A Formalisation of the Myhill-Nerode Theorem based on Regular Expressions

Christian Urban

joint work with Chunhan Wu and Xingyuan Zhang
from the PLA University of Science and
Technology in Nanjing

A Formalisation of the Myhill-Nerode Theorem based on Regular Expressions

or, Regular Languages Done Right

Christian Urban

joint work with Chunhan Wu and Xingyuan Zhang
from the PLA University of Science and
Technology in Nanjing

In Most Textbooks...

- A regular language is one where there is a DFA that recognises it.

In Most Textbooks. . .

- A regular language is one where there is a DFA that recognises it.

I can think of three reasons why this is a good definition:

- string matching via DFAs (yacc)
- pumping lemma
- closure properties of regular languages (closed under complement)

Really Bad News!

DFAs are bad news for formalisations in theorem provers. They might be represented as:

- graphs
- matrices
- partial functions

All constructions are messy to reason about.

Really Bad News!

DFAs are bad news for formalisations in theorem provers. They might be represented as:

- graphs
- matrices
- partial functions

All constructions are messy to reason about.

Alexander and Tobias: "...automata theory ...does not come for free ..."

Really Bad News!

DFAs are bad news for formalisations in theorem provers. They might be represented as:

- graphs
- matrices
- partial functions

All constructions are messy to reason about.

Constable et al needed (on and off) 18 months for a 3-person team to formalise automata theory in Nuprl including Myhill-Nerode. There is only very little other formalised work on regular languages I know of in Coq, Isabelle and HOL.

Really Bad News!

DFAs are bad news for formalisations in theorem provers. They might be represented as:

- graphs
- matrices
- partial functions

All constructions are messy to reason about.

typical textbook reasoning goes like: "...if M and N are any two automata with no inaccessible states ..."

Regular Expressions

. . . are a simple datatype:

$$
\begin{array}{lcl}
\text{rexp} & ::= & \text{NULL} \\
 & \mid & \text{EMPTY} \\
 & \mid & \text{CHR } c \\
 & \mid & \text{ALT rexp rexp} \\
 & \mid & \text{SEQ rexp rexp} \\
 & \mid & \text{STAR rexp}
\end{array}
$$

Regular Expressions

...are a simple datatype:

$$
\begin{array}{rcl}
r & ::= & \mathbf{0} \\
 & | & [] \\
 & | & c \\
 & | & r_1 + r_2 \\
 & | & r_1 \cdot r_2 \\
 & | & r^\star
\end{array}
$$

Regular Expressions

...are a simple datatype:

$$
\begin{array}{rcl}
r & ::= & \mathbf{0} \\
  & \mid & [] \\
  & \mid & c \\
  & \mid & r_1 + r_2 \\
  & \mid & r_1 \cdot r_2 \\
  & \mid & r^\star
\end{array}
$$

Induction and recursion principles come for free.

Semantics of Rexps

$$
\begin{array}{rcl}
\mathbb{L}(\mathbf{0}) & = & \varnothing \\
\mathbb{L}([]) & = & \{[]\} \\
\mathbb{L}(c) & = & \{[c]\} \\
\mathbb{L}(r_1 + r_2) & = & \mathbb{L}(r_1) \cup \mathbb{L}(r_2) \\
\mathbb{L}(r_1 \cdot r_2) & = & \mathbb{L}(r_1) \;;\; \mathbb{L}(r_2) \\
\mathbb{L}(r^\star) & = & \mathbb{L}(r)^\star
\end{array}
$$

$$
L_1 ; L_2 \stackrel{\text{def}}{=} \{s_1 @ s_2 \mid s_1 \in L_1 \wedge s_2 \in L_2\}
$$

$$
\frac{}{[] \in L^\star} \qquad \frac{s_1 \in L \quad s_2 \in L^\star}{s_1 @ s_2 \in L^\star}
$$

Regular Expression Matching

- Harper in JFP'99: "Functional Pearl: Proof-Directed Debugging"
- Yi in JFP'06: "Educational Pearl: 'Proof-Directed Debugging' revisited for a first-order version"
- Owens et al in JFP'09: "Regular-expression derivatives re-examined"

Regular Expression Matching

- Harper in JFP'99: "Functional Pearl: Proof-Directed Debugging"
- Yi in JFP'06: "Educational Pearl: 'Proof-Directed Debugging' revisited for a first-order version"
- Owens et al in JFP'09: "Regular-expression derivatives re-examined"

> "Unfortunately, regular expression derivatives have been lost in the sands of time, and few computer scientists are aware of them."

