EC3204: Programming Languages and Compilers (Fall 2023)
Homework 2: Regex to DFA Compiler

100 points in total, 10% of the total score
Due: 10/18, 23:59 (submit via GIST LMS)

Instructor: Sunbeom So

Important Notes

- **Evaluation criteria**
  For each problem, 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), and "Passed" indicates testcases whose expected outputs match with the outputs produced by your implementations.

- **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**
  For each problem, your job is to complete the "`(* TODO *)`" parts in provided templates. You can define new helper functions for it. However, you should not modify existing code templates such as variant type declarations.

- **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 a file `hw.ml` only. Do not change the file name. Do not zip the file.

1 Assignment Summary

Your goal in this assignment is to implement a compiler that translates regular expressions into deterministic finite automata (DFAs).

2 How to Build

1. Install the dependencies via 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. Download `hw2.zip` file from GIST LMS and unzip `hw2.zip`.

3. In the hw2 directory, activate the build script via the following command:

```
$ chmod +x build
```

4. To compile the project, run the following command in your shell:

```
$ ./build
```

   You should run the above command whenever you modify your code and test it.

3 Structure of the Project

You can find the following files in the `hw2` directory.

- `main.ml`: contains driver code with some testcases. You can add your own testcases here.
- `regex.ml`: contains the definition of regular expressions:

```
type alphabet = A | B
type t =
  | Empty
  | Epsilon
  | Alpha of alphabet
  | OR of t * t
  | CONCAT of t * t
  | STAR of t
```

  where we assume an alphabet is either a or b, i.e., $\Sigma = \{a, b\}$.

- `nfa.ml`: contains utilities for implementing NFAs. See `nfa.mli` to see how to use this module.

- `nfa.mli`: contains the interface file for `nfa.ml`.
- `dfa.ml`: contains utilities for implementing DFAs. See `dfa.mli` to see how to use this module.
- `dfa.mli`: contains the interface file for `dfa.ml`.
- `hw.ml`: contains the three unimplemented functions: `regex2nfa`, `nfa2dfa`, `run_dfa`. Your job is to complete these three functions and submit this file.

4 How to Run

Once you build the project, the executable named `main.native` will be generated. Run the executable by the following command:

```
$ ./main.native
```

The command above will execute the below code in `main.ml` at the top-level:

```
let _ =
  List.iter (fun (regex, str) ->
    prerr_endline (string_of_bool (match_regex regex str))
  ) testcases
```

- Initially, the command above will raise an exception with the following message:

  ```
  Fatal error: exception Hw.Not_implemented
  ```

- If `hw.ml` is successfully completed, the executable will output some string values for given testcases. For example, given the testcases in `main.ml`:

  ```
  let testcases : (Regex.t * alphabet list) list =
    [(Empty, []);
     (Epsilon, []);
     (Alpha A, [A]);
     (Alpha A, [B]);
     (OR (Alpha A, Alpha B), [B]);
     (CONCAT (STAR (Alpha A), Alpha B), [B]);
     (CONCAT (STAR (Alpha A), Alpha B), [A;B]);
     (CONCAT (STAR (Alpha A), Alpha B), [A;A;B]);
     (CONCAT (STAR (Alpha A), Alpha B), [A;B;B]);
     (CONCAT (CONCAT (STAR (CONCAT (Alpha A, Alpha A)),
                      STAR (CONCAT (Alpha B, Alpha B))), Alpha B), [B]);
     (CONCAT (CONCAT (STAR (CONCAT (Alpha A, Alpha A)),
                      STAR (CONCAT (Alpha B, Alpha B))), Alpha B), [A;A;B]);
     (CONCAT (CONCAT (STAR (CONCAT (Alpha A, Alpha A)),
                      STAR (CONCAT (Alpha B, Alpha B))), Alpha B), [B;B;B]);
     (CONCAT (CONCAT (STAR (CONCAT (Alpha A, Alpha A)),
  ```

```
                    STAR (CONCAT (Alpha B, Alpha B))), Alpha B), [A;A;A;A;B;B;B]);
   (CONCAT (CONCAT (STAR (CONCAT (Alpha A, Alpha A)),
                    STAR (CONCAT (Alpha B, Alpha B))), Alpha B), [A;A;A;B;B;B])
]
```

the executable should print the following strings:

```
false
true
true
false
true
true
true
true
false
true
true
true
true
false
```