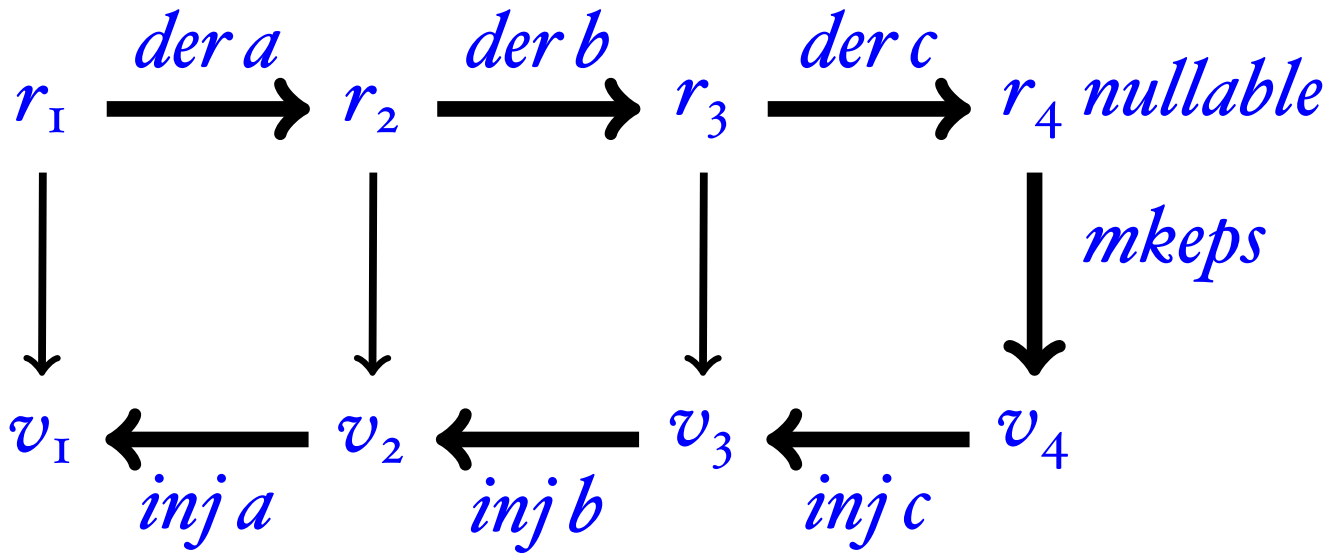
Automata and Formal Languages (5)

