Compilers Design

M. T. Bennani
Assistant Professor, FST - University of Tunis El Manar,
URAPOP-FST

-Academic Year 2020-2021-

Outline

- Specifying lexical structure using regular expressions
- Finite automata
  - Deterministic Finite Automata (DFAs)
  - Non-deterministic Finite Automata (NFAs)
- Implementation
  - RegExp $\Rightarrow$ NFA $\Rightarrow$ DFA $\Rightarrow$ Tables
- Exercises

Regular expressions extension

Variation in regular expression notation

- Union : A | B ⇔ A + B
- Option : A + ε ⇔ A?
- Range : 'a'+'b'+...+'z' ⇔ [a-z]
- Excluded range : complement of [a-z] ⇔ [^a-z]

Regular expression to lexical specification (1)

1. Write a rexp for the lexemes of each token
   - Number = digit+
   - Keyword = 'if'+'else'+...
   - Identifier = letter(letter+digit)*
2. Construction R, matching all lexemes for all tokens
   - R = Keyword + Identifier + Number + ...
   - R = $R_1 + R_2 + \ldots$
3. Verify if the input $(X_1 \ldots X_n)$ belongs to the language
   - For $1 \leq i \leq n$ check $X_1 \ldots X_i \in L(R)$
4. If success
   - $X_1 \ldots X_i \in L(R_j)$ for some j

   remove the lexeme $X_1 \ldots X_i$ from the input and go to (3)

Ambiguities

- How much input is used? What if
  - $X_1 \ldots X_i \in L(R)$ and also
  - $X_1 \ldots X_k \in L(R)$

Rule 1

Pick longest possible string in L(R): "The maximal munch" algorithm

- Which token is used? What if
  - $X_1 \ldots X_i \in L(R_j)$ and also
  - $X_1 \ldots X_i \in L(R_k)$

Rule 2

Use rule listed first (j if $j < k$)
- Treats "if" as a keyword, not an identifier

Error Handling

- What if: No rule matches a prefix of input?
  - Write a rule matching all "bad" strings
  - Put it last (lowest priority)

Definitions

- Regular expressions = Specification
- Finite automata = implementation

Finite automaton consists of

- An input alphabet Σ
- A set of states S
- A start state n
- A set of accepting states $F \subseteq S$
- A set of transitions state $\rightarrow^{input}$ state

Notations

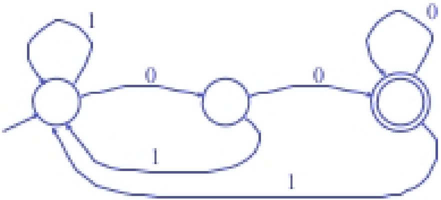
- Transition : $S_1 \to^a S_2$
  - In state S_1 on input "a" go to state S_2
- If end of input and in accepting state $\Rightarrow$ accept
- Otherwise $\Rightarrow$ reject

Examples

- Design a finite automaton that accepts only "1"
- Design a finite automaton that accepts numbers of 1's followed by a single 0.

Automata examples

- Alphabet{0,1}
- What language does this recongnize?



Epsilon moves

- Another kind of transition: ε-moves
  - $A \rightarrow^{\varepsilon} B$
- Machine can move from state A to state B without reading input

Deterministic and Nondeterministic Automata

Deterministic Finite Automata (DFA)

- One transition per input per state
- No ε-moves

$\Rightarrow$ A DFA can take only one path through the state graph
- Completely determined by input

Nondeterministic Finite Automata (NFA)

- Can have multiple transitions for one input in a given state
- Can have ε-moves

$\Rightarrow$ NFAs can choose

- Whether to make ε-moves
- Which of multiple transitions for a single input to take

NFA and DFA

NFA acceptance

- Let the following NFA

State	0	1
-A	{A,B}	A
B	C	$\varnothing$
C+	$\varnothing$	$\varnothing$

  - For input: 1 0 0, The NFA accepts the input if it **can** get to the final state

Comparaison

- NFAs and DFAs recognize the same set of languages (regular languages)
- DFSs are faster to execute because there are no choices to consider
- For a given language NFA can be simpler than DFA

NFA to DFA (1/2)

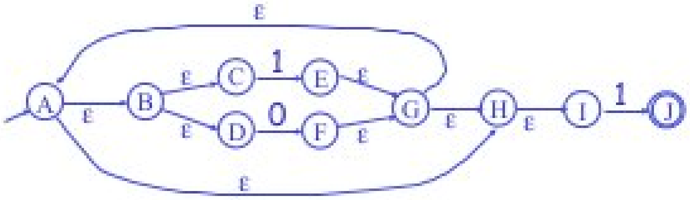
Step 1 - Create state table from the given NDFA.
Step 2 - Create a blank state table under possible input alphabets for the equivalent DFA.
Step 3 - Mark the start state of the DFA by q0 (Same as the NDFA).
Step 4 - Find out the combination of States Q0, Q1,... , Qn for each possible input alphabet.
Step 5 - Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.
Step 6 - The states which contain any of the final states of the NDFA are the final states of the equivalent DFA.

NFA to DFA (2/2)

- Each state of DFA is a non-empty subset of states of the NFA
- Start state
  - The set of NFA states reachable through ε-moves from NFA start state
- Add a transition S $\rightarrow^a$ S' to DFA iff
  - S' is the set of NFA states reachable from any state in S after seeing the input a, considering ε-moves as well

Example

The regular expression is: (1+0)*1



- A DFA can be implemented by a 2D table T
  - One dimension is "states"
  - Other dimension is "input symbol"
  - For each transition $S_i \to^a S_k$ define T[i,a] = k
- DFA execution
  - If in state S_i and input a, read T[i,a]=k and skip to the state S_k
  - very efficient

Note

- NFA to DFA conversion is the heart of the tools such as flex
- But, DFAs can ve huge
- In practive, flex-like tools trade off speed for space in the choice of NFA and DFA representations