Visualization module solving logical problems for the FORMULA system

Ksenia Panova
Student of Peter the Great
Saint-Petersburg Polytechnic University
Email: k-nazarova@mail.ru

Abstract

Visualization programs are used for various purposes (training, debugging and optimization) by programmers. However, such programs are often highly dependent on programming paradigms for which they were created, and not always successfully adapt to the calculations made in other paradigms. This paper describes designed system which intended for data visualization. It focuses on information visualization of solutions of logical problems derived from the FORMULA system.

1. INTRODUCTION

We have designed the module of visualization of solutions obtained via the smt solver Formula. Solvers for satisfiability modulo theories (SMT) check the satisfiability of first-order formulas containing operations from various theories such as the booleans, bit-vectors, arithmetic. A thorough consideration of various methods of visualization was made. As a result of studies the declarative programming language for solutions of logic problems such as Einstein task or the problem of queens was developed.

2. PROBLEMS

The developed system is designed for data visualization. The purpose of the study is to consider an information visualization solutions of logical problems derived from the FORMULA system, which is fundamentally different from existing visualization tools. The most obvious advantage of the new system is that the developed module operates with a much smaller amount of information, which saves time and expense of resources for the visualization process.

The main task of the investigation is to design an integrated module of visualization solutions and the results of logical tasks.

The developed module should maintain the following functions:

1. Integration module logical solution FORMULA tasks, analysis of the results of using it
2. Display of static solutions with support for the following types of visualization:
   (a) Table
   (b) Counts
   (c) Vectors
3. Display of the process of solving problems by means of a dynamic visualization:
   (a) Moving objects
   (b) Time Management

3. ARCHITECTURE DESIGN

The project structure is shown in Fig. 2 The system is a program running in Windows. It analyzes configured via FORMULA file, and loads the content to the respective variable depending on the type of task and the format:

- For a static display in a table - table
- For dynamic display - sequence

The first component of this structure - FORMULA system, in which the task of first-order logic is implemented and executed. This step tests satisfiability of task, and if it is, the result is written to the output file. Visualization system uses this file to obtain the solution of the proposed problem and its graphical display.

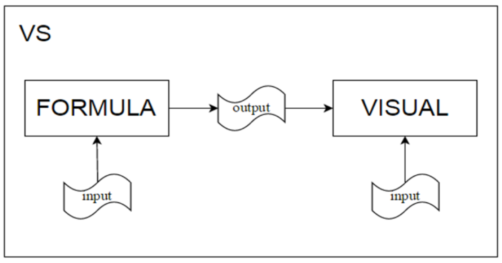

Figure 1. Project structure

4. LANGUAGE MODULE

Starting the system designed by the programming language is accompanied by its conversion into a form that is meaningful to the computer. Therefore, we need a program that reads the text written in one language (the source) and translates it into an equivalent text in another language.

- For a static display in a table - table
- For dynamic display - sequence

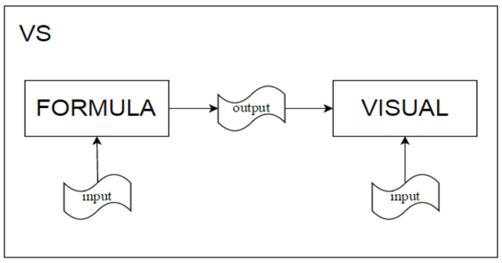

Figure 2. Project structure

In order to provide this, compiler that converts a program written in the language of visualization, in a program written in C ++ was developed.

For that, lexer performs lexical analysis which is the process of converting a sequence of characters into a sequence of tokens (strings with an identified ”meaning”). Lexical analysis was performed using parser generator ”Lex”.

After lexical analysis is the syntactic analysis. Parsing or syntactic analysis is the process of analysing a string of symbols, either in natural language or in computer languages, conforming to the rules of a formal grammar. The purpose of syntactic analysis is to determine the structure of the input text. This structure consists of a hierarchy of phrases, the smallest of which are the basic symbols and the largest of which is the sentence[1]. The structure can be described by a tree with one node for each phrase. Basic symbols are represented by values stored at the nodes. The root of the tree represents the sentence. an example is shown in Fig. 3. Syntactic analysis was performed using generator ”Yacc”, which provides automatic construction of grammatical analysis procedures.

