A nice paper relevant to this course is titled “The Glory of the Past”

NICTA Researcher,
Adjunct at the Australian National University and Griffith University

$\epsilon$-Closure, Kleene’s Theorem, and Kleene’s Algebra

Charles Gretton
24 February 2014
First, a Quick Summary of $\varepsilon$-Closure

- I’ve previously waived my hands about $\varepsilon$-labelled arcs.

- Recall, $\varepsilon$ is our notation for the empty string.

- How can we formally treat $\varepsilon$-labelled arcs in an NFA?

- Using a function called ECLOSE...

- A construction similar to that for NFA $\Longleftrightarrow$ DFA is used to prove $\varepsilon$-NFA $\Longleftrightarrow$ DFA
Quick Summary of $\epsilon$-Closure

- $\epsilon$-NFA is $\langle Q, \Sigma, \delta, q_0, F \rangle$.

- $\text{ECLOSE}: Q \rightarrow 2^Q$ – i.e. gives an element in the powerset of states.

- It is defined inductively, as follows.
  
  - **BASIS**: $q \in \text{ECLOSE}(q)$ – i.e. the argument is in the basis.
  
  - **INDUCTION**: Suppose $p \in \text{ECLOSE}(q)$ and $r \in \delta(p, \epsilon)$, then $r \in \text{ECLOSE}(q)$.
Quick Example of $\epsilon$-Closure

- **ECLOSE**: $Q \rightarrow 2^Q$ – i.e. gives an element in the powerset of states.
- It is defined inductively, as follows.
  - **BASIS**: $q \in \text{ECLOSE}(q)$ – i.e. the argument is in the basis.
  - **INDUCTION**: Suppose $p \in \text{ECLOSE}(q)$ and $r \in \delta(p, \epsilon)$, then $r \in \text{ECLOSE}(q)$.

```
1 --ε→ 2 --ε→ 3 --ε→ 6
  |      |      |
  ε      ε      ε
```

```
4 --a→ 5 --ε→ 7
  |      |
  ε      ε
```

???
Quick Example of $\epsilon$-Closure

- **ECLOSE**: $Q \rightarrow 2^Q$ – i.e. gives an element in the powerset of states.
- It is defined inductively, as follows.
  - **BASIS**: $q \in \text{ECLOSE}(q)$ – i.e. the argument is in the basis.
  - **INDUCTION**: Suppose $p \in \text{ECLOSE}(q)$ and $r \in \delta(p, \epsilon)$, then $r \in \text{ECLOSE}(q)$.

ECLOSE(1) = \{1, 2, 3, 4, 6\}
\( \varepsilon \)-Closure Subset Construction

- For DFA \( \rightarrow \varepsilon \)-NFA, just add \( \delta(q, \varepsilon) = \emptyset \) transitions to all states – i.e. \( \varepsilon \) is **not** an alphabet symbol.

- \( \varepsilon \)-NFA \( \rightarrow \) DFA
  - Same as for NFA \( \rightarrow \) DFA,
  - Each DFA state corresponds to a unique set of \( \varepsilon \)-NFA states.
  - Only take the \( \varepsilon \)-closure at \( \varepsilon \)-NFA transitions.
Summarizing

- NFA $\leftrightarrow$ DFA
- The equivalent NFA can be exponentially smaller than the corresponding DFA.
- $\epsilon$-NFA $\leftrightarrow$ DFA
- From transitivity: $\epsilon$-NFA $\leftrightarrow$ NFA
- All formalisms so far are expressively equivalent.
Languages and Regular Expressions

- Regular expression implementations in many UNIX tools: `grep`, `sed`, `awk`, `emacs` and `vi`

- Following Kleene's seminal work, we deal with regularity formally.

- We describe the language intended by a given regular expression.

- We consider the class of languages that can be expressed.

- We examine algebraic laws which inform re-writing rules for simplifying expressions.
Operations on Languages

- \( L_1, L_2 \subseteq \Sigma^* \)
- product of languages

\[
L_1 \cdot L_2 = L_1 L_2 = \{WX | W \in L_1, X \in L_2\}
\]

- inductive \( i^{th} \) product

\[
\begin{align*}
L^0 & = \{\varepsilon\} \\
L^* & = \bigcup_{i \geq 0} L^i \\
\forall i > 0, L^i & = L L^{i-1}
\end{align*}
\]

- positive closure

\[
L^+ = LL^*
\]
A regular expression is a well formed word over a given alphabet $\Sigma$.

