Tutorial 02

- Regular Expressions
- DFAs
- Scanning

Regular Expressions

- Alphabet (Σ) = finite non-empty set of symbols
  ↳ eg: $\Sigma = \{a\}$, $\Sigma = \{a, b, c\}$, $\Sigma = \{hello, world\}$
- word (w) = finite sequence of symbols in Σ
  ↳ ε = sequence with no symbols
  ↳ eg: if $\Sigma = \{a, b, c\}$

  valid words $\in \Sigma = \varepsilon, a, b, c, aa, abc, abbca, \ldots$

  eg: if $\Sigma = \{hi, bye\}$

  valid words $\in \Sigma = \varepsilon$, hi, bye, hihi, hibye, ...
- Regular Language (L) over Σ is a set of words iteratively defn:
  1) $L = \emptyset$
  2) $L = \{\varepsilon\}$
  3) $L = \{a\}$
  4) $L = L_1 \cup L_2 = \{x : x \in L_1 \text{ or } x \in L_2\}$ (union)
  5) $L = L_1 L_2 = \{xy : x \in L_1, y \in L_2\}$ (concatenation)
  6) $L = L_1{*} = \bigcup_{i=0}^{\infty} L_1^i = \{\text{""}, L_1, L_1L_1, L_1L_1L_1, \ldots\}$

- Regular expressions → concise/simple defⁿ of regular languages
  1) $R = \emptyset$
  2) $R = \varepsilon$
  3) $R = a$ where $a \in \Sigma$
  4) $R = R_1 | R_2 = \{x : x \in R_1 \text{ or } x \in R_2\}$ union
  5) $R = R_1 R_2 : \{xy : x \in R_1, y \in R_2\}$ concatenation
  6) $R = R_1^* = \{"", R_1, R_1 R_1, \ldots\}$ Kleene Star
- Operator Precedence: $R^* > R_1 R_2 > R_1 \cup R_2$
  ↳ eg: $aa|bb^* \equiv aa|b(b^*) \equiv (aa)|(b(b^*))$

eg: Provide a regex for $\Sigma = \{a,b\}$, $L = \{aa, ab, ba, bb\}$
  ↳ valid words: $aa, ab, ba, bb \in L$
  ↳ solⁿ 1) $R = aa|ab|ba|bb$
  ↳ solⁿ 2) $R = (a|b)(a|b)$

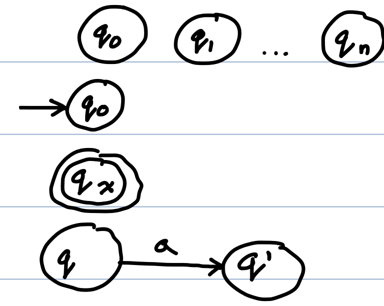
eg: Provide a regex for $\Sigma = \{0,1\}$, $L = \{x \in \Sigma^* : x\text{'s } 2^{nd} \text{ symbol} = 0 \ \& \ 5^{th} \text{ symbol is } 1\}$
  ↳ 00111, 1000110101 $\in L$
  ↳ solⁿ) $(0|1)0(0|1)(0|1)1(0|1)^*$

eg: Provide a regex for $\Sigma = \{a, b, +, -, \cdot, /\}$
  $L = \{x \in \Sigma^* : x \text{ represents a valid arithmetic operation}\}$
  ↳ Note: No unary ops ($-b \notin L$) & no implicit mult ($ab \notin L$)
  ↳ $a, b, a+b, a-b, a \cdot b, a/b, a+a+a \in L$
  ↳ Pattern: term op term op term ... op term
  solⁿ) $(a|b)[(+|-|\cdot|/)(a|b)]^*$

Deterministic Finite Automata (DFA)

- DFA's model computations & algos $(\Sigma, Q, q_0, A, \delta)$

1) Σ : input alphabet
2) Q : finite set of states
3) $q_0 \in Q$: starting state
4) $A \subseteq Q$: set of accepting states
5) $\delta : Q \times \Sigma \to Q$: transition fn



