LING/C SC/PSYC 438/538

Lecture 14

Sandiway Fong
Administrivia

- Homework 6 graded
Applications of FSA

• Let’s take a look at one application of regular language technology:
  – Efficient String Matching
String Matching

• Example:
  – Pattern (P): aaba
  – Corpus (C): aaabbaabab

• Naïve algorithm:
  – compare Pattern against Corpus from left to right a character at a time

  – P: aaba
  – C: aaabbaabab
  – P: _aaba
  – C: aaabbaabab
  – P: ___aaba
  – C: aaabbaabab
  – P: ____aaba
  – C: aaabbaabab
  – P: ______aaba
  – C: aaabbaabab
  – Matched!
Knuth-Morris-Pratt (KMP)

• can do better (i.e. *use fewer comparisons*) if we use a FSA to represent the Pattern

• plus extra links for restarts in the event of a failure

• Example
  • Pattern: aaba

*all backpointers are restarts for failed character*

• Suppose the alphabet was limited to just \{a,b\}, then restarts can be for the character following the failed character
Knuth-Morris-Pratt (KMP)

- Example:
  - Pattern: aaba
  - Corpus \{a,b\}: aaabbaabab

- KMP:
  - P: aaba
  - C: aaabbaabab  (mismatch at state 2)
  - P: _aaba
  - C: aaabbaabab  (mismatch at state 3)
  - P: _____aaba
  - C: aaabbaabab  (matched)

- Other possible algorithms:
  - e.g. Boyer-Moore (start match from the back of the pattern)
Beyond Regular Languages

• Beyond regular languages
  – $a^n b^n = \{ab, aabb, aaabbb, aaaabbbb, \ldots \} \ n \geq 1$
  – is not a regular language

• That means no FSA, RE (or Regular Grammar) can be built for this set

• Informally, let’s think about a FSA implementation ...

1. We only have a finite number of states to play with ...
2. We’re only allowed simple free iteration (looping)
Beyond Regular Languages

- \( L = a^+ b^+ \)

```bash
Having a frequency table is not permitted. Not allowed:
%freq = ();
...
$freq->{$state}++;`
A Formal Tool: The Pumping Lemma

[See also discussion in JM 16.2.1, pages 533–534]

• Let $L$ be a regular language,
• then there exists a number $p > 0$
  – where $p$ is a pumping length (sometimes called a magic number)
  such that every string $w$ in $L$ with $|w| \geq p$ can be written in the following form
    $$w = xyz$$
• with strings $x$, $y$ and $z$ such that $|xy| \leq p$, $|y| > 0$ and $xy^iz$ is in $L$
• for every integer $i \geq 0$.

BTW: there is also a pumping lemma for Context-Free Languages
A Formal Tool: The Pumping Lemma

Restated:

• For every *(sufficiently long)* string $w$ in a regular language
• there is always a way to split the string into three adjacent sections, call them $x$, $y$ and $z$, ($y$ nonempty), i.e. $w$ is $x$ followed by $y$ followed by $z$
• And $y$ can be repeated as many times as we like (or omitted)
• And the modified string is still a member of the language

Essential Point!
To prove a language is non-regular: show that no matter how we split the string, there will be modified strings that can't be in the language
A Formal Tool: The Pumping Lemma

• Example:
  – show that $a^n b^n$ is not regular

• Proof (by contradiction):
  – pick a sufficiently long string in the language
  – e.g. a..aab..bb ($\#a’s = \#b’s$)
  – Partition it according to $w = xyz$
  – then show $xy^i z$ is not in $L$
  – i.e. string does not pump
A Formal Tool: The Pumping Lemma

aaaa..aabbbb..bb

Case 1: \( w = xyz, y \) straddles the ab boundary
what happens when we pump \( y \)?

Case 2: \( w = xyz, y \) is wholly within the a’s
what happens when we pump \( y \)?

Case 3: \( w = xyz, y \) is wholly within the b’s
what happens when we pump \( y \)?
\[ a^n b^n \]

- In Perl?
  - we could use ( ?{...Perl code...} )
A Formal Tool: The Pumping Lemma

• Prime number testing
  prime number testing using Perl’s extended “regular expressions”

• Using unary notation, e.g. 5 = “11111”

• /^(11+?)\1+$/ will match anything that’s greater than 1 that’s not prime

$L = \{1^n \mid n \text{ is prime}\}$ is not a regular language
A Formal Tool: The Pumping Lemma

\[ 1^n = 111..1111..11111 \text{ such that } n \text{ is a prime number} \]

\[ x \quad y \quad z \]

For any split of the string
Pump \( y \) such that \( i = \text{length}(x+z) \), giving \( y^i \)

What is the length of string \( w = xy^i z \) now?

In \( x y^{xz} z \), how many copies of \( xz \) do we have?
Answer is \( y+1 \)
i.e. pumped number can be factorized into \( (1+|y|)|xz| \)

The resulting length is non-prime since it can be factorized
A Formal Tool: The Pumping Lemma

$1^n = 111\ldots1111\ldots11111$ such that $n$ is a prime number

Illustration of the calculation:
1111 1111 111 (eleven)
1111 1111 1111 1111 1111 1111 1111 1111 1111 111
4 + 4*7 + 3
= 5*7
which isn't prime
A Formal Tool: The Pumping Lemma

- Another angle to reduce the mystery, let's think in terms of FSA. We know:
  1. we can't control the loops
  2. we are restricted to a finite number of states
  3. assume (without loss of generality) there are no e-transitions

- Suppose there are a total of p states in the machine
- Suppose we have a string in the language longer than p
- What can we conclude?  
  **Answer:** we must have visited some state(s) more than once!

  **Also:** there must be a loop (or loops) in the machine!

  **Also:** we can repeat or skip that loop and stay inside the language!
Homework 7
SWI Prolog

- Install on your laptop (Mac or PC):

<table>
<thead>
<tr>
<th>Binaries</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Binary Icon]</td>
</tr>
<tr>
<td>11,849,467 bytes</td>
</tr>
<tr>
<td>![Binary Icon]</td>
</tr>
<tr>
<td>18,508,889 bytes</td>
</tr>
<tr>
<td>![Binary Icon]</td>
</tr>
<tr>
<td>18,561,463 bytes</td>
</tr>
</tbody>
</table>
SWI Prolog

• Mac problems:

• option-click on application
SWI Prolog

1. every command ends in a period
2. case sensitive: variables begin with an uppercase letter

Control-D (EOF) to terminate.

halt.
SWI Prolog

• Install on your laptop (Linux, Debian-based):

  • sudo apt-add-repository ppa:swi-prolog/stable
  • sudo apt-get update
  • sudo apt-get install swi-prolog
SWI Prolog
SWI Prolog

http://xquartz.macosforge.org/landing/