CS421 Topic 11: Regular Expressions and Finite Automata[1]

Sameer Sundresh
sundresh@uiuc.edu

University of Illinois at Urbana-Champaign

June 14, 2007

[1] Based on slides by Mattox Beckman, as updated by Vikram Adve, Gul Agha, Elsa Gunter, and Mark Hills

Language Syntax

Regular Languages

Finite Automata

Language Syntax

Syntax is the description of which strings of symbols are meaningful expressions in a language

- Note that syntax tells us what a language *looks like*, but not what it *does* – we need semantics to assign the language meaning
- Syntax gives us an entry point into the language – if well-designed, the syntax should reflect well what the language does

Elements of Syntax

- Character set: the characters that can be used in a language; previously always ASCII, now also Unicode to support other languages
- Keywords: words which have predefined importance in a language
- Reserved words: keywords that cannot be changed by the user
- Identifiers: words which act as names for program values (variables, functions, classes, etc)
- Special constants: language-defined identifiers that cannot be assigned to

Elements of Syntax, cont.

- Operator symbols: symbolic representations for operations (arithmetic, list concatenation, etc) – often cannot be changed
- Delimiters: parentheses, braces, commas, semicolons, etc
- Whitespace: blanks, newlines, tabs – comments often treated as whitespace

Elements of Syntax, cont.

- Expressions (`a + b`, `this->add(3,4)`)
- Type expressions (`a:int,f:'a -> 'b`)
- Declarations
- Statements
- Subprograms (functions, procedures)
- Modules
- Interfaces
- Classes (OO languages)

Formal Language Descriptions

Various language description formalisms are available for different classes of formal languages.

- Regular languages
  - Regular expressions
  - Regular grammars
  - Finite state automata
- Context-free languages
  - Context-free grammars
  - BNF/EBNF notation
  - Syntax diagrams

This list is *not* exhaustive – more information in automata theory/formal languages

Grammars

A *grammar* is a formal description of which strings over a given character set (*alphabet*) are in a particular language

- Language designers write a grammar
- Language implementors use this grammar to know which input programs to accept
- Language users use this grammar to know how to write legitimate programs

Regular Languages

Regular languages are set of *words*, or *strings*, over an *alphabet* of characters valid for the language.

$$\Sigma = \{a, b, c, \cdots\}$$

The language does **not** include every possible combination of characters, so we need a way to define which combinations are valid.

Regular Expressions

Regular expressions act as definitions for the language – they allow us to give a *finite* definition of a possibly *infinite* set of words in the language. Words matching the expression are in the language, while words that do not match are not.

Forming Expressions

$$\Sigma = \{a, b, \cdots\}$$

- Any single character in the alphabet is a regular expression representing a set with one string of just that character.

$$a = \{a\}, b = \{b\}, \cdots$$

- ϵ is a regular expression representing the set with just the empty string

$$\epsilon = \{\epsilon\}$$

Forming Expressions

$$\Sigma = \{a, b, \cdots\}$$

- Given regular expressions **x** and **y**, **xy** is a regular expression representing the set of all strings formed by first taking a string from **x** and then appending a string from **y**

$$\mathbf{x} = \{a, b\}, \mathbf{y} = \{c, d\}, \mathbf{xy} = \{ac, ad, bc, bd\}$$

- Given regular expressions **x** and **y**, $\mathbf{x} + \mathbf{y}$ is a regular expression representing the set of all strings in either **x** or **y** (set union)

$$\mathbf{x} = \{a, b\}, \mathbf{y} = \{c, d\}, \mathbf{x} + \mathbf{y} = \{a, b, c, d\}$$

Forming Expressions

$$\Sigma = \{a, b, \cdots\}$$

- **(x)** is identical to **x** for any regular expression **x**
  $$x = \{a, b\}, (x) = \{a, b\}$$
- Given regular expression **x**, x^* is the regular expression representing 0 or more copies of **x** concatenated together
  $$x = \{a\}, x^* = \{\epsilon, a, aa, aaa, \cdots\}$$

Sample regular expressions

- $(0+1)^*1$
  The set of all strings of 0s and 1s ending in 1: $\{1, 01, 11, \cdots\}$
- $a^*b(a^*)$
  The set of all strings of as and bs with exactly one b
- $((01)+(10))^*$
  What do you think this is?

