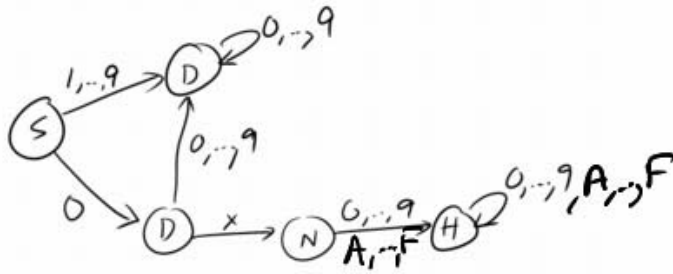


1. A decimal constant is any sequence of one or more decimal digits (including one starting with zero). A hexadecimal constant has the form 0xW where W is a sequence of one or more decimal digits or the letters A-F.

(a) Write a finite-state machine to recognize either decimal or hexadecimal constants. Label each state with either S (start state), D (decimal constant), H (hexadecimal constant), or N (neither).

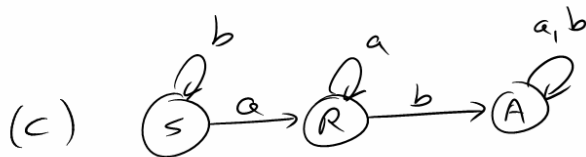
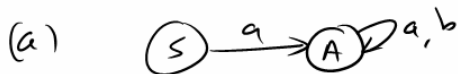


(b) Write a regular expression to recognize either a decimal or hexadecimal constant.

'0' 'x' (['0' - '9'] | ['A' - 'F'])+ | ['0' - '9']+

2. Define finite-state machines for the following regular expressions over the characters a and b. Label each node either S (start), A (accept the string) or R (reject it).

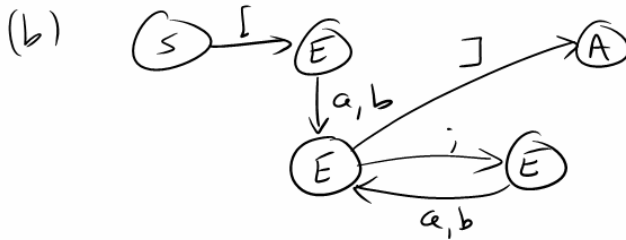
- (a) $a(a|b)^*$
- (b) $aa^*b(b|a)^*$
- (c) $(b|a)^*a^*ab(b|a)^*$



3. (a) Write a regular expression for this language: semicolon-separated lists of a' and b's, in square brackets. Examples of strings in the language are [], [a], and [a;b;a;a].

$[' ((' a ' | ' b ') (; (' a ' | ' b ')) ^ *) ? ']$

(b) Then write a finite-state machine for the language. Each state should be labeled either as S (for the start state), A (for an accepting state), or E (for an error state).



4. Write an ocamllex specification for tokens of this type:

type token = LESSTHAN | LEFTSHIFT | LEFTSHIFTEQ | IDENT of string

where these represent, respectively, the sequence "<", "<<", "<<=", and any string starting with a letter and consisting solely of letters and digits. Your specification should return the next token in the input, ignoring any character other than the ones that constitute tokens.

```

let letter = ['A' - 'Z'] | ['a' - 'z']
let digit = ['0' - '9']

rule tokenize = parse
| "<" { LESSTHAN }
| "<<" { LEFTSHIFT }
| "<<=" { LEFTSHIFTEQ }
| (letter (letter | digit)*) as id { IDENT id }
| _ { tokenize lexbuf }
  
```