Automata and Formal Languages (4)

Email: christian.urban at kcl.ac.uk
Office: S1.27 (1st floor Strand Building)
Slides: KEATS (also home work is there)

Last Week

Last week I showed you

- tokenizer
- tokenization identifies lexeme in an input stream of characters (or string) and categorizes them into tokens
- maximal munch rule

`http://www.technologyreview.com/tr10/?year=2011`

The Derivative of a Rexp

$$
\begin{array}{lcl}
\text{der}\,c\,(\varnothing) & \stackrel{\text{def}}{=} & \varnothing \\
\text{der}\,c\,(\epsilon) & \stackrel{\text{def}}{=} & \varnothing \\
\text{der}\,c\,(d) & \stackrel{\text{def}}{=} & \text{if}\ c = d\ \text{then}\ \epsilon\ \text{else}\ \varnothing \\
\text{der}\,c\,(r_1 + r_2) & \stackrel{\text{def}}{=} & (\text{der}\,c\,r_1) + (\text{der}\,c\,r_2) \\
\text{der}\,c\,(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \text{if nullable}\ r_1 \\
 & & \text{then}\ ((\text{der}\,c\,r_1) \cdot r_2) + (\text{der}\,c\,r_2) \\
 & & \text{else}\ (\text{der}\,c\,r_1) \cdot r_2 \\
\text{der}\,c\,(r^*) & \stackrel{\text{def}}{=} & (\text{der}\,c\,r) \cdot (r^*)
\end{array}
$$

"the regular expression after c has been recognised"

For this we defined the set $\text{Der}\ c\ A$ as

$$\text{Der}\ c\ A \stackrel{\text{def}}{=} \{\, s \mid c\!::\!s \in A \}$$

which is called the semantic derivative of a set and proved

$$L(\text{der}\ c\ r) = \text{Der}\ c\ (L(r))$$

The Idea of the Algorithm

If we want to recognise the string abc with regular expression r then

1. $Der\ a\ (L(r))$

The Idea of the Algorithm

If we want to recognise the string abc with regular expression r then

1. Der a (L(r))
2. Der b (Der a (L(r)))
3. Der c (Der b (Der a (L(r))))

The Idea of the Algorithm

If we want to recognise the string abc with regular expression r then

1. Der a (L(r))
2. Der b (Der a (L(r)))
3. Der c (Der b (Der a (L(r))))
4. finally we test whether the empty string is in set

The Idea of the Algorithm

If we want to recognise the string abc with regular expression r then

1. Der a (L(r))
2. Der b (Der a (L(r)))
3. Der c (Der b (Der a (L(r))))
4. finally we test whether the empty string is in set

The matching algorithm works similarly, just over regular expression than sets.

Input: string abc and regular expression r

1. der a r
2. der b (der a r)
3. der c (der b (der a r))

Input: string abc and regular expression r

1. der a r
2. der b (der a r)
3. der c (der b (der a r))
4. finally check whether the latter regular expression can match the empty string

We need to prove

$$L(\text{der c r}) = \text{Der c } (L(\text{r}))$$

by induction on the regular expression.

Proofs about Rexp

- P holds for $\varnothing$, ϵ and c
- P holds for $r_1 + r_2$ under the assumption that P already holds for r_1 and r_2.
- P holds for $r_1 \cdot r_2$ under the assumption that P already holds for r_1 and r_2.
- P holds for r^* under the assumption that P already holds for r.

Proofs about Natural Numbers and Strings

- P holds for 0 and
- P holds for $n + 1$ under the assumption that P already holds for n

- P holds for `""` and
- P holds for $c :: s$ under the assumption that P already holds for s

Regular Expressions

r	$::=$	$\varnothing$	null
	$\mid$	ϵ	empty string / "" / []
	$\mid$	c	character
	$\mid$	$r_1 \cdot r_2$	sequence
	$\mid$	$r_1 + r_2$	alternative / choice
	$\mid$	r^*	star (zero or more)

Regular Expressions

$$
\begin{array}{rcll}
r & ::= & \varnothing & \text{null} \\
 & | & \epsilon & \text{empty string / "" / []} \\
 & | & c & \text{character} \\
 & | & r_1 \cdot r_2 & \text{sequence} \\
 & | & r_1 + r_2 & \text{alternative / choice} \\
 & | & r^* & \text{star (zero or more)}
\end{array}
$$

Languages

A language is a set of strings.

A regular expression specifies a set of strings or language.

A language is regular iff there exists a regular expression that recognises all its strings.

Languages

A language is a set of strings.

A regular expression specifies a set of strings or language.

A language is regular iff there exists a regular expression that recognises all its strings.

not all languages are regular, e.g. $a^n b^n$.

Regular Expressions

$$
\begin{array}{llll}
r & ::= & \varnothing & \text{null} \\
 & \mid & \epsilon & \text{empty string / "" / []} \\
 & \mid & c & \text{character} \\
 & \mid & r_1 \cdot r_2 & \text{sequence} \\
 & \mid & r_1 + r_2 & \text{alternative / choice} \\
 & \mid & r^* & \text{star (zero or more)}
\end{array}
$$

How about ranges [a-z], r^+ and !r?

Negation of Regular Expr's

- !r (everything that r cannot recognise)
- $L(!r) \stackrel{\text{def}}{=} \text{UNIV} - L(r)$
- nullable (!r) $\stackrel{\text{def}}{=}$ not (nullable(r))
- der c (!r) $\stackrel{\text{def}}{=}$!(der c r)

Regular Exp's for Lexing

Lexing separates strings into "words" / components.

- Identifiers (non-empty strings of letters or digits, starting with a letter)
- Numbers (non-empty sequences of digits omitting leading zeros)
- Keywords (else, if, while, ...)
- White space (a non-empty sequence of blanks, newlines and tabs)
- Comments

Automata

A deterministic finite automaton consists of:

- a set of states
- one of these states is the start state
- some states are accepting states, and
- there is transition function

  which takes a state as argument and a character and produces a new state

  this function might not always be defined