PAPER • OPEN ACCESS

Design and implementation of an efficient program interpreter for industrial robot

To cite this article: Mingyou Xie *et al* 2021 *J. Phys.: Conf. Ser.* **1884** 012018

Design and implementation of an efficient program interpreter for industrial robot

Mingyou Xie[1], Di Li[1], Minghao Cheng[1], Shipeng Li[1] and Yongchao Luo[2*]

[1] School of Mechanical and Automotive Engineering, South China University of Technology, Guangzhou 510640, China

[2] School of Electrical Engineering, Guangzhou College of South China University of Technology, Guangzhou 510800, China

[*]Corresponding author’s e-mail: luoyc@gcu.edu.cn

Abstract. Robot program interpreter is an important tool for robot to realize real-time control and fast feedback. In order to simplify the process of robot programming and improve the interpretation efficiency of robot program, an easy-to-use and efficient industrial robot program interpreter is designed and implemented in this paper. The interpreter divides the robot program into two parts: variable definition and instruction call. The process of interpretation includes lexical analysis, syntactic analysis, semantic analysis and instruction interpretation modules. Using flex and bison tools to assist in the generation of lexical and syntactic analysis programs, this paper proposes child-sibling notation (CSN) to construct a syntax tree. In semantic analysis, the red-black tree structure of the map container is used to create a symbol table and record variable information. By the way of presetting type checking codes, errors in the program can be reported and handled. Finally, the interpreter traverses the syntax tree with depth-first algorithm, and calls the corresponding control function while interpreting the instruction sentence to execute the motion of robot. The experimental results show that the designed interpreter has high efficiency and stability in interpreting the robot program and meets the operational requirements of industrial robots.

1. Introduction
Industrial robot programming language is used to program for robot. Whether the design of a set of robot programming language is reasonable or not is directly related to the programming ability of robot control system, which determines the function that the robot can provide[1]. The function of robot program interpreter is to interpret and execute the robot program, which directly affects the performance and efficiency of the robot[2]. Scholars all over the world have done some research on the implementation and optimization of robot programming language and its interpreter[3-5]. For example, Yang and Chen (2015) proposed a design scheme of industrial robot language system based on QT and regular expression, which has good portability[6]. Based on PCRE regular expression, Wang and Lv (2018) proposed a interpretation mode including twice scans, which reduced the complexity of the interpreter algorithm[7]. Chen and Chen (2019) proposed a layered architecture for interpreter to reduce the coupling between modules[8]. However, they have not done much exploration in improving the ease of use of the interpreter, and the interpretation efficiency of the interpreter needs to be further improved.

In order to improve the programming ability and execution efficiency of robot control system, an easy-to-use and efficient robot program interpreter is designed and implemented in this paper. The rest of the paper is organized as follows: Section 2 introduces the general design of the robot programming language and its interpreter. Section 3 uses relevant theories and methods proposed in this paper to implement the robot program interpreter. This part includes lexical analysis, syntactic analysis, semantic analysis and instruction interpretation. Section 4 verifies the accuracy, efficiency and stability of the robot program interpreter through experiment. Section 5 presents the analysis and conclusion of the experimental results.

2. General design

2.1. Design of robot programming language

According to the operation requirements of industrial robots, and in order to simplify the syntax of programming language and enrich the programming functions of robots, the robot programming language designed in this paper mainly includes ten data types and five control instruction groups. The data type is mainly composed of general basic data types, such as integer, floating-point, string, Boolean, and special data types for robot, such as position, reference coordinate system, tool coordinate system, motion parameters, input and output. The control instructions include the following five types of instruction groups.

- Motion command: PTP (point-to-point motion), Lin (linear motion), CIRC (circular motion), etc.
- Setting instructions: Dyn (setting motion parameters), Refsys (setting reference coordinate system), Tool (setting tool coordinate system), etc.
- System function instructions: WaitTime, Stop, CLOCK, etc.
- Process control instructions: Call (subroutine call), IF (conditional branch), LOOP, etc.
- Input and output instructions: DIN (input), DOUT (output), etc.

