Derivatives of Languages and Regular Expressions

Robert M. Keller
Harvey Mudd College
April 2014
Why This?

- The concept of language derivative is related to the Myhill-Nerode Theorem.

- Furthermore, the concept can applied computationally to regular expressions directly, giving an alternate way to construct a DFA from a regular expression, *without* an intermediate NFA.

- Derivatives provide a systematic method for contacting the states of the *minimal* language machine (= the Myhill-Nerode classes).
Derivative of a Language

- Let $L \subseteq \Sigma^*$ be a language.
- Let $x \in \Sigma^*$ (not necessarily in the language).
- Define $L/x = \{y \mid xy \in L\}$.
- $L/x$ is called the derivative of $L$ wrt $x$.
- Informally, $L/x$ consists of members of $L$ with any initial $x$ “lopped off”. If $x$ cannot be lopped of from a member of $L$, then it is not included.
- $L/x$ is $[x]$, the Myhill-Nerode Equivalence class of $x$. 
Examples

- \( \{01, 011\}/0 = \{1, 11\} \)
- \( \{01, 11\}/0 = \{1\} \) (no contribution from 11)
- \( \{(01)^n | n \in \mathbb{N}\}/0 = \{1(01)^n | n \in \mathbb{N}\} \)
- \( \{(01)^n | n \in \mathbb{N}\}/1 = \emptyset \)
- \( \{0^n1^n | n \in \mathbb{N}\}/0^m = \{0^{(n-m)}1^n | n \geq m\} \)
Chained Derivatives

• For any $x, y \in \Sigma^*$
  \[ L/(xy) = (L/x)/y \]

• That is, we can infer the derivative for a long string in steps.
Derivatives of Unions

If $L, M \subseteq \Sigma^*$

$\frac{L \cup M}{x} = \frac{L}{x} \cup \frac{M}{x}$
Derivatives of Language Concatenations

(This doesn’t work for arbitrary strings \( x \), only for letters \( \sigma \).)

If \( L, M \subseteq \Sigma^* \) and \( \sigma \in \Sigma \)

\[
(LM)/\sigma = (L/\sigma)M \quad \text{if } \varepsilon \notin L
\]

\[
(LM)/\sigma = (L/\sigma)M \cup M/\sigma \quad \text{if } \varepsilon \in L
\]
Note on Distinctness of Derivatives

• \( L/x = \{ y \mid xy \in L \} \).

• It is entirely possible that for some \( x' \neq x \) it is still possible that \( L/x = L/x' \).

• Examples:
  • \( \{0\}^*/0 = \{0\}^* = \{0^*\}/00 \)
  • \( \{0\}/1 = \emptyset = \{0\}/11 \)
Example

- Consider \( L = \{x \in \{0, 1\}^* \mid x \text{ has at most 2 1's}\} \).
We can derive derivatives as follows, then construct a DFA:
- \( L/\epsilon = L \)
- \( L/0 = L \) (because 0 doesn’t change the number of 1’s).
- \( L/1 = \{x \in \{0, 1\}^* \mid x \text{ has at most one 1}\} \)
- \( L/01 = (L/0)/1 = L/1 \)
- \( L/11 = (L/1)/1 = \{x \in \{0, 1\}^* \mid x \text{ has no 1}\} = \{0\}^* \)
- \( L/011 = (L/0)/11 = L/11 = \{0\}^* \)
- \( L/111 = \emptyset \)
Derivatives of $L = \{0^n1^n \mid n \in \mathbb{N}\}$

$L/\varepsilon = \{0^n1^n \mid n \in \mathbb{N}\}$
$L/0^m = \{0^{n-m}1^n \mid n \geq m\}$
$L/0^m1^k = \{0^n1^{n-k} \mid n \geq k > 0\}$
$L/1 = \text{everything else}$
Myhill-Nerode Theorem
Restated Using Derivatives

A language $L$ is regular iff it has a finite number of distinct derivatives.
Derivatives of Regular Expressions

• An useful result is that the derivative concept can be extended to regular expressions, giving us a way to construct a DFA from a regular expression without going to an NFA first.