↳ $\delta(q, a) = q'$ $\quad q, q' \in Q$ & $a \in \Sigma$

↳ if $\delta(q,a)$ DNE, transition to non-accepting error state

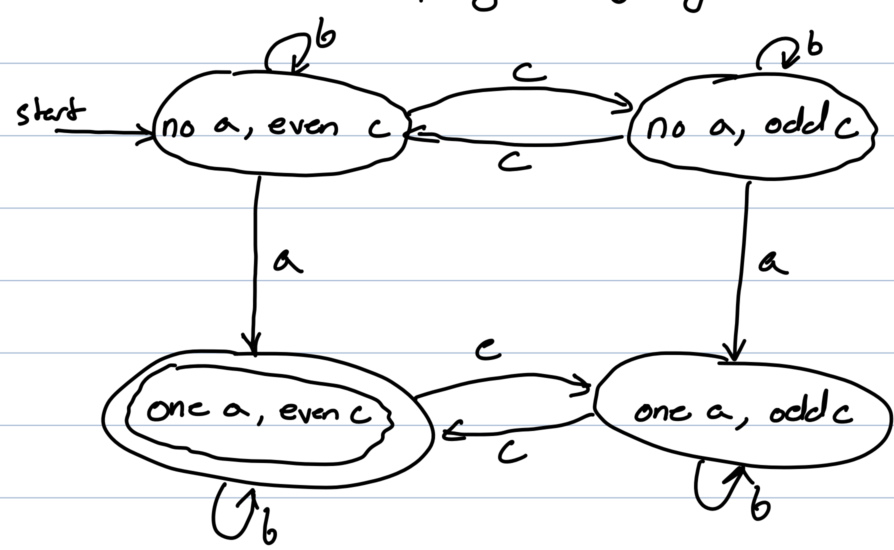
→ $\delta(\text{error}, a) = \text{error} \quad \forall a \in \Sigma$

eg: Draw a DFA for $\Sigma = \{a, b, c\}$, $L = \{x \in \Sigma^* :$ x contains exactly 1 a & an even # of c's $\}$

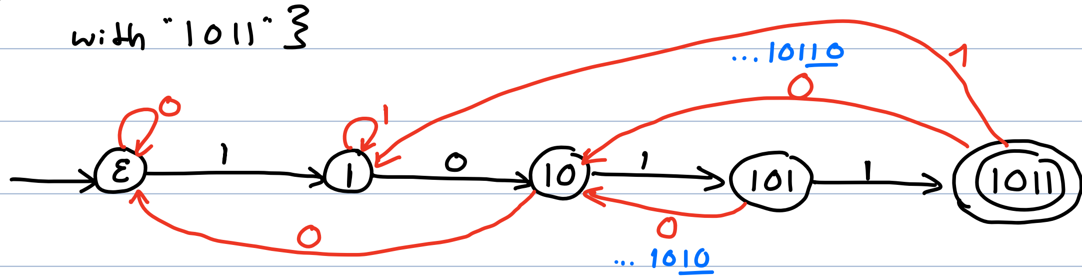
↳ Accept: a, acc, cac, accb, bbcacbcc $\in L$

Reject: "", cc, aa, ...

↳ Tip: think about what states are needed to differentiate between accepting & rejecting states



eg: DFA for $\Sigma = \{0,1\}$ accepting $L \{x \in \Sigma^* :$ x ends with "1011" $\}$