We use the following symbols:

\[
\Sigma \cup \{ \epsilon, \emptyset, ( ), +, ., * \}
\]

- alphabet
- empty string
- empty set
- open parenthesis
- close parenthesis
- “union” or “choice”
- concatenation – $X.Y$ often written as $XY$
- Kleene closure – the “star” of this show
Regular Expressions

- A **regular expression** is a well formed word over a given alphabet $\Sigma$
- We use the following symbols:
  
  $\Sigma \cup \{ \epsilon, \emptyset, ( ), +, ., * \}$

  - **“symbol”** :: alphabet
  - **constant (or 0-ary operator)** :: empty string
  - **constant (or 0-ary operator)** :: empty set
  - **punctuation** :: open parenthesis
  - **punctuation** :: close parenthesis
  - **infix binary operator** :: “union” or “choice”
  - **infix binary operator** :: concatenation – $X.Y$ often written as $XY$
  - **postfix unary operator** :: Kleene closure – the “star” of this show

  "Kleene's Theorem Copyright NICTA 2014"
Regular Expressions

- We use the following symbols:

\[ \Sigma \cup \{ \epsilon, \emptyset, ( ), +, ., * \} \]

- Taking \( a \in \Sigma \cup \{ \epsilon, \emptyset \} \), regular expressions satisfy the following grammar:

\[
\text{regexp} ::= a | (\text{regexp}) | \text{regexp} + \text{regexp} | \text{regexp} . \text{regexp} | \text{regexp}^*
\]

- The Kleene closure operator, \( * \), has the highest precedence.
- Concatenation, \( . \), is usually represented by juxtaposition – e.g. concatenation \( a.b \) is written \( ab \).
- Concatenation has the next highest precedence.
- Union/choice has the least precedence.
Regular Expression Examples

- Taking \( a \in \Sigma \cup \{\epsilon, \emptyset\} \), regular expressions satisfy the following grammar:

\[
\text{regexp ::= } a | (\text{regexp}) | \text{regexp} + \text{regexp} | \text{regexp} \cdot \text{regexp} | \text{regexp}^*
\]

- So, which of the following is correct?

\[
\begin{align*}
ab^* + a &= (a(b^* + a)) \\
ab^* + a &= (ab)^* + a \\
ab^* + a &= (a(b^*)) + a
\end{align*}
\]
Regular Expression Examples

- Taking $a \in \Sigma \cup \{\epsilon, \emptyset\}$, regular expressions satisfy the following grammar:

$$\text{regexp} ::= a | (\text{regexp}) | \text{regexp} + \text{regexp} | \text{regexp} \cdot \text{regexp} | \text{regexp}^*$$

- So, which of the following is correct?

$$ab^* + a = (a(b^* + a))$$
$$ab^* + a = (ab)^* + a$$
$$ab^* + a = (a(b^*)) + a$$
Semantics of Regular Expressions

• BASIS

<table>
<thead>
<tr>
<th>Language of Expression</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{L}(\epsilon)$</td>
<td>${\epsilon}$</td>
</tr>
<tr>
<td>$\mathcal{L}(\emptyset)$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\mathcal{L}(a), \ a \in \Sigma$</td>
<td>${a}$</td>
</tr>
</tbody>
</table>

• INDUCTION

Below, $L_1$ and $L_2$ are well formed regular expressions.

