MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
SUMY STATE UNIVERSITY
UKRAINIAN FEDERATION OF INFORMATICS

PROCEEDINGS

OF THE IV INTERNATIONAL SCIENTIFIC CONFERENCE

ADVANCED INFORMATION SYSTEMS AND TECHNOLOGIES

AIST-2016

May 25 –27, 2016
Sumy, Ukraine

Software implementation of calculating the value of a logical expression in compilers

Z.I. Maslova[1], T.V. Lavryk[2]
Sumy State University, Ukraine, [1]maslova@sumdu.edu.ua, [2]metodist@dl.sumdu.edu.ua

Abstract. This paper describes an algorithm to optimize a process of determining the value of a logical expression. This algorithm is based on the principles of the algebra of logic, graphs and automata theory. Fast calculation of a logical expression is achieved by a reduction in the number of operations. The program is a multi-functional simulator.

Keywords. Logical Expression, Compiler, Boolean Variable, Graph, Finite State Automaton.

INTRODUCTION

Since the fifth version of the Pascal language, compiler has had a function to compute the value of a logical expression fast. Initially, this function could be added on a programmer's request. Modern compilers except for program translators from algorithmic language automatically have additional functions to optimize the program [1].

One of this function is a rapid calculation of the value of a logical expression.

PROBLEM STATEMENT

Fast calculation of the value of a logical expression can be achieved by optimizing the number of logical operations that affect the result. The goal of this work is to create an algorithm for calculating a logical expression and to design a program code on its base. Initial data in the given problem are logical expression.

DESCRIPTION OF THE ALGORITM

The algebra of logic, graph theory and the theory of automata is used to create an algorithm for logic expression calculation.

Reduction in the number of operations is achieved through the use of the basic properties of the logical constants:

$$x \wedge 1 = x;$$
$$x \vee 1 = 1;$$
$$x \wedge 0 = 0;$$
$$x \vee 0 = x.$$

Boolean variable x in terms of graph theory is represented as a binary tree, with edges x and $\overline{x}$, and the leaves – 0 and 1.

The graph for the whole logical expression is constructed according to the rules [2]:

- for elementary logic conjunctions $x_1 \wedge x_2$ graph x_2 is attached to output one of the graph for x_1;
- for disjunction graph of x_2 is attached to output zero of the graph x_1;
- for the graphical implementation of negation x values of outputs are inverted.

Connection of the outputs of subgraphs with the same names depends on the sequence operations and the presence of brackets.

The graph for the logical expression (1) will have form figure 1.

$$f(x_1, x_2, x_3, x_4, x_5) = x_1 \vee x_2 \quad \overline{x_3} x_4 \vee x_5 \quad (1)$$

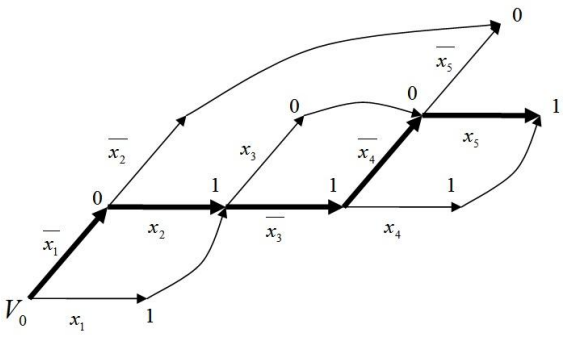

Figure 1 –The graph for the a logical expression

A visual representation of the graph (fig. 1) is given by graph (fig. 2).

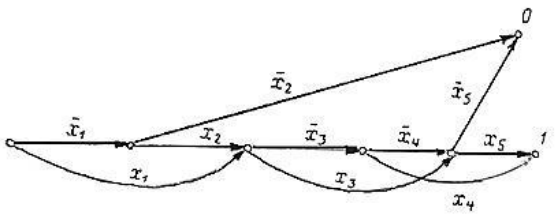

Figure 2 –The modified graph.

The next step of the algorithm is to build a finite automaton for a calculated expression. Automaton table (fig. 3) is constructed using the graph. The number of states of the automaton is equal to the number of variables in the expression. Input alphabet (0, 1) are the values of logical variables. The letter of output alphabet is equal to the output value of the previous variable, to which the current variable is joined.

Variable	Output "0"	Output "1"
x_1	x_2, –	x_3, –
x_2	x_5, 0	x_3, 0
x_3	x_5, 1	x_4, 1
x_4	x_5, 1	x_5, 1
x_5	x_5, –	x_5, –

Figure 3 – Automaton table.

By using this table the program quickly calculates the value of a logical expression. So, if the value of the variable x_1 is zero, the program proceeds to check the value of the variable x_2. And if the value of x_2 is 0, the program will immediately receive the value of the whole expression. If the value of the variable x_1 is 1, the program proceeds to the calculation value of (2):

$$\overline{x_3}x_4 \vee x_5 \quad (2)$$

And the value of the whole expression will be determined by the value of the expression (2).

PROGRAM REALIZATION

The complex [multi-functional] simulator could be created based on the program languages PHP, Javascript, CSS and HTML. PHP was used for this project. JavaScript was used to create the database, while CSS stylizes pages based on the determined parameters. HTML language is used for the graphic design of the project. The output of the project is created automaton for a fast calculation of a logic expression, logic processor.

The program is constructed so that it can be used in the educational process as a multifunctional simulator. Functions of this simulator are the following: demonstration of the algorithm for solving the problem, training and monitoring of students' knowledge. The solution process could be demonstrated without participation of a student. The learning function is realized with participation of a student in solving the problem. Program monitors all actions of a student. If the program detects an error, the process stops. A student gets an opportunity to consult with a teacher. If necessary, a student can read theoretical material. It is possible to correct errors. If needed, the steps may be repeated. The final step is the monitoring phase. At this stage, a student solves the problem by him-/her-self. The solution is performed without any interruptions. It is not allowed to correct errors and to consult with the teacher.

CONCLUSIONS

The theory of logic algebra, graph theory and automata theory is important section of discrete mathematics. The developed program deepens students' knowledge of discrete mathematics. In addition, students enhance knowledge on system programming and familiarize themselves with the functions of the compiler.

REFERENCES

[1] Waite W. M. An introduction to compiler construction / Waite W.M., Carter L. R. – New York: HarperCollins, 1993. – 438 p.

[2] Kuznetsov, O. P. Diskretnaya matematika dlya inzhenera [Tekst] / O. P. Kuznetsov, G. M. Adelson-Velskiy. – 2-e izd., pererab. i dop. – M. : Energoatomizdat, 1988. – 480 s.