As a result of the joint compilation of the generated lexer and parser , executable file visual.exe is created for handling of code with new syntax.

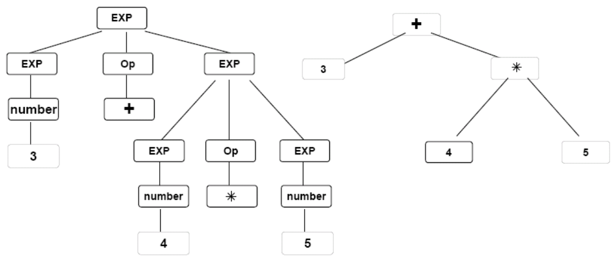

Figure 3. Example of syntactic analysis

The code generator solves the basic problem of the translator: in the process of traversal of the syntax tree it takes tokens in the machine instruction sequence. Tokens are extracted from the tree nodes in a certain order . When the amount of tokens is enough to determine the desired operation, code generator generates corresponding machine instructions and tree traversal resumes.

Connectivity between Lex and Yacc is shown in Fig. 4.

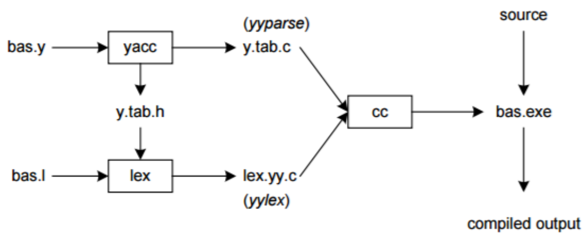

Figure 4. Building a Compiler with Lex/Yacc

Yacc reads the grammar descriptions in bas.y and generates a syntax analyzer (parser), that includes function yyparse, in file y.tab.c. Included in file bas.y are token declarations. Also yacc generates definitions for tokens and places them in file y.tab.h. Lex reads the pattern descriptions in bas.l, includes file y.tab.h, and generates a lexical analyzer, that includes function yylex, in file lex.yy.c.

Finally, the lexer and parser are compiled and linked together to create executable bas.exe. Then we call yyparse to run the compiler. Function yyparse automatically calls yylex to obtain each token[2].

5. SYNTAX

Alphabet of the visualization language consists of latin alphabet letters (uppercase and lowercase), underscores, arabic numerals, and special characters (- + =, () []!). This syntax is case sensitive.

```
<name> = (<letter>|_)
            {<letter>|<letter>|_}
<letter> = A|B|...|Y|Z|a|b|...|y|z
<digit> = 0|1|...|9
```

As identity is unacceptable to use the reserved keywords that have an impact on the translator and are used to form a variety of language structures, or which are the names of standard functions: *print, show, move, backgnd, slab, table, listing, if, foreach, in*.

In the statement, all the tokens can be separated by any number of whitespace characters, which includes spaces, tabs, and newline, carriage return, and newline.

Conditions:

```
if  (<condition>)
        <statement>
```

or

```
if  (<condition>)
        <statement>
else
        <statement>
```

Enumeration is used to declare all imaging facilities:

```
enum <enum_name>{<item>, <item>, ...};
```

The syntax allows one-dimensional and two-dimensional arrays consisting of sets of integers or strings. Moreover, one-dimensional arrays can be of different types.

