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 $length(w) \geq \ell$ can be expressed as a concatenation of three strings, $w = u_1 v u_2$, where u_1, v and u_2 satisfy:

- $length(v) \geq 1$
  (i.e. $v \neq \varepsilon$)
- $length(u_1 v) \leq \ell$
- for all $n \geq 0$, $u_1 v^n u_2 \in L$
  (i.e. $u_1 u_2 \in L$, $u_1 v u_2 \in L$ [but we knew that anyway], $u_1 v v u_2 \in L$, $u_1 v v v u_2 \in L$, 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 Σ is just a set of strings in Σ^*.

Thus any subset $L \subseteq \Sigma^*$ determines a language over Σ.

The ***language determined by a regular expression*** r over Σ is

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

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 $\begin{cases} r \equiv s & \text{to mean} \quad L(r) = L(s) \\ r \leq s & \text{"} \quad\quad \text{"} \quad\quad L(r) \subseteq L(s) \end{cases}$

Kleene algebra

$$(r|s)|t \equiv r|(s|t)$$
$$r|s \equiv s|r$$
$$r|r \equiv r$$
$$r|\emptyset \equiv r$$

$$(rs)t \equiv r(st)$$
$$r\varepsilon \equiv r \equiv \varepsilon r$$
$$r\emptyset \equiv \emptyset \equiv \emptyset r$$

Kleene algebra

$$(r|s)|t \equiv r|(s|t)$$
$$r|s \equiv s|r$$
$$r|r \equiv r$$
$$r|\emptyset \equiv r$$

$$(rs)t \equiv r(st)$$
$$r\varepsilon \equiv r \equiv \varepsilon r$$
$$r\emptyset \equiv \emptyset \equiv \emptyset r$$

$$r(s|t) \equiv rs \mid rt$$
$$(r|s)t \equiv rt \mid st$$

Kleene algebra

$$(r|s)|t \equiv r|(s|t)$$
$$r|s \equiv s|r$$
$$r|r \equiv r$$
$$r|\emptyset \equiv r$$

$$(rs)t \equiv r(st)$$
$$r\varepsilon \equiv r \equiv \varepsilon r$$
$$r\emptyset \equiv \emptyset \equiv \emptyset r$$

$$r(s|t) \equiv rs \mid rt$$
$$(r|s)t \equiv rt \mid st$$

$$\varepsilon|rr^* \equiv r^* \equiv r^*r|\varepsilon$$

Kleene algebra

$$(r|s)|t \equiv r|(s|t)$$
$$r|s \equiv s|r$$
$$r|r \equiv r$$
$$r|\emptyset \equiv r$$

$$(rs)t \equiv r(st)$$
$$r\varepsilon \equiv r \equiv \varepsilon r$$
$$r\emptyset \equiv \emptyset \equiv \emptyset r$$

$$r(s|t) \equiv rs|rt$$
$$(r|s)t \equiv rt|st$$

$$\varepsilon|rr^* \equiv r^* \equiv r^*r|\varepsilon$$

$$r \leq s \text{ iff } r|s \equiv s$$

Kleene algebra

$$(r|s)|t \equiv r|(s|t)$$
$$r|s \equiv s|r$$
$$r|r \equiv r$$
$$r|\emptyset \equiv r$$

$$(rs)t \equiv r(st)$$
$$r\varepsilon \equiv r \equiv \varepsilon r$$
$$r\emptyset \equiv \emptyset \equiv \emptyset r$$

$$r(s|t) \equiv rs|rt$$
$$(r|s)t \equiv rt|st$$

$$\varepsilon|rr^* \equiv r^* \equiv r^*r|\varepsilon$$

if $r|st \leq t$
then $s^*r \leq t$

if $r|ts \leq t$
then $rs^* \leq t$

$r \leq s$ iff $r|s \equiv s$

Qu:

$$b^*a(b^*a)^* \overset{?}{\equiv} (a|b)^*a$$

Qu:

$$b^* a (b^* a)^* \stackrel{?}{\equiv} (a|b)^* a$$

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 $r_1 \equiv r_2$

$$\boxed{\text{Suffices to decide } r_1 \leqslant r_2}$$

(since $r_1 \equiv r_2$ if & only if $r_1 \leqslant r_2$ AND $r_2 \leqslant r_1$)

Decision procedure for $r_1 \leqslant r_2$

Note: $r_1 \leqslant r_2$ iff $L(r_1) \subseteq L(r_2)$

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

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

iff $L(r_1 \;\&\; (\sim r_2)) = \emptyset$

Decision procedure for $r_1 \leq r_2$

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

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

iff $L(r_1 \,\&\, (\sim r_2)) = \emptyset$

So suffices to decide, given any r, whether $L(r) = \emptyset$

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 ℓ 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