2.2. Design of architecture of program interpreter

The robot program interpreter in this paper is applied to the teaching software developed by the laboratory. In order to improve the usability of the interpreter and the user's programming experience, the interpreter makes the robot program consist of two parts. One part is the variable definition file, which contains the variable definition statements and is edited and generated by the variable management module of the teaching software. The other part is the instruction call file, which contains all kinds of instruction statements of the robot and is edited and generated by the program management module. An instruction call file and a variable definition file with the same name form a robot program, and one or more robot programs form a robot project. When robot language interpreter interprets the program, it takes the project as the unit. First, it analyzes the morphology and syntax, and constructs the syntax tree. Then, it carries out semantic analysis, creates the symbol table, detects and deal with the error types, and displays them in the information report module of the teaching software. Finally, it traverses the syntax tree, interprets the instructions, and calls the corresponding control function to make the robot perform the motion. The architecture of the robot interpreter is shown in Figure 1.

Dividing the robot program into variable definition and instruction call, the user only needs to make a few clicks and inputs in the variable management and program module of the teaching software to complete the generation of program code. On the one hand, it simplifies the process of programming, on the other hand, it greatly reduces the possibility of errors in the robot program. Each module in the interpreter is relatively independent and has a clear structure, which reduces the coupling between them and makes the system easier to expand and upgrade.

3. Theories and methods

3.1. Lexical analysis

The main task of lexical analysis is to scan the robot program character by character from left to right, recognize word symbols according to the lexical rules of robot programming language, and generate symbol sequences for syntactic analysis[9].

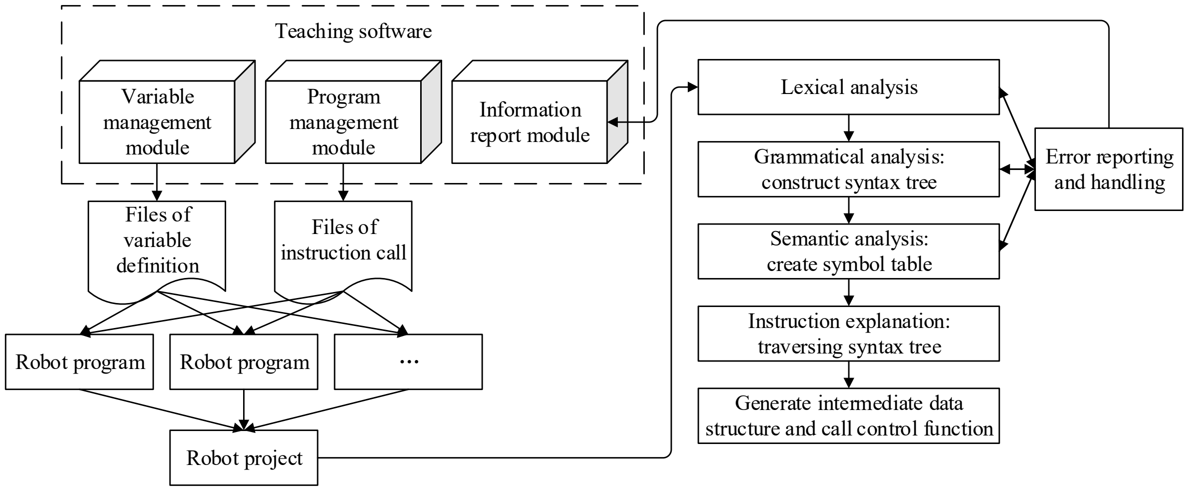

Teaching software
Variable management module
Program management module
Information report module
Files of variable definition
Files of instruction call
Robot program
Robot program
...
Robot project
Lexical analysis
Grammatical analysis: construct syntax tree
Semantic analysis: create symbol table
Instruction explanation: traversing syntax tree
Generate intermediate data structure and call control function
Error reporting and handling

Figure 1. Architecture of robot interpreter

The lexical analysis of this paper is realized by the assistance of flex tool software, which generates a lexical analysis program by compiling file of lexical rules. The flex file of lexical rules consists of declaration, translation rule and auxiliary process[10]. The declaration includes header file of functions, symbol declaration and its definition of regular expression. The translation rule consists of symbol and its corresponding action. The auxiliary process is the function code that creates syntax tree node for each symbol. In the robot lexical analyzer, whenever a declared symbol is matched, the syntax tree node of the symbol is created, and the symbol is returned to the parser. For variables, integer constants, floating-point constants, Boolean constants and strings, id, intgr, flt, bl and str symbols are returned respectively. For keywords of program statements, keyword symbols are returned directly. For operators and punctuations, their word names are returned as symbols. For unmatched characters, lexical errors are reported.

