1 Test Section

2 Abstract Syntax

Figure 1 gives the abstract syntax that of EMU. An EMU module can have a number of mutation applications that each one targets one property of an entity in the model. A mutation is associated with a number of binding roles (matching roles) that query the target models and obtain instances of the matching result from the model. Querying the model is facilitated by an EOL expression in terms of a *domain* (to specify the scope of instances of the target model) and *guard* (to impose more filtering conditions and constraints) over the result of each query.

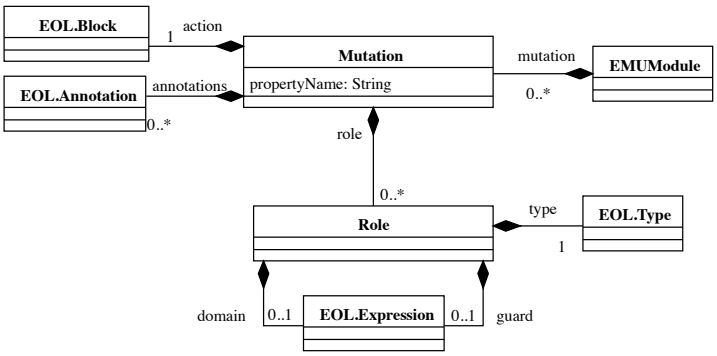

Figure 1: The abstract syntax of EMU

3 Concrete Syntax

A simplified version of the concrete syntax of EMU is given in Listing 1. To begin with, any mutation must start with the keyword *mutate* followed by a target property name of an entity followed by multiple binding roles.

```
1   EMUModule:
2     '@max'
3     'mutation' ID
4     ROLE (',' ROLE)* ACTION
5   ROLE:
6     ID ':' EOL.Type (DOMAIN|GUARD)*
7   DOMAIN:
8     ('in'|'from') ':' EOL.Expression
9   GUARD:
10    'guard' ':' EOL.Expression
11  ACTION:
12    ('byAdd'|'byReplace','byDelete') '{' EOL.StatementBlock '}'
```

Listing 1: EMU concrete syntax

Binding roles are concepts that accumulate a set of model instances obtained from querying and filtering the target model. A binding role starts with an idenfitication (that is a variable to hold the collection of instances) followed by a colon ':' and then the type of collection (line 4). By defualt, a variable name and a type specificaion only will fetch all instances created in the model for

the type. Users can filter the combination collection using EOL expressions in terms of the *DOMAIN* and *GUARD* language concepts (line 8 and line 10).

Using a domain language concept, a scope of model instances can be specified by *in* (for static query) or *from* (for dynamic query) keywords followed by the query using an EOL expression (in line 8). The difference between the static and dynamic model queries is that in the first the result of evaluating the query (EOL expressions) is computed once for all executions of the *action* block of the mutation and the results are not accessible by other query expressions of other binding roles. On the contrary, the dynamic query result is re-computed whenever the containing binding role's values are iterated and executed. Its values are accessible by other binding roles.

In using the *GUARD* language concept, the user can impose further filtering and constraints over the resulting values of the role query using the keyword *guard*, as in line 10.

After defining binding roles, the user must specify the mutation action to perform. The user can choose one of the define actions which are "byAdd","byReplace", and "byDelete". The action block that follows this should contain EOL statements that only mainupluates the property (specificed in line 3).

4 EMU Examples

The purpose of EMU is to facilitate the implementation of concrete mutation operators for a given metamodel. This section gives examples of using EMU and were based on the metamodel Figure 2.

