The Pumping Lemma

For every regular language \( L \), there is a number \( \ell \geq 1 \) satisfying the *pumping lemma property*:

all \( w \in L \) with \( \text{length}(w) \geq \ell \) can be expressed as a concatenation of three strings, \( w = u_1 vu_2 \), where \( u_1, v \) and \( u_2 \) satisfy:

- \( \text{length}(v) \geq 1 \)  
  (i.e. \( v \neq \varepsilon \))

- \( \text{length}(u_1 v) \leq \ell \)

- for all \( n \geq 0 \), \( u_1 v^n u_2 \in L \)  
  (i.e. \( u_1 u_2 \in L \), \( u_1 vu_2 \in L \) [but we knew that anyway], \( u_1 vv u_2 \in L \), \( u_1 vv v u_2 \in L \), etc).
Some questions

(a) Is there an algorithm which, given a string $u$ and a regular expression $r$ (over the same alphabet), computes whether or not $u$ matches $r$?

(b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?

(c) Is there an algorithm which, given two regular expressions $r$ and $s$ (over the same alphabet), computes whether or not they are equivalent? (Cf. Slide 8.)

(d) Is every language of the form $L(r)$?
Languages

A (formal) language $L$ over an alphabet $\Sigma$ is just a set of strings in $\Sigma^*$. Thus any subset $L \subseteq \Sigma^*$ determines a language over $\Sigma$.

The language determined by a regular expression $r$ over $\Sigma$ is

$$L(r) \overset{\text{def}}{=} \{ u \in \Sigma^* \mid u \text{ matches } r \}.$$ 

Two regular expressions $r$ and $s$ (over the same alphabet) are equivalent iff $L(r)$ and $L(s)$ are equal sets (i.e. have exactly the same members).

Write $r \equiv s$ to mean $L(r) = L(s)$

Write $r \leq s$ to mean $L(r) \subseteq L(s)$
Kleene algebra

\[(r\parallel s)\parallel t \equiv r\parallel (s\parallel t)\]
\[r\parallel s \equiv s\parallel r\]
\[r\parallel r \equiv r\]
\[r\parallel \emptyset \equiv r\]

\[(rs)\parallel t \equiv r(st)\]
\[r\varepsilon \equiv r \equiv \varepsilon r\]
\[r\emptyset \equiv \emptyset \equiv \emptyset r\]
Kleene algebra

\[(rt|st) = r|s|t\]
\[rs = s|r\]
\[r|r = r\]
\[r|\emptyset = r\]

\[(rs)t = r(st)\]
\[r\varepsilon = r = \varepsilon r\]
\[r\emptyset = \emptyset = \emptyset r\]

\[r(s|t) = rs|rt\]
\[(r|s)t = rt|st\]
Kleene algebra

\[(r|s)t \equiv r|(s|t)\]
\[r|s \equiv s|r\]
\[r|r \equiv r\]
\[r|\emptyset \equiv r\]

\[(rs)t \equiv r(st)\]
\[r \varepsilon \equiv r \equiv \varepsilon r\]
\[r \emptyset \equiv \emptyset \equiv \emptyset r\]

\[\varepsilon | rr^* \equiv r^* \equiv r^* r | \varepsilon\]
Kleene algebra

\[(\text{r|s})\text{t} \equiv \text{r|}(\text{s|t})\]
\[
\text{r|s} \equiv \text{s|r}
\]
\[
\text{r|r} \equiv \text{r}
\]
\[
\text{r|}\emptyset \equiv \text{r}
\]

\[(\text{rs})\text{t} \equiv \text{r(st)}\]
\[
\text{r}\varepsilon \equiv \text{r} \equiv \varepsilon \text{r}
\]
\[
\text{r}\emptyset \equiv \emptyset \equiv \emptyset \text{r}
\]

\[
\varepsilon \text{r} \equiv \varepsilon \equiv \varepsilon \text{r|}\varepsilon
\]

\[
\text{rs} \equiv \text{rs|rt}
\]

\[
\text{r|s} \equiv \text{s|if } \text{r|s} \equiv \text{s}
\]
Kleene algebra

\[(rs)t \equiv r(st)\]
\[r|s = s|r\]
\[r|r = r\]
\[r|\emptyset = r\]

\[(rs)t \equiv r(st)\]
\[r\emptyset = r \equiv \emptyset r\]

\[r(s|t) = rs \mid rt\]
\[(rs)t \equiv rt \mid st\]

\[r \leq s \text{ if } r|s = s\]

\[\varepsilon|rr^* \equiv r^* \equiv r^*r \mid \varepsilon\]

if \[r|st \leq t\]
then \[s^*r \leq t\]

if \[r|ts \leq t\]
then \[rs^* \leq t\]
Qu:

\[ b^* a (b^* a)^* \equiv (a | b)^* a \]
Qu: \( b^*a (b^*a)^* \equiv (a|b)^*a \) ?

Ans: YES!
Some questions

(a) Is there an algorithm which, given a string $u$ and a regular expression $r$ (over the same alphabet), computes whether or not $u$ matches $r$?

(b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?

(c) Is there an algorithm which, given two regular expressions $r$ and $s$ (over the same alphabet), computes whether or not they are equivalent? (Cf. Slide 8.)

