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

Date: 26 August 2014 **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.

Wiet Mazairac, Jakob Beetz, "BIMQL – An open query language for building information models", Advanced Engineering Informatics, Volume 27, Issue 4, October 2013, Pages 444-456

[Q1] The given article explains the open query language for building information models. The proposed query language is intended for selecting, updating and deleting of data stored in a class models (Ifc). The production rules of the grammar given in **Listing 2** of the page 3.

(a) Draw a flow diagram for the following BIMQL statement. (Note that the variable *VAR1* may be INTEGER or REAL and *VAR2* may be INTEGER or STRING and *ET_PL* may be EntityType or PLUGIN) [20 Marks]

```
Select  $AllDoors
Where $AllDoors.ET_PL = IfcDoor
And     $AllDoors.Property.Width /= VAR1
Select  $AllDoorsHeights := $AllDoors.Attribute.OverallHeight
Set     $AllDoors.Attribute.Description := VAR2
```

(b) Draw a derivation tree for the following BIMQL expression. [15 Marks]

```
Select  $AllDoors
Where $AllDoors.EntityType = IfcDoor
Or      $AllDoors.Property.Height = 123
And     $AllDoors.Attribute.Description = ProductXYZ
```

(c) Define the grammar **G** for the BIMQL. Clearly show the terminals and the non-terminals. [10 Marks]

(d) Write LEX implementation syntax for token of the BIMQL. [10 Marks]

Part B – Answer only Three questions

[Q2] The DESK grammar rules define as follows (*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) Draw a derivation tree for string: ***print id + id where id = int id = int*** [04 Marks]

(b) Draw a NFA for string: ***print id*** $(+ \ \textbf{\textit{id}})^*$ ***where*** $(\textbf{\textit{id}} = \textbf{\textit{int}})^*$ [04 Marks]

(c) Draw a DFA equivalent to NFA in (b) [07 Marks]

[Q3] Consider the grammar rules given below (*VendingMachine, STOCK, SALES, ITEM, SALE* are non-terminals and others are terminals).

VendingMachine → stock STOCK sales SALES
STOCK → STOCK ITEM | ε
ITEM → item price qty
SALES → SALES SALE | ε
SALE → item price

(a) Derive the string: ***stock item price qty sales item price*** [01 Marks]

(b) Define the Chomsky Normal Form (CNF) for CFGs. [02 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 (FRUITS is a non-terminal and others are terminals).

FRUITS → mango FRUITS apple | mango apple

(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: *mango mango apple apple* [03 Marks]

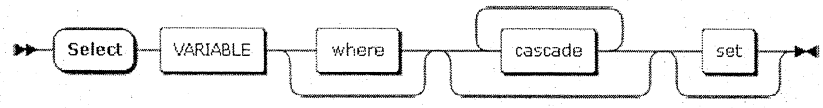
[Q5]

(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. [07 Marks]

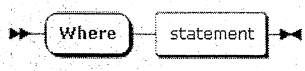
```
BIMQL          ::= select
select         ::= 'Select' VARIABLE where? cascade* set?
cascade        ::= 'Select' VARIABLE ':=' VARIABLE ('.Attribute.' STRING | '.Property.'
                   STRING) where?
where          ::= 'Where' statement
set            ::= 'Set' VARIABLE '.Attribute.' STRING ':=' (INTEGER | REAL | STRING)
statement      ::= relation ('And' relation | 'Or' relation)*
relation       ::= relationleft ('=' relationright | '/=' relationright | '<' relationright
                   |'<=' relationright | '>='    relationright | '>' relationright)
relationleft   ::= (VARIABLE '.EntityType' | VARIABLE '.Attribute.' STRING | VARIABLE
                   '.Property.' STRING | VARIABLE PLUGIN)
relationright  ::= (INTEGER | REAL | STRING)
VARIABLE       ::= '$' STRING
PLUGIN         ::= ':' STRING
INTEGER        ::= '0..9'+
REAL           ::= INTEGER+ ('.' INTEGER+ )?
STRING         ::= ('0..9' | 'A..Z' | 'a..z' | '!' | '#' | '$' | '%' | '&' | '^' | '|' |
                   '*' | '+' | ',' | '-' | '.' | '/' | ':' | ';' |'<' | '=' | '>' | '?'
                   | '~' | '`' | '@' | '_')+
```

Listing 2. Backus-Naur form of the proposed BIMQL query language.