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

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 $n < \ell$. $\square$

Decision procedure for $r_1 \equiv r_2$

Given r_1 and r_2 :

1. Construct DFAs M_1 and M_2 such that

$$L(M_1) = L(r_1 \,\&\, {\sim} r_2)$$
$$L(M_2) = L(r_2 \,\&\, {\sim} r_1)$$

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

$$\texttt{SENTENCE} \longrightarrow \texttt{SUBJECT VERB OBJECT}$$

$$\texttt{SUBJECT} \longrightarrow \texttt{ARTICLE NOUNPHRASE}$$

$$\texttt{OBJECT} \longrightarrow \texttt{ARTICLE NOUNPHRASE}$$

$$\texttt{ARTICLE} \longrightarrow \texttt{a}$$
$$\texttt{ARTICLE} \longrightarrow \texttt{the}$$

$$\texttt{NOUNPHRASE} \longrightarrow \texttt{NOUN}$$
$$\texttt{NOUNPHRASE} \longrightarrow \texttt{ADJECTIVE NOUN}$$

$$\texttt{ADJECTIVE} \longrightarrow \texttt{big}$$
$$\texttt{ADJECTIVE} \longrightarrow \texttt{small}$$

$$\texttt{NOUN} \longrightarrow \texttt{cat}$$
$$\texttt{NOUN} \longrightarrow \texttt{dog}$$

$$\texttt{VERB} \longrightarrow \texttt{eats}$$

A derivation

$$
\begin{aligned}
\underline{\texttt{SENTENCE}} &\longrightarrow \underline{\texttt{SUBJECT}}\ \texttt{VERB OBJECT} \\
&\longrightarrow \underline{\texttt{ARTICLE}}\ \texttt{NOUNPHRASE VERB OBJECT} \\
&\longrightarrow \texttt{the NOUNPHRASE}\ \underline{\texttt{VERB}}\ \texttt{OBJECT} \\
&\longrightarrow \texttt{the}\ \underline{\texttt{NOUNPHRASE}}\ \texttt{eats OBJECT} \\
&\longrightarrow \texttt{the}\ \underline{\texttt{ADJECTIVE}}\ \texttt{NOUN eats OBJECT} \\
&\longrightarrow \texttt{the big}\ \underline{\texttt{NOUN}}\ \texttt{eats OBJECT} \\
&\longrightarrow \texttt{the big cat eats}\ \underline{\texttt{OBJECT}} \\
&\longrightarrow \texttt{the big cat eats}\ \underline{\texttt{ARTICLE}}\ \texttt{NOUNPHRASE} \\
&\longrightarrow \texttt{the big cat eats a}\ \underline{\texttt{NOUNPHRASE}} \\
&\longrightarrow \texttt{the big cat eats a}\ \underline{\texttt{ADJECTIVE}}\ \texttt{NOUN} \\
&\longrightarrow \texttt{the big cat eats a small}\ \underline{\texttt{NOUN}} \\
&\longrightarrow \texttt{the big cat eats a small dog}
\end{aligned}
$$

Example of Backus-Naur Form (BNF)

Terminals:

$$\mathtt{x} \quad ' \quad + \quad - \quad * \quad (\quad)$$

Non-terminals:

$$\mathtt{id} \quad \mathtt{op} \quad \mathtt{exp}$$

Start symbol:

$$\mathtt{exp}$$

Productions:

$$\begin{aligned} \mathtt{id} \quad &::= \quad \mathtt{x} \mid \mathtt{id}' \\ \mathtt{op} \quad &::= \quad + \mid - \mid * \\ \mathtt{exp} \quad &::= \quad \mathtt{id} \mid \mathtt{exp}\ \mathtt{op}\ \mathtt{exp} \mid (\mathtt{exp}) \end{aligned}$$

Regular expressions over an alphabet Σ

- each symbol $a \in \Sigma$ is a regular expression
- ε 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 ε, $\emptyset$, $($, $)$, $|$, and * are not symbols in Σ.)

A context free grammar for regular expressions over alphabet Σ

set of terminals $\Sigma \cup \{\varepsilon, \emptyset, (,), |, *\}$

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

start symbol r

productions

$$r ::= a \mid \varepsilon \mid \emptyset \mid (r|r) \mid rr \mid (r)^*$$

(where $a \in \Sigma$)