1. Standard one-dimensional array

   ```
   <array_name>[<size>] =
             {<statement>, ...};
   ```

2. A one-dimensional array associated with a list of visualization objects, which consist of a set of strings, namely paths to images of the respective objects

   ```
   <array_name>[<enum_name>] =
             {<path>, ...};
   ```

3. A one-dimensional array consisting of a set of pairs of values of the object and its initial state, required to initiate dynamic visualization

   ```
   <pair> = (<object_name>,<init_state>)
   <array_name>[<size>] =
             {<pair>, ...};
   ```

4. Two-dimensional array

   ```
   <array_name>[<size>][<size>] = {
             <statement>,...;
             <statement>,...;
              ...
   };
   ```

Also, the syntax requires a foreach statement, which defines a cyclic structure, repeating groups of operators in relation to each element of the array.

```
foreach (<item> in <array_name>)
        <statement>
```

One of the standard functions is print, which prints the value received on its input variable to standard output:

```
print <name>;
```

To determine the format of the visualization, dynamic or static (table), keywords *table* and *sequence* are used. *table* is the ID of a two-dimensional array, which is loaded from the result of the FORMULA, *sequence* - a standard function that initializes the dynamic visualization.

The function *sequence* takes two arguments: **operator** is the number of states between which the transitions, and **array_name** is identifier of the array that contains the objects for display and their initial states.

```
sequence (<statement>, <array_name>);
```

Standard function *move* is used to switch between the states. *move* obtains object name, which should change its current state. The direction of movement of the object is based on the previous states.

```
move(<object_name>);
```

The primary function of language is function for downloading of the format and objects into visualizer and graphical representation of these objects. This role performs the function *show*:

```
<show_statement> = table |
         sequence (<statement>,
                  <array_name>) |
         move(<object_name>);
show <show_statement>;
```

6. VISUALIZATION MODULE

A key part of the system architecture is graphical module. As a GUI library use Qt 5.4.1 - cross-platform toolkit software development with C ++ programming. The following classes are the basis of the implementation of the visualization module:

1. Class *Visual* - a class that is responsible for the display of objects in the dialog box.
2. Class *NewItem* - a class that is responsible for a separate display object, describes its size, location, and its direction of movement in the case of dynamic visualization.
3. Class *NewState* - a class inherited from QGraphicsItem which describes a separate state, its size and location in the dialog box. It is used in dynamic imaging.
4. Class *MakeMove* - a class that initiates the download of status objects and their movement when dynamic visualization.
5. Class *NewTable* - a class inherited from QGraphicsItem. It describes and visualizes the table.

   UML diagram of this module in Fig. 5.

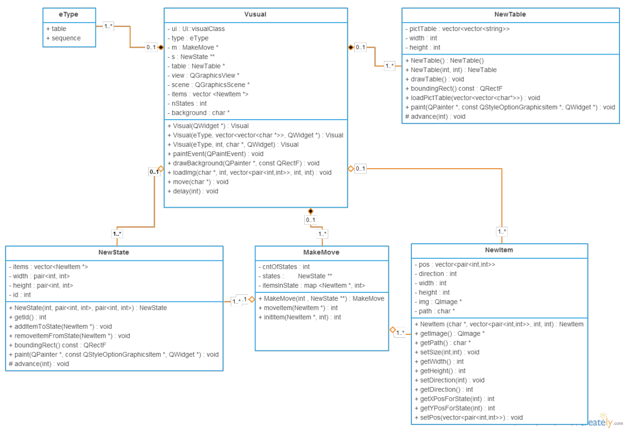
eType
+ table
+ sequence
Vusual
- ui : Ui::visualClass
- type : eType
- m : MakeMove *
- s : NewState **
- table : NewTable *
- view : QGraphicsView *
- scene : QGraphicsScene *
- items : vector <NewItem *>
- nStates : int
- background : char *
+ Visual(QWidget *) : Visual
+ Visual(eType, vector<vector<char *>>, QWidget *) : Visual
+ Visual(eType, int, char *, QWidget) : Visual
+ paintEvent(QPaintEvent) : void
+ drawBackground(QPainter *, const QRectF) : void
+ loadImg(char *, int, vector<pair<int,int>>, int, int) : void
+ move(char *) : void
+ delay(int) : void
NewTable
- pictTable : vector<vector<string>>
- width : int
- height : int
+ NewTable() : NewTable()
+ NewTable(int, int) : NewTable
+ drawTable() : void
+ boundingRect() const : QRectF
+ loadPictTable(vector<vector<char*>>) : void
+ paint(QPainter *, const QStyleOptionGraphicsItem *, QWidget *) : void
# advance(int) : void
NewState
- items : vector<NewItem *>
- width : pair<int, int>
- height : pair<int, int>
- id : int
+ NewState(int, pair<int, int>, pair<int, int>) : NewState
+ getId() : int
+ addItemToState(NewItem *) : void
+ removeItemFromState(NewItem *) : void
+ boundingRect() const : QRectF
+ paint(QPainter *, const QStyleOptionGraphicsItem *, QWidget *) : void
# advance(int) : void
MakeMove
- cntOfStates : int
- states : NewState **
- itemsInState : map <NewItem *, int>
+ MakeMove(int , NewState **) : MakeMove
+ moveItem(NewItem *) : int
+ initItem(NewItem *, int) : int
NewItem
- pos : vector<pair<int,int>>
- direction : int
- width : int
- height : int
- img : QImage *
- path : char *
+ NewItem (char *, vector<pair<int,int>>, int, int) : NewItem
+ getImage() : QImage *
+ getPath() : char *
+ setSize(int,int) : void
+ getWidth() : int
+ getHeight() : int
+ setDirection(int) : void
+ getDirection() : int
+ getXPosForState(int) : int
+ getYPosForState(int) : int
+ setPos(vector<pair<int,int>>) : void
1..*
0..1
creately.com

