Regular Expressions and Finite State Automata

Gul Agha
2104 SC
http://www.cs.uiuc.edu/class/fa05/cs421/current/

Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Elsa Gunter

Language Syntax

- Syntax is the description of which strings of symbols are meaningful expressions in a language
- It takes more than syntax to understand a language; need meaning (semantics) too
- Syntax is the entry point

Elements of Syntax

- Character set – previously always ASCII, now often 64 character sets
- Keywords – usually reserved
- Special constants – cannot be assigned to
- Identifiers – can be assigned to
- Operator symbols
- Delimiters (parenthesis, braces, brackets,)
- Blanks (aka white space)

Elements of Syntax

- Expressions
- Type expressions
- Declarations (in functional languages)
- Statements (in imperative languages)
- Subprograms

Elements of Syntax

- Modules
- Interfaces
- Classes (for object-oriented languages)

Formal Language Descriptions

- Regular expressions, regular grammars, finite state automata
- Context-free grammars, BNF grammars, syntax diagrams
- Whole family more of grammars and automata – covered in automata theory
Grammars

- Grammars are formal descriptions of which strings over a given character set are in a particular language
- Language designers write grammar
- Language implementers use grammar to know what programs to accept
- Language users use grammar to know how to write legitimate programs

Regular Expressions

- Start with a given character set – a, b, c…
- Each character is a regular expression
- The regular expression b represents the set { b }

Regular Expressions

- If x and y are regular expressions, then xy is a regular expression
  - xy represents the set of all strings made concatenating a string in x with a string in y
- If x and y are regular expressions, then x | y is a regular expression
  - x | y represents the set of strings in x or in y (or both)

Example Regular Expressions

- (01)*1
  - The set of all strings of 0's and 1's ending in 1. {1, 01, 11,…}  
- a*b(a*)
  - The set of all strings of a's and b's with exactly one b
- (01) (10)*
  - You tell me
- Regular expressions (equivalently, regular grammars) important for lexing, breaking strings into recognized words

Example: Lexing

- Regular expressions good for describing lexemes (words) in a programming language
- Identifier = (a | b | … | z | A | B | … | Z) (a | b | … | z | A | B | … | Z) (0 | 1 | … | 9)*
- Digit = (0 | 1 | … | 9)
- Number = (1 | … | 9)(0 | … | 9) | ~ (1 | … | 9)(0 | … | 9)
- Keywords: if = if, while = while,…
Implementing Regular Expressions
- Regular expressions can generate strings in a language
- Not so good for recognizing when a string is in language
- Regular expressions: which option to choose, how many repetitions to make
- Answer: finite state automata

Finite State Automata
- A finite state automata over an alphabet is:
  - a directed graph
  - edges are labeled with elements of alphabet, or empty string
  - some nodes (or states), marked as final
  - one node marked as start state
- Syntax of FSA

Example FSA

Deterministic FSA’s
- If FSA has for every state exactly one edge for each letter in alphabet then FSA is deterministic
  - No edge labeled with $\varepsilon$
- In general FSA is non-deterministic.
  - NFSA also allows edges labeled by $\varepsilon$
- Deterministic FSA special kind of non-deterministic FSA

DFSA Language Recognition
- Think of a DFSA as a board game; DFSA is board
- You have string as a deck of cards; one letter on each card
- Start by placing a disc on the start state

DFSA Language Recognition
- Move the disc from one state to next along the edge labeled the same as top card in deck; discard top card
- When you run out of cards:
  - if you are in final state, you win; string is in language
  - if you are not in a final state, you lose; string is not in language
DFSA Language Recognition - Summary

- Given a string (over the alphabet)
- Start at \textit{start} state
- Move over edge labeled with first letter to new state
- Remove first letter from string
- Repeat until string gone
- If end in final state then string in language

- Semantics of FSA

Example DFSA

- Regular expression: (0 \mid 1)^* 1
- Deterministic FSA

Example DFSA

- Regular expression: (0 \mid 1)^* 1
- Accepts string \texttt{0 1 1 0 1}

Example DFSA

- Regular expression: (0 \mid 1)^* 1
- Accepts string \texttt{0 1 1 0 1}

Example DFSA

- Regular expression: (0 \mid 1)^* 1
- Accepts string \texttt{0 1 1 0 1}

Example DFSA

- Regular expression: (0 \mid 1)^* 1
- Accepts string \texttt{0 1 1 0 1}
Example DFSA

- Regular expression: \((0 | 1)^* 1\)
- Accepts string \(\ldots 0 1\)

Example DFSA

- Regular expression: \((0 | 1)^* 1\)
- Accepts string \(\ldots\)

Example DFSA

- Regular expression: \((0 | 1)^* 1\)
- Accepts string \(\ldots\)

Example DFSA

Non-deterministic FSA’s

- NFSA generalize DFSA in two ways:
  - Include edges labeled by \(\varepsilon\)
  - Allows process to non-deterministically change state
- Each state can have zero, one or more edges labeled by each letter
- Given a letter, non-deterministically choose an edge to use

NFSA Language Recognition

- Play the same game as with DFSA
- Free move: move across an edge with empty string label without discarding card
- When you run out of letters, if you are in final state, you win; string is in language
- You can take one or more moves back and try again
- If have tried all possible paths without success, then you lose; string not in language

FSA Language Recognition

- Move the disc from one state to next if edge between labeled the same as top card in deck; discard top card
- Free move: move across an edge with empty string label without discarding card
- When you run out of cards, if you are in final state, you win; string is in language
- You can take a move back and try another
- If you have cards left and you have tried all possible edges without success, then you lose; string not in language
Example NFSA

- Regular expression: \((0 \mid 1)^* 1\)
- Non-deterministic FSA

Accepts string: 0 1 1 0 1

Guess: 0 1 1 0 1
Example NFSA

- Regular expression: (0 | 1)* 1
- Accepts string: 0 1 0 1
- Backtrack

Example NFSA

- Regular expression: (0 | 1)* 1
- Accepts string: 0 1 0 1
- Guess again

Example NFSA

- Regular expression: (0 | 1)* 1
- Accepts string: 0 1 0 1
- Guess

Example NFSA

- Regular expression: (0 | 1)* 1
- Accepts string: 0 1 0 1
- Backtrack

Example NFSA

- Regular expression: (0 | 1)* 1
- Accepts string: 0 1 0 1
- Guess again
Example NFSA

- Regular expression: \((0 | 1)^* 1\)
- Accepts string 01101
- Guess (Hurray!!)

Rule Based Execution

- Search
- When stuck backtrack to last point with choices remaining
- Executing the NFSA in last example was example of rule based execution
- FSA's are rule-based programs; transitions between states (labeled edges) are rules; set of all FSA's is programming language