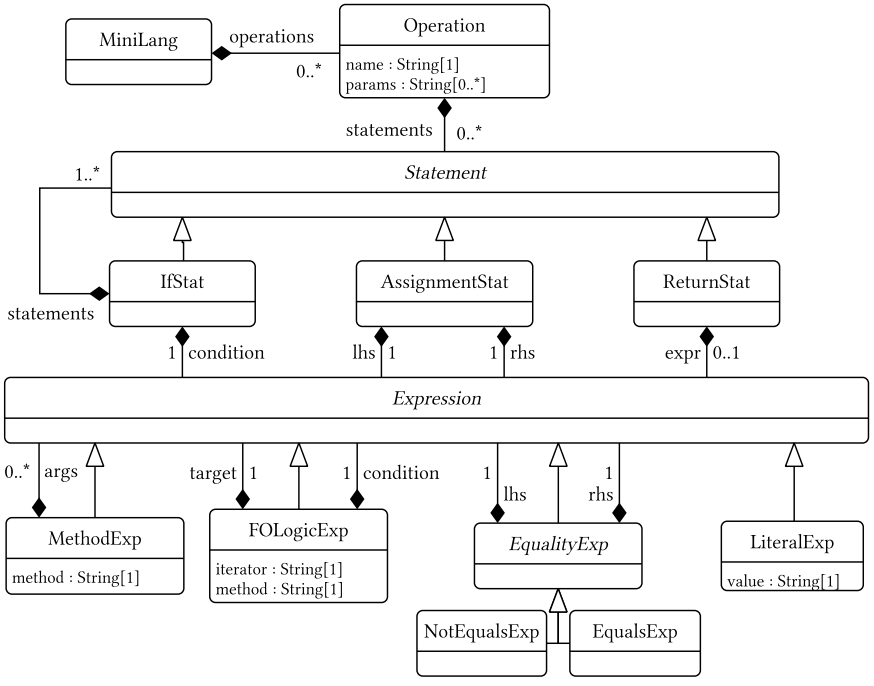

Figure 2: The abstract syntax of the MiniLang metamodel

4.1 EMU for Replacement

This example is to write an EMU program that allows the replacement of the value of the property condition of instances of FOLogicExp. The intention is to replace an equal comparison operator with a not-equal comparison operator (as shown in Figure 3). This is to demonstrate how a common mutation in programming languages in which an equality operator is modified with replacement action by using EMU. This mutation has preserved the left-hand-side and the right-hand-side of the instance (equalExp1 by assigning them to the new instance notEqualExp1.

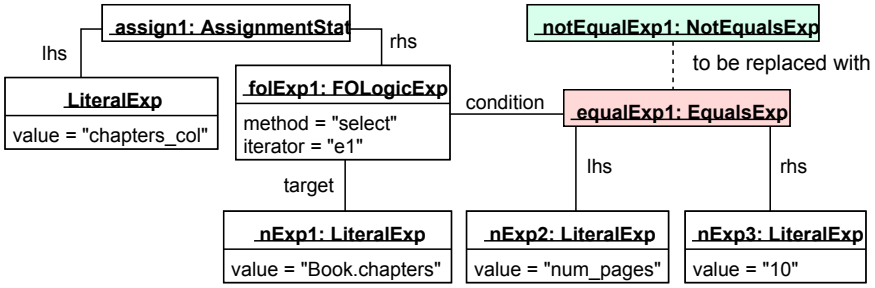

Figure 3: A fragment of an instance model of the MiniLang metamodel 2

```
1  mutate condition
2  instance:FOLogicExp
3    in:FOLogicExp.all.select(e|
4      e.condition.isTypeOf(EqualsExp))
5  byReplace {
6     //create new Boolean object
7     var n_equal = new NotEqualExp();
8
9     //copy over lhs from old instance
10    n_equal.lhs=instance.condition.lhs;
11
12    //copy over rhs from old instance
13    n_equal.rhs=instance.condition.rhs;
14
15    //assign new condition
16    //to this FOLogicExp
17    instance.condition = n_equal;
18 }
```

Listing 2: A CMO implementation for mutating model 3

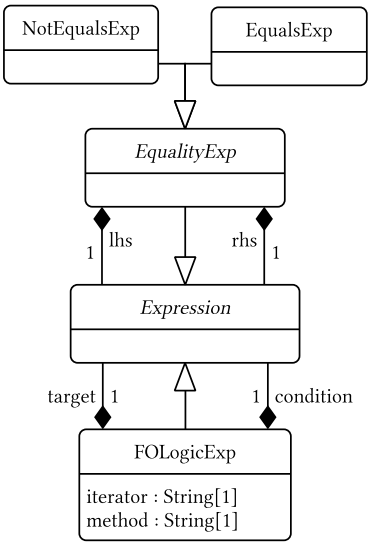

Figure 4: A fragment of the metamodel in Fig. 2

4.2 EMU for Addition

This example is to write an implemntation of EMU program that adds a return statement to an operation statement block as shown in Figure 2.

```
1   mutate statements
2   instance:IfStat
3   byAdd {
4      // create new ReturnStat object
5      var ret_stat = new ReturnStat();
6
7      // add the new object to the
           collection
8      // of statements
9      instance.statements.add(ret_stat);
10  }
```

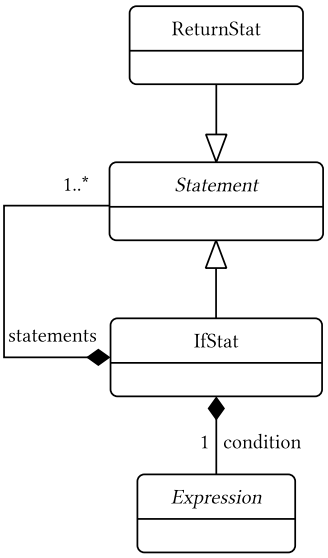

Figure 5: A fragment of the metamodel in Fig. 2

4.3 EMU for Deletion

This EMU example code deletes statements of an instance of operation as given in Figure 2. In this example, there is two role bindings: one of fetchs all instances IfStat and the other is to iterate through instances represented by statements property.

```
1   mutate statements
2   instance:IfStat,
3   stat:Statement
4    from:instance.statements
5   byDelete {
6      // remove a statement from collection
7      // of statements
8      instance.statements.remove(stat);
9   }
```

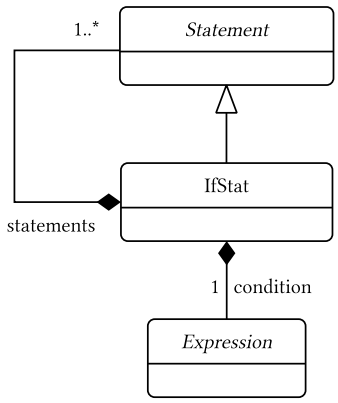

Figure 6: A fragment of the metamodel in Fig. 2