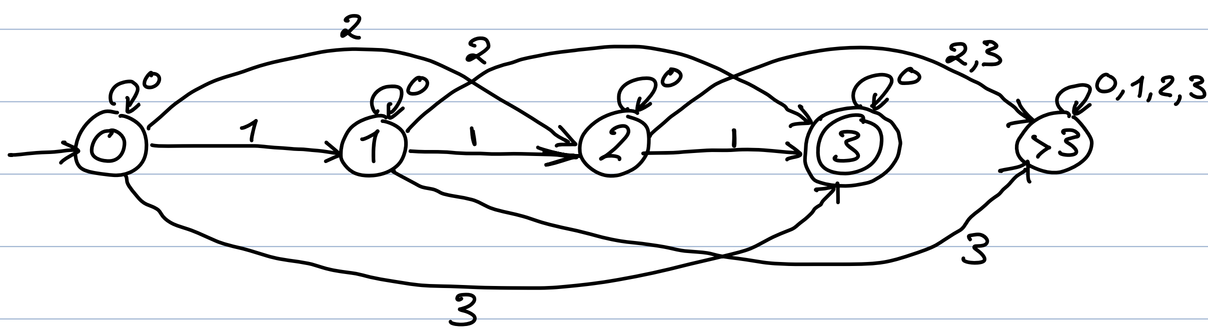
eg: DFA for $\Sigma = \{0,1,2,3\}$ accepting $L = \{x \in \Sigma^* :$ x's digit sum is 3$\}$

↳ Accept: 3, 03, 0300, 012, 12, 111, 0101010, ...

(012: 0+1+2=3; 0101010: 0+1+0+1+0+1+0=3)

Reject: 0, 1, 2, 00, 10, 20, 31, 13, 33333, ...

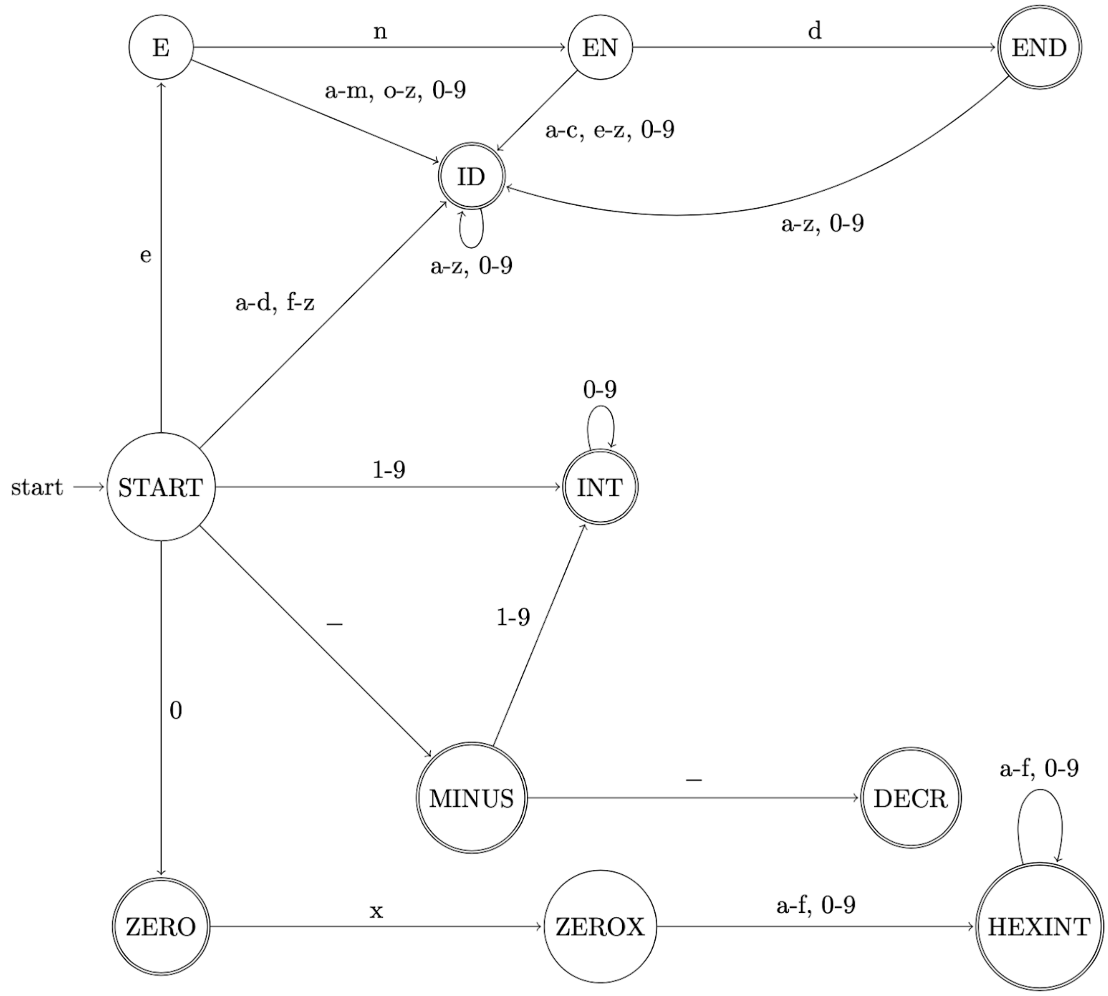
(0, 1, 2, 00, 10, 20: sum < 3; 31, 13, 33333: sum > 3)



