A Puzzle For You

Construct a FSM that accepts all decimal integers.
\[ \Sigma = \{0, 1, 2, \ldots, 9\} \]
\[ L = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, \ldots\} \]
Leading zeroes are disallowed. E.g., \(012 \notin L\)
Regular Languages

CMPT 125
Mar. 25
Lecture 30

Today:

● Regular Languages
● Regular Expressions
● FSM Implementations
● Finite State Transducers
A formal language is used to distinguish precisely what sequences are allowed

- expressed mathematically, often recursively

Three important definitions:

- alphabet ($\Sigma$) - a set of characters / symbols
- word ($w$) - a finite sequence of characters / symbols
- language ($L$) - a [possibly infinite] set of words

Parse a word $w$ to decide if it is in the language $L$

- Accept if $w$ is in $L$, Reject if not in $L$
Modelling Computation (Review)

To decide a language, use a finite state machine (FSM).

Rules of the Game:

- Finite number of states: one of them is the *Start* state; one or more are the *Final* states.
- The FSM reads one character at a time.
- Transitions are based solely on the current state and the next character.
- A missing transition defaults to the dead state, which is not a Final state.
- If the FSM ends in a final state, then: *Accept*
- Else: *Reject*

$L = \{\text{all words that have substring } abc\}$
Modelling Computation (Review)

To decide a language, use a *finite state machine* (FSM).

**Rules of the Game:**

- Finite number of states: one of them is the *Start* state; one or more are the *Final* states.
- The FSM reads one character at a time.
- Transitions are based solely on the current state and the next character.
- A missing transition defaults to the dead state, which is not a Final state.
- If the FSM ends in a final state, then: *Accept*
- Else: *Reject*

\[ \Sigma = \{a, b, c\} \]

\[ L = \{ \text{all words that don’t have substring abc} \} \]
Regular Languages

A *regular language* can be decided by a FSM.

- If you complement a regular language, i.e., swap Accept ↔ Reject, the result is a regular language.
- Regular languages are *closed* under complement

Regular languages are also closed under:

- union
- catenation
- Kleene star

Write them using *regular expressions*. 
Regular Expressions

If $L_1$ and $L_2$ are two regular languages, then

1. $L_1 \cup L_2$ is their union, i.e., use a word from $L_1$ or a word from $L_2$
2. $L_1L_2$ is their catenation, i.e., use a word from $L_1$ followed by one from $L_2$
3. $L_1^*$ is its Kleene closure, i.e., use 0 or more catenations of words from $L_1$

Examples:

- 0 or more b’s: $b^*$
- begins with a b: $b(a|b)^*$
- begins and ends with a b: $b(a|b)^*b$
- begins or ends with a b: $b(a|b)^* | (a|b)^*b$
- begins and ends with different: $\lambda | a(a|b)^*b | b(a|b)^*a$
- exactly 3 long: $(a|b)(a|b)(a|b)$ OR $(a|b)^3$
- has substring abc: $(a|b|c)^*abc(a|b|c)^*$
- even number of a’s: $b^*(ab*ab^*)^*$
Follow transitions in a simple loop.

**Algorithm:**

```
state ← Start
while there is still input {
    c ← next input symbol
    if transition(state, c) exists then
        state ← transition(state, c)
    else
        Reject (OR . . . state ← Dead)
}
if state is a Final state then Accept
else Reject
```
Case Method

Algorithm:

- Use a large if / else if / ... 
- Use a nested switch / case

```java
switch (state) {
    case Start:
        switch (c) {
            case '0':
                state = BeginWith0;
                break;
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                state = BeginWith1to9;
                break;
            default:
                state = Dead;
        }
    break;
}
```

Σ = {0, 1, 2, ..., 9}
FSM Augmentation: Actions

While following a transition, perform an action

- place actions on transitions following a slash
- should compute a useful property of the word

E.g., What might be a useful property?

- the integer’s value
- A1: \( \text{val} = c - '0' \);
- A2: \( \text{val} = 10 \times \text{val} + (c - '0') \);

\[ \Sigma = \{0, 1, 2, \ldots, 9\} \]
Another possible action: output

- need to add a special symbol for EOF (usually $\$)$

Problem: Construct a FSM with output that reports the parity of a sequence of bits

- E.g., $1011 \rightarrow 1$, $11011 \rightarrow 0$, $\lambda \rightarrow 0$

![FSM Diagram]
Example: Block Reduction

Problem: Construct a FSM with output that reports the 0/1 blocks of a binary sequence

- E.g., $111000010011100011 \rightarrow 1010101$

Strategy:
- Output the first of each block.