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

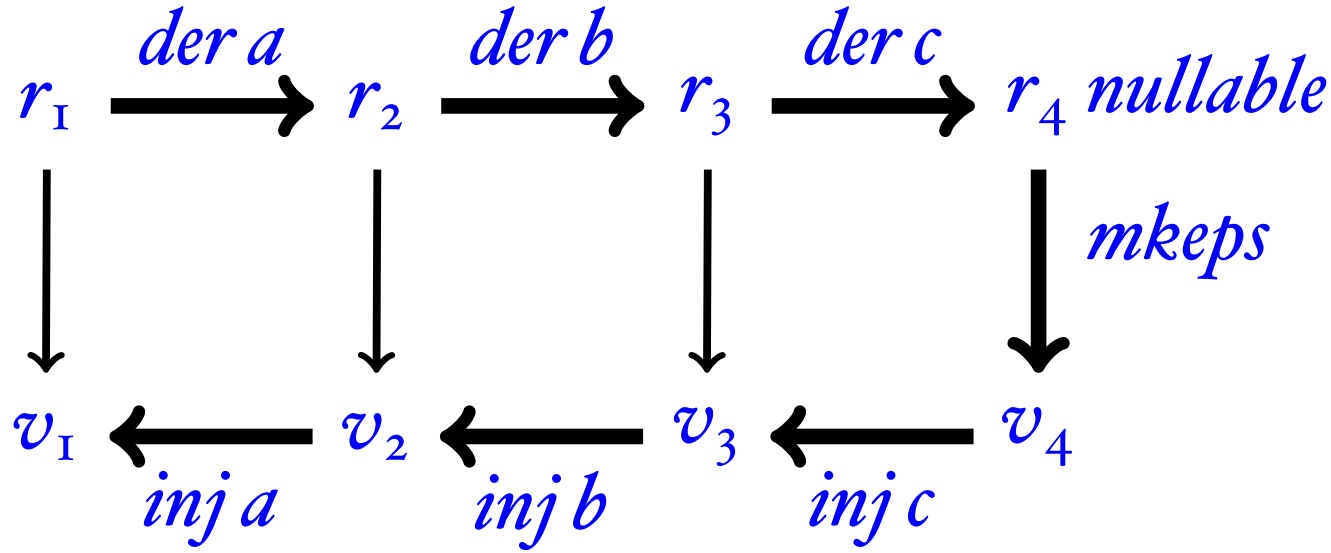
Last Week
Regexes and Values

Regular expressions and their corresponding values:

$$
\begin{array}{rcl}
r & ::= & \varnothing \\
 & | & \epsilon \\
 & | & c \\
 & | & r_1 \cdot r_2 \\
 & | & r_1 + r_2 \\
 & | & r^* \\
\end{array}
\qquad\qquad
\begin{array}{rcl}
v & ::= & \\
 & & \mathit{Empty} \\
 & | & \mathit{Char}(c) \\
 & | & \mathit{Seq}(v_1, v_2) \\
 & | & \mathit{Left}(v) \\
 & | & \mathit{Right}(v) \\
 & | & [v_1, \ldots v_n] \\
\end{array}
$$


der a
der b
der c
r_1
r_2
r_3
r_4
nullable
mkeps
v_1
v_2
v_3
v_4
inj a
inj b
inj c

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



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

Mkeps

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

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

Inject

Injecting ("Adding") a character to a value

$$
\begin{array}{ll}
inj\,(c)\,c\,Empty & \stackrel{\text{def}}{=} Char\,c \\
inj\,(r_1 + r_2)\,c\,Left(v) & \stackrel{\text{def}}{=} Left(inj\,r_1\,c\,v) \\
inj\,(r_1 + r_2)\,c\,Right(v) & \stackrel{\text{def}}{=} Right(inj\,r_2\,c\,v) \\
inj\,(r_1 \cdot r_2)\,c\,Seq(v_1, v_2) & \stackrel{\text{def}}{=} Seq(inj\,r_1\,c\,v_1, v_2) \\
inj\,(r_1 \cdot r_2)\,c\,Left(Seq(v_1, v_2)) & \stackrel{\text{def}}{=} Seq(inj\,r_1\,c\,v_1, v_2) \\
inj\,(r_1 \cdot r_2)\,c\,Right(v) & \stackrel{\text{def}}{=} Seq(mkeps(r_1), inj\,r_2\,c\,v) \\
inj\,(r^*)\,c\,Seq(v, vs) & \stackrel{\text{def}}{=} inj\,r\,c\,v :: vs
\end{array}
$$