Scanning

- Goal: Take non-empty input (w) & split it into non-empty token sequences.
  - ↳ input $w = w_1 + w_2 + \ldots + w_n$ where $w_1, \ldots, w_n \in L$ for $n > 0$
    - ($w_1 + w_2 + \ldots + w_n$: string concat; $w_1, \ldots, w_n$: tokens)
  - ↳ word w can be scanned w.r.t. to L if $\exists\ w = w_1 + \ldots + w_n$ $n > 0$
- Maximal munch & simplified maximal munch are scanning algos:
- Simplified Maximal Munch (SMM)
  - ↳ Run DFA accepting L with input w
  - ↳ If transition fn in state q on char a does not exist:
    - → If q is accepting, output current token & reset DFA with remaining input
    - → If q is not accepting, produce ERROR & exit
  - ↳ Consume input chars from w until ERROR or no more input.
- Maximal Munch (MM):
  - ↳ Run DFA accepting L with input w
  - ↳ Backtrack input & DFA if transition on state q with char a does not exist to last seen accepting state
    - → Output token & reset DFA if accepting state is found / reached
    - → ERROR if backtracking didn't reach an accepting state
- Possible that MM & SMM might scan/accept different w

eg: Suppose we have the DFA:

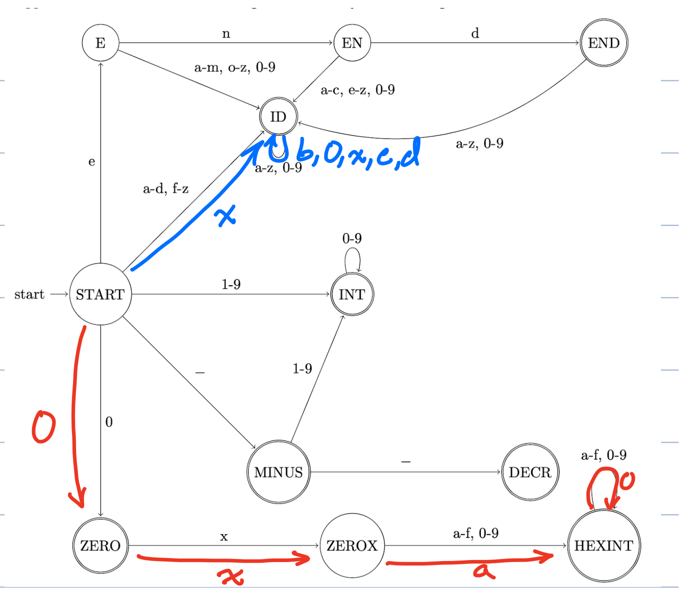


give the sequence of tokens produced by SMM for each input below.

