EECS483 D2: Project 1 Details

Chun-Hung Hsiao

Jan 18, 2013

Announcements

- We won’t open the additional discussion session.
- Online submission system will be open at 12:00am, Jan 19.
- Email me your group information before the end of the day if you have not done so!
  - If you mailed me, you should have got a short reply from me.

Project 1: Deadline & Policy

- Due on 11:59pm, Jan 25.
  - You will still be able to submit your source code after the deadline, but we will check ALL submission times and reduce your late days accordingly.
- The submission of highest points (before applying the late penalty) will be used for grading.
- You will be able to get feedback of the first 3 submissions of each day.
- DO NOT try to exploit the submission system. There will be severe punishment if we detect malicious behavior in your source code.

Project 1: Submission

- Login on a CAEN Linux machine and place your source code in a separated folder.
- Use the following command to submit your code

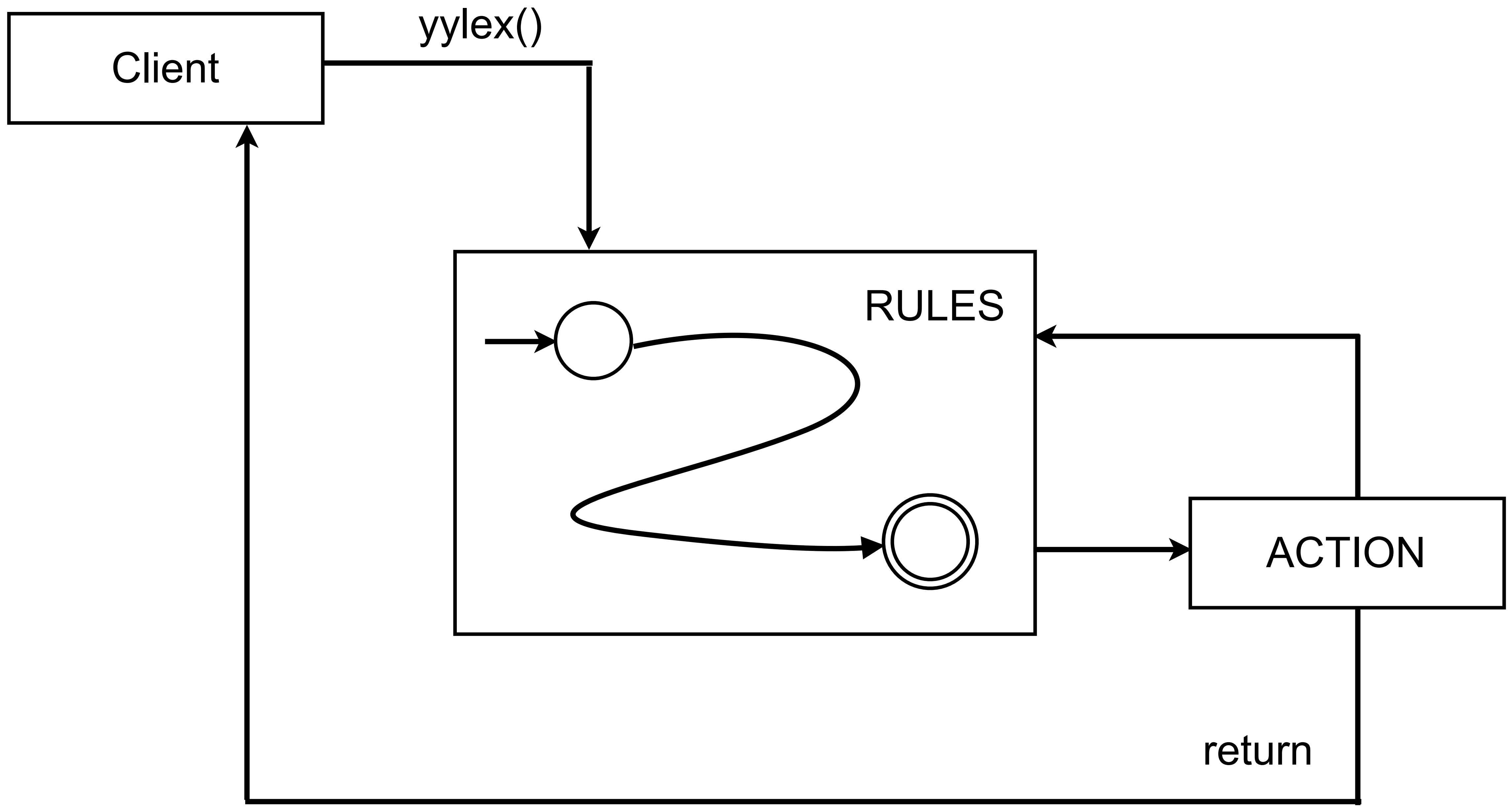
```
$ ~chhsiao/Public/submit.sh <project#> <folder>
```