• For any regular expression $R$ over $\Sigma$ and any $\sigma \in \Sigma$ we can construct $R/\sigma$ such that $L(R/\sigma) = L(R)/\sigma$.


http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.98.4378
Derivative Rules for Regular Expressions

- $\emptyset/\sigma = \emptyset$
- $\varepsilon/\sigma = \emptyset$
- $\sigma/\sigma = \varepsilon$
- $\sigma'/\sigma = \emptyset$ if $\sigma' \neq \sigma$
- $(R\cup S)/\sigma = (R/\sigma) \cup (S/\sigma)$
- $(RS)/\sigma = (R/\sigma)S$, if $\varepsilon \notin L(R)$
- $(RS)/\sigma = (R/\sigma)S \cup (S/\sigma)$, if $\varepsilon \in L(R)$
- $(R^*)/\sigma = (R/\sigma)R^*$
Note on the condition $\varepsilon \in L(R)$

This condition (called the empty-word property) is determinable without constructing a DFA, using the following inductive rules on regular expressions:

- not $\varepsilon \in L(\emptyset)$
- $\varepsilon \in L(\varepsilon)$
- not $\varepsilon \in L(\sigma)$
- $\varepsilon \in L(R \cup S)$ iff $\varepsilon \in L(R)$ or $\varepsilon \in L(S)$
- $\varepsilon \in L(RS)$ iff $\varepsilon \in L(R)$ and $\varepsilon \in L(S)$
- $\varepsilon \in L(R^*)$
Examples of Derivatives of REs (applying the rules recursively)

- $0/0 = \varepsilon$
- $1/0 = \emptyset$
- $(0 \cup 1)/0 = \varepsilon$
- $01/0 = 1$
- $01/1 = \emptyset$
- $0^*/0 = 0^*$
- $1^*/0 = \emptyset$
- $(01)^*/0 = 1(01)^*$
- $(01)^*/1 = \emptyset$
- $(0 \cup 01)^*/0 = (\varepsilon \cup 1)(0 \cup 01)^*$
- $(01 \cup 10)^*/0 = 1(0 \cup 01)^*$
Try These

• \((0 \cup 1)^*/0 = ((0 \cup 1)/0)(0 \cup 1)^* = (0/0 \cup 1/0)(0 \cup 1)^* = (\varepsilon \cup \emptyset)(0 \cup 1)^* = \varepsilon(0 \cup 1)^* = (0 \cup 1)^*\)

• \((11^* \cup 01)/0 =\)

• \((11^* \cup 01)/1 =\)
Example: Using derivatives to construct a DFA

- Suppose $R = 0^* \cup 0^*11^*$. Then
- $R/0 = 0^* \cup 0^*11^* = R$
- $R/1 = 1^*$
- $R/10 = 1^*/0 = \emptyset$
- $R/11 = 1^*/1 = 1^* = R/1$
- At this point we have closure, so can diagram the DFA.
- $R$ is the initial state.
- The accepting states are those having $\varepsilon$ as an element.
DFA for $0^* \cup 0^*11^*$
constructed using derivatives of regular expressions
Caution

- Closure can sometimes be tricky to detect.

- Two regular expressions might be equivalent, but not identical.

- The same state is ideally used for both.
Example: Using derivatives to construct DFA

- Suppose $R = (0 \cup 1)^*11$.
- $R/0 = (0 \cup 1)^*11 = R$
- $R/1 = (0 \cup 1)^*11 \cup 1$ (note: 2\textsuperscript{nd} case of rule)
- $R/10 = (R/1)/0 = R$
- $R/11 = (R/1)/1 = (0 \cup 1)^*11 \cup 1 \cup \varepsilon$
- $R/110 = R$
- $R/111 = R/11$
Some Regular Expression Identities