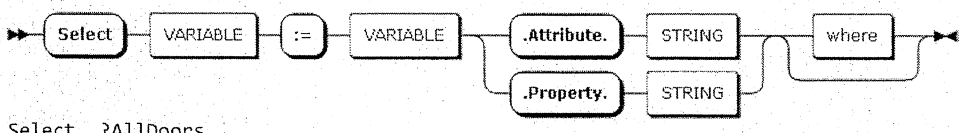



```
Select ?Everything
```

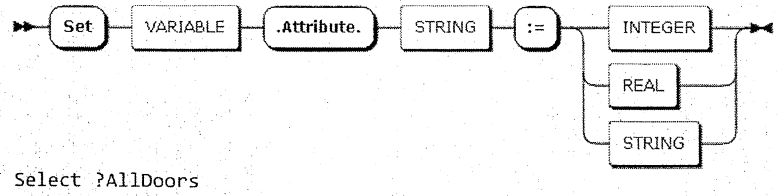
Fig. 2. Syntax diagram of the 'select' statement along with a BIMQL example selecting all entities of a model.




```
Select ?AllDoors
Where  ?AllDoors.EntityType = IfcDoor
```




```
Select  ?AllDoors
Where   ?AllDoors.EntityType = IfcDoor
Select  ?AllDoorsHeights := ?AllDoors.Attribute.OverallHeight
```

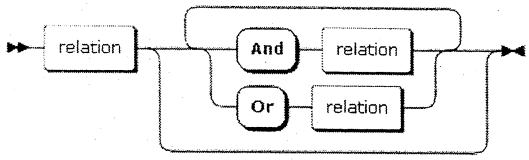



```
Select ?AllDoors
Where  ?AllDoors.EntityType = IfcDoor
Set    ?AllDoors.Attribute.Description := ProductXYZ
```

Fig. 3. Flow diagrams and illustrative examples of 'where', 'select' and 'set' rules.

also yields performance enhancements esp. on larger models. The plugin is triggered by the ':' colon character, after which an arbitrary string is identifying the keyword associated with a specific plugin implementation.

As a proof of concept, two such plugins have been implemented: The "storey plugin" allows to filter building elements by building storey by going through the 'IfcRelContainedInSpatialStructure' objectified relationship instances of the model and looking whether the 'RelatingStructure"s associated with an object (IfcProduct) name matches with the right-hand side. The "is-a plugin" (Listing 3) allows for a natural-language selection of building objects. It uses the ISO 12006-3:3-based buildingSMART Data Dictionary (bsDD) [53] to map natural language names to IFC entity definitions: The 'IfcDoor' class has been associated as one of the names of a concept (IfdSubject) which has the name "door" in the language "International English" as one of its names. The plugin implements a cached reverse-lookup that allows to retrieve names of the same concept in other languages ("Tür", "Deur", "Dør" etc.) and matches them with the right-hand side of the where statement. This allows for natural language selections of




```
Select ?AllDoors
Where  ?AllDoors.EntityType = IfcDoor
And    ?AllDoors.Attribute.Description = ProductXYZ
```

Fig. 4. Syntax diagram and illustrative example of the 'relation'-rule and its combinations.

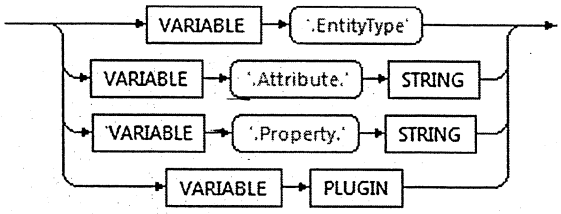


Fig. 5. 'relationleft'-rule syntax diagram.

objects. The following examples selects all IfcDoors (searched for by its Dutch word "Deur") and IfcWindows (searched for by its French term "fenêtre") that are either on the first or second floor (searched for by the German names provided in the particular model.