Figure 5. UML diagram of visualization module

7. TESTING

Consider the example of visualization of Einstein problem.

Task: 5 different people live in 5 houses of different colors, smoke 5 different brands of cigarettes, look after 5 different animals, drink 5 different beverages. At the same time, there are 15 conditions:

1. The Norwegian lives in the first house.
2. The Englishman lives in the red house.
3. The green house is immediately to the left of the white.
4. The Dane drinks tea.
5. The one who smokes Rothmans, lives next to the one who has cat.
6. Anyone who lives in the yellow house smokes Dunhill.
7. The German smokes Marlboro.
8. He who lives in the center drinks milk.
9. A neighbor of the one who smokes Rothmans, drinks water.
10. The man who smokes Pall Mall, has bird.
11. The Swede has dog.
12. The Norwegian lives next to the blue house.
13. Whoever has horse lives in the blue house.
14. The man who smokes Philip Morris, drinking beer.
15. In the green house drinks coffee.

Question: Who has fish?

Result of visualization is static table with the solution of the problem. Here see that the owner of fish is the German (Fig. 6):

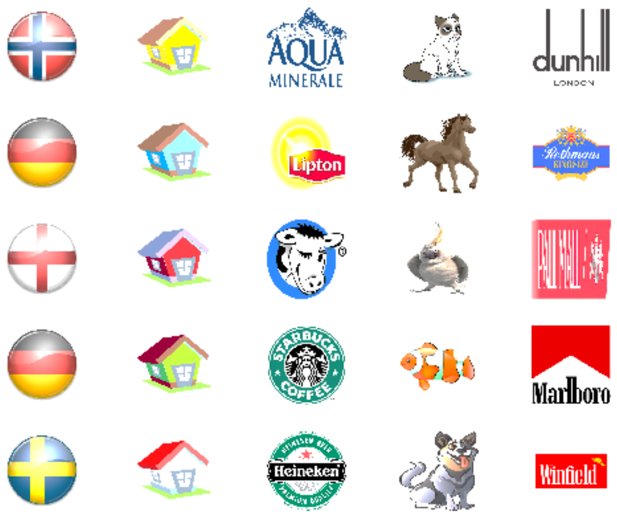

Figure 6. Result visualization solutions of Einstein's problem

8. CONCLUSIONS

As a result of studies the integrated module of visualization solutions and the results of logical tasks was designed. Also visualization language was developed. For that rules and a new programming language syntax have been formulated, on the basis of which carried out the lexical, syntactic and semantic analysis. Graphical system module is also designed to visualize input data in a predetermined format, namely in the form of static and dynamic table moving objects.

References

[1] A. Aho, M. Lam, R. Sethi, Compilers: Principles, Techniques, and Tools, 2008.

[2] T. Niemann, Lex & Yacc tutorial.