

Automata and Formal Languages (5)

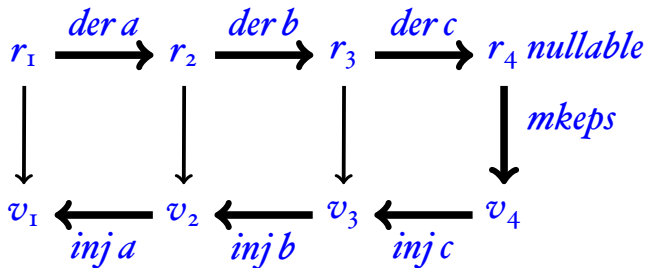
Email: christian.urban at kcl.ac.uk
Office: SI.27 (1st floor Strand Building)
Slides: KEATS (also home work is there)

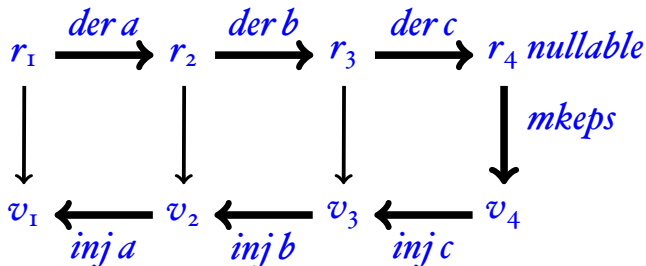
Last Week

Regexes and Values

Regular expressions and their corresponding values:

$r ::=$	\emptyset	$v ::=$	<i>Empty</i>
	ϵ		<i>Char</i> (c)
	c		<i>Seq</i> (v_1, v_2)
	$r_1 \cdot r_2$		<i>Left</i> (v)
	$r_1 + r_2$		<i>Right</i> (v)
	r^*		$[v_1, \dots, v_n]$



$$\begin{aligned}
 r_1: & a \cdot (b \cdot c) \\
 r_2: & \epsilon \cdot (b \cdot c) \\
 r_3: & (\emptyset \cdot (b \cdot c)) + (\epsilon \cdot c) \\
 r_4: & (\emptyset \cdot (b \cdot c)) + ((\emptyset \cdot c) + \epsilon)
 \end{aligned}$$


$$\begin{aligned}
 v_1: & \text{Seq}(\text{Char}(a), \text{Seq}(\text{Char}(b), \text{Char}(c))) \\
 v_2: & \text{Seq}(\text{Empty}, \text{Seq}(\text{Char}(b), \text{Char}(c))) \\
 v_3: & \text{Right}(\text{Seq}(\text{Empty}, \text{Char}(c))) \\
 v_4: & \text{Right}(\text{Right}(\text{Empty}))
 \end{aligned}$$

Mkeps

Finding a (posix) value for recognising the empty string:

$$\begin{aligned} \text{mkeps } \epsilon & \stackrel{\text{def}}{=} \text{Empty} \\ \text{mkeps } r_1 + r_2 & \stackrel{\text{def}}{=} \text{if nullable}(r_1) \\ & \quad \text{then Left}(\text{mkeps}(r_1)) \\ & \quad \text{else Right}(\text{mkeps}(r_2)) \\ \text{mkeps } r_1 \cdot r_2 & \stackrel{\text{def}}{=} \text{Seq}(\text{mkeps}(r_1), \text{mkeps}(r_2)) \\ \text{mkeps } r^* & \stackrel{\text{def}}{=} \square \end{aligned}$$

Inject

Injecting (“Adding”) a character to a value

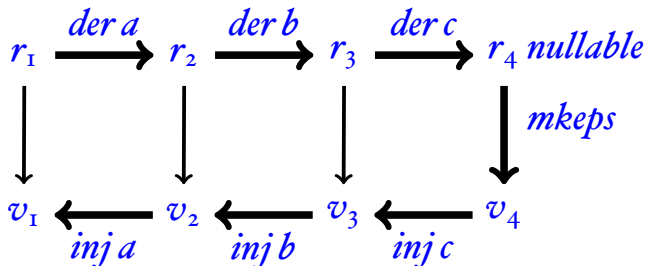
$$\begin{aligned} inj(c) c Empty & \stackrel{\text{def}}{=} Char\ c \\ inj(r_I + r_2) c Left(v) & \stackrel{\text{def}}{=} Left(inj\ r_I\ c\ v) \\ inj(r_I + r_2) c Right(v) & \stackrel{\text{def}}{=} Right(inj\ r_2\ c\ v) \\ inj(r_I \cdot r_2) c Seq(v_I, v_2) & \stackrel{\text{def}}{=} Seq(inj\ r_I\ c\ v_I, v_2) \\ inj(r_I \cdot r_2) c Left(Seq(v_I, v_2)) & \stackrel{\text{def}}{=} Seq(inj\ r_I\ c\ v_I, v_2) \\ inj(r_I \cdot r_2) c Right(v) & \stackrel{\text{def}}{=} Seq(mkeps(r_I), inj\ r_2\ c\ v) \\ inj(r^*) c Seq(v, vs) & \stackrel{\text{def}}{=} inj\ r\ c\ v :: vs \end{aligned}$$

inj: 1st arg \mapsto a rexp; 2nd arg \mapsto a character; 3rd arg \mapsto a value