3.2. Syntactic analysis

The purpose of syntactic analysis is to identify the syntactic components from the symbol sequence of robot program according to the syntactic rules of robot programming language, and at the same time to check the syntax, so as to prepare for semantic analysis and instruction interpretation.

Constructing abstract syntax tree (AST) is an important task of syntactic analysis[11]. With the help of bison tool software, this paper generates a parser. According to the analysis method of LALR(1), starting from the symbol sequence generated by lexical analysis, a new sentence pattern is obtained by searching for the "reducible string" of the current sentence pattern and using rules to reduce it to the corresponding non terminal symbol. This reducing process is repeated until the statute reaches the beginning symbol "Program" of the syntax. And finally, from the bottom to the top, a syntax analysis tree is constructed.

The original syntactic structure of the robot program in this paper is shown in Figure 2. It is a multi-tree structure. "Program" represents the entry of the whole program. It includes two branches: "DefinitionList" and "InstructionList". They are composed of several specific variable definition statements named "Definition" and instruction call statements named "Instruction" respectively.

The storage form of syntax tree and the rationality of design of node structure are important factors that affect the performance of program interpreter. In this paper, the child-sibling notation (CSN) is proposed to construct the syntax tree. The implementation idea of the child-sibling notation is: Starting from the root node of the tree, the child node and sibling node of each node are stored in the chain list in turn. The design of structure of syntax tree node is shown in Figure 3, in which the semantic values corresponding to different data types are put in a union named "value" to save memory.

Taking the robot program in Figure 4 as an example, the program contains three variable definition statements and two instruction call statements. After syntactic analysis, the syntax tree as shown in

Figure 5 is constructed, in which "c" and "s" on the arrow point to child node and sibling node respectively. Figure 5 shows the structure of the first variable definition statement and the first instruction call statement, and the structure of other statements is similar, so omitted.

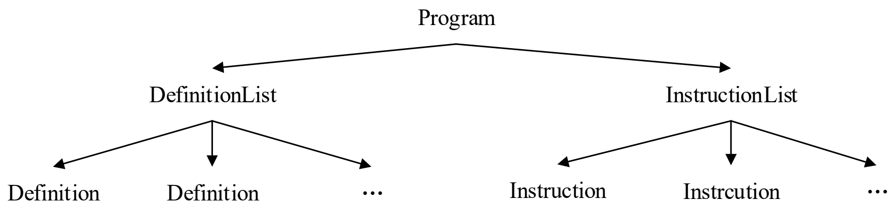


Figure 2. Original syntactic structure of robot program

```
struct AST
{
   struct AST *child; //Pointer to child node
   struct AST *sibling; //Pointer to sibling node
   string name; //Name of node
   string content; //Literal value of node
   string type; //type of node
   union{
      int intgr; //Integer value
      double flt; //Floating point value
      string str; //String value
      bool bl; //Boolean value
      struct AXISPOS axisPos; //Axis position
      struct CARTPOS cartPos; //Cartesian position
      struct TOOL tool; //Tool coordinate system
      struct CARTREFSYS cartRefSys; //Reference coordinate system
      struct DYNAMIC dynamic; //Motion parameter value
      struct DIO dio; //Input and output value
   }value; //The semantic value of the corresponding type of node
};
```

Figure 3. Structure of syntax tree node

```
//Variable definition
b:INTEGER=5 //Variable definition of integer
ap0:AXISPOS=(a1=10,a2=20,a3=30,a4=40,a5=50,a6=60) //Variable definition of axis position
cp0:CARTPOS=(x=100,y=200,z=300,a=20,b=40,c=60) //Variable definition of Cartesian position
//Instruction call
PTP(ap0) //Point-to-point motion
Lin(cp0) //Linear motion
```

