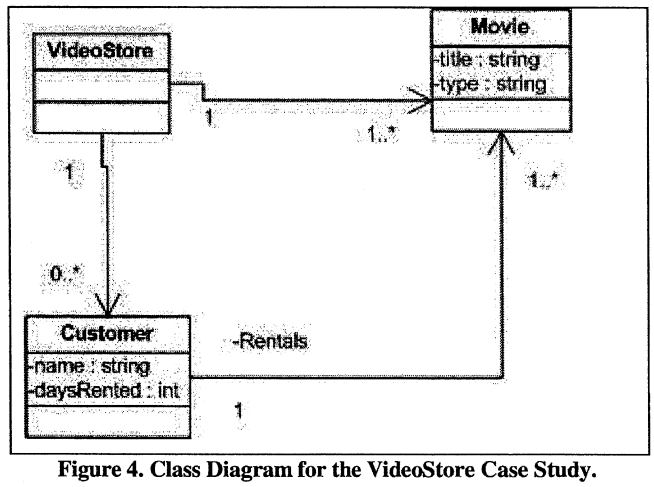

Figure 4. Class Diagram for the VideoStore Case Study.

Initially, the class diagram is converted into an XML representation. During Stage 2 of the validation process, the XML representation of the class diagram is converted into a CFG representation, as shown in Table 2.

1. VideoStore → MOVIES CUSTOMERS | CUSTOMERS
MOVIES | MOVIES | CUSTOMERS
2. MOVIES → MOVIES MOVIE | MOVIE
3. MOVIE → title type
4. CUSTOMERS → CUSTOMERS CUSTOMER | eps
5. CUSTOMER → name days RENTALS
6. RENTALS → RENTALS RENTAL | RENTAL
7. RENTAL → MOVIE1
8. MOVIE1 → title type

Table 2. Video Store Class Diagram Represented as a CFG

A CFG consists of a start symbol, a set of production rules, terminals, and non-terminals. The CFG in Table 2 contains seven productions with the start symbol indicated by *VideoStore*.

would need to be revised. Correspondingly, if a positive sample fails, then it would mean that a required functionality cannot be obtained from the class diagram, and the class diagram needs to be reconsidered. As an example, it can be observed that the *Add Customer* use case cannot be satisfied by the class diagram in Figure 4. In other words, the derived CFG is not able to generate any strings corresponding to the *Add Customer* scenario. Upon noticing this lack of functionality, the UML class diagram can be refactored as follows: note that class *Customer* contains the attributes *name* and *daysRented*. A new class *Rental* is added, and attribute *daysRented* is shifted to class *Rental*. After this refactoring, the *Add Customer* use case is possible. Figure 5 shows the refactored class diagram.

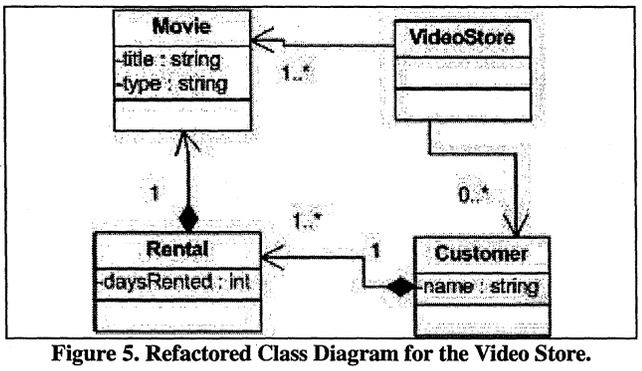

Figure 5. Refactored Class Diagram for the Video Store.

Step 5 of the process provides feedback to the user in the event that a string is not in the language of the CFG. In this case, the user is provided a set of strings that are similar (to a certain extent) to the string provided by the user in the hope that one of the strings in the feedback set would be what the user really desires. The total size of the feedback string set and the similarity degree of the strings are parameters provided by the user. As an example, consider the following string, which is rejected by the CFG in Figure 5:

TheRing

The closest match for this string is rule 3. Although this string will fail to be parsed, the user might be provided with the feedback set given in Table 4.

Similarity: 60%
Total number of strings: 4
1) TheRing a
2) TheRing aa
3) TheRing aaa
4) TheRing aaaa

Table 4. A Sample Feedback Set Provided by the System

The similarity measure of 60% requires that the total number of strings in the set is at least 60% similar to the original string. As an example, compare the following two strings: *TheRing*, and *TheRing a*. These two strings differ by only 1 character.

4.3 CYCLICAL RELATIONS

Naïve approaches to generating CFG's from UML diagrams can encounter complications like an infinite, non-terminating recursive grammar. Our approach is able to derive a correct infinite terminating recursive grammar in the presence of such relationships, and we demonstrate this by proposing a small modification to the *VideoStore* case study example.

Assume that in the class diagram of Figure 4, a 1-to-many relation *Rental2* exists between *Movie* and *Customers*. This would make sense if multiple copies of a movie exist, and can be rented by many customers. An example of a naïve CFG produced from this class diagram would be:

1. VideoStore → MOVIES CUSTOMERS | CUSTOMERS MOVIES | MOVIES | CUSTOMERS
2. MOVIES → MOVIES MOVIE | MOVIE
3. MOVIE → title type RENTALS2
4. RENTALS2 → RENTALS2 RENTAL2 | RENTAL2
5. RENTAL2 → CUSTOMER
6. CUSTOMERS → CUSTOMERS CUSTOMER | CUSTOMER
7. CUSTOMER → name days RENTALS
8. RENTALS → RENTALS RENTAL | RENTAL
9. RENTAL → MOVIE

Production sets (3, 4, 5) and (7, 8, 9) indicate a cyclical non-terminating relationship in the grammar. In this situation, use case strings will fail to be parsed because the grammar is non-terminating and infinite. This problem arises whenever a class, modeled by a non-terminal, is referred to by another class via a relation, which is also modeled by a non-terminal. We propose using a new non-terminal to model the destination class pointed to by the source class in a relation. This prevents the problem of inheriting the relation non-terminals of the destination class by the source class. Using this technique, the CFG for the modified *VideoStore* example changes to the CFG in Table 5.

1. VideoStore → MOVIES CUSTOMERS | CUSTOMERS MOVIES | MOVIES | CUSTOMERS
2. MOVIES → MOVIES MOVIE | MOVIE
3. MOVIE → title type RENTALS2
4. RENTALS2 → RENTALS2 RENTAL2 | RENTAL2
5. RENTAL2 → CUSTOMER1
6. CUSTOMER1 → name days
7. CUSTOMERS → CUSTOMERS CUSTOMER | CUSTOMER
8. CUSTOMER → name days RENTALS
9. RENTALS → RENTALS RENTAL | RENTAL
10. RENTAL → MOVIE1
11. MOVIE1 → title type

Table 5. VideoStore CFG to Handle Cyclical Relations

An example of a string in this CFG would be:
1. theRing horror
2. Jack 5 Ann 5 Mike 10
3. jurassicPark child
4. John 1 Jane 5
5. Bruce 10
6. theRingTwo horror

Sentences 1-4 correspond to two instances of the *Rentals2* relation, and sentences 5 and 6 are an instance of the *Rental* relation.