Flatten

Obtaining the string underlying a value:

$ Empty $	$\stackrel{\text{def}}{=} []$
$ Char(c) $	$\stackrel{\text{def}}{=} [c]$
$ Left(v) $	$\stackrel{\text{def}}{=} v $
$ Right(v) $	$\stackrel{\text{def}}{=} v $
$ Seq(v_1, v_2) $	$\stackrel{\text{def}}{=} v_1 @ v_2 $
$ [v_1, \dots, v_n] $	$\stackrel{\text{def}}{=} v_1 @ \dots @ v_n $

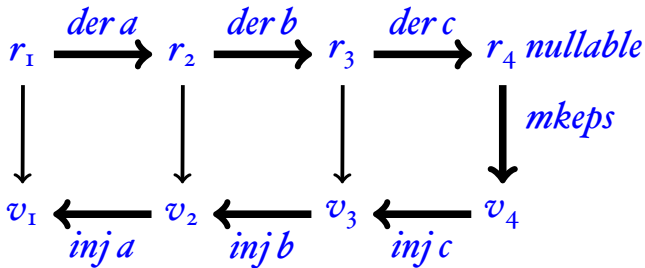
$$\begin{aligned}
 r_1: & a \cdot (b \cdot c) \\
 r_2: & \epsilon \cdot (b \cdot c) \\
 r_3: & (\emptyset \cdot (b \cdot c)) + (\epsilon \cdot c) \\
 r_4: & (\emptyset \cdot (b \cdot c)) + ((\emptyset \cdot c) + \epsilon)
 \end{aligned}$$


$$\begin{aligned}
 v_1: & \text{Seq}(\text{Char}(a), \text{Seq}(\text{Char}(b), \text{Char}(c))) \\
 v_2: & \text{Seq}(\text{Empty}, \text{Seq}(\text{Char}(b), \text{Char}(c))) \\
 v_3: & \text{Right}(\text{Seq}(\text{Empty}, \text{Char}(c))) \\
 v_4: & \text{Right}(\text{Right}(\text{Empty}))
 \end{aligned}$$

$$\begin{aligned}
 |v_1|: & abc \\
 |v_2|: & bc \\
 |v_3|: & c \\
 |v_4|: & []
 \end{aligned}$$

Simplification

- If we simplify after the derivative, then we are building the value for the simplified regular expression, but *not* for the original regular expression.



$$(\emptyset \cdot (b \cdot c)) + ((\emptyset \cdot c) + \epsilon) \mapsto \epsilon$$

Rectification

$\mathit{simp}(r)$:

case $r = r_1 + r_2$

let $(r_{1s}, f_{1s}) = \mathit{simp}(r_1)$

$(r_{2s}, f_{2s}) = \mathit{simp}(r_2)$

case $r_{1s} = \emptyset$: return $(r_{2s}, \lambda v. \mathit{Right}(f_{2s}(v)))$

case $r_{2s} = \emptyset$: return $(r_{1s}, \lambda v. \mathit{Left}(f_{1s}(v)))$

case $r_{1s} = r_{2s}$: return $(r_{1s}, \lambda v. \mathit{Left}(f_{1s}(v)))$

otherwise: return $(r_{1s} + r_{2s}, \mathit{falt}(f_{1s}, f_{2s}))$

$\mathit{falt}(f_1, f_2) \stackrel{\text{def}}{=}$

$\lambda v. \text{case } v = \mathit{Left}(v') : \text{return } \mathit{Left}(f_1(v'))$

$\text{case } v = \mathit{Right}(v') : \text{return } \mathit{Right}(f_2(v'))$

Rectification

$simp(r):...$

case $r = r_I \cdot r_2$

let $(r_{1s}, f_{1s}) = simp(r_I)$

$(r_{2s}, f_{2s}) = simp(r_2)$

case $r_{1s} = \emptyset$: return (\emptyset, f_{error})

case $r_{2s} = \emptyset$: return (\emptyset, f_{error})

case $r_{1s} = \epsilon$: return $(r_{2s}, \lambda v. Seq(f_{1s}(Empty), f_{2s}(v)))$

case $r_{2s} = \epsilon$: return $(r_{1s}, \lambda v. Seq(f_{1s}(v), f_{2s}(Empty)))$

otherwise: return $(r_{1s} \cdot r_{2s}, f_{seq}(f_{1s}, f_{2s}))$

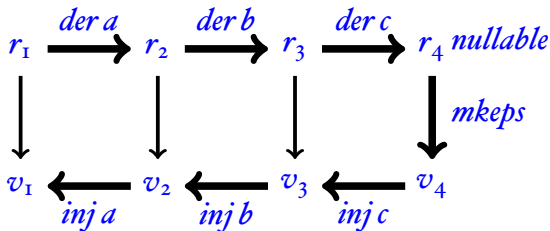
$f_{seq}(f_1, f_2) \stackrel{\text{def}}{=}$