Figure 4. Example of robot program

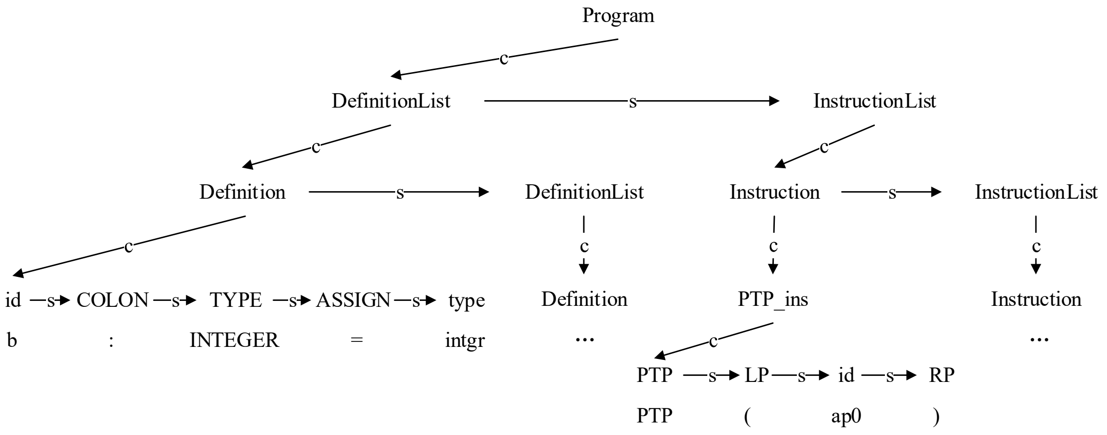


Figure 5. The syntax tree constructed by the child-sibling notation

In this paper, the syntax tree constructed by child-sibling notation (CSN) stores the syntactic structure of multi tree in the form of binary tree, which makes the construction, traversal and analysis of syntax tree more convenient and fast, and speeds up the interpretation efficiency of robot program interpreter. On the other hand, all syntax tree nodes adopt a unified data structure named "AST", which helps to reduce the complexity of interpretation algorithm.

3.3. Semantic analysis

Semantic analysis is an important work in the process of program interpretation. It links the definition and reference of variables in robot program to check whether each syntactic component has correct semantics[12, 13]. In the process of semantic analysis, we usually need to design a data structure called symbol table to save the related information in the context, and then perform type checking to report and deal with the errors in the program.

3.3.1. Establishment of symbol table

When analyzing variable definition statements in robot programs, for each newly defined variable, its literal value, data type and corresponding semantic value need to be recorded in the symbol table for subsequent search and use. Since the variable cannot be redefined, that is, the literal value of each variable should be different from each other, the literal value of the variable can be used as the unique identification of the variable. In this paper, the key-value pair of map container in C++ STL library is used as the storage structure of symbol table, the literal value of variable is used as the key, and the data type and semantic value of variable are encapsulated into a structure named "VAR" as the value shown in Figure 6, among which the value union is same as in "AST". They form a one-to-one relationship and are stored in the symbol table. Figure 7 is the UML class diagram of the map container. Variables can be inserted and searched in the symbol table by calling its insert and find member functions.

```
struct VAR
{
   string type; //type of node
   union{
         int intgr; //Integer value
         double flt; //Floating point value
         ...
   }value; //The semantic value of the corresponding type of node
};
```

Figure 6. var structure

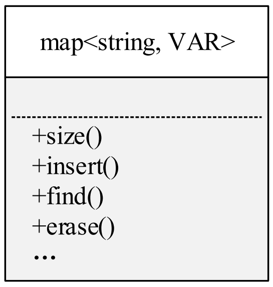


Figure 7. UML class diagram of map container

Because the internal structure of map container is red black tree, the average time complexity of inserting and searching elements on the red black tree is O(log n), which is less than the O(n) time complexity of symbol table with storage structure of linked list. Therefore, using map container as the storage structure of symbol table can make the creation and use of symbol table more efficient, and further improve the efficiency of robot program interpretation.