- \( R \cup S = S \cup R \)
- \( R \ (S \ T) = (R \ S) \ T \)
- \( (R \cup S)T = RT \cup ST \)
- \( T(R \cup S) = TR \cup TS \)
- \( RR^* = R^*R \)
- If \( \varepsilon \in L(R) \) then \( RR^* = R^* \)
- \( \varepsilon \cup R^* = R^* \)
- \( \varepsilon \cup RR^* = R^* \)
- \( (\varepsilon \cup R \cup R^2 \cup \ldots \ R^{n-1}) \ (R^n)^* = R^* \), for every \( n \)
- \( \varepsilon R = R \)
- \( R\varepsilon = R \)
- \( (R^*)^* = R^* \)
- \( R^*R^* = R^* \)
- \( R \cup R^* = R^* \)
- \( (R \cup S)^* = (R^*S)^* \)
- \( (R \cup S)^* = (RS^*)^* \)
- \( R(SR)^* = (RS)^*R \)
- \( \emptyset^* = \varepsilon \)
Arden’s Rule

- The *smallest solution* for $X$ of the RE equation
  $$X = B \cup AX$$
  is
  $$X = A^*B.$$ 

If $\varepsilon \notin L(A)$ then the solution is unique.

- Intuition: $X = B \cup AX = B \cup A(B \cup AX)$
  $= B \cup AB \cup A^2X = B \cup AB \cup A^2(B \cup AX)$
  $= B \cup AB \cup A^2B \cup A^3X = ... \subseteq A^*B$

- Arden’s rule is implicit in the derivation of a regular expression from a DFA
Arden’s Rule Illustrated

\[ X = B \cup AX \]

NFA accepting \( X \)

This is an example of a “fixed point” theorem.
Alternate Conversion of DFA to RE

- Use Arden’s Rule + Gaussian Elimination Pattern

Regular Expressions in Software Tools


<table>
<thead>
<tr>
<th>Utility</th>
<th>Regular Expression Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>vi</td>
<td>Basic</td>
</tr>
<tr>
<td>sed</td>
<td>Basic</td>
</tr>
<tr>
<td>grep</td>
<td>Basic</td>
</tr>
<tr>
<td>csplit</td>
<td>Basic</td>
</tr>
<tr>
<td>dbx</td>
<td>Basic</td>
</tr>
<tr>
<td>dbxtool</td>
<td>Basic</td>
</tr>
<tr>
<td>more</td>
<td>Basic</td>
</tr>
<tr>
<td>ed</td>
<td>Basic</td>
</tr>
<tr>
<td>expr</td>
<td>Basic</td>
</tr>
<tr>
<td>lex</td>
<td>Basic</td>
</tr>
<tr>
<td>pg</td>
<td>Basic</td>
</tr>
<tr>
<td>nl</td>
<td>Basic</td>
</tr>
<tr>
<td>rdist</td>
<td>Basic</td>
</tr>
<tr>
<td>awk</td>
<td>Extended</td>
</tr>
<tr>
<td>nawk</td>
<td>Extended</td>
</tr>
<tr>
<td>egrep</td>
<td>Extended</td>
</tr>
<tr>
<td>EMACS</td>
<td>EMACS Regular Expressions</td>
</tr>
<tr>
<td>PERL</td>
<td>PERL Regular Expressions</td>
</tr>
</tbody>
</table>
Example, using program egrep

- `egrep = “Extended Global Regular Expressions Print”`

- (This is a Unix program. Windows users can use it in Cygwin.)

- This program identifies **lines** in a file that **contain** a match a regular expression on the command line, e.g. to match lines containing the literal letter “x”

  ```
  egrep x /usr/share/dict/propernames
  ```
A sample file: /usr/share/dict/propernames

$ head /usr/share/dict/propernames
Aaron
Adam
Adlai
Adrian
Agatha
Ahmed
Ahmet
Aimee
Amy
Ami

$ egrep x /usr/share/dict/propernames
Alex
Alexander
Alexis
Axel
Felix
Lex
Marnix
Max
Rex
Roxana
Roxane
Roxanne
Roxie

# head is first 10
Match lines containing “aa”

$ egrep aa /usr/share/dict/propernames
Isaac
Lievaart
Maarten
Raanan
Saad
Sjaak
Use | for Union

- Because | is significant to the operating system (for “piping”), regular expressions using it must be quoted.

```bash
$ egrep 'az|za' /usr/share/dict/propernames
Elizabeth
Hazel
Kazuhiro
Liza
Ozan
Suzan
Suzanne
```
Anchor Characters $^\$ and $^\\$ $