<table>
<thead>
<tr>
<th>Language of Expression</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{L}(L_1 + L_2)$</td>
<td>$\mathcal{L}(L_1) \cup \mathcal{L}(L_2)$</td>
</tr>
<tr>
<td>$\mathcal{L}(L_1L_2)$</td>
<td>$\mathcal{L}(L_1)\mathcal{L}(L_2)$</td>
</tr>
<tr>
<td>$\mathcal{L}(L_1^*)$</td>
<td>$\mathcal{L}(L_1)^*$</td>
</tr>
<tr>
<td>$\mathcal{L}((L_1))$</td>
<td>$\mathcal{L}(L_1)$</td>
</tr>
</tbody>
</table>
Every Finite Language is Regular

- A regular language is a language that can be given by a regular expression.
- A finite language is a finite set of strings:
  \[ \{ X_1, X_2, \ldots, X_n \} \]
- The regular expression that gives that finite language is:
  \[ X_1 + X_2 + \ldots + X_n \]
- QED
Algebra and Regular Expressions

- Commutativity laws:

\[ L_1 + L_2 = L_2 + L_1 \]
Algebra and Regular Expressions

- Cancellation laws:
  - Identities
    - $L + \emptyset = L$
    - $\emptyset + L = L$
    - $\epsilon L = L$
    - $L \epsilon = L$

- Annihilators
  - $L \emptyset = \emptyset$
  - $\emptyset L = \emptyset$
Algebra and Regular Expressions

- **Associative laws:**

  \[ L_1 + (L_2 + L_3) = (L_1 + L_2) + L_3 \]
  \[ L_1(L_2L_3) = (L_1L_2)L_3 \]

- **Power associativity**

  \[ L_1(L_1L_1) = (L_1L_1)L_1 \]
  \[ \ldots \]
  \[ (L_1L_1)(L_1L_1) = (L_1(L_1L_1))L_1 \]
Algebra and Regular Expressions

- *Left* and *right* distributive laws:

\[
L_1(L_2 + L_3) = L_1L_2 + L_1L_3 \\
(L_2 + L_3)L_1 = L_2L_1 + L_3L_1
\]

- Idempotence:

\[
L + L = L
\]

- *Closure laws* – There are *exactly* two languages whose closures are finite. *What are they?*
Algebra and Regular Expressions

- **Left and right distributive laws:**

  \[ L_1(L_2 + L_3) = L_1L_2 + L_1L_3 \]

  \[ (L_2 + L_3)L_1 = L_2L_1 + L_3L_1 \]

- **Idempotence:**

  \[ L + L = L \]

- **Closure laws**

  \[ \epsilon^* = \epsilon \]

  \[ \emptyset^* = \epsilon \]

  \[ (L^*)^* = L^* \]
Application of Algebraic Laws

- $a + ab^*$

- **precedence:** $(a + (a(b^*)))$

- **annihilators:** $(a\epsilon + (a(b^*)))$

- **left distributive law:** $a(\epsilon + b^*)$

- **cancellation law:** $a(b^*)$

- **CONCLUSION:** $ab^* = a + ab^*$
When I write \((L^*)^* = L^*\), I intended that \(L\) could be any regular expression.

For example, the following concrete expressions are instances of this identity.

1. \(((a)^*)^* = a^*\)
2. \(((\epsilon(a + bab + \emptyset) + bab)^*)^* = (\epsilon(a + bab + \emptyset) + bab)^*\)

The above are called **concrete** regular expressions, because they contain no **variables**.

Our textbook talks about a specific class of concrete expression (above, Type 1), where each variable symbol from the lifted expression is replaced with an alphabet constant.
Theorem Relating Concrete and Lifted Expressions

Let $E$ be a regular expression with variable symbols $L_i$, $i \in [1..m]$.

For example taking $m = 2$, $E$ could be $(L_1 + L_2)^*$, $(L_1 L_2^*)^*$, etc.

Let $E'$ be an expression where each variable $L_i$ is replaced by a constant $a_i$.

Every $X \in \mathcal{L}(E)$ can be written as $X_1 X_2 .. X_k$, where each substring $X_j$ is in one of the languages $\mathcal{L}(L_i)$.

Writing $L(X_j) = i$ if $X_j$ is from the $i^{th}$ language,

$a_{L(X_1)} a_{L(X_2)} .. a_{L(X_k)} \in \mathcal{L}(E')$
Theorem Relating Concrete and Lifted Expressions

- **Basis**: \( E \in \{\epsilon, \emptyset, L_1\} \).

- The cases \( E \in \{\epsilon, \emptyset\} \) are trivial, because the concrete and lifted expressions are identical.