- The feedback will be emailed to you in a few minutes
  - If you do not get it in hours, please email me or reflect on the forum. I will resolve the problem once I saw the message.

Lex Example: Simple In-Order Calculator

```
%{ enum { INT = 1, ADD, SUB, MUL, DIV, ENTER, ERROR }; %}
%%
[ \t]
\+          return ADD;
-           return SUB;
\*          return MUL;
\/          return DIV;
[0-9]+      return INT;
\n          return ENTER;
.           return ERROR;
%%
int compute(int a, int op, int b) {
    switch(op) {
        case ADD: return a + b;
        case SUB: return a - b;
        case MUL: return a * b;
        case DIV: return a / b;
    }
    return b;
}
int main() {
    int val = 0, op = 0, token;
    while(token = yylex()) {
        switch(token) {
            case INT: val = compute(val, op, atoi(yytext)); break;
            case ENTER: printf("%d\n", val); val = 0; op = 0; break;
            case ERROR: puts("error!"); return 1;
            default: op = token;
        }
    }
    return 0;
}
```

Lex Flow



Lex Built-ins

- `char* yytext` - C string of the matched lexeme
- `int yyleng` - the length of the match lexeme
- `yylval`, `yylloc` - bridge to the parser
  - Not a necessary part of lex
- `ECHO;` - output the lexeme
- More in the manual
  - http://flex.sourceforge.net/manual/Index-of-Functions-and-Macros.html#Index-of-Functions-and-Macros
  - http://flex.sourceforge.net/manual/Index-of-Variables.html#Index-of-Variables

Lex Rule Matching

- Longest possible match
  - "`supercalifragilisticexpialidocious`" is considered one token matched by "`[a-z]*`" rather than two tokens matched by "`[a-z]{17}`"
- Matches the earlier rule if tie
- Print a one-character token if none matched
  - See `dpp.l` for the simplest lex file!

Lex Conditions

- You can use conditions to specify when a rule should be turned on
  - "`<COND>[a-z]+`" is active only if `COND` is on
  - "`<C1,C2>[0-9]+`" is active when either `C1` or `C2` is on
  - "`<*>[HM]ar*y`" is always active
- Declare condition variables in the Definition section
  - `%x COND` - only rules with COND are active
  - `%s COND` - rules with no conditions are also active
- `BEGIN(COND);` to trigger the condition
  - Only one condition is on at a time
- Initial condition: `INITIAL`

Compiling and Running Lex Program

- First compile to C:

```
lex myscanner.l
```

  - Outputs `lex.yy.c`
  - Specify output filename with `-o` option

- Then compile to executable:

```
gcc lex.yy.c -ll -o myscanner
```

  - Can also use `g++`, as in Project 1

- Scan file through I/O redirection:

```
./myscanner < file
```

Project 1: What to Do

- Main quest: complete scanner.l to write a scanner for Decaf
  - Recognizes keywords, operators, identifiers, strings and numeric literals
  - Reports the line and column numbers of each token
  - Reports errors for invalid tokens
- Optional: preprocessor for Decaf
  - Strip comments
  - Implement simple macro substitution
  - You may choose to use either C or Lex to implement it

