

1. Data l'espressione regolare

$a((a|ab)c)^*b$

- Si disegni il diagramma dell'automa deterministico che riconosce il linguaggio descritto da tale espressione, mostrando anche i passi intermedi della costruzione.
- Dato l'automa ricavato in precedenza, si derivi l'espressione regolare che rappresenta il linguaggio riconosciuto da tale automa. Ovviamente, l'espressione che si ottiene sarà equivalente a quella di partenza, ma l'esercizio richiede di mostrare i passi necessari per derivarla.

2. Data l'espressione regolare

$cb(cb|ca)^*bb$

- Si disegni il diagramma dell'automa deterministico che riconosce il linguaggio descritto da tale espressione, mostrando anche i passi intermedi della costruzione.
- Dato l'automa ricavato in precedenza, si derivi l'espressione regolare che rappresenta il linguaggio riconosciuto da tale automa. Ovviamente, l'espressione che si ottiene sarà equivalente a quella di partenza, ma l'esercizio richiede di mostrare i passi necessari per derivarla.

3. Il comando Unix "ps aux" mostra i processi attivi. Per ogni processo, vengono visualizzati da destra a sinistra, divisi da caratteri di spaziatura: il nome dell'utente proprietario del processo; un numero progressivo che identifica il processo; la percentuale del tempo di CPU usata; la percentuale di memoria usata; la dimensione virtuale (SIZE); la dimensione in memoria (RSS); il tty che controlla il processo; lo stato del processo; la data di inizio (o l'ora di inizio se la data è oggi); il tempo complessivo di CPU usato; il comando che ha iniziato il processo.

USER	PID	%CPU	%MEM	SIZE	RSS	TTY	STAT	START	TIME	COMMAND
at	198	0.0	0.0	892	112	? S		Jun 12	0:00	/usr/sbin/atd
bin	131	0.0	0.1	820	212	? S		Jun 12	0:04	/sbin/portmap
pippo	29455	0.0	1.0	1908	1280	p4 S		15:11	0:00	-bash
pippo	29500	0.0	3.5	6216	4524	p6 S		15:14	0:00	emacs surname.tex
pippo	30075	0.0	0.4	956	536	p4 R		18:13	0:00	ps aux
pluto	28271	0.0	1.0	1948	1324	p1 S		11:04	0:00	-bash
root	1	0.0	0.0	260	76	? S		Jun 12	0:05	init [2]
root	2	0.0	0.0	0	0	? SW		Jun 12	0:00	(kflushd)
root	3	0.0	0.0	0	0	? SW		Jun 12	0:03	(kupdate)
root	4	0.0	0.0	0	0	? SW		Jun 12	0:00	(kpiod)
root	5	0.0	0.0	0	0	? SW		Jun 12	0:01	(kswapd)
root	74	0.0	0.2	1120	348	? S		Jun 12	0:05	/usr/lib/postfix/mast

Si fornisca un'espressione per grep (posix 1003.2) che consenta di individuare tutte le righe relative a processi che: occupano la cpu più del 75%; sono controllati da un tty il cui nome è un "p" seguito da un intero (ad esempio, p1, p13,...); sono iniziati dopo il 15 giugno; sono una shell bash, cioè sono stati lanciati da un comando "-bash".

4. Il comando Unix “ls -l” mostra il contenuto di una directory. Per ogni file, vengono visualizzati da destra a sinistra, divisi da caratteri di spaziatura: 10 caratteri che definiscono il tipo di file e i diritti di accesso; per le directory, il numero di file che contiene (o 1 per i file); il nome dell’utente proprietario; il gruppo a cui appartiene il file; la dimensione; mese, giorno e ora dell’ultima modifica, se fatta nell’anno corrente, oppure mese, giorno e anno; il nome del file.

```
-rw-r--r-- 1 pippo users 44691 Jun 10 16:58 Fondamenti4.tex
-rwxr-xr-x 1 pippo users 100 Jun 10 17:21 Fourpages
-rw-r--r-- 1 pippo users 19968 Jun 12 17:06 MPS2.doc
drwx----- 2 pippo users 2048 Jun 10 17:31 Mail
-rw-r--r-- 1 pippo users 22528 Jun 12 10:21 Preventivo.doc
drwxr-xr-x 2 pippo users 2048 Oct 17 2001 bibbiena
drwx----- 2 pippo users 2048 Feb 15 11:24 mail
drwxr-xr-x 2 pippo users 2048 Jun 27 2000 public_html
drwxr-xr-x 3 pippo users 2048 Feb 21 17:24 seminari
drwxr-xr-x 59 pippo users 2048 May 13 11:03 tex
-rwxr-xr-x 1 pippo users 53 Jun 10 17:29 twopage
```

Si fornisca un’espressione per grep (posix 1003.2) che consenta di individuare tutte le righe relative a directory che: contengono da 8 a 128 files; di cui l’utente “pippo” è proprietario; il cui nome inizia per “sys”; creati l’ultimo giorno del mese di dicembre o novembre.