(d) Is every language of the form $L(r)$?
Decision procedure for $r_1 \equiv r_2$

Suffices to decide $r_1 \leq r_2$

(since $r_1 \equiv r_2$ if and only if $r_1 \leq r_2$ AND $r_2 \leq r_1$.)
Decision procedure for $r_1 \leq r_2$

Note: $r_1 \leq r_2$ if $L(r_1) \subseteq L(r_2)$
Decision procedure for \( r_1 \leq r_2 \)

Note: \( r_1 \leq r_2 \) iff \( L(r_1) \subseteq L(r_2) \)

iff \( L(r_1) \cap (\Sigma^* - L(r_2)) = \emptyset \)
Decision procedure for $r_1 \leq r_2$

Note: $r_1 \leq r_2$ if $L(r_1) \subseteq L(r_2)$

iff $L(r_1) \cap (\Sigma^* \setminus L(r_2)) = \emptyset$

iff $L(r_1 \& (\neg r_2)) = \emptyset$
Decision procedure for $n_1 \leq n_2$

Note: $n_1 \leq n_2$ iff $L(n_1) \subseteq L(n_2)$

iff $L(n_1) \cap (\Sigma^* - L(n_2)) = \emptyset$

iff $L(n_1 \land (\neg n_2)) = \emptyset$

So suffices to decide, given any $n$, whether $L(n) = \emptyset$
Lemma If a DFA $M$ accepts any string at all, it accepts one whose length is less than the number of states in $M$.

Proof. Suppose $M$ has $\ell$ states (so $\ell \geq 1$). If $L(M)$ is not empty, then we can find an element of it of shortest length, $a_1 a_2 \ldots a_n$ say (where $n \geq 0$). Thus there is a transition sequence

$$s_M = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots \xrightarrow{a_n} q_n \in \text{Accept}_M.$$ 

If $n \geq \ell$, then not all the $n + 1$ states in this sequence can be distinct and we can shorten it as on Slide 30. But then we would obtain a strictly shorter string in $L(M)$ contradicting the choice of $a_1 a_2 \ldots a_n$. So we must have $n < \ell$. $\Box$
Decision procedure for $r_1 \equiv r_2$

Given $r_1$ and $r_2$:

1. Construct DFAs $M_1$ and $M_2$ such that
   \[ L(M_1) = L(r_1 \land \neg r_2) \]
   \[ L(M_2) = L(r_2 \land \neg r_1) \]

2. Check whether $L(M_1) = \emptyset$ and $L(M_2) = \emptyset$
   (in which case $r_1 \equiv r_2$)
   or not
   (in which case $r_1 \not\equiv r_2$)
Chapter 6: Grammars

(p 47)
Some production rules for ‘English’ sentences

SENTENCE → SUBJECT VERB OBJECT
SUBJECT → ARTICLE NOUNPHRASE
OBJECT → ARTICLE NOUNPHRASE
ARTICLE → a
ARTICLE → the
NOUNPHRASE → NOUN
NOUNPHRASE → ADJECTIVE NOUN
ADJECTIVE → big
ADJECTIVE → small
NOUN → cat
NOUN → dog
VERB → eats
A derivation

\[
\text{SENTENCE} \rightarrow \text{SUBJECT} \text{ VERB OBJECT} \\
\rightarrow \text{ARTICLE NOUNPHRASE VERB OBJECT} \\
\rightarrow \text{the NOUNPHRASE VERB OBJECT} \\
\rightarrow \text{the NOUNPHRASE eats OBJECT} \\
\rightarrow \text{the ADJECTIVE NOUN eats OBJECT} \\
\rightarrow \text{the big NOUN eats OBJECT} \\
\rightarrow \text{the big cat eats OBJECT} \\
\rightarrow \text{the big cat eats ARTICLE NOUNPHRASE} \\
\rightarrow \text{the big cat eats a NOUNPHRASE} \\
\rightarrow \text{the big cat eats a ADJECTIVE NOUN} \\
\rightarrow \text{the big cat eats a small NOUN} \\
\rightarrow \text{the big cat eats a small dog}
\]
Example of Backus-Naur Form (BNF)

Terminals:
\[
x' + - * ( )
\]

Non-terminals:
\[
id \ op \ exp
\]

Start symbol:
\[
exp
\]

Productions:
\[
id ::= x \mid id'\\
op ::= + \mid - \mid *
exp ::= id \mid exp \ op \ exp \mid (exp)
\]
Regular expressions over an alphabet $\Sigma$

- each symbol $a \in \Sigma$ is a regular expression
- $\varepsilon$ is a regular expression
- $\emptyset$ is a regular expression
- if $r$ and $s$ are regular expressions, then so is $(r|s)$
- if $r$ and $s$ are regular expressions, then so is $rs$
- if $r$ is a regular expression, then so is $(r)^*$

Every regular expression is built up inductively, by finitely many applications of the above rules.

(N.B. we assume $\varepsilon$, $\emptyset$, $( )$, $|$, and $*$ are not symbols in $\Sigma$.)
A context-free grammar for regular expressions over alphabet $\Sigma$

set of terminals $\Sigma \cup \{ \epsilon, \emptyset, ( , ), 1 , * \}$

set of non-terminals $\{ r \}$

start symbol $r$

productions

$$ r ::= a | \epsilon | \emptyset | ( r | r ) | r r | ( r )^* $$

(where $a \in \Sigma$)