- Taking \( E = L_1 \), then \( \mathcal{L}(E) = \mathcal{L}(L_1) \). Also \( E' = a_1 \) and \( \mathcal{L}(E') = \mathcal{L}(a_1) = \{a_1\} \).

- Above, the \( L_1/a_1 \) correspondence is clearly satisfied.
Theorem Relating Concrete and Lifted Expressions

- **INDUCTION:** \( E \in \{ E_1 + E_2, E_1E_2, E_1^* \} \).

- \( E'_1 \) and \( E'_2 \) are constructed so that if a variable appears in both subexpressions, then that is substituted with the same concrete symbol.

- For example, take \( E = L_1L_2 + L_2L_1 \).

- Then a valid concrete expression is: \( E = a_1a_2 + a_2a_1 \).

- An invalid concrete expression would be: \( E = a_1a_2 + a_3a_4 \).
Theorem Relating Concrete and Lifted Expressions

- **INDUCTION**: \( E \in \{ E_1 + E_2, E_1 E_2, E_1^* \} \).

- \( E_1' \) and \( E_2' \) are constructed so that if a variable appears in both subexpressions, then that is substituted with the same concrete symbol.

- \( \mathcal{L}(E') = \mathcal{L}(E_1' + E_2') = \mathcal{L}(E_1') \cup \mathcal{L}(E_2') \)

- \( \mathcal{L}(E) = \mathcal{L}(E_1 + E_2) = \mathcal{L}(E_1) \cup \mathcal{L}(E_2) \)

- Our inductive hypothesis gives us the correspondence for strings in \( \mathcal{L}(E_1)/\mathcal{L}(E_1') \).

- Also for strings in \( \mathcal{L}(E_2)/\mathcal{L}(E_2') \).

- Following the definition of union, we have proved this step case.
Theorem Relating Concrete and Lifted Expressions

- **INDUCTION**: \( E \in \{ E_1 + E_2, E_1 E_2, E_1^* \} \).

- \( E'_1 \) and \( E'_2 \) are constructed so that if a variable appears in both subexpressions, then that is substituted with the same concrete symbol.

- \( \mathcal{L}(E') = \mathcal{L}(E'_1 \cdot E'_2) = \mathcal{L}(E'_1) \cdot \mathcal{L}(E'_2) \)

- \( \mathcal{L}(E) = \mathcal{L}(E_1 \cdot E_2) = \mathcal{L}(E_1) \cdot \mathcal{L}(E_2) \)

- Our inductive hypothesis gives us the correspondence for strings in \( \mathcal{L}(E_1) / \mathcal{L}(E'_1) \).

- Also for strings in \( \mathcal{L}(E_2) / \mathcal{L}(E'_2) \).

- Following the definition of concatenation, we have proved this step case.
Theorem Relating Concrete and Lifted Expressions

- **INDUCTION**: $E \in \{E_1 + E_2, E_1 E_2, E_1^*\}$.

- $\mathcal{L}((E_1')^*) = \mathcal{L}(\epsilon) \cup \mathcal{L}(E_1')^+$

- $\mathcal{L}(E_1^*) = \mathcal{L}(\epsilon) \cup \mathcal{L}(E_1)^+$

- Our inductive hypothesis gives us the correspondence for strings in $\mathcal{L}(E_1)/\mathcal{L}(E_1')$.

- Because closure yields zero or more occurrences of the language strings, we also have a correspondence between $\mathcal{L}(E_1)^*/\mathcal{L}(E_1')^*$. 

- QED
Theorem Relating Concrete and Lifted Expressions

• What if we added to the step cases:
  \( E \in \{ E_1 \cap E_2, E_1 + E_2, E_1 E_2, E_1^* \} \)?

• Let's propose a law \( L_1 \cap L_2 \cap L_3 = L_1 \cap L_2 \)

• The law holds for a concretization: \( \mathcal{L}(a \cap b \cap c) = \mathcal{L}(a \cap b) = \emptyset \)

• But fails generally: \( \mathcal{L}(a \cap a \cap \emptyset) \neq \mathcal{L}(a \cap a) \)

• So, the theorem we just proved is rather specific to regular expressions.

Kleene's Theorem  Copyright NICTA 2014

Charles Gretton^2  30/30