Regular Languages, Continued

March 28, 2011
CS 81: Computability and Logic
A Jewel of Theoretical Computer Science

The following are equivalent:

1. There is a DFA accepting the language $L$
2. [Rabin and Scott] There is an NFA accepting $L$
3. [Kleene] $L$ is a regular set.
Completing the Equivalence: Automata to Regular Expressions

Two approaches:

1. Solving equations
2. Generalized NFAs
Let $L_q$ be the set of strings are accepted when starting from state $q$.

✓ What is $L_{q_0}$, $L_{q_1}$, $L_{q_2}$, ...?

✓ How is $L_{q_1}$ related to $L_{q_2}$?
**Automaton as a System of Equations**

\[ L_A = \varepsilon L_B \cup bL_D \]

\[ L_B = \]

\[ L_C = \]

\[ L_D = \]
Solving Equations using Arden’s Rule

✓ The equation

\[ L = AL \cup B \]

has the solution

\[ L = A^*B \]

✓ This is the smallest solution

- If \( \epsilon \notin A \), the unique solution
- Otherwise \( A^*C \) is a solution for any \( B \subseteq C \).

\[
\begin{align*}
L_A &= L_B \cup bL_D \\
L_B &= \epsilon \cup bL_A \cup aL_C \\
L_C &= \epsilon \cup aL_D \\
L_D &= (a \cup b)L_D \cup bL_C
\end{align*}
\]
**Generalized NFAs**

Just like an NFA, but edges have regular expressions rather than single symbols.

![Diagram](image)

Since regular expressions can be turned into NFAs, we aren’t adding any extra power.
**Regexp by Removing States**

The strategy:

✓ Make sure our NFA has
  - One start state, with edges only going out
  - One accept state, with edges only going in.
**Reexp by Removing States**

The strategy:
- Make sure our NFA has
  - One start state, with edges only going out
  - One accept state, with edges only going in.

![Diagram of a graph with labeled nodes and edges representing a regular expression.](image)
**Regexp by Removing States**

The strategy:

- Make sure our NFA has
  - One start state, with edges only going out
  - One accept state, with edges only going in.

- Remove all the intermediate states (A–D), one at a time.
- In the end, we have one edge, labeled by our regexp.
Removing States

✓ When removing state $q$, replace every pair of in/out edges by a single edge

$UV*W$
**Example**

A NFA with states labeled A, B, C, and D, and transitions labeled with symbols a, b, and ε (epsilon). The starting state is S, and the accepting state is Z.
Regular Expression Review

✓ What are the usual “primitive” regular expressions?

✓ What other regular expressions have you seen?

Give two strings matching this regular expression.
Give two strings matching this regular expression.

<table>
<thead>
<tr>
<th>String</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muammar Qaddafi</td>
<td>Moamar Gaddafi</td>
</tr>
<tr>
<td>Mo'ammar Gadhafi</td>
<td>Mu'ammar Qadhdhafi</td>
</tr>
<tr>
<td>Muammar Kaddafi</td>
<td>Muammar al-Khaddafi</td>
</tr>
<tr>
<td>Muammar Qadhafi</td>
<td>Mu'am al-Kadafi</td>
</tr>
<tr>
<td>Moammar El Kadhafi</td>
<td>Muammar Ghaddafy</td>
</tr>
<tr>
<td>Muammar Gadafi</td>
<td>Muammar Ghadafi</td>
</tr>
<tr>
<td>Mu'ammar al-Qadafi</td>
<td>Muammar Ghaddafy</td>
</tr>
<tr>
<td>Moamer El Kazzafi</td>
<td>Muammar Kaddafi</td>
</tr>
<tr>
<td>Moamar al-Gaddafi</td>
<td>Muammar Quathafi</td>
</tr>
<tr>
<td>Mu'ammar Al Qathafi</td>
<td>Muammar Gheddafi</td>
</tr>
<tr>
<td>Muammar Al Qathafi</td>
<td>Muammar Al-Kaddafi</td>
</tr>
<tr>
<td>Mo'ammar el-Gadhafi</td>
<td>Moammar Khadafy</td>
</tr>
<tr>
<td>Moamar El Kadhafi</td>
<td>Moammar Qudhafi</td>
</tr>
<tr>
<td>Muammar al-Qadhafi</td>
<td>Mu'ammar al-Qaddafi</td>
</tr>
<tr>
<td>Mu'ammar al-Qadhdhafi</td>
<td>Mu'ammar Muhammad Abu Minyar al-Qadhafi</td>
</tr>
<tr>
<td>Mu'ammar Qadafi</td>
<td></td>
</tr>
</tbody>
</table>
**Exercise**

