ECS 140A Programming Languages

Fall 2022

Homework 5

About This Assignment

- This assignment asks you to complete programming tasks using the Go programming language.
- **You are only allowed to use the subset of Go that we have discussed in class. No credit will be given in this assignment if any of the problem solutions use material not discussed in class.** Please use Piazza for any clarifications regarding this issue.
  - Especially, the use of the `runtime` and `reflect` packages is forbidden.
  - Although the package `time` is discussed in class, you are NOT allowed to use it in this homework (e.g. `time.Sleep`).
- **Since the focus of this homework is concurrency, homework solutions are required to be concurrent in order to receive credit.**
- To complete the assignment (i) download `hw5-handout.zip` from Canvas, (ii) modify the `members.txt` and `.go` files in the `hw5-handout` directory as per the instructions in this document, and (iii) zip the `hw5-handout` directory into `hw5-handout.zip` and upload this `zip` file to Canvas by the due date.

  Do not change the file names, create new files, or change the directory structure of `hw5-handout`.
- This assignment can be worked on in a group. Even if working in a group, each student has to submit a copy of the work to Canvas to get credit. See the Syllabus for more details.
- List all the names and email addresses of all members of the group in the `members.txt` file in the `hw5-handout` directory, one per line in the format `name <email>`.

  If you are working individually, then only add your name and email to `members.txt`.
- Refer to **Homework 0** for instructions on installing the correct versions of the programming language as well as using CSIF computers.
- Begin working on the homework early.
- Apart from the description in this document, look at the unit tests provided to understand the requirements for the code you have to write.

- Post questions on Piazza if you require any further clarifications. Use private posts if your question contains part of the solution to the homework.
- Keep your homework solution after you submit it. You may need to use it for later assignments.

1 Bug1 (10 points)

- Modify code in `hw5-handout/bug1/bug1.go` to fix the concurrency bug.
- Some unit tests are provided for you in `hw5-handout/bug1/bug1_test.go`. **You are not allowed to make changes in `hw5-handout/bug1/bug1_test.go` to fix the bug.**

  From the `hw5-handout/bug1` directory, run the `go test` command to run the unit tests.

  Run the `go test -race` command to run the unit tests with the race detector,[1] a tool for finding race conditions in Go code.
- From the `hw5-handout/bug1` directory, run `go test -cover` command to see the current code coverage.
- If needed, add more unit tests in `hw5-handout/bug1/bug1_test.go` to get 100% code coverage for the code in `hw5-handout/bug1/bug1.go`.
- From the `hw5-handout/bug1` directory, run the following two commands to see which statements are covered by the unit tests:

```
$ go test -coverprofile=temp.cov
$ go tool cover -html=temp.cov
```

2 NFA (20 points)

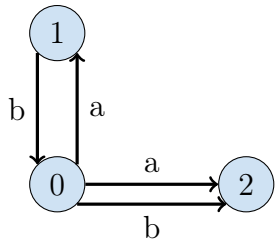
A nondeterministic finite automaton (NFA) is defined by a set of states, symbols in an alphabet, and a transition function. A state is represented by an integer. A symbol is represented by a rune, i.e., a character. Given a state and a symbol, a transition function returns the set of states that the NFA can transition to after reading the given symbol. This set of next states could be empty.

A graphical representation of an NFA is shown below:

In this example, $\{0, 1, 2\}$ are the set of states, $\{a, b\}$ are the set of symbols, and the transition function is represented by labelled arrows between states.

- If the NFA is in state 0 and it reads the symbol a, then it can transition to either state 1 or to state 2.

[1] https://blog.golang.org/race-detector



- If the NFA is in state 0 and it reads the symbol b, then it can only transition to state 2.
- If the NFA is in state 1 and it reads the symbol b, then it can only transition to state 0.
- If the NFA is in state 1 and it reads the symbol a, it cannot make any transitions.
- If the NFA is in state 2 and it reads the symbol a or b, it cannot make any transitions.

A given final state is said to be reachable from a given start state via a given input sequence of symbols if there exists a sequence of transitions such that if the NFA starts at the start state it would reach the final state after reading the entire sequence of input symbols.

In the example NFA above,

- The state 1 is reachable from the state 0 via the input sequence *abababa*.
- The state 1 is *not* reachable from the state 0 via the input sequence *ababab*.
- The state 2 is reachable from state 0 via the input sequence *abababa*.

The transition function for the NFA described above is represented by the expTransitions function in `hw5-handout/nfa/nfa_test.go`. Some unit tests have also been given to you in `hw5-handout/nfa/nfa_test.go`. From the `hw5-handout/nfa` directory, run the `go test` command to run the unit tests.

- Write a **concurrent** implementation of the `Reachable` function in `hw5-handout/nfa/nfa.go` that returns `true` if a final state is reachable from the start state after reading an input sequence of symbols, and `false`, otherwise. **The goal of this assignment is to test your knowledge on concurrency in Go, thus no credit will be given if your implementation is not concurrent.**
- If needed, write new tests, in `hw5-handout/nfa/nfa_test.go` to ensure that you get 100% code coverage for your code.

  From the `hw5-handout/nfa` directory, run the `go test -cover` command to see the current code coverage.

  From the `hw5-handout/nfa` directory, run the following two commands to see which statements are covered by the unit tests:

```
$ go test -coverprofile=temp.cov
$ go tool cover -html=temp.cov
```

- Implementing a sequential version of `Reachable` will get you no points.