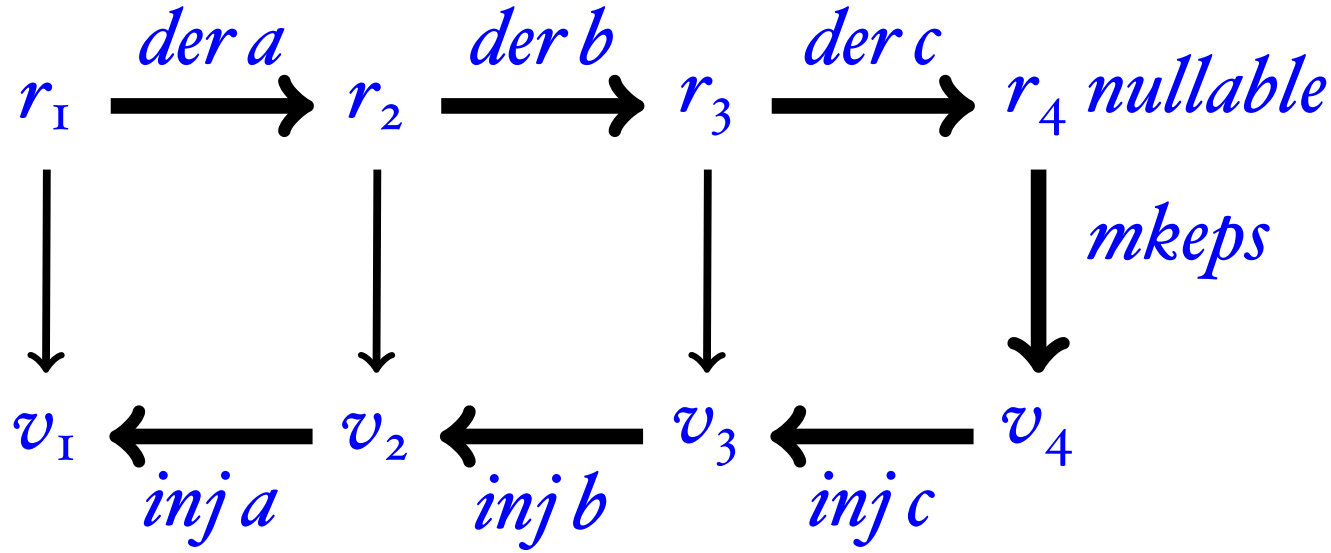
inj: 1st arg $\mapsto$ a rexp; 2nd arg $\mapsto$ a character; 3rd arg $\mapsto$ a value

Flatten

Obtaining the string underlying a value:

$$
\begin{array}{lcl}
|Empty| & \stackrel{\text{def}}{=} & [] \\
|Char(c)| & \stackrel{\text{def}}{=} & [c] \\
|Left(v)| & \stackrel{\text{def}}{=} & |v| \\
|Right(v)| & \stackrel{\text{def}}{=} & |v| \\
|Seq(v_1, v_2)| & \stackrel{\text{def}}{=} & |v_1| \mathbin{@} |v_2| \\
|[v_1, \ldots, v_n]| & \stackrel{\text{def}}{=} & |v_1| \mathbin{@} \ldots \mathbin{@} |v_n|
\end{array}
$$

