CSE 401 Project Report, Spring 2022

ag

Vishnu Kartha, Varun Pradeep

vkartha, varun10p

We used 1 late day for the semantics portion of the project, 2 late days for the type checking version of the project, and 2 late days for the code generation portion but we also made changes to it a day later which we will detail. The version of code generation that was submitted and tagged on time was version 7ce4b728; we will refer to this as CG1. We made additional changes to our code on Tuesday 5/31 which is the fully updated version 4b43ab1e which currently has the codegen-final tag; we will refer to this as CG2.

Working/Broken Language Features:

In CG1, most of the MiniJava language features work. Of course we had a working scanner as well as a parser according to the BNF of MiniJava. In terms of the code generation, we were able to get arithmetic expressions and boolean expressions to work. We were able to implement variables, parameters, and assignment with a few caveats. We had working control flow. We were able to correctly implement extended classes and dynamic dispatch. Finally, we were able to do arrays correctly. However, this version had its issues. One of the issues was when we had multiple variables that were of the same Class Type. This was because of the way we were storing offsets which was in the ClassType itself, and when the ClassTableBuilder found a new variable with the same type we believe this was causing it to overwrite the previous variables offset which had the same time. We took a safety first approach and solved it in CG2. The other issue we had was with “this” which was more of an accidental issue. We realized that we were accidentally prefixing the offset with “-” which made the instruction look for the memory in the wrong direction from (%rdi) which is where “this” is stored. We made the obvious fix in CG2.

In CG2, we believe all of the required features for MiniJava are working. In this version we fixed the bugs that were happening in CG1. To fix the variable offsets issue, to make sure they would not get overwritten, we added maps to both the ClassTable and SymbolTable which stored the offsets as values where the keys were the identifier names for variables. This map was built when we were building the SymbolTables and we used it to get variable offsets in the code generation. To fix the problem with the negative offsets for getting class variables we just simply removed “-” from our assembly instructions where necessary. The next change we made was to the error function we call when there is an array index error. One issue was that it tried to continue to run the problem even when it called this function. We solved this by calling exit(1) in the C file which is just System Exit with error status. We also changed the error to clearly print “Array Index Out of Bounds Error.” We made no other functional changes but we did tidy up the code by factoring out common chunks of code and getting rid of unnecessary lines and creating better-named variables.

Testing

For the initial parts of the project we tried to develop brief automated test suites just to do some verification. The majority of our testing was functional testing as we went along through the project, we would make short java files that uses language features we wanted to test. For the code generation, due to time crunch and lack of ideas no how to make an automated test, we most did functional testing of language features which we did by having a minimal Java file which used the feature that we wanted to test. Then since we did not do automated testing, we felt as though the provided Sample Java Programs were sufficiently complex to do testing on so we produced code generation for those files and compared them to the output of the files compiled with javac and they were the same.

Additional Features:

We did not implement any features other than the core Minijava features or any compiler optimizations as we prioritized getting a working version. However, if we did have more time, we see the potential to optimize some instruction. For example, when we do multiplying by 8 we could have done shlq $8,[register].

Work Split:

We employed several different collaboration techniques to split up the work in this project. Initially, we gave each other different tasks and communicated over discord when problems arose. When there were large problems we would hop in a call and debug together so that we could be more effective. We would have one person share the screen so the other person could follow along and offer suggestions in real time. Later on through the project, we discovered the Intellij code with me plugin so that we could code together in real time. This made our work more interactive and efficient as we were able to bounce ideas off of each other. All in all, we believe that the collaboration went well for this project.

Conclusion:

Some things that went well were our communication with each other, talking to the TAs when problems arose, and the final quality of the project. The TAs were extremely forthcoming about how they graded the TypeChecking in particular and we were able to get regraded fairly. Somethings which could have been done better are the extensiveness of the testing, time management of the code gen, and not being able to utilize the opportunity to add additional features. If we were to do the project again, we would have liked more proactive grading of the checkpoints as we found that errors in the previous checkpoints in the project would carry over to later ones as we were not given prompt feedback.

Nonetheless, in the future we would start code generation much sooner as it was incredibly time consuming to identify the causes of some of the bugs we were getting.