Regular expressions give us a method for finding recognized words in strings, making them crucial for lexing

Lexing

Regular expressions are good for describing *lexemes* (words) in a programming language:

- Identifier = $(\mathtt{a}+\mathtt{b}+\cdots+\mathtt{z})(\mathtt{a}+\mathtt{b}+\cdots+\mathtt{z}+\mathtt{A}+\mathtt{B}+\cdots+\mathtt{Z}+\mathtt{0}+\cdots+\mathtt{9})^*$
- Digit $= (0+\cdots+9)$
- Number $= (1+\cdots+9)$**Digit**$^* + -(1+\cdots+9)$**Digit***
- Keywords: **if** $= \mathtt{if}$, **while** $= \mathtt{while}, \cdots$

Implementing Regular Expressions

Regular expressions provide a good way to *generate* strings in the language, but not a good way to *recognize* strings in the language

- Which option do I choose?
- How many times should I repeat?
- etc

Solution: *finite automata*

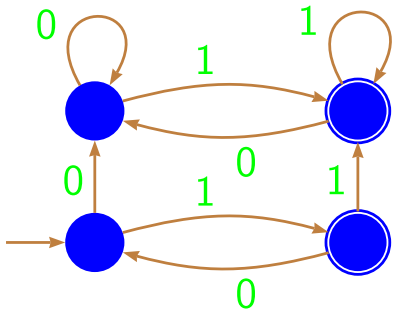
Finite Automata

A *finite automaton* over an alphabet Σ is

- a directed graph
- with edges labeled with elements of the alphabet or ϵ
- and one or more nodes marked as *final*
- and a single node marked as the *start* node

Each node represents a *state*, with each edge representing a *transition* between states.

Example Finite Automaton



Deterministic Finite Automata (DFAs)

An finite automaton is *deterministic* if

- every state has one edge for each letter in the alphabet
- no edge is labeled ϵ

Most finite automata are *non-deterministic*, violating at least one of the above requirements. DFAs are a special case of NFAs.

DFA Language Recognition

How to think of language recognition with a DFA:

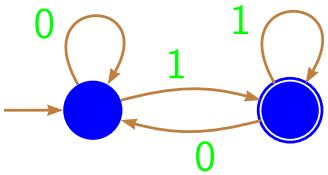
- Think of language recognition as a board game; DFA is the board
- The string is a deck of cards, one card for each letter in the string
- A disc is placed on the start state to begin the game
- Now, draw the top card. Move the disc along the edge with the label that matches the character on the card, and discard the just-drawn card.
- Repeat until you run out of cards.

Was the String Recognized?

- If the disc is in a final state, the string is part of the language and is recognized by the DFA – you win!
- If the disc is on any other state, the start is not recognized – it is not part of the language – you lose!

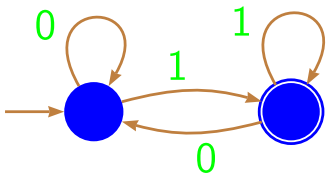
Example DFA

Regular Expression: $(0+1)^*1$



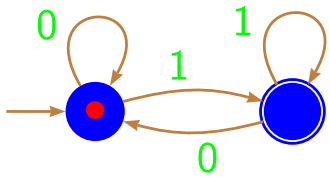
Example DFA

Regular Expression: $(0+1)^*1$
Does this accept 01101?