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>^A</td>
<td>&quot;A&quot; at the beginning of a line</td>
</tr>
<tr>
<td>A$</td>
<td>&quot;A&quot; at the end of a line</td>
</tr>
<tr>
<td>A^</td>
<td>&quot;A^&quot; anywhere on a line</td>
</tr>
<tr>
<td>$A</td>
<td>&quot;$A&quot; anywhere on a line</td>
</tr>
<tr>
<td>^^</td>
<td>&quot;^^&quot; at the beginning of a line</td>
</tr>
<tr>
<td>$$</td>
<td>&quot;$&quot; at the end of a line</td>
</tr>
</tbody>
</table>
Example: Lines ending in “ay”

$ egrep ay$ /usr/share/dict/propernames
Clay
Fay
Jay
Kay
Lindsay
Murray
Ray
Sanjay
Vijay
Wild Card .

. by itself matches any single character except end-of-line
Example: 4-letter lines beginning with “A”

$ egrep ^A...$ /usr/share/dict/propernames
Adam
Alan
Alex
Amir
Amos
Andy
Anna
Anne
Arne
Axel
Use [ ] to enumerate several characters

```
$ egrep 'of[aeiou]' /usr/share/dict/propernames
Christofer
Hirofumi
Sofia
Sofoklis

# Here we want ‘of’, but only if followed by vowel
```
Matching Ranges of Characters

[A-Z]
[a-z]
[0-9]
[A-Za-z0-9_]
Iterators

? matches 0 or 1 instances of what comes before
+ matches 1 or more instances
* matches 0 or more instances
For further info

http://en.wikipedia.org/wiki/Regular_expression

http://www.grymoire.com/Unix/Regular.html

http://www.regular-expressions.info/reference.html

$man egrep

$man regex
The Pumping Lemma


- The pumping lemma provides another way to show that a language is not finite-state. It cannot be used to show that a language is finite-state.

- A string of length \( k \) accepted by an \( n \)-state machine takes the machine through a sequence of \( k+1 \) states, some of which may repeat.

- If \( k > n \), then some of those states must repeat. [This is an instance of the “pigeonhole principle”.

- Repeated states implies that there is a shorter string that is also accepted by the machine, as well as certain longer strings that are too.
Pigeonhole Principle

• Also called “Dirichlet Drawer Principle”.

Illustrated with 10 pigeons in 9 holes:

http://en.wikipedia.org/wiki/Pigeonhole_principle
Pumping Lemma Setup

\{q_1, q_2, q_3, \ldots, q_{k+1}\} states in a sequence, with \( k \geq n \).

But only \( n \) distinct states exist.

Therefore some state is repeated at least once.
Pumping Lemma Setup

Suppose $q_j$ is the first repetition of any state. Let $q_i$ be the prior occurrence of that state.

The input sequence $x_1 \ldots x_{i-1}$ takes $q_1$ to $q_i$.

The input sequence $x_i \ldots x_{j-1}$ takes $q_i$ to $q_j$. And $q_i = q_j$.

The input sequence $x_j \ldots x_{k-1}$ takes $q_j$ to $q_{k+1}$, an accepting state.
Pumping Conditions

Call this **u**: The input sequence $x_1 \ldots x_{i-1}$ takes $q_1$ to $q_i$.

Call this **v**: The input sequence $x_i \ldots x_{j-1}$ takes $q_i$ to $q_j$. And $q_i = q_j$.

Call this **w**: The input sequence $x_j \ldots x_{k-1}$ takes $q_j$ to $q_{k+1}$, an accepting state.

Then

$|uv| \leq n$, since $q_j$ is the first repetition of any state.

$|v| > 0$, i.e. $v \neq \varepsilon$, since $q_j$ is the first repetition of $q_i$.

