EC3204: Programming Languages and Compilers (Fall 2023)
Homework 4: Translator and Optimizer

100 points for each, 30% of the total score
Due: 12/12, 23:59 (submit via GIST LMS)

Instructor: Sunbeom So

Important Notes

- **No Plagiarism**
  Cheating (i.e., copying assignments by any means) will get you 0 points for the entire HWs. See the slides for Lecture 0. Code-clone checking will be conducted irregularly.
- **Executable**
  Before you submit your code, please make sure that your code can be successfully compiled by the OCaml compiler. That is, the command `./build` should not report any errors. Otherwise, you will get 0 points for that HW.
- **No Template Changes**
  Your job is to complete `translator.ml` and `optimizer.ml`. However, you should not modify the other existing code templates.
- **No Printing Functions**
  The parts implemented by you should not contain printing functions such as `print_int`.
- **File Extension**
  The submitted files should have `.ml` extensions, not the others (e.g., `.pdf`).
- **File Naming Rule**
  Submit `translator.ml` and `optimizer.ml` only. Do not change the file names. Do not zip the files.

1 Assignment Summary

Your goal in this assignment is to implement a translator (10%) and an optimizer (20%) for the source and target languages in lecture 18.

2 Structure of the Project

You can find the following files in the `hw4` directory.

- `main.ml`: contains the driver code.
- `s.ml`: contains the definition of the abstract syntax and the interpreter for our source language ("S").
- `lexer.mll`: replace this file with your lexer specification from hw3.
- `parser.mly`: replace this file with your parser specification from hw3.
- `t.ml`: contains the definition of the abstract syntax and the interpreter for our target language ("T").
- `translator.ml`: should contain your implementation that translates the source program (the program written in S) into the target program (the program written in T).
- `optimizer.ml`: should contain your implementation that optimizes the target program passed from `translator.ml`.

Your job is to complete `translator.ml` (10%) and `optimizer.ml` (20%). Submit these two files only.

3 How to Run

Once you completed `translator.ml` and `optimizer.ml`, you can build the project as follows.

```
$ make
```

Then, the executable `run` will be generated. To remove some dummy files (not mandatory), run the following command:

```
$ make clean
```

Your executable `run` should output a correct value for a given source program. For example, if you run the command

```
$ ./run test/t0.s
```

the executable `run` should output the value 1 as follows.

```
== source program ==
{
 int x;
 x = 0;
 print x+1;
}
== execution result (source) ==
1
== target program ==
0 : x = 0
0 : t4 = 0
0 : x = t4
0 : t1 = x
0 : t2 = 1
0 : t3 = t1 + t2
0 : write t3
0 : HALT
== execution result (translated) ==
1
#instructions executed: 7
== optimized target program ==
0 : t3 = 1
0 : write t3
0 : HALT
== execution result (optimized) ==
1
#instructions executed: 2
```

Note that different optimized target programs are acceptable as long as they produce the same results as before optimization. For example, the following optimize program is acceptable although suboptimal (hence full credits may not be given).

```
== optimized target program ==
0 : x = 0
0 : t4 = 0
0 : x = t4
0 : t1 = x
0 : t3 = t1 + 1
0 : write t3
0 : HALT
== execution result (optimized) ==
1
#instructions executed: 6
```

4 Evaluation Criteria

4.1 Translator (100 points, 10%)

The correctness of your code will be evaluated using testcases:

$$\frac{\#\text{Passed}}{\#\text{Total}} \times \text{point per problem}$$

- "Total" indicates a set of testcases prepared by the instructor (undisclosed before the evaluation).
- "Passed" indicates testcases whose expected outputs match with the outputs produced by your implementations.

4.2 Optimizer (100 points, 20%)

Relative evaluation will be introduced. Your code will be evaluated based on the correctness (i.e., the target program and optimized program should output the same results) and optimization performance (in terms of the reduction in the number of instructions being executed).

- Preserve the semantics for all testcases && Top 30% in terms of instruction reduction: 100 points
- Preserve the semantics for all testcases && Top 70% in terms of instruction reduction: 90 points
- Preserve the semantics for all testcases && Bottom 30% in terms of instruction reduction: 80 points
- Fails to preserve the semantics (i.e., the target program and optimized program output the different results) for some testcase && Produce the optimized program for some testcase: 50 points
- The rest: 0 points