CMSC 141 Automata and Language Theory

Regular Languages

Mark Froilan B. Tandoc

September 10, 2014

Finite Automata and Regular Expressions

Finite automata and regular expressions describe exactly the same class of languages - the class of *regular languages*

What's next?

Simplification of regular expressions and Minimization of finite automata

Simplification of Regular Expressions
Exercise: proofs

- **Identity elements for union and concatenation**
  $\emptyset + x = x + \emptyset = x$
  and
  $\varepsilon x = x\varepsilon = x$
- **Annihilation element for concatenation**
  $\emptyset x = x\emptyset = \emptyset$
- **Commutativity of union**
  $x + y = y + x$
- **Associativity of union, concatenation**
  $(x + y) + z = x + (y + z)$
  and
  $(xy)z = x(yz)$

More Identities
Exercise: proofs

- **Distributive properties**
  $x(y+z) = xy + xz$
  and
  $(x+y)z = xz + yz$
  but
  $(x+y)^* \neq x^* + y^*$
- **Idempotency of union and Kleene closure**
  $x + x = x$
  and
  $(x^*)^* = x^*$
  and
  $(x^+)^+ = x^+$

More Identities
Exercise: proofs

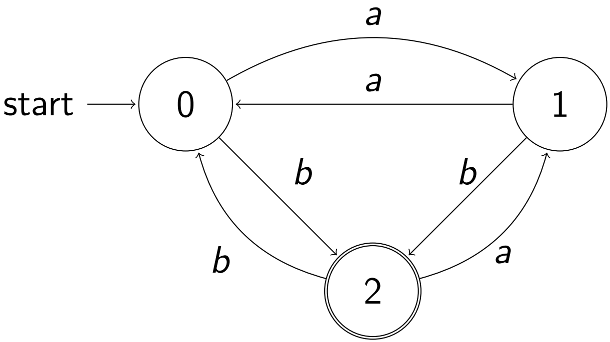
- **Absorption property**
  If $x \subseteq y$ then $x + y = y$
- **Kleene star properties**
  $\varepsilon^* = \varepsilon^+ = \varepsilon$
  and
  $\emptyset^* = \varepsilon$
  and
  $(x^* y^*)^* = (x + y)^*$

Minimization of Finite Automata

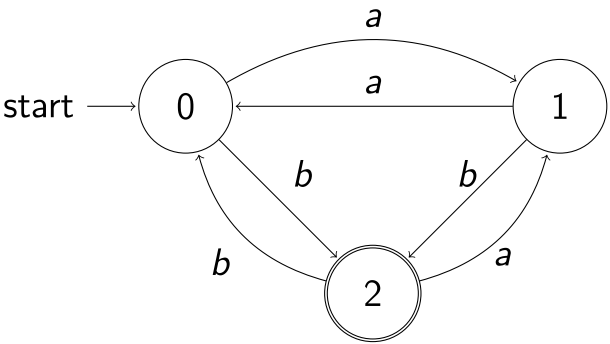
- Idea is to identify states which are essentially the same, or *indistinguishable*, and merge them into a single state.
- Easy to do with graphical tools like JFLAP where we can drag states around.

Distinguishable States

- States p and q are **distinguishable** if
  - one is a final state and the other is a non-final state, or
  - there is some string $x \in \Sigma^*$, such that $\delta(p, x)$ and $\delta(q, x)$ are distinguishable



Minimization of FA

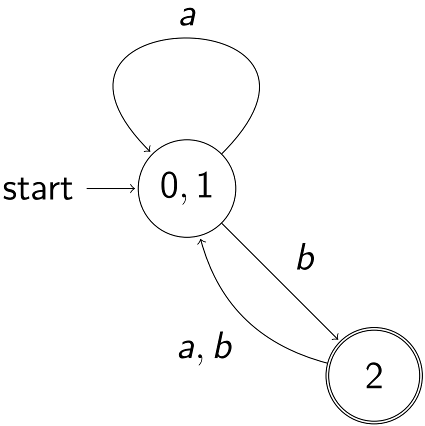


	0	1	2
0	-		X
1		-	X
2	X	X	-

States 0 and 1 are indistinguishable

for any string x, $\delta(0, x)$ and $\delta(1, x)$ are both non-final states.