$\forall m \geq 0 \ uv^m w$ is accepted by the same machine.
Pumping Lemma for Regular Languages

• For any regular language $L$, there exists a natural number $p = \text{pump}(L)$, such that for any string $x \in L$, if $|x| \geq p$, then there exist $u, v, w$ such that:
  • $x = uvw$
  • $|uv| \leq p$
  • $|v| > 0$
  • $\forall m \geq 0 \ uv^m w \in L$.
  • [$v$ is the “pumped” string.]
As a single formula

\[ \forall L \text{ regular}(L) \rightarrow \\
\exists p \in \mathbb{N} \ \forall x \in L \\
| x | \geq p \rightarrow \\
\exists u, v, w \\
[x = uvw \\
\land |uv| \leq p \\
\land |v| > 0 \\
\land (\forall m \geq 0 \ uv^m w \in L)] \]

We can view pump(L) as the Skolem function for p given L.
Proof that $L = \{0^n1^n \mid n \in N\}$ is not regular using the Pumping Lemma

- Suppose the language is regular.
- Let $p = \text{pump}(L)$.
- Let $x = 0^p1^p$.
  According to the pumping lemma, $x$ can be written $uvw$, where $v \neq \epsilon$, and $(\forall m \geq 0)$ $uv^m w \in L$.
- Whatever $v$ is, it is one of the following forms:
  - $0^r$, $r > 0$  
  - $1^s$, $s > 0$
  - $0^r1^s$, $r > 0$ and $s > 0$
- In the first case, we would have $u0^{2r}w \in L$, which has too many 0’s.
- In the second case, we would have $u1^{2r}w \in L$, which has too many 1’s.
- In the third case, we have $u0^r1^s0^r1^sw \in L$, which is not of the correct form, as 1’s cannot come before 0’s.

- Each possibility yields a contradiction, so $L$ is not regular after all.
More Examples of Non-Regular Languages
Which can be proved with the pumping lemma? Which with the Myhill-Nerode theorem / derivatives?

- \{x \cdot x \mid x \in \{0, 1\}^*\}
- \{xx^R \mid x \in \{0, 1\}^*\}
- \{0^n1_m \mid m, n \in \mathbb{N}, m < n\}
- \{x \in \{0, 1\}^* \mid \#_0(x) = \#_1(x)\}
- \{1^n \mid n \text{ a perfect square}\}
- The set of palindromes over an alphabet with more than one symbol.
- The set of well-formed strings of parens =
  \{ (, ()(), (()), (())(), (()()), ((())(), ()()()), ()(()()), (())(), ()()(), \ldots \}
Pumping Lemma Corollary

• The language accepted by a DFA of n states is infinite iff it accepts a string of length \( \geq n \).

• Proof:
  • (⇒) If the DFA accepts an infinite language, there must be strings of arbitrary length it accepts.
  • (⇐) If a DFA accepts a string of length \( \geq n \), we can generate an infinite number of strings that it also accepts by pumping one such string.
Decision Problems

- A **decision problem** is the problem of finding an algorithm that will answer questions about formal systems, such as automata, regular expressions, grammars, etc.

- If an algorithm exists, the problem is said to be **solvable**, otherwise **unsolvable**. (The words **decidable** and **undecidable** are also used.)

- Later we will see many examples of unsolvable decision problems. But for systems related to **regular languages**, many are solvable.
Examples of Decision Problems

- Given two DFA’s (expressed as a string representation of their graphs) are their languages equal?
- Given a DFA, is its language empty?
- Given a DFA, is its language finite?
- Given two DFA’s, is the language of one included in the language of the other?
Solution to Decision Problems

- To describe the solution of a decision problem, we must give an algorithm.
- Consider the first decision problem on the previous page:
  - Given two DFA’s (expressed as a string representation of their graphs) are their languages equal?
- An algorithm for this problem might be as follows:
  - Consider the union machine of the DFA’s: the machine that includes all states and transitions of each. Determine whether what would have been the initial states of each are equivalent using the equivalence checking algorithm. If they are equivalent, then the machines accept the same language. Otherwise they don’t.
Solution to the Finiteness Decision Problem

• An algorithm that will determine whether or not any DFA accepts an infinite set:
  • By the same kind of reasoning as in the pumping lemma, we can show that the language an n-state DFA is infinite iff it accepts a string of length between n and 2n.
  • Since the number of strings in this range is finite, we can check them all to see if there is an acceptance.

• Can you devise another algorithm?