Decaf Scanner

- Recognizing each valid token
  - Record the location of the lexeme in `yylloc`
  - Set the value attribute of `yylval` if it is a literal
  - Set the name attribute of `yylval` if it is an identifier
- Reporting valid tokens
  - Just return the type of the tokens to `main()`
- Reporting invalid tokens
  - Generate error messages through the library function in class `ReportError`
  - Some tokens are skipped, some are fixed

Decaf Preprocessor

- Handle comments across multiple lines
  - Need to preserve line numbers for scanner
  - Column numbers are not preserved after preprocessing
- Macro substitution
  - "`#define ABC 10`" substitutes "#`ABC`" with "`10`"
  - "`#define ABC   10`" substitutes "#`ABC`" with "`  10`"
  - Skip bad `#define` to end of line
  - Look up the latest definition before substitution
  - Skip invalid # tokens (# followed by a series of letters)

Handling # Directives

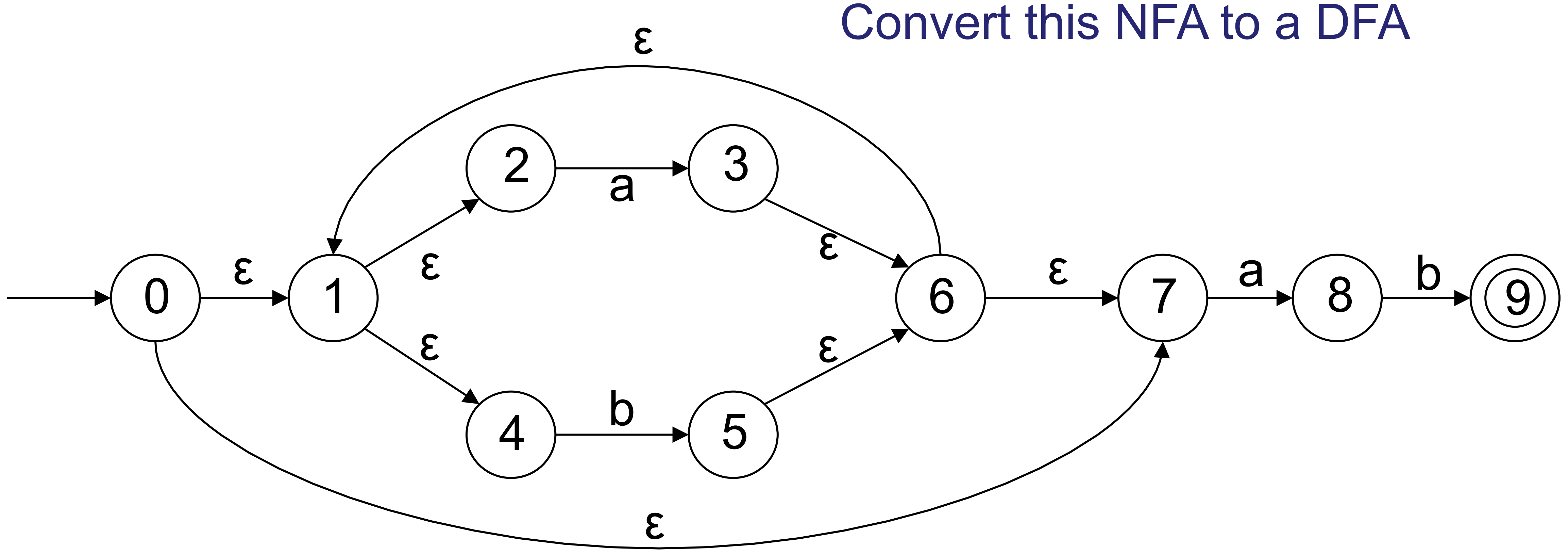
- Need a symbol table to remember each macro definition
- Update the table when seeing a redefinition
- Check the table to see if a macro has been defined when seeing it
- If defined, retrieve the replacement and output it!