Demo

The Myhill-Nerode Theorem

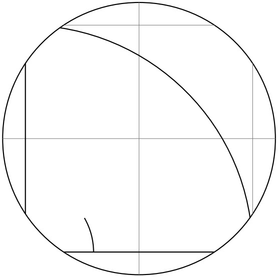
- provides necessary and sufficient conditions for a language being regular (pumping lemma only necessary)
- will help with closure properties of regular languages

The Myhill-Nerode Theorem

- provides necessary and sufficient conditions for a language being regular (pumping lemma only necessary)
- will help with closure properties of regular languages
- key is the equivalence relation:

$$x \approx_L y \stackrel{\text{def}}{=} \forall z.\ x@z \in L \Leftrightarrow y@z \in L$$

The Myhill-Nerode Theorem



- finite $(UNIV // \approx_L) \;\Leftrightarrow\; L$ is regular

Equivalence Classes

- $L = []$

$$\Big\{\{[]\},\ UNIV - \{[]\}\Big\}$$

- $L = [c]$

$$\Big\{\{[]\},\ \{[c]\},\ UNIV - \{[], [c]\}\Big\}$$

- $L = \varnothing$

$$\Big\{UNIV\Big\}$$

Regular Languages

- L is regular $\stackrel{\text{def}}{=}$ if there is an automaton M such that $\mathbb{L}(M) = L$

- Myhill-Nerode:

  finite $\Rightarrow$ regular
  finite $(UNIV // \approx_L) \Rightarrow \exists r. L = \mathbb{L}(r)$

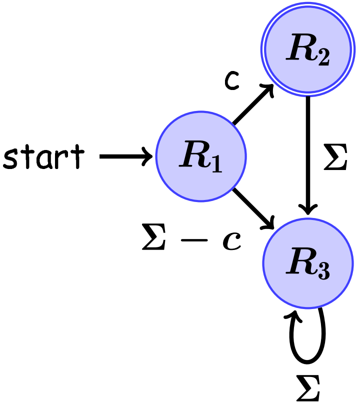
  regular $\Rightarrow$ finite
  finite $(UNIV // \approx_{\mathbb{L}(r)})$

Final States

- $\textsf{final}_L\ X \stackrel{\textsf{def}}{=}$
  $X \in (UNIV // \approx_L)\ \wedge\ \forall s \in X.\ s \in L$
- we can prove: $L = \bigcup \{X.\ \textsf{final}_L\ X\}$

Transitions between Equivalence Classes

$L = \{[c]\}$

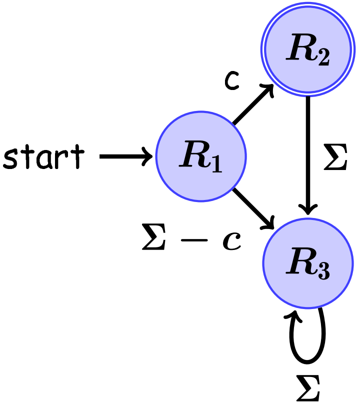


$UNIV // \approx_L$ produces

R_1: $\{[]\}$
R_2: $\{[c]\}$
R_3: $UNIV - \{[], [c]\}$

Transitions between Equivalence Classes

$L = \{[c]\}$



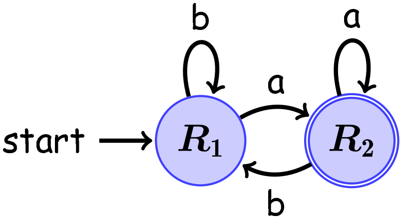
$UNIV // \approx_L$ produces

R_1: $\{[]\}$
R_2: $\{[c]\}$
R_3: $UNIV - \{[], [c]\}$

$$X \stackrel{c}{\longrightarrow} Y \stackrel{\text{def}}{=} X ; [c] \subseteq Y$$