3.3.2. Error reporting and handling

In the teaching software, in addition to variable naming and specific value input, the user only needs to click to select a predetermined statement and insert it to complete the generation of robot program code, which greatly reduces the possibility of lexical and *syntactic* errors in the program, making the program errors concentrated in the semantic part, so error reporting and processing are particularly important in semantic analysis.

When analyzing the semantics of a robot program, we need to visit the symbol table many times to check whether the program conforms to the context sensitive features of the robot programming language according to the variable information recorded in the symbol table. In this paper, the error type checking code is preset in the process of constructing the syntax tree to report the locations and types of the errors in the robot program, and deal with the errors in time. Take a PTP statement for example, its processing flow is as follows:

- Step 1: Construct sub-syntax tree for the statement.
- Step 2: Find the record of the variable of the statement in the symbol table. If it doesn't exist, report type mismatch error and set the global variable "flag" to 1 indicating that there's error in the program.
- Step 3: If the record of the variable exists, assign the relevant information of the variable in the symbol table to the syntax tree node. And then judged whether the type of the variable is a position type, if not, report type mismatch error and also set the global variable "flag" to 1.
- Step 4: Continue to check the following statements, so as to ensure that the errors in the robot program can be detected at one time before the end of the process of semantic analysis.

This method of error reporting and handling is convenient for users to modify and improve the robot program according to the error prompts, and optimize the use experience of the robot program interpreter.

3.4. Instruction interpretation

After lexical, syntactic and semantic analysis of the robot program, if no error is reported, it means that the program can run normally. At this time, it is necessary to interpret the instructions in the program sentence by sentence, and call the corresponding control function to let the robot execute the motion.
The instruction interpretation takes the root node of the syntax tree as the entry, and uses the depth-first algorithm to traverse the syntax tree constructed by the child-sibling notation. When a certain instruction sentence is accessed, its corresponding interpretation function is called. The algorithm flow of instruction interpretation is shown in Figure 8.

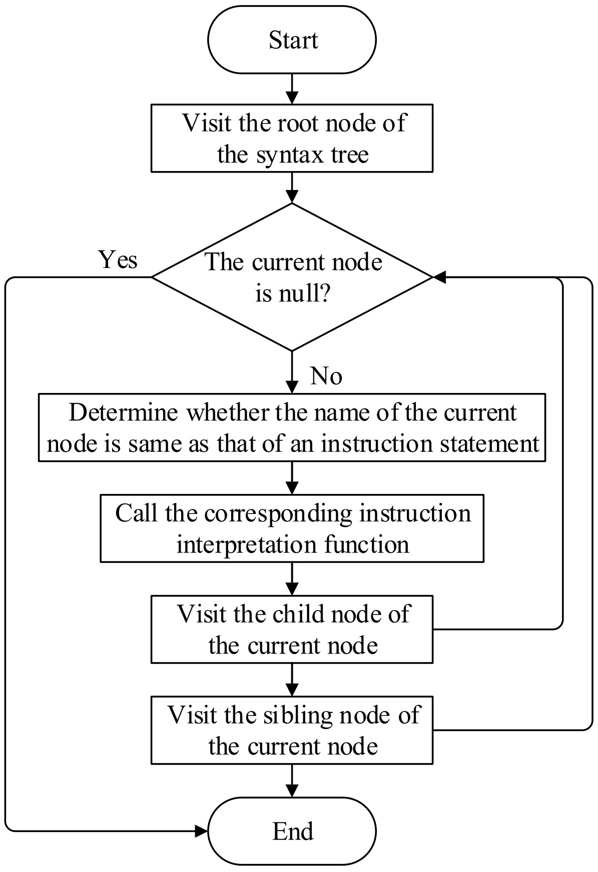

Start
Visit the root node of the syntax tree
Yes
The current node is null?
No
Determine whether the name of the current node is same as that of an instruction statement
Call the corresponding instruction interpretation function
Visit the child node of the current node
Visit the sibling node of the current node
End

Figure 8. Algorithm flow chart of instruction interpretation

In this paper, the robot program interpreter programs corresponding interpretation functions for each instruction statement. In the interpretation function, it will continue depth-first traversal of the syntax tree nodes that make up the instruction statement, and assign the semantic values of the nodes of instruction parameters to the intermediate data structures such as robot position and velocity, which are used as the actual parameters of the control function. And then, the control function is called to execute the motion of the robot. Using this way of interpreting the program while executing the motion can speed up the response speed of the robot when the program is running.

