Administrivia

• Reminder:
  – no lecture next Monday
  – (I'm at a conference)

• Homework 6
  – due date next Monday night by midnight
  – usual submission rules
ε-transitions

• jump from state to another state with the empty character
  – ε-transition *(textbook)* or λ-transition
  – no increase in expressive power

• examples

```
ε  a  b
```
what’s the equivalent without the ε-transition?
Non-Deterministic Finite State Automata (NDFSA)

- non-deterministic FSA (NDFSA)
- no restriction on ambiguity (surprisingly, no increase in power)
- Example:
Non-Deterministic Finite State Automata (NDFSA)

Strategies for keeping track of multiple states

- Backtracking (**backup**)
- Lookahead
- Parallelism

*algorithm gets complicated fast*

---

**Figure 2.19** An algorithm for NPSA recognition. The word *node* means a state of the FSA, and state or *search-state* means "the state of the search process", i.e., a combination of node and tape position.
NDFSA $\rightarrow$ (D)FSA

[discussed at the end of section 2.2 in the textbook]

- construct a new machine
  - each state of the new machine represents the \textit{set of possible states} of the original machine when stepping through the input

- \textbf{Note:}
  - new machine is equivalent to old one \textit{(but has more states)}
  - new machine is deterministic

- \textbf{example}

\[\text{[Powerpoint Animation]}\]
Ungraded Homework Exercise

• do not submit, do the following exercise to check your understanding

  – apply the set-of-states construction technique to the two machines on the \(\varepsilon\)-transition slide (repeated below)

  – self-check your answer:

    • verify in each case that the machine produced is deterministic and accurately simulates its \(\varepsilon\)-transition counterpart
Perl Regular Expressions

• Perl regex can include backreferences to groupings (i.e. \1, etc.)
  – backreferences give Perl regexes expressive power beyond regular languages:

```
/\1er they \1er we \2/  
```

will match *The faster they ran, the faster we ran* but not *The faster they ran, the faster we ate.* These numbered memories are called registers (e.g. \1, \2).

• the set of prime numbers is **not** a regular language
  
  \[L_{\text{prime}} = \{2, 3, 5, 7, 11, 13, 17, 19, 23, \ldots \}\]

  can be proved using the Pumping Lemma for regular languages (later)
Backreferences and FSA

• Deep question:
  – why are backreferences impossible in FSA?

Example:
  Suppose you wanted a machine that accepted /(a+b+)\1/
  One idea: link two copies of the machine together

Doesn’t work! Why?

• Perl implementation:
  – how to modify it get the backreference effect?

```perl
ustralian delta =
  ("s", {"a","x"}, "x", {"a","x", "b", "y"}, "y", {"b","y", "a","x2"},
   "x2", {"a","x2", "b","y2"}, "y2", {"b","y2"});
$state = "s";
foreach $c (split(/,ARGV[0]))) {
  $state = $delta{$state}{$c};
}
print (($state eq "y2") ? "Accept\n" : "Reject\n");
```
Regular Languages and FSA

• Formal (constructive) set-theoretic definition of a regular language

1. $\emptyset$ is a regular language
2. $\forall a \in \Sigma \cup \epsilon, \{a\}$ is a regular language
3. If $L_1$ and $L_2$ are regular languages, then so are:
   a. $L_1 \cdot L_2 = \{xy | x \in L_1, y \in L_2\}$, the concatenation of $L_1$ and $L_2$
   b. $L_1 \cup L_2$, the union or disjunction of $L_1$ and $L_2$
   c. $L_1^+$, the Kleene closure of $L_1$

• Correspondence between REs and Regular Languages
  • concatenation (juxtaposition)
  • union (also [ ])
  • Kleene closure ($^*$) $= (x^+ = xx^*)$

• Note:
  • backreferences are memory devices and thus are too powerful
  • e.g. $L = \{ww\}$ and prime number testing (see earlier lectures)
Regular Languages and FSA

- **Other closure properties:**

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersection</td>
<td>If $L_1$ and $L_2$ are regular languages, then so is $L_1 \cap L_2$, the language consisting of the set of strings that are in both $L_1$ and $L_2$.</td>
</tr>
<tr>
<td>Difference</td>
<td>If $L_1$ and $L_2$ are regular languages, then so is $L_1 - L_2$, the language consisting of the set of strings that are in $L_1$ but not $L_2$.</td>
</tr>
<tr>
<td>Complementation</td>
<td>If $L_1$ is a regular language, then so is $\Sigma^* - L_1$, the set of all possible strings that aren’t in $L_1$.</td>
</tr>
<tr>
<td>Reversal</td>
<td>If $L_1$ is a regular language, then so is $L_1^R$, the language consisting of the set of reversals of all the strings in $L_1$.</td>
</tr>
</tbody>
</table>

- Not true higher up: *e.g. context-free grammars as we’ll see later*
Equivalence: FSA and Regexs

Textbook gives one direction only

• Case by case:
  a) Empty string
  b) Empty set
  c) Any character from the alphabet