Give a regular expression for C identifiers:
- ✓ Can contain letters, digits, and underscores
- ✓ Must begin with a letter or underscore.
- ✓ E.g., `main` or `__Z6rotateiii`

Give a regular expression for Ada identifiers, which
- ✓ Can contain letters, digits and underscores
- ✓ Begin with a letter
- ✓ Have no consecutive underscores or an underscore at the end
- ✓ E.g., `woohoo32` or `Last_Nonzero_Row`
A2.5.1 Integer Constants

An integer constant consisting of a sequence of digits is taken to be octal if it begins with 0 (digit zero), decimal otherwise. Octal constants do not contain the digits 8 or 9. A sequence of digits preceded by 0x or 0X (digit zero) is taken to be a hexadecimal integer. The hexadecimal digits include a or A through f or F with values 10 through 15.

An integer constant may be suffixed by the letter u or U, to specify that it is unsigned. It may also be suffixed by the letter l or L to specify that it is long.

The type of an integer constant depends on its form, value and suffix. (See §A4 for a discussion of types.) If it is unsuffixed and decimal, it has the first of these types in which its value can be represented: int, long int, unsigned long int. If it is unsuffixed octal or hexadecimal, it has the first possible of these types: int, unsigned int, long int, unsigned long int. If it is suffixed by u or U, then unsigned int, unsigned long int. If it is suffixed by l or L, then long int, unsigned long int.

The elaboration of the types of integer constants goes considerably beyond the first edition, which merely caused large integer constants to be long. The U suffixes are new.

A2.5.2 Character Constants
Exercise: Integer Constants in C

Does your regular expression match:
Exercise: Integer Constants in C

Does your regular expression match:

✓ 6
Exercise: Integer Constants in C

Does your regular expression match:

✓ 6
✓ 9
Exercise: Integer Constants in C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
Exercise: Integer Constants in C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
Exercise: Integer Constants in C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
✓ 2UL
**Exercise: Integer Constants in C**

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
✓ 2UL
✓ 2lu
EXERCISE: INTEGER CONSTANTS IN C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
✓ 2UL
✓ 2lu
✓ 2uu
EXERCISE: INTEGER CONSTANTS IN C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
✓ 2UL
✓ 2lu
✓ 2uu
✓ 02
Exercise: Integer Constants in C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
✓ 2UL
✓ 21u
✓ 2uu
✓ 02
✓ 002
**Exercise: Integer Constants in C**

Does your regular expression match:

- ✓ 6
- ✓ 9
- ✓ 2u
- ✓ 2L
- ✓ 2UL
- ✓ 2lu
- ✓ 2uu
- ✓ 02
- ✓ 002
- ✓ 09
Exercise: Integer Constants in C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
✓ 2UL
✓ 2lu
✓ 2uu
✓ 02
✓ 002
✓ 09
✓ 02u1
Exercise: Integer Constants in C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
✓ 2UL
✓ 2lu
✓ 2uu
✓ 02
✓ 002
✓ 002
✓ 02u1
✓ 0Xff90
✓ 0x
Exercise: Integer Constants in C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
✓ 2UL
✓ 2lu
✓ 2uu
✓ 02
✓ 002
✓ 09
✓ 02u1
✓ 0Xff90
✓ 0x0
Exercise: Integer Constants in C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
✓ 2UL
✓ 2lu
✓ 2uu
✓ 02
✓ 002
✓ 09
✓ 02u1
✓ 0Xff90
✓ 0x0
✓ 0x0F
Exercise: Integer Constants in C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
✓ 2UL
✓ 2lu
✓ 2uu
✓ 02
✓ 002
✓ 09
✓ 02u1
✓ 0Xff90
✓ 0x0
✓ 0x0F
✓ 0x0F1U
**Exercise: Integer Constants in C**