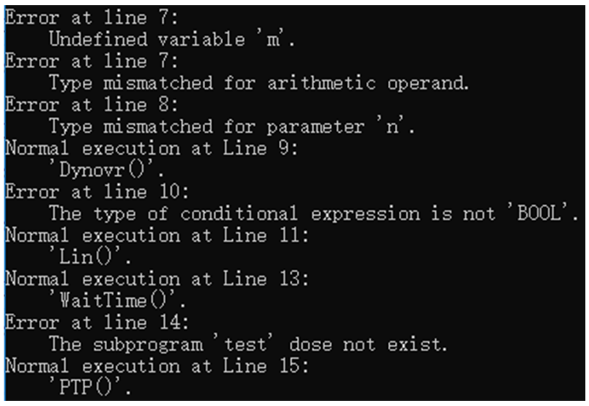
4. Experiments and results

In order to verify the effectiveness of the robot program interpreter designed in this paper, the robot program is tested to simulate the actual operation of the interpreter. For the wrong statements in the program, the error information should be printed, and for the correct instruction statements, the name of the instruction should be printed. After inputting the test program of Figure 9 into the robot language interpreter and running, the printed output information is shown in Figure 10. It can be seen that the robot language interpreter designed in this paper can accurately report the locations and types of errors in the robot program and deal with them. If there is no error in the program, it can execute the interpretation of the instruction statement and call the corresponding robot control function.

In order to test the interpretation efficiency of the robot program interpreter in this paper, several programs with different number of instruction statements are used as input, and the time required for the interpreter to interpret each program is calculated and recorded. The statistical chart is shown in Figure 11. It can be seen from the figure that the robot program interpreter designed and implemented in this paper can complete the interpretation of 5000 instruction sentences in 200 ms. And the interpretation time does not increase significantly when the number of instructions is multiplied, which reflects the high efficiency and stability of the interpretation.

```
//Variable definition
n:INTEGER=10
ap0:AIXSPOS=(a1=10,a2=20,a3=30,a4=40,a5=50,a6=60)
cp0:CARTPOS=(x=200,y=300,z=400,a=20,b=40,c=60)
dyn0:DYNAMIC=(velAxis=100,accAxis=1000,jerkAxis=2000)
//Instruction call
n=ap0+m //Wrong
Dyn(n) //Wrong
Dynovr(n) //Right
IF ap0 THEN //Wrong
  Lin(cp0) //Right
END_IF
WaitTime(10) //Right
CALL test //Wrong
PTP(ap0) //Right
```

Figure 9. Tested robot program



```
Error at line 7:
    Undefined variable 'm'.
Error at line 7:
    Type mismatched for arithmetic operand.
Error at line 8:
    Type mismatched for parameter 'n'.
Normal execution at Line 9:
    'Dynovr()'.
Error at line 10:
    The type of conditional expression is not 'BOOL'.
Normal execution at Line 11:
    'Lin()'.
Normal execution at Line 13:
    'WaitTime()'.
Error at line 14:
    The subprogram 'test' dose not exist.
Normal execution at Line 15:
    'PTP()'.
```