$\lambda v. \text{case } v = Seq(v_1, v_2): \text{return } Seq(f_1(v_1), f_2(v_2))$

Lexing with Simplification

$\text{lex } r \ [] \stackrel{\text{def}}{=} \text{if } \text{nullable}(r) \text{ then } \text{mkeps}(r) \text{ else } \text{error}$

$\text{lex } r \ c \ :: \ s \stackrel{\text{def}}{=} \text{let } (r', \text{frect}) = \text{simp}(\text{der}(c, r))$
 $\text{inj } r \ c \ (\text{frect}(\text{lex}(r', s)))$



Records

- new regex: $(x : r)$ new value: $Rec(x, v)$

Records

- new regex: $(x : r)$ new value: $Rec(x, v)$
- $nullable(x : r) \stackrel{\text{def}}{=} nullable(r)$
- $der\ c(x : r) \stackrel{\text{def}}{=} (x : der\ c\ r)$
- $mkeps(x : r) \stackrel{\text{def}}{=} Rec(x, mkeps(r))$
- $inj(x : r)\ c\ v \stackrel{\text{def}}{=} Rec(x, inj\ r\ c\ v)$

Records

- new regex: $(x : r)$ new value: $Rec(x, v)$
- $nullable(x : r) \stackrel{\text{def}}{=} nullable(r)$
- $der\ c(x : r) \stackrel{\text{def}}{=} (x : der\ c\ r)$
- $mkeps(x : r) \stackrel{\text{def}}{=} Rec(x, mkeps(r))$
- $inj(x : r)\ c\ v \stackrel{\text{def}}{=} Rec(x, inj\ r\ c\ v)$

for extracting subpatterns $(z : ((x : ab) + (y : ba)))$

While Tokens

WHILE_REGS $\stackrel{\text{def}}{=} (($ "k" : KEYWORD) +
("i" : ID) +
("o" : OP) +
("n" : NUM) +
("s" : SEMI) +
("p" : (LPAREN + RPAREN)) +
("b" : (BEGIN + END)) +
("w" : WHITESPACE))*

”if true then then 42 else +”

KEYWORD(if),
WHITESPACE,
IDENT(true),
WHITESPACE,
KEYWORD(then),
WHITESPACE,
KEYWORD(then),
WHITESPACE,
NUM(42),
WHITESPACE,
KEYWORD(else),
WHITESPACE,
OP(+)

”if true then then 42 else +”

KEYWORD(if),
IDENT(true),
KEYWORD(then),
KEYWORD(then),
NUM(42),
KEYWORD(else),
OP(+)

There is one small problem with the tokenizer.
How should we tokenize:

"x - 3"

OP:

"+" , "-"

NUM:

(NONZERODIGIT · DIGIT*) + "0"

NUMBER:

NUM + ("-" · NUM)

Two Rules

- Longest match rule (“maximal munch rule”): The longest initial substring matched by any regular expression is taken as next token.
- Rule priority: For a particular longest initial substring, the first regular expression that can match determines the token.

Environment

Obtaining the “recorded” parts of a regular expression:

$env(Empty)$	$\stackrel{\text{def}}{=} []$
$env(Char(c))$	$\stackrel{\text{def}}{=} []$
$env(Left(v))$	$\stackrel{\text{def}}{=} env(v)$
$env(Right(v))$	$\stackrel{\text{def}}{=} env(v)$
$env(Seq(v_1, v_2))$	$\stackrel{\text{def}}{=} env(v_1) @ env(v_2)$
$env([v_1, \dots, v_n])$	$\stackrel{\text{def}}{=} env(v_1) @ \dots @ env(v_n)$
$env(Rec(x : v))$	$\stackrel{\text{def}}{=} (x : v) :: env(v)$

- Regular expression for email addresses

(name: $[a-z0-9_-.]^+$).@.
(domain: $[a-z0-9_-.]^+$) ..
(top_level: $[a-z.]{2,6}$)

christian.urban@kcl.ac.uk

- result environment:

$[(name : christian.urban),$
 $(domain : kcl),$
 $(top_level : ac.uk)]$

Coursework

$nullable([c_1 c_2 \dots c_n]) \stackrel{\text{def}}{=} ?$

$nullable(r^+) \stackrel{\text{def}}{=} ?$

$nullable(r^?) \stackrel{\text{def}}{=} ?$

$nullable(r^{\{n,m\}}) \stackrel{\text{def}}{=} ?$

$nullable(\sim r) \stackrel{\text{def}}{=} ?$

$der\ c\ ([c_1 c_2 \dots c_n]) \stackrel{\text{def}}{=} ?$

$der\ c\ (r^+) \stackrel{\text{def}}{=} ?$

$der\ c\ (r^?) \stackrel{\text{def}}{=} ?$

$der\ c\ (r^{\{n,m\}}) \stackrel{\text{def}}{=} ?$

$der\ c\ (\sim r) \stackrel{\text{def}}{=} ?$