Minimized FA



Regular Grammars

- **Grammars** are rule-based systems for describing languages.

  Regular grammars is for regular languages
- Example: The regular grammar below consists of a single variable $\{S\}$, two terminals $\{rose, red\}$ and two production rules $\{S \to rose, S \to red\ S\}$
  (can also be written as $\{S \to rose | red\ S\}$)
- This grammar generates the language $\{rose, red\ rose, red\ red\ rose, ...\}$

Formal Definition of Regular Grammars

A **regular grammar** is a 4-tuple (V, T, P, S) where

- V is a finite set of variables
- T is a finite set of terminal symbols $= \Sigma$
- P is a finite set of production rules, each of the form

  $< variable > \rightarrow < terminal >^*$

  or

  $< variable > \rightarrow < terminal >^* < variable >$
- S is the start variable, $S \in V$

Derivation

- A grammar G is said to generate a string x, if x can be derived from the start variable S, by finite sequence of variable replacements based on the production rules

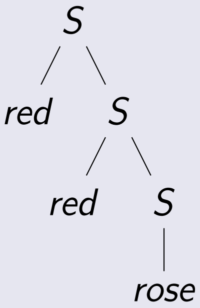
Example Derivation

Production rules: $\{S \rightarrow rose | red\ S\}$
String: "red red rose"

linear derivation

$$
\begin{aligned}
S &\rightarrow red\ S \\
&\rightarrow red\ red\ S \\
&\rightarrow red\ red\ rose
\end{aligned}
$$

parse tree



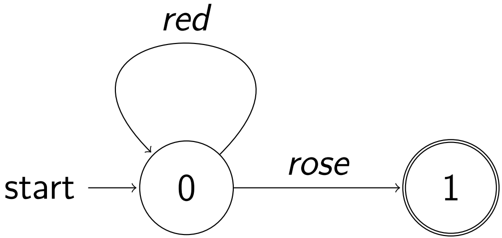
Note that $L(G) = (red)^* rose$

Regular Grammars = FA

Idea of proof:

- FA *states* are *variables* in the regular grammar
- The *start state* is the *start symbol*
- $\delta(X, a) = Y$ if and only if the rule $X \rightarrow aY$ is present
- *X is a final state* if and only if the rule $X \rightarrow \varepsilon$ is present

$S \rightarrow \mathit{red}\ S | \mathit{rose}\ T$
$T \rightarrow \varepsilon$



Example

Regular Expression

$$(0+1)^*11(0+1)^*$$

Regular Grammar

$$S \to 0S \mid 1S \mid 11T$$
$$T \to 0T \mid 1T \mid \varepsilon$$

Derivation Example

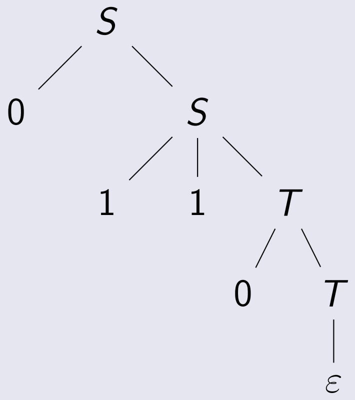
Regular Grammar

$$S \to 0S \mid 1S \mid 11T$$
$$T \to 0T \mid 1T \mid \varepsilon$$

0110

$$\begin{aligned} S &\to 0S \\ &\to 011T \\ &\to 0110T \\ &\to 0110 \end{aligned}$$

parse tree



References

- Previous slides on CMSC 141
- M. Sipser. Introduction to the Theory of Computation. Thomson, 2007.
- J.E. Hopcroft, R. Motwani and J.D. Ullman. Introduction to Automata Theory, Languages and Computation. 2nd ed, Addison-Wesley, 2001.
- E.A. Albacea. Automata, Formal Languages and Computations, UPLB Foundation, Inc. 2005
- JFLAP, www.jflap.org