南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

COMPILIER @Liu Yepang 2019

for SUSTech CSE

Project 3

Edited by

汪至圆

11610634

2019

SHENZHEN

1 Lab Demand

In last two project, I have finished the parser for the SPL source code and generated the syntex tree and do the semantic analysis for the SPL source code. In this project I will generate the intermediate code for my syntex tree.

2 Environment

The environment for my project is g++ 9.1, flex 2.6, bison 3.4, urwid 2.1. All the work of coding and test was finished in the Manjaro 18.1 which based on Linux 4.19.

3 Run

My project was compiled by the make. If you want to compile the project, you just need run 'make splc'. And then it will output a executable file splc.out. The splc.out will receive a argument which is the path of SPL source code file, and the output of the program will at the same folder of the source code and the extend name of the file will be ".ir". The context of the output file will be the intermediate-code.

4 Generate Process

When generate the intermediate-code, I use the translation schemes introduced in the document. I define the translation schemes for the expression, the condition, the statement, the arguments and the CompSt node. For the assumptions in the document, I traversal the node and find all the definitions of the function and apply the translation schemes to it. For the translation schemes for CompSt node, I will do the declaration for the variable and add the variable to the variable map which storage the correspondence information of the variable name and the name I used in the intermediate-code. And for the Stat node in the CompSt node, I will apply the statement translation scheme to it, and in the statement translation scheme, there will use other translation schemes to translate specific node.

5 Optimization

After generated the intermediate-code, I find there are many repeated statements, so I want to optimize it.

- For all the expression with a const value or a single variable which has been defined, the expression translation scheme won't return the code to set a code to define a new variable but a string with length 0, and it will set the value of the expression be the expression node be the const value or the name used in the intermediate-code of the single variable. And while using this expression node in the right value expression, we can use the value of it directly.

- In the intermediate-code, we will use so many temporary variables to transfer the value, and most of them are so redundant. In these step, I will find and analysis all the temporary variables. If the temporary which won't be used in the code below it, I will delete the line define the temporary variable. If the temporary variable just used in the right value individually, I will delete the line which used it and set the left value of the line define the temporary variable be the left value of the line be deleted.
- And I find, there are may be some line which used to define the variable has right value consist of immediate value, for these lines, I will give them the result of the expression immediately.
- There are some lines have a feature, they have the same left value, and in their right value, there are only the left value of them will appear except the immediate value. For these lines, I will compress them to one line.
- The last one is a very useful optimization but it can't implement because of the limitation of the simulation. In the simulation, the immediate value can't be used in the logical calculation, so we need to define a variable with the value to do the calculation, it's so wasteful to define a variable for each const value in the logical calculation. So I want to set a const value pool, all the const values which appear in the logical expressions in the program will be distributed to a variable, and when we need to use the const value, we can use the corresponding variable. But this optimization can't be implemented because that the simulation don't support global variable. Maybe can create a const value pool for each function, but it is also wasteful.

5.1 Performance

For the test case privaded by the TA, I test their performance.

Test Case	Input	Number of Instructions
test_3_r01		20
test_3_r02	2019,10	26
test_3_r03		3715
test_3_r04	10	18
test_3_r05		17
test_3_r06		124
test_3_r07	5,10	45
test_3_r08	-10	13
test_3_r09		52212
test_3_r10	10	95

6 Bonus

I haven't implemented bonus features yet.