Notice how the storey plugin can be triggerd by different keywords "Storey" (English) and "Verdieping" (Dutch), as an arbitrary number of keywords can be provided by a plugin registered into the query engine implementation at runtime. More details on the implementation of the extension mechanism can be found in Section 4.

3.7. 'Relationright'-rule

The 'relationright'-rule (Fig. 6) for the assignment of a comparison can be any string. If the string is numeric, it will be automatically compared with property definitions of simple and derived types provided in the property such as REAL, INTEGER or IfcPositiveLengthMeasure through automatic casting.

It is also possible to specify patterns by using asterisks, question marks and other terms familiar from regular expression terms [54]. The underlying functionality will try to match the pattern with the value the 'relationleft'-rule returns. These patterns make is possible to e.g. return both 'OverallHeight' and 'OverallWidth' attributes of an IfcDoor by querying for 'Overall*' or return the 'SecurityRating', 'FireRating' and 'AccousticRating' properties form the PSet_Door_common in one go by asking for '*Rating'

3.8. Shortcuts

The introduction of shortcuts serve as an illustration as to why a domain specific language that provides syntactic simplifications compared with a general purpose language is useful for complex models such as the IFC. The relation of an entity (IfcSpace in this example) with its properties that go beyond the few direct attributes (Listing 4) defined in the core schema constitutes a complex sub graph (Fig. 7). This requires several graph network 'hops' or nested iterations in procedural programming approaches and nested joins in traditional SQL based query languages.

The BIMQL-code in the above row of Listing 5 shows how BIMQL can be used to navigate the traditional graph-connection. Seven lines of code are needed. The first two lines select an object and the

```
Select ?Var1 Where ?Var1:Storey = Obergeschoss
    Or ?Var1:Verdieping = Erdgeschoss
   And ?Var1:is-a = Deur
    Or ?Var1:is-a = fenêtre
```

Listing 3. Natural language in a BIMQL query.

other five lines are required to retrieve the 'volume'-property of that object. When the property shortcut is used (lower part) those five lines which were needed first are replaced by only one line.

In Section 1.1 the IFC model specification and specifically the concept of objectified relationships have been introduced. Objectified relationships could be a starting point for additional shortcuts. The next example, in which the boundary object for a given space are retrieved from the model (Listing 6) illustrates this principle. A space is related to its boundaries by the 'IfcRelSpaceBoundary'-entity.

By introducing a new shortcut, based on the 'IfcRelSpaceBoundary'-entity and named 'SpaceBoundary', the query not only becomes one line shorter, but also more comprehensible (Listing 7).

4. Implementation

The bimserver.org platform [16] already provides some means to extract partial building information models from a repository. These models can be downloaded after which they can be viewed or edited. An altered partial model can be uploaded to the server again on which it will be merged with the original model still present. Selections on individual model revisions can be made by specifying object IDs (including revision and authoring information), the IFC GUID, or all instances of a selected entity in the IFC schema. It is also possible to create custom queries by writing Java code which can be compiled and loaded during the runtime of the server, however the threshold to actually use this feature is high and the learning curve steep. In order to overcome this high entry threshold and because the bimserver.org is an accessible open source project we have chosen to integrate the proposed query language as a Domain Specific Language (DSL) that wraps the underlying querying mechanisms and hides the low-level technicalities from end-users. Next to the possibility to write Java code, the bimserver.org platform will be extended, so it will be possible to write BIMQL code.

The Model Driven Architecture (MDA) approach of software engineering is one of the architectural cornerstones of the bimserver.org framework. Instead of developing the source code itself, the programmer develops a model, which is used for automatically generating the source code. Although it increases the initial planning and writing resources required to produce the system that automatically generates source code from a model, this method increases portability, productivity and cross-platform interoperability. First the EXPRESS schema is converted to an Eclipse Modeling Framework (EMF) model. This model is then used to

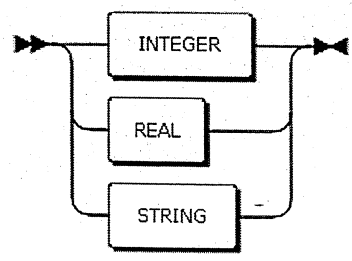


Fig. 6. 'Relationright'-rule syntax diagram.