Course Project
Phase #3. Optimization

Prof. Jaeseung Choi

Dept. of Computer Science and Engineering

Sogang University

General Information

- **Check "Project #3" in *Assignment* tab of *Cyber Campus***
  - Skeleton code (**`Prj3.tgz`**) is attached in the post
    - Test cases and checker script will be updated by next Monday
  - Submission will be accepted in the same post
  - **Deadline: 12/26 Tue. 23:59**
    - Sorry for taking away your Christmas holiday ☹
    - **No late submission this time**
- **Please read the instructions in this slide carefully**
  - Important information and submission guidelines are included
  - **This time, you will have to submit two files**

And let me warn you that this phase will be difficult

Remind: Course Policy

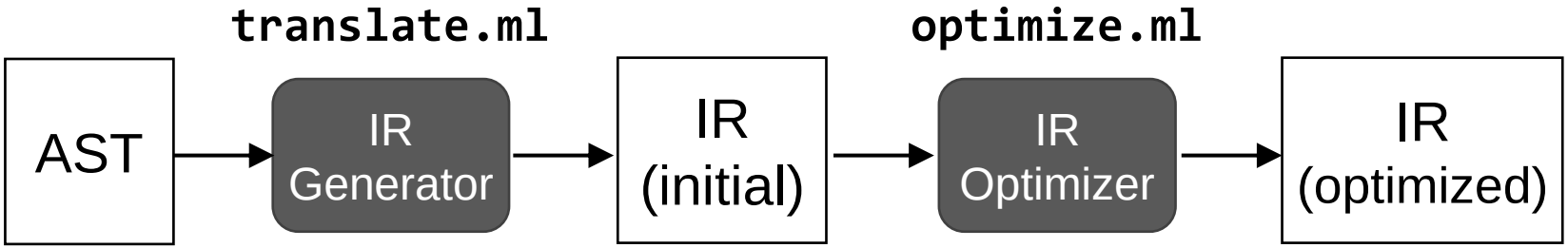
- **Cheating (code copy) is strictly forbidden in this course**
  - Read the orientation slide once more
- **Don’t ask for solutions in the online community**
  - TA will regularly monitor the communities
- **Don’t ask ChatGPT to write your code**
- **Even after the end of the course, please do not upload your code at GitHub or share it with your friends**
  - This makes it hard to manage the course in the following years

Skeleton Code

- **Copy Prj3.tgz into CSPRO server and decompress it**
  - You can use cspro**5**.sogang.ac.kr / cspro.sogang.ac.kr
- **`src/`: Source files you have to work with**
- **`Makefile`: Type `make` to build the whole project**
  - Internally redirects to **`src/Makefile`**
- **The following three components will be updated later**
  - **`testcase/`:** Sample test cases and their answers
  - **`check.py`:** Script for self-grading with test cases
  - **`config`:** Used by the grading script (you can ignore)

Outline

- **In this phase, you will implement IR-level optimization**
  - Make the IR code more efficient, while preserving its behavior
- **We use the same language and IR executor to phase #2**
  - But this time, **IR executor will measure and report the execution cost of your code**
  - This *"cost"* simulates the execution time of the code
- **The project structure is almost the same to phase #2**
  - `optimize.ml` file is newly added for optimization



Where do I have to read and fix?

- **First, read the `cost.ml` file added in `src/`**
  - This file defines the execution cost for each IR instruction
  - Ex) **`Set`** instruction is cheaper than **`Copy`** instruction
- **And your have to fill in `translate.ml` and `optimize.ml`**
  - You will **submit these two files**, and the whole code must compile when I copy your files into the skeleton code
  - In `translate.ml`, you have to implement the same function as before (you can start with the file that you submitted in phase #2)

    ```
    let run (p: program) : ir_code = ...
    ```
  - In `optimize.ml`, you have to implement the optimization logic

    ```
    let run (ir: ir_code) : ir_code = ...
    ```

Modes in `main.bin`

- **Once you compile the skeleton code and run it, it will print out the usage as follow**
  - There are total five modes supported in this phase
  - **`print-opt`** and **`run-opt`** modes are added for this phase

```
$ make
$ ./main.bin
<Usage>
[*]./main.bin print-ast <source file>
[*]./main.bin print-ir <source file>
[*]./main.bin run-ir <source file> <input file>
[*]./main.bin print-opt <source file>
[*]./main.bin run-opt <source file> <input file>
```

Printing the Optimized IR

- **If you run the `print-opt` mode, source program will be translated, optimized, and then printed out**
  - Same interface with **`print-ir`** mode in the previous phase
  - Provided **`translate.ml`** currently generates dummy IR code
  - Provided **`optimize.ml`** currently returns the input IR without performing any optimization

```
$ ./main.bin print-opt testcase/prog-1
f(i, b) : [
  $r0 = 0,
  ret $r0
]
```

Executing the Optimized IR

- **If you run the `run-opt` mode, source program will be translated, optimized, and then executed**
  - Same interface with **`run-ir`** mode in the previous phase
  - This time, the executor will print out the **execution result** and the **execution cost**
  - By optimizing your IR code to have **less instructions** and use **cheaper instructions**, you can reduce the execution cost

```
$ ./main.bin run-opt testcase/prog-1 testcase/inp-1
Result: 0, Cost: 3
...
```

Optimizations to Implement

- **We will focus on the optimization covered in the lecture**
  - Mem2Reg (★)
  - Constant folding
  - Constant propagation
  - Copy propagation
  - Common subexpression elimination
  - Dead-code elimination
- **Among these, Mem2Reg optimization is the easiest one to implement, and will have the most notable impact**
- **If you correctly implement all these optimizations, then of course you will get the full point**
  - Even if you miss 1~2 optimizations, you will still get many points

Tips for IR Translation

- **You can also fix `translate.ml` file to improve the efficiency of the generated IR code**
  - For example, think about more efficient implementation of short-circuit evaluation in IR code
- **In our project, Mem2Reg can be easily implemented in `translate.ml` file**
  - During the translation to IR, you can put all the non-array variables in registers (instead of memory)
  - But remember that this is only possible in our project, because our source language does not have pointers (`&x`, `*p`)

Evaluation Criteria

- **I will use 5~8 test programs to grade your compiler, and each program will be executed with N inputs**
  - **(Correctness)** If your IR code returns wrong output for any of the *N inputs*, you lose the whole point for that test program
  - **(Performance)** If your IR code returns correct output for all the inputs, then your score will be decided by the execution cost
- **For each test program, I will set two reference points**
  - Ex) If (the sum of) execution cost < **200**, you will get full point
  - Ex) If execution cost > **400**, you will get zero point
  - Ex) If execution cost is between these two reference points, you will get partial point

Self-Grading

- I am still working on preparing the test cases and setting their reference points
- Around next Monday (12/18), I will upload the test cases and checker script (`check.py`) for self-grading
- When you run `check.py` this time, it will print out the performance score for each test program

Submission Guideline

- **You should submit the following two files (be careful not to submit compile by-product files like `*.cmo`)**
  - **`translate.ml`**
  - **`optimize.ml`**
- **Submission format**
  - Upload these files directly to *Cyber Campus* (**do not zip them**)
  - **Do not change the file name** (e.g., adding any prefix or suffix)
  - If your submission format is wrong, you will get **-20% penalty**