Regular expressions

The fundamental building blocks are the regular expressions that match a single character. Most characters, including all letters and digits, are regular expressions that match themselves. Any metacharacter with special meaning may be quoted by preceding it with a backslash. A list of characters enclosed by '[' and ']' matches any single character in that list; if the first character of the list is the caret '^', then it matches any character **not** in the list. For example, the regular expression '[0123456789]' matches any single digit. A range of ASCII characters may be specified by giving the first and last characters, separated by a hyphen. Finally, certain named classes of characters are predefined, as follows. Their interpretation depends on the LC_CTYPE locale; the interpretation below is that of the POSIX locale, which is the default if no LC_CTYPE locale is specified.

```
`[:alnum:]'
    Any of `[:digit:]' or `[:alpha:]'
`[:alpha:]'
    Any letter:
    a b c d e f g h i j k l m n o p q r s t u v w x y z,
    A B C D E F G H I J K L M N O P Q R S T U V W X Y Z .
`[:blank:]'
    Space or tab.
`[:cntrl:]'
    Any character with octal codes 000 through 037, or DEL (octal code 177).
`[:digit:]'
    Any one of 0 1 2 3 4 5 6 7 8 9.
`[:graph:]'
    Anything that is not a `[:alnum:]' or `[:punct:]'.
`[:lower:]'
    Any one of a b c d e f g h i j k l m n o p q r s t u v w x y z.
`[:print:]'
    Any character from the `[:space:]' class, and any character that is not in the `[:graph:]' class.
`[:punct:]'
    Any one of ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~.
`[:space:]'
    Any one of CR FF HT NL VT SPACE.
`[:upper:]'
    Any one of A B C D E F G H I J K L M N O P Q R S T U V W X Y Z.
`[:xdigit:]'
    Any one of a b c d e f A B C D E F 0 1 2 3 4 5 6 7 8 9.
```

For example, '[[:alnum:]]' means '[0-9A-Za-z]', except the latter form is dependent upon the ASCII character encoding, whereas the former is portable. (Note that the brackets in these class names are part of the symbolic names, and must be included in addition to the brackets delimiting the bracket list.) Most metacharacters lose their special meaning inside lists. To include a literal ']', place it first in the list. Similarly, to include a literal '^', place it anywhere but first. Finally, to include a literal '-', place it last.

The period `'.'` matches any single character. The symbol `'\w'` is a synonym for `'[:alnum:]'` and `'\W'` is a synonym for `'[^:alnum:]'`.

The caret `'^'` and the dollar sign `'$'` are metacharacters that respectively match the empty string at the beginning and end of a line. The symbols `'<'` and `'>'` respectively match the empty string at the beginning and end of a word. The symbol `'\b'` matches the empty string at the edge of a word, and `'\B'` matches the empty string provided it's not at the edge of a word.

A regular expression may be followed by one of several repetition operators:

- `'?'` The preceding item is optional and will be matched at most once.
- `'*'` The preceding item will be matched zero or more times.
- `'+'` The preceding item will be matched one or more times.
- `'{n}'` The preceding item is matched exactly n times.
- `'{n,}'` The preceding item is matched n or more times.
- `'{n,m}'` The preceding item is matched at least n times, but not more than m times.

Two regular expressions may be concatenated; the resulting regular expression matches any string formed by concatenating two substrings that respectively match the concatenated subexpressions.

Two regular expressions may be joined by the infix operator `'|'`; the resulting regular expression matches any string matching either subexpression.

Repetition takes precedence over concatenation, which in turn takes precedence over alternation. A whole subexpression may be enclosed in parentheses to override these precedence rules.

The backreference `'\n'`, where n is a single digit, matches the substring previously matched by the n th parenthesized subexpression of the regular expression.

In basic regular expressions the metacharacters `'?'`, `'+'`, `'{'`, `'|'`, `'('`, and `')'` lose their special meaning; instead use the backslashed versions `'\?'`, `'\+'`, `'\{'`, `'\|'`, `'\('`, and `'\)'`.

5. Si consideri la grammatica

$S \rightarrow Rc \mid cR$
 $R \rightarrow aBR \mid Rab \mid \varepsilon$
 $B \rightarrow b \mid baB$

- Si dica quale delle seguenti stringhe appartiene al linguaggio, mostrando, in caso positivo, una generazione
 - cabab
 - cababa
 - babac
- Si disegni un albero sintattico delle stringhe che appartengono alla grammatica
- Si verifichi che la grammatica è ambigua mostrando un esempio

6. Si consideri la grammatica

$S \rightarrow aBa \mid bAb \mid abSba \mid baSab \mid c$
 $A \rightarrow aSa \mid \varepsilon$
 $B \rightarrow bSb \mid \varepsilon$

- Si dica quale delle seguenti stringhe appartiene al linguaggio, mostrando, in caso positivo, una generazione
 - babbab
 - abba
 - abcba
- Si disegni un albero sintattico delle stringhe che appartengono alla grammatica
- Si verifichi che la grammatica è ambigua mostrando un esempio

7. Si consideri la grammatica

$S \rightarrow SABd \mid d$
 $A \rightarrow aAa \mid ac$
 $B \rightarrow Bb \mid \varepsilon$

- Eliminare la ricorsione a sinistra e fattorizzare la grammatica
- Si calcolino i First e i Follow e si produca la tabella di analisi
- Dimostrare il funzionamento dell'analizzatore sintattico, facendo vedere lo stack e l'input passo per passo, quando si analizza la stringa: daacad

Gli esercizi b e c sono facoltativi

8. Si consideri la grammatica

$S \rightarrow SA \mid d$
 $A \rightarrow aBa \mid aa$
 $B \rightarrow Bb \mid c$

- Eliminare la ricorsione a sinistra e fattorizzare la grammatica
- Si calcolino i First e i Follow e si produca la tabella di analisi
- Dimostrare il funzionamento dell'analizzatore sintattico, facendo vedere lo stack e l'input passo per passo, quando si analizza la stringa: dacbba