Does your regular expression match:

- ✓ 6
- ✓ 9
- ✓ 2u
- ✓ 2L
- ✓ 2UL
- ✓ 2lu
- ✓ 2uu
- ✓ 02
- ✓ 002
- ✓ 09
- ✓ 02u1
- ✓ 0Xff90
- ✓ 0x0
- ✓ 0x0F
- ✓ 0x0F1U
- ✓ 0x0x3
Exercise: Integer Constants in C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
✓ 2UL
✓ 2lu
✓ 2uu
✓ 02
✓ 002
✓ 09
✓ 02u1
✓ 0Xff90
✓ 0x0
✓ 0xF
✓ 0x0F1U
✓ 0x0xF1U
✓ 0x0xF1U
✓ 0x0x3
✓ 0
Exercise: Integer Constants in C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
✓ 2UL
✓ 2lu
✓ 2uu
✓ 02
✓ 002
✓ 09
✓ 02ul
✓ 0Xff90
✓ 0x0
✓ 0x0F
✓ 0x0F1U
✓ 0x0x3
✓ 0
✓ 01u
EXERCISE: INTEGER CONSTANTS IN C

Does your regular expression match:

✓ 6
✓ 9
✓ 2u
✓ 2L
✓ 2UL
✓ 2lu
✓ 2uu
✓ 02
✓ 002
✓ 09
✓ 02ul
✓ 0Xff90
✓ 0x0
✓ 0x0F
✓ 0x0F1U
✓ 0x00x3
✓ 0
✓ 0lu
✓ 0x
**Closure Properties**

A family of languages is a set of languages.

- The family of all finite languages
- The family of all languages
- The family of all regular languages

A family $\mathcal{F}$ is **closed under an operation** if applying the operation to languages in $\mathcal{F}$ always produces a result in $\mathcal{F}$.
Finite Languages

Is the family of finite languages closed under:

✓ Union? \((A \cup B)\)
✓ Intersection \((A \cap B)\)
✓ Concatenation? \((AB)\)
✓ Star \((A^*)\)
✓ Complement \((A^c)\)
Regular Languages

The regular languages are closed under

✓ Union? \((A \cup B)\)
✓ Intersection \((A \cap B)\)
✓ Concatenation? \((AB)\)
✓ Star \((A^*)\)
✓ Complement \((A^c)\)

Proofs: Consider the corresponding automata...
COMPLEMENT?
COMPLEMENT!

A \xrightarrow{a} X \xrightarrow{b,c} B

A \xrightarrow{a} X \xrightarrow{b,c} Z \xrightarrow{a,b,c} B

X \xrightarrow{b} b
Z \xrightarrow{a,b,c} a,b,c
COMPLEMENT!
**Complement!**

\[ \text{DFA } M = (\Sigma, Q, \rightarrow, q_0, F) \]
\[ \text{DFA } M^c = (\Sigma, Q, \rightarrow, q_0, Q \setminus F) \]
**Intersection: Inputs**

- 

  - **A** → **X** → **B**
  - **L** → **K**

  - Transitions:
    - **A** to **X** on **a**
    - **X** to **B** on **a**
    - **L** to **K** on **a**
    - **L** to **K** on **b**
**INTERSECTION: PRODUCT AUTOMATON**
**Intersection: Product Automaton**

\[
\text{DFA } M = (\Sigma, Q, \rightarrow, q_0, F) \\
\text{DFA } M' = (\Sigma, Q', \rightarrow', q'_0, F') \\
\text{DFA } M \cap M' = (\Sigma, Q \times Q', \rightarrow_{\text{both}}, \langle q_0, q'_0 \rangle, F \times F').
\]