Systems of Equations

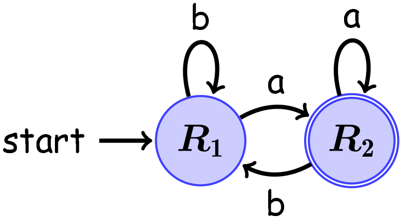
Inspired by a method of Brzozowski '64, we can build an equational system characterising the equivalence classes:



$$R_1 \equiv R_1; b + R_2; b$$
$$R_2 \equiv R_1; a + R_2; a$$

Systems of Equations

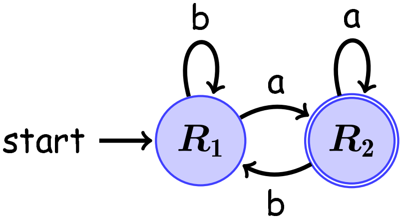
Inspired by a method of Brzozowski '64, we can build an equational system characterising the equivalence classes:



$$R_1 \equiv R_1; b + R_2; b + \lambda; []$$
$$R_2 \equiv R_1; a + R_2; a$$

Systems of Equations

Inspired by a method of Brzozowski '64, we can build an equational system characterising the equivalence classes:



$$R_1 \equiv R_1; b + R_2; b + \lambda; []$$
$$R_2 \equiv R_1; a + R_2; a$$

we can prove

$$R_1 = R_1; \mathbb{L}(b) \cup R_2; \mathbb{L}(b) \cup \{[]\}; \{[]\}$$
$$R_2 = R_1; \mathbb{L}(a) \cup R_2; \mathbb{L}(a)$$

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a + R_2; a$$

A Variant of Arden's Lemma

Arden's Lemma:

If $[] \not\in A$ then

$$X = X; A + \text{something}$$

has the (unique) solution

$$X = \text{something}; A^\star$$

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a + R_2; a$$

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a + R_2; a$$

by Arden

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a + R_2; a$$

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a + R_2; a$$

by Arden

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a \cdot a^\star$$

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a + R_2; a$$

by Arden

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a \cdot a^\star$$

by Arden

$$R_1 = R_2; b \cdot b^\star + \lambda; b^\star$$
$$R_2 = R_1; a \cdot a^\star$$

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a + R_2; a$$

by Arden

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a \cdot a^\star$$

by Arden

$$R_1 = R_2; b \cdot b^\star + \lambda; b^\star$$
$$R_2 = R_1; a \cdot a^\star$$

by substitution

$$R_1 = R_1; a \cdot a^\star \cdot b \cdot b^\star + \lambda; b^\star$$
$$R_2 = R_1; a \cdot a^\star$$

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a + R_2; a$$

by Arden

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a \cdot a^\star$$

by Arden

$$R_1 = R_2; b \cdot b^\star + \lambda; b^\star$$
$$R_2 = R_1; a \cdot a^\star$$

by substitution

$$R_1 = R_1; a \cdot a^\star \cdot b \cdot b^\star + \lambda; b^\star$$
$$R_2 = R_1; a \cdot a^\star$$

by Arden

$$R_1 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star$$
$$R_2 = R_1; a \cdot a^\star$$

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a + R_2; a$$

by Arden

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = R_1; a \cdot a^\star$$

by Arden

$$R_1 = R_2; b \cdot b^\star + \lambda; b^\star$$
$$R_2 = R_1; a \cdot a^\star$$

by substitution

$$R_1 = R_1; a \cdot a^\star \cdot b \cdot b^\star + \lambda; b^\star$$
$$R_2 = R_1; a \cdot a^\star$$

by Arden

$$R_1 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star$$
$$R_2 = R_1; a \cdot a^\star$$

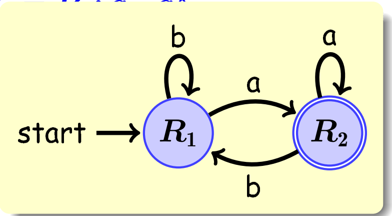
by substitution

$$R_1 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star$$
$$R_2 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star \cdot a \cdot a^\star$$

$R_1 = R_1; b + R_2; b + \lambda; []$
$R_2 = R_1; a + R_2; a$

by Arden

$R_1 = R_1; b + R_2; b + \lambda; []$
$R_2 = R_1; a \cdot a^\star$



by Arden

R
R

by substitution

R
$R_2 = R_1; a \cdot a^\star$

by Arden

$R_1 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star$
$R_2 = R_1; a \cdot a^\star$

by substitution

$R_1 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star$
$R_2 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star \cdot a \cdot a^\star$

The Equ's Solving Algorithm