Example DFA

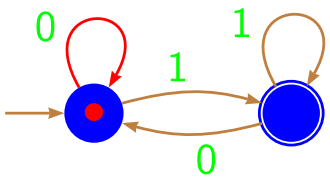
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 0: $\boxed{0}$ 1 1 0 1

Example DFA

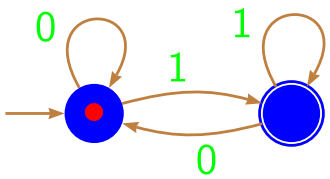
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Move and Discard: ~~0~~ 1 1 0 1

Example DFA

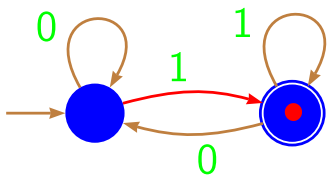
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 1: ~~0~~ $\boxed{1}$ 1 0 1

Example DFA

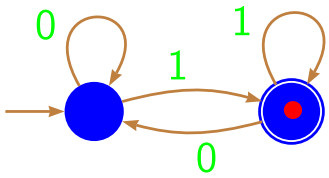
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Move and Discard: ~~0~~ ~~1~~ 1 0 1

Example DFA

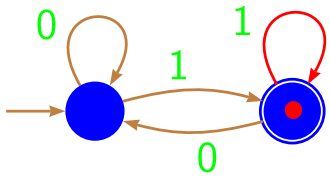
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 1: ~~0~~ ~~1~~ $\boxed{1}$ 0 1

Example DFA

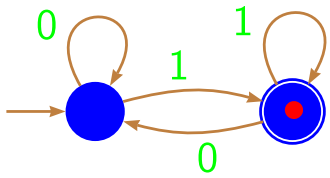
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Move and Discard: ~~0~~ ~~1~~ ~~1~~ 0 1

Example DFA

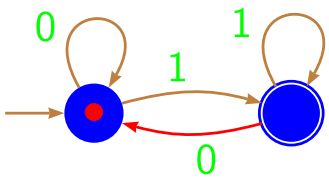
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 0: ~~0~~ ~~1~~ ~~1~~ $\boxed{0}$ 1

Example DFA

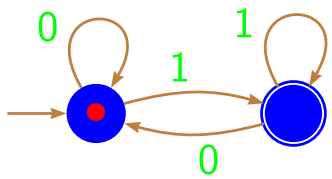
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Move and Discard: $\theta \ \ddagger \ \ddagger \ \theta \ 1$

Example DFA

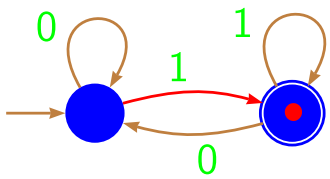
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 1: $\theta \ \ddagger \ \ddagger \ \theta \ \boxed{1}$

Example DFA

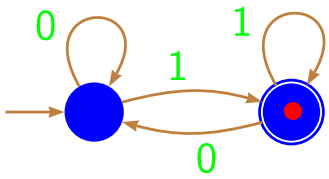
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Move and Discard: $\theta \ \ddagger \ \ddagger \ \theta \ \ddagger$

Example DFA

Regular Expression: $(0+1)^*1$
Does this accept 01101?



Now, where are we? In a final state, so we win! $\theta \ \ddagger \ \ddagger \ \theta \ \ddagger$

Non-Deterministic Finite Automata

NFAs are similar to DFAs, but generalize them in two ways:

- Edges can be labeled with ϵ, allowing the state to change nondeterministically (we can follow it on no input, or stay where we are)
- Each state can have 0 or more edges labeled with each letter, allowing us to nondeterministically choose from multiple edges on a transition

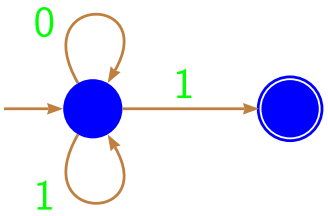
NFA Language Recognition

We'll play a similar game to the DFA game, with a few changes in the rules:

- Edges labeled ϵ give us a free move – no need to discard
- When you run out of letters, if you are in a final state you win!
- You can "take moves back" and try again – back up the disc and put the cards back in the pile
- If you try all possible moves and still can't get to a final state, you lose!