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

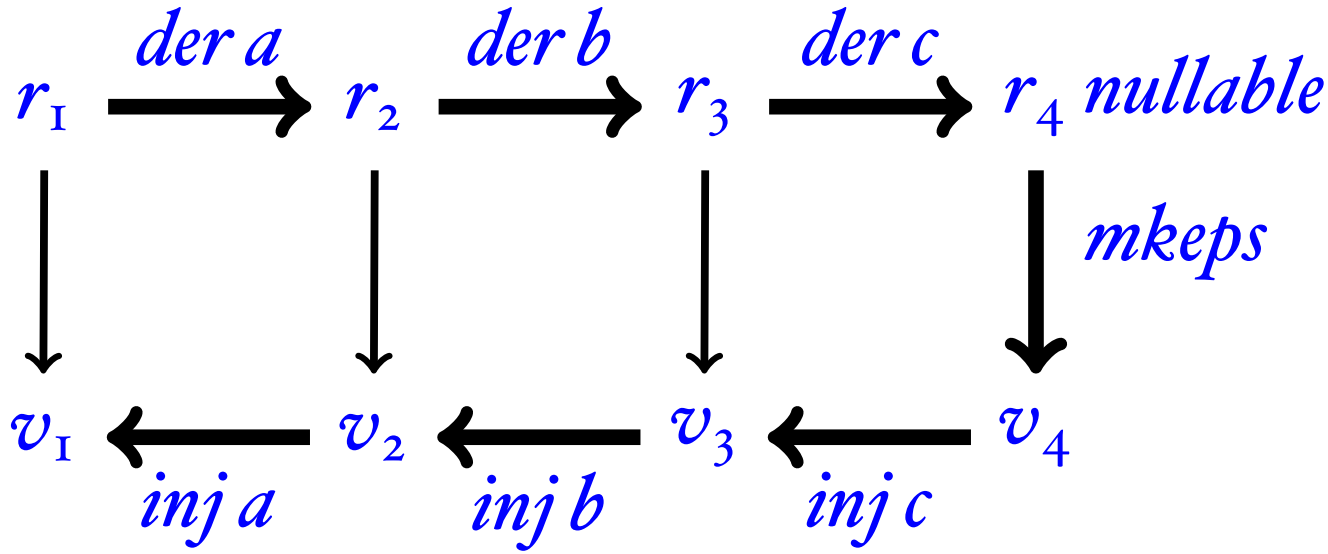


$$
\begin{array}{ll}
v_1: & Seq(Char(a), Seq(Char(b), Char(c))) \\
v_2: & Seq(Empty, Seq(Char(b), Char(c))) \\
v_3: & Right(Seq(Empty, Char(c))) \\
v_4: & Right(Right(Empty))
\end{array}
$$

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

Simplification

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



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

Rectification

$simp(r)$:

case $r = r_1 + r_2$

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

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

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

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

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

otherwise: return $(r_{1s} + r_{2s}, f_{alt}(f_{1s}, f_{2s}))$

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

$\lambda v.$ case $v = Left(v')$: return $Left(f_1(v'))$

case $v = Right(v')$: return $Right(f_2(v'))$

Rectification

$simp(r)$:...

case $r = r_1 \cdot r_2$

let $(r_{1s}, f_{1s}) = simp(r_1)$
$(r_{2s}, f_{2s}) = simp(r_2)$

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

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

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

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

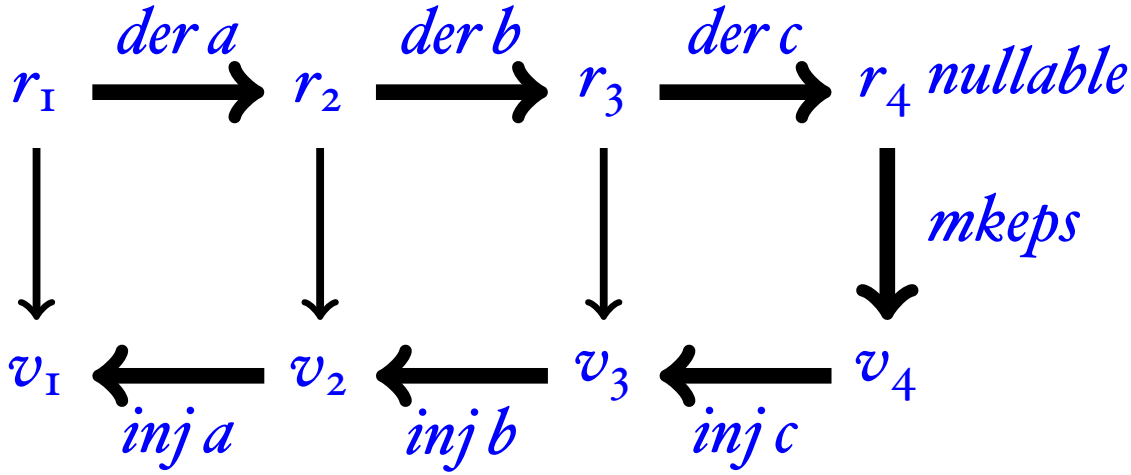
otherwise: return $(r_{1s} \cdot r_{2s}, f_{seq}(f_{1s}, f_{2s}))$

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

Lexing with Simplification

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

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



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

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

”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.