EC3204: Programming Languages and Compilers (Fall 2024)
Homework 3: Translator and Optimizer

Due: 12/28, 14:00 (submit on GIST LMS)

Instructor: Sunbeom So

Important Notes

- **Evaluation criteria**
  See Chapter 5 of this document.
- **Compilable**
  Make sure that your submission is successfully compiled. If your code cannot be compiled, you will get 0 points for the corresponding HW.
- **Academic Integrity**
  Violating academic integrity will result in an F, even after the end of the semester.

  Click here to check the rules related to academic integrity.

  Be aware that proving your academic integrity is entirely your responsibility.
- **No Changes on Templates, File Names, and File Extensions**
  Your job is to complete `(* TODO *)` parts in provided code templates. You should not modify the other templates. Do not change the file names. The submitted files should have `.ml` extensions, not the others (e.g., `.pdf`, `.zip`, `.tar`).
- **Regarding Using LLM-based Tools**
  You can use LLM-based tools to complete your HW, as long as (1) you use LLMs by yourself, (2) you can reproduce all the steps assisted by LLMs, and (3) you can clearly explain all the details of your implementation (including LLM-generated parts).

  Even though you claim that you have completed HW with the help of LLMs, if your submission is highly similar to other students' code and you cannot reproduce certain steps by any reason (including the nondeterminism of LLMs), I will consider that you cheated on your assignments.

1 Goal

Your goal is to implement an `S`-to-`T` translator and an optimizer for `T` programs.

2 Structure of the Project

You can find the following files in the directory `hw3/code`.

- `main.ml`: contains the driver code.
- `s.ml`: contains the abstract syntax (p.8) and the interpreter (including the semantic evaluation rules in p.12–13) of the source language `S` covered in Lecture 14.
- `t.ml`: contains the abstract syntax (p.14) and the interpreter (p.16–17) of the target IR language `T` covered in Lecture 14.
- `translator.ml`: implements a translator that converts an `S` program into an equivalent `T` program. **Your job is to replace "raise NotImplemented" with your implementation based on the translation procedure in Lecture 14.** You can add your own helper functions.
- `optimizer.ml`: implements an optimizer that transforms a given T program into an equivalent yet more efficient (simpler) `T` program. **Your job is to replace "raise NotImplemented" with your implementation based on Lecture 17.** You can add your own helper functions.

3 How to Build

(1) Install the dependencies using the following commands.

```
$ sudo apt-get update
$ sudo apt-get install -y opam ocamlbuild ocaml-findlib
$ opam init
$ eval $(opam env)
$ opam update
$ opam install -y batteries
```

(2) After completing `translator.ml` and `optimizer.ml`, to build the project, run the following command in the directory `hw3/code`.

```
$ make
```

Then, the executable `run` will be generated. You can run it as follows.

```
$ ./run TESTFILE
```

where `TESTFILE` is a file containing an `S` program. Examples of `S` program files can be found in the directory `hw3/code/test`.

Optionally, to remove dummy files after the build, run the following command:

```
$ make clean
```

4 Running Example

For example, if you run the command

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

you may obtain the following results:

```
1  == source program ==
2  {
3   int x;
4   x = 0;
5   print x+1;
6  }
7  == execution result (source) ==
8  1
9  == target program ==
10 0 : x = 0
11 0 : t4 = 0
12 0 : x = t4
13 0 : t1 = x
14 0 : t2 = 1
15 0 : t3 = t1 + t2
16 0 : write t3
17 0 : HALT
18 == execution result (translated) ==
19 1
20 #instructions executed: 7
21 == optimized target program ==
22 0 : t3 = 1
23 0 : write t3
24 0 : HALT
25 == execution result (optimized) ==
26 1
27 #instructions executed: 2
```

where line 8 is the execution result of the `S` program (lines 2–6) in `test/t0.s`, line 19 is the execution result of the `T` program (lines 10–17) produced by your translator (`translator.ml`), and line 26 is the execution result of the optimized `T` program (lines 22–24) produced by your optimizer (`optimizer.ml`).