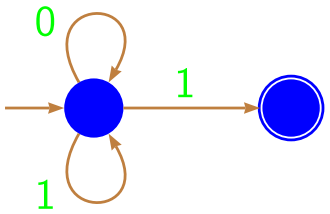
Example NFA

Regular Expression: $(0+1)^*1$



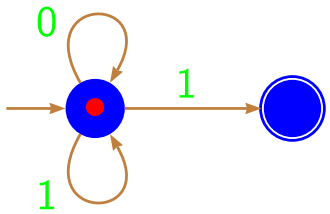
Example NFA

Regular Expression: $(0+1)^*1$
Does this accept 01101?



Example NFA

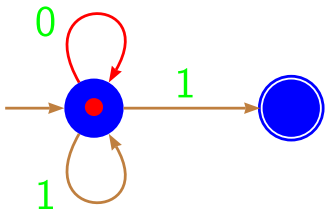
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 0: $\boxed{0}$ 1 1 0 1

Example NFA

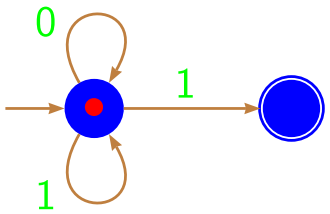
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Move and Discard: ~~0~~ 1 1 0 1

Example NFA

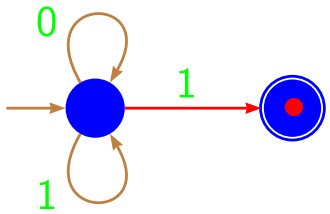
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 1: ~~0~~ $\boxed{1}$ 1 0 1

Example NFA

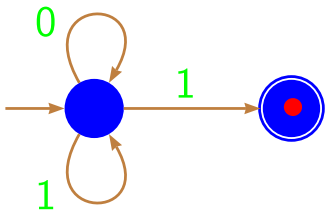
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Guess our move and Discard: ~~0~~ ~~1~~ 1 0 1

Example NFA

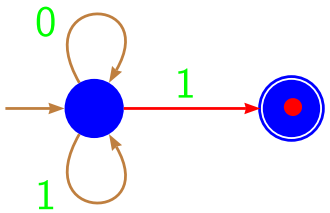
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 1: $\theta \pm \boxed{1}\ 0\ 1$

Example NFA

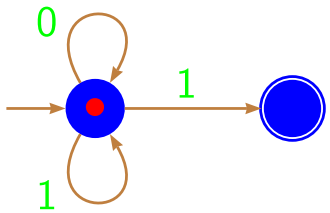
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Oops, need to backtrack... $\theta \pm \boxed{1}\ 0\ 1$

Example NFA

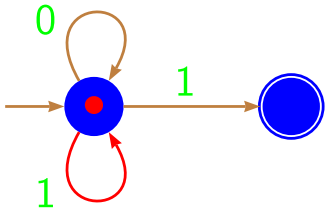
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 1: $\theta\ \boxed{1}\ 1\ 0\ 1$

Example NFA

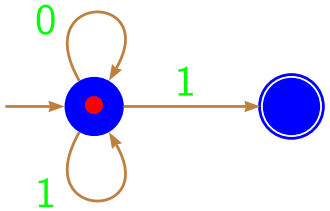
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Try the other move and Discard: $\theta \pm 1\ 0\ 1$

Example NFA

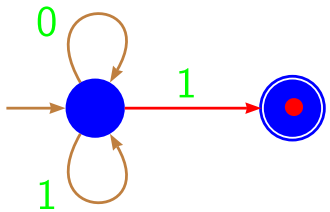
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 1: $\theta \pm \boxed{1}\ 0\ 1$