1) Input : 0xa0xb0xcd

HEXINT 0xa0

ID xb0xcd



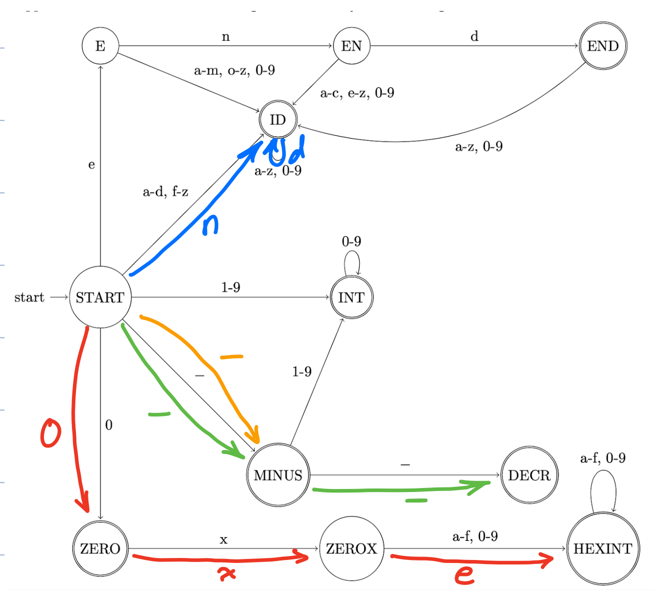
2) Input : 0xend---

HEXINT 0xe

ID nd

DECR --

MINUS -

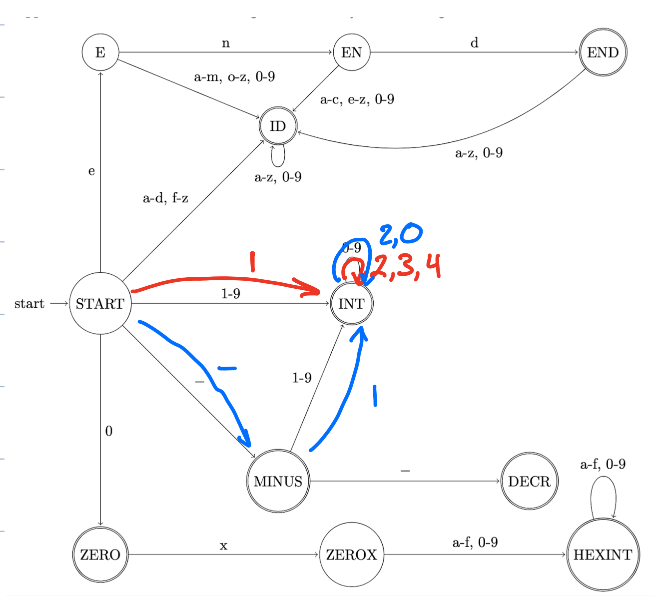


3) 1234-120xb

INT 1234

INT -120

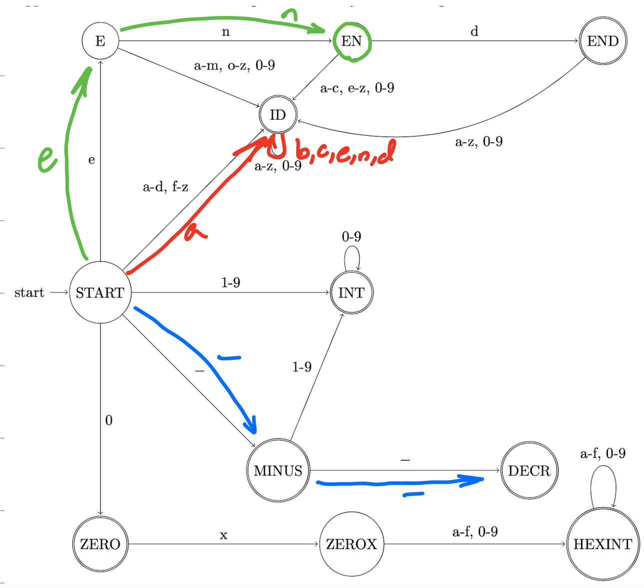
ID xb



4) abcend--en-3

↑ error!

ID	abcend
DECR	--
ERROR	(δ(EN,-) DNE)



5) 01end-end10

ZERO	0
INT	1
END	end
MINUS	-
ID	end10