Note that syntactically different optimized `T` programs are acceptable too, as long as they are semantically equivalent to a given `S` program (i.e., they produce the same results with a given `S program`). For example, the following optimized `T` program is acceptable as it outputs `1` (line 11), although not the minimum size.

```
1 ...
2 == optimized target program ==
3 0 : x = 0
4 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
```

5 Evaluation Criteria

The quality of your code will be evaluated using testcases:

$$\frac{\sum_{i=1}^{\#\text{Total}} \mathsf{score}(\texttt{trans}, \texttt{opt}, s_i)}{\#\text{Total}} \times 100$$

where #Total is the number of testcases, s_i is a testcase (an `S` program), `trans` and `opt` are the translator (`translator.ml`) and the optimizer (`optimizer.ml`) implemented by you. score is a function that evaluates your implementations based on the correctness (semantics-preservation) and the optimization effectiveness (in terms of `#instructions executed`):

$$\mathsf{score}(\texttt{trans}, \texttt{opt}, s_i) = \begin{cases} 1 & \text{if } [\![\texttt{trans}(s_i)]\!] = [\![s_i]\!],\ [\![\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{i}))]\!] = [\![s_i]\!],\ 1 - \frac{|\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{i}))|}{|\texttt{trans}(s_i)|} \geq 0.4 \\ 0.9 & \text{if } [\![\texttt{trans}(s_i)]\!] = [\![s_i]\!],\ [\![\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{i}))]\!] = [\![s_i]\!],\ 0.3 \leq 1 - \frac{|\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{i}))|}{|\texttt{trans}(s_i)|} < 0.4 \\ 0.8 & \text{if } [\![\texttt{trans}(s_i)]\!] = [\![s_i]\!],\ [\![\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{i}))]\!] = [\![s_i]\!],\ 0.2 \leq 1 - \frac{|\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{i}))|}{|\texttt{trans}(s_i)|} < 0.3 \\ 0.7 & \text{if } [\![\texttt{trans}(s_i)]\!] = [\![s_i]\!],\ [\![\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{i}))]\!] = [\![s_i]\!],\ 0.1 \leq 1 - \frac{|\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{i}))|}{|\texttt{trans}(s_i)|} < 0.2 \\ 0.6 & \text{if } [\![\texttt{trans}(s_i)]\!] = [\![s_i]\!],\ [\![\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{i}))]\!] = [\![s_i]\!],\ 0 < 1 - \frac{|\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{i}))|}{|\texttt{trans}(s_i)|} < 0.1 \\ 0.5 & \text{if } [\![\texttt{trans}(s_i)]\!] = [\![s_i]\!],\ [\![\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{i}))]\!] = [\![s_i]\!],\ 1 - \frac{|\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{i}))|}{|\texttt{trans}(s_i)|} = 0 \\ 0.0 & \text{otherwise} \end{cases}$$

In the above, $[\![x]\!]$ means the execution result of x, and $1 - \frac{|\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{i}))|}{|\texttt{trans}(s_i)|}$ denotes the reduction ratio of the executed instructions.

For instance, in our first running example, $\mathsf{score}(\texttt{trans}, \texttt{opt}, s_1)$, where $s_1 = \texttt{test/t0.s}$, outputs 1. This is because $[\![s_1]\!] = 1$, $[\![\texttt{trans}(s_1)]\!] = 1$, $[\![\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{1}))]\!] = 1$, and $1 - \frac{|\texttt{opt}(\texttt{trans}(\texttt{s}_\texttt{1}))|}{|\texttt{trans}(s_1)|} = 1 - \frac{2}{7} \geq 0.4$.

Observe that, if your optimizer just returns a given input `T` program (i.e., if your optimizer is the identity function), you can still obtain 50 points.