Example NFA

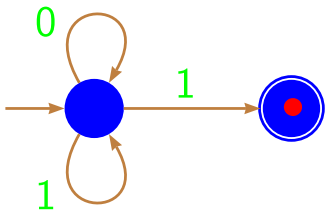
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Guess our move and Discard: $\theta \pm \pm\ 0\ 1$

Example NFA

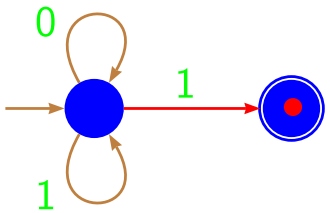
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 0: $\theta \ddagger \ddagger \boxed{0}\ 1$

Example NFA

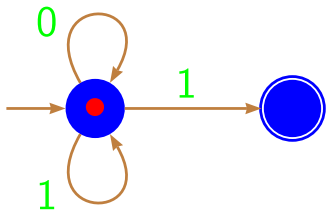
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Oops, need to backtrack... $\theta \ddagger \ddagger \boxed{0}\ 1$

Example NFA

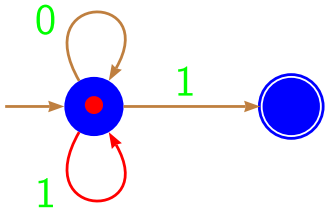
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 1: $\theta \ddagger \boxed{1}\ 0\ 1$

Example NFA

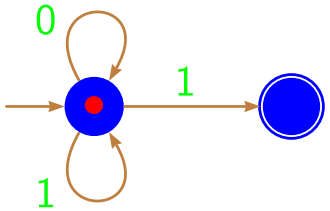
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Try the other move and Discard: $\theta \ddagger \ddagger 0\ 1$

Example NFA

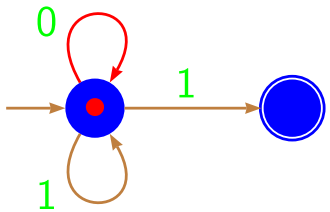
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 0: $\theta \ddagger \ddagger \boxed{0}\ 1$

Example NFA

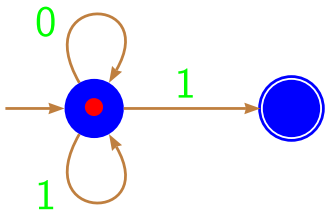
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Move and Discard: $\theta \ddagger \ddagger \theta\ 1$

Example NFA

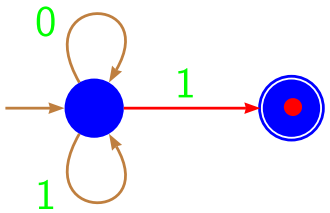
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Draw 1: $\theta \ \ddagger \ \ddagger \ \theta \ \boxed{1}$

Example NFA

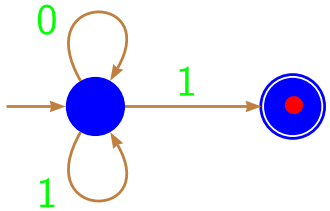
Regular Expression: $(0+1)^*1$
Does this accept 01101?



Guess our move and Discard: $\theta \ \ddagger \ \ddagger \ \theta \ \ddagger$

Example NFA

Regular Expression: $(0+1)^*1$
Does this accept 01101?



No more input, and we are in a final state: we win! $\theta \ \ddagger \ \ddagger \ \theta \ \ddagger$

Rule-based Execution

At its core, this is a *search* problem. We try to move forward to find a solution and, when stuck, *backtrack* to the last choice point and try to proceed differently. If we backtrack all the way back to the start and run out of choices, we know there is no solution. This is the same way rule-based languages (think logic languages like Prolog) work – in our case, the rules are the transitions.