COMP 323: Programming languages implementation
Homework 1: Regular expressions and ML-ulex specification

1. Written problems (5 points)

Write regular expressions for each of the following, where you may use ϵ, concatenation, disjunction, Kleene-$*$, character class notation $[a - b]$ and fixed repetition $r\{n\}$, where if r is a regular expression, $r\{n\}$ is $r \ldots r$ (n times):

Problem 1. *Strings over $\{a, b, c\}$ where the first a preceeds the first b (note that a string with no as or with not bs satisfies this criterion).*

Problem 2. *Strings over $\{a, b, c\}$ with an even number of as.*

Problem 3. *Strings over $\{0, 1\}$ that represent numbers divisible by 4 in binary (assume most-significant bit first and no leading 0s).*

Problem 4. *The language of non-negative octal and decimal integer literals in C. These consist of:*

- *The digit 0;*
- *The octal (base-8) numerals, which start with the digit 0 followed by one or more base-8 digits, where the first such is not 0.*
- *The decimal (base-10) numerals, which consist of one or more base-10 digits, where the first such is not 0.*

Problem 5. *The language of non-negative integers written in groups of three separated by commas as appropriate. Example words in the language are the following:*

$$0 \qquad 12 \qquad 762 \qquad 9,652 \qquad 92,100,542$$

(the spaces that appear after the commas are imaginary; the only ASCII characters are digits and commas).

2. Coding problems (5 points)

Write an ML-ulex specification with a single rule that matches the language of non-negative integers written in groups of three separated by commas. In the code distribution I have provided build files, a structure that defines the token type, and a driver you can use to test your specification. You need write only the ML-ulex specification.

The token type is `NumsTokens.token`, which is defined by

```
datatype token = Num of int | EOF
```

Your `eof` function must return `NumTokens.EOF`. On matching input, your ML-ulex rule must yield a value of the form `Num`(n), where n is the number represented by the string that matches your RE. Thus your ML-ulex rule will look something like

```
re => ( NumsTokens.Num (...) ) ;
```

where *re* is your regular expression and ... is code to convert the matching string (named by the identifier `yytext`) to the corresponding `int`-type value.

3. Code distribution, submission, and grading

The code distribution consists of the following files:

- `Makefile` and `.cm` files: build files. The `make` target to build the test executable is `tests` and to build the driver executable is `driver`.
- `unit_test.sml`, `tests.sml` and `driver.sml`: source code for the test and driver executables.
- `tokens.sml`: definition of a structure with the token type for the lexer.

You must submit the following files:

- `hw1.pdf`: your solutions to the written problems.
- `nums.lex`: your ML-ulex specification.

Your work will be graded according to the following criteria:

A: Written answers correct, appropriately concise. Lexer passes all tests, good lexer specification.

B: Few minor errors in written answers, maybe poor choice of factoring. Lexer fails for very minor reasons, or not such a good lexer specification.

C: Many minor errors, or a few minor errors and a serious error in the written work. Lexer fails for what look like non-trivial reasons.

D: Many serious errors in the written work. Serious misunderstandings of how to write a lexer specification.

F: Written work missing significant portions. Lexer does not compile.