1. ∅ is a regular language
2. ∀a ∈ Σ ∪ ε, {a} is a regular language

Automata for the base case (no operators) for the induction showing that any regular expression can be turned into an equivalent automaton.
Equivalence: FSA and Regexs

• Concatenation:

3. If $L_1$ and $L_2$ are regular languages, then so are:
   
   (a) $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$, the concatenation of $L_1$ and $L_2$ 
   
   (b) $L_1 \cup L_2$, the union or disjunction of $L_1$ and $L_2$ 
   
   (c) $L_1^*$, the Kleene closure of $L_1$

– Link final state of FSA$_1$ to initial state of FSA$_2$ using an empty transition

![Diagram](image.png)

**Note:** empty transition can be eliminated using the set of states construction  
*(see earlier slides in this lecture)*
Equivalence: FSA and Regexs

- Kleene closure:

3. If $L_1$ and $L_2$ are regular languages, then so are:
   
   (a) $L_1 \cdot L_2 = \{ xy \mid x \in L_1, y \in L_2 \}$, the concatenation of $L_1$ and $L_2$
   
   (b) $L_1 \cup L_2$, the union or disjunction of $L_1$ and $L_2$
   
   (c) $L_1^*$, the Kleene closure of $L_1$

- repetition operator: zero or more times
- use empty transitions for loopback and bypass

*Figure 2.24* The closure (Kleene $*$) of an FSA.
Equivalence: FSA and Regexs

- **Union**: aka disjunction

3. If $L_1$ and $L_2$ are regular languages, then so are:
   (a) $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$, the concatenation of $L_1$ and $L_2$
   (b) $L_1 \cup L_2$, the union or disjunction of $L_1$ and $L_2$
   (c) $L_1^+$, the Kleene closure of $L_1$

- Non-deterministically run both FSAs at the same time, accept if either one accepts
Regular Languages and FSA

• Other closure properties:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersection</td>
<td>If $L_1$ and $L_2$ are regular languages, then so is $L_1 \cap L_2$, the language consisting of the set of strings that are in both $L_1$ and $L_2$.</td>
</tr>
<tr>
<td>difference</td>
<td>If $L_1$ and $L_2$ are regular languages, then so is $L_1 - L_2$, the language consisting of the set of strings that are in $L_1$ but not $L_2$.</td>
</tr>
<tr>
<td>complementation</td>
<td>If $L_1$ is a regular language, then so is $\Sigma^* - L_1$, the set of all possible strings that aren’t in $L_1$.</td>
</tr>
<tr>
<td>reversal</td>
<td>If $L_1$ is a regular language, then so is $L_1^R$, the language consisting of the set of reversals of all the strings in $L_1$.</td>
</tr>
</tbody>
</table>

Let’s consider building the FSA machinery for each of these guys in turn...
Regular Languages and FSA

• Other closure properties:

| intersection | if $L_1$ and $L_2$ are regular languages, then so is $L_1 \cap L_2$, the language consisting of the set of strings that are in both $L_1$ and $L_2$. |
Regular Languages and FSA

• Other closure properties:

<table>
<thead>
<tr>
<th>difference</th>
<th>if $L_1$ and $L_2$ are regular languages, then so is $L_1 - L_2$, the language consisting of the set of strings that are in $L_1$ but not $L_2$.</th>
</tr>
</thead>
</table>

Regular Languages and FSA

• Other closure properties:

| complementation | If $L_1$ is a regular language, then so is $\Sigma^* - L_1$, the set of all possible strings that aren’t in $L_1$. |
Regular Languages and FSA

• Other closure properties:

| reversal | If $L_1$ is a regular language, then so is $L_1^R$, the language consisting of the set of reversals of all the strings in $L_1$. |
Regular Expressions from FSA

Textbook Exercise: find a RE for
4. the set of all strings from the alphabet $a, b$ such that each $a$ is immediately preceded by and immediately followed by a $b$;

Examples (* denotes string not in the language):
- *ab  *ba
- bab
- $\lambda$ (empty string)
- bb
- *baba
- babab
Regular Expressions from FSA

• Draw a FSA and convert it to a RE:

\[ b^* \ b \ (ab^+)^+ \]

\[ = b+(ab^+)^* \mid \varepsilon \]
Regular Expressions from FSA

• Perl implementation:

```perl
$s = "ab ba bab bb baba babab";
while ($s =~ /(b+(ab+)*\b)/g) {
    print "<$1> match!\n";
}
```

• Output:

```bash
perl test.perl
<bab> match!
<bb> match!
<babab> match!
```

Note: doesn’t include the empty string case

Note: /../g global flag for multiple matches
Homework 6

• Question 1:
  – Assume \( \Sigma = \{a,b\} \).
  – Draw a non-deterministic FSA that accepts all strings that contain the substring \( abba \).
  – Examples:
    • *abbba, babba, *abb, abba, *aaba, aabbbbaba

• Question 2:
  – convert the machine from Question 1 into a deterministic FSA.
Homework 6

• Question 3:
  – Implement your machine in Perl and give sample runs.