Figure 10. Output of the tested program

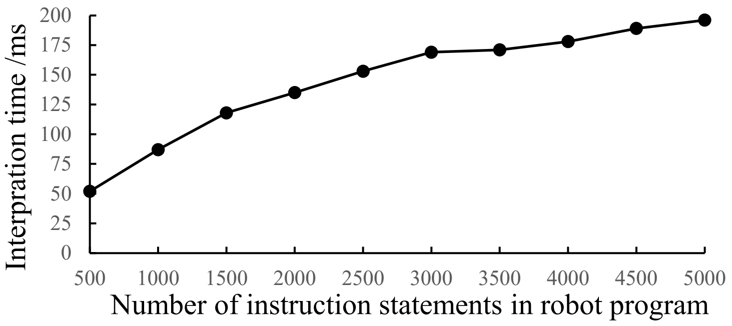


Figure 11. Interpretation time of robot program

5. Conclusion

In this paper, a robot programming language is designed and its efficient interpreter is implemented. The architecture of interpreter divides robot program into two parts: variable definition and instruction call. The user only needs to make a few clicks and inputs in the corresponding management module to complete the generation of program code, which simplifies the process of programming and greatly reduces the possibility of errors in the robot program. Using flex and bison tools to assist of generating lexical and syntactic analysis program, the development efficiency and code reliability are improved. The syntax tree is constructed by the child-sibling notation (CSN) proposed in this paper. The unified node structure of the syntax tree makes the storage, traversal and analysis of the syntax tree more convenient and efficient, and reduce the complexity of the interpretation algorithm. In semantic analysis, the red-black tree structure of map container is used as the storage structure of symbol table, which makes it easier and faster to insert and find variables in symbol table, and further improves the interpretation efficiency of robot program. In the process of constructing the syntax tree, the type checking code is preset, which can report the locations and types of all errors and deal with them in the robot program at one time. It is convenient for users to modify and improve the robot program according to the error prompts. By traversing the syntax tree with depth-first algorithm, the corresponding control functions are called while interpreting the instruction statements, which improves the response speed of the robot in the execution of program.

Experiments show that the interpreter designed and implemented in this paper can detect and deal with errors in the robot program quickly and accurately. And it can interpret and execute the robot instructions effectively and steadily. At present, the robot program interpreter has been applied in the self-developed teaching software, which meets the actual needs of robot operation. This article has important significance and reference value for further research on robot program interpretation.

Acknowledgments
This work was supported in part by the Science and Technology Planning Project of Guangdong Province, Grant No. 2017B090914002 and the National Key R&D Program of China, Grant No. 2018YFB1700500.

References
[1] Wang, Z., Ma X. (2015) Design and implementation of industrial robot language interpreter. Industrial Control Computer, 28: 6-8.
[2] Wang, Y., Li, L. (2013) Implementation of industrial Robot interpreter of open architecture. Machinery Design & Manufacture, 51: 253-255.
[3] Hong, H., Yu, D., Zhang, X., Chen, L. (2010) Research on a new model of numerical control program interpreter. In: IEEE International Conference on Advanced Computer Control. Shenyang. 467-472.
[4] Quante, J. (2016) A program interpreter framework for arbitrary abstractions. In: IEEE International Working Conference on Source Code Analysis and Manipulation. Raleigh. 91-96.
[5] Liu, L., Yao, Y., Du, J. (2019) A universal and scalable CNC interpreter for CNC systems. International Journal of Advanced Manufacturing Technology, 103: 4453-4466.
[6] Yang, X., Chen, F., Zhou, Fei. (2015) Design of the industry robot language system based on Qt. Modular Machine Tool & Automatic Manufacturing Technique, 57: 71-74.
[7] Wang, F., Lv, C. (2018) Design and implementation of interpreter for industrial robot Machine Building & Automation, 47: 177-180.
[8] Chen, X., Chen, F. (2019) The study and implement of industrial robot language and interpreter. Modular Machine Tool & Automatic Manufacturing Technique, 61: 111-113 + 118.
[9] Wilhelm, R., Seidl, H., Hack, S. (2013) Compiler design: Syntactic and semantic analysis. Springer Science & Business Media, Berlin.
[10] Li, W. (2016) Principles and Techniques of Compilation. Tsinghua University Press, Beijing.
[11] Levine, J. (2010) flex & bison. Southeast University Press, Nanjing.
[12] Fang, D. (2018) Development of Static Code Defect Detection Tool Based on Abstract Syntax Tree. Beijing University of Posts and Telecommunications, Beijing.
[13] Lu, T., Wang, Z., He, G., Wang, Yi. (2015) Design and implement of new industrial robot language and interpreter. Industrial Control Computer, 28: 33-34.