Project 1 Hints

- Go through `main()` to know how the program executes
- Arrange the order of the rules carefully
- Think about when to increase the line and column numbers
  - You can use `DoBeforeEachAction()` to simplify the update
- Some errors need individual rules to detect!
  - Consider the rules for each possible valid and invalid token

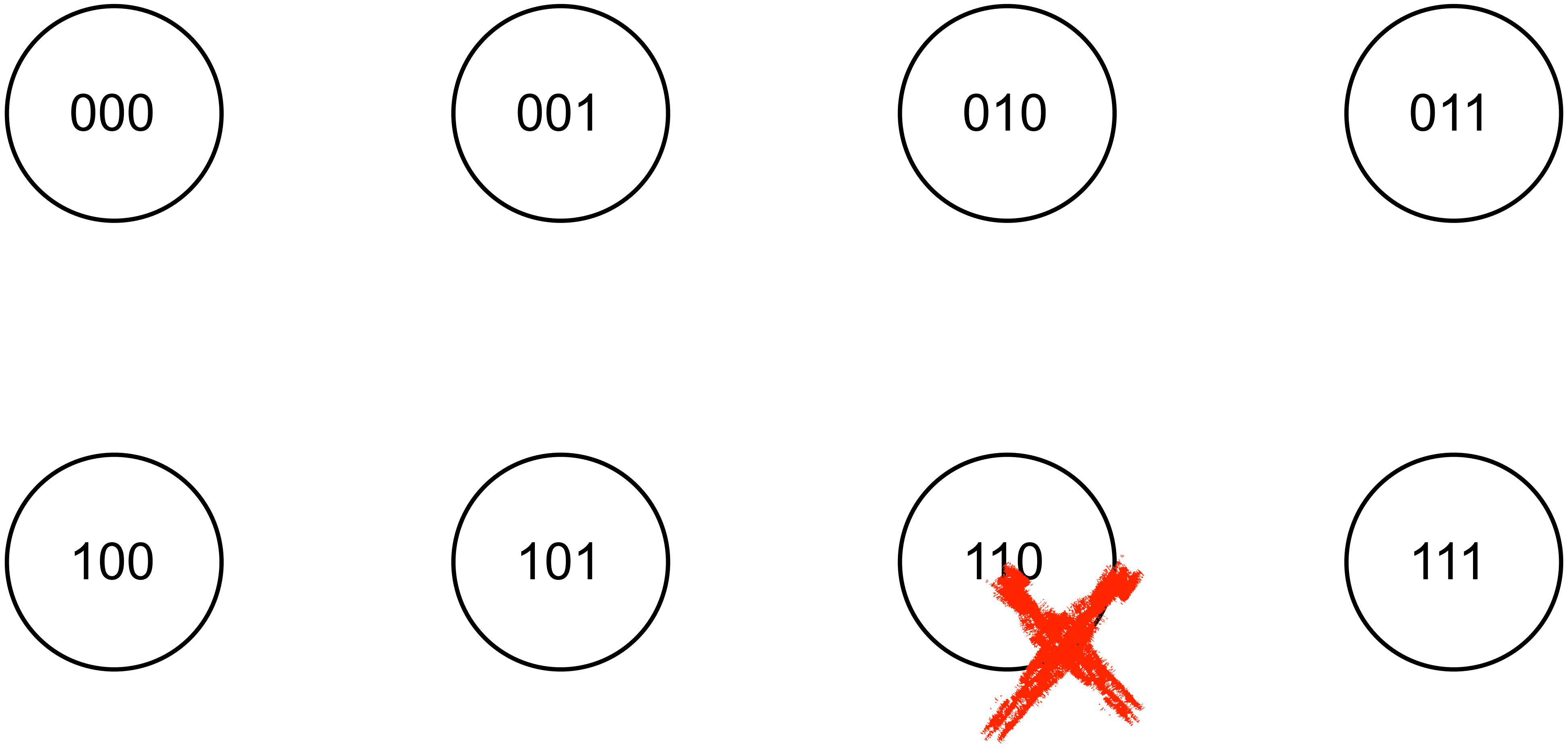
Exercise 1

Convert this NFA to a DFA



Exercise 2

- How to construct a DFA that accepts anything but strings containing 110?



Exercise 3

- How to write down an RE that recognizes all strings with even numbers of a’s and b’s?
  - It’s hard to come up with an RE that pairs all a’s and b’s!
  - How about split them into 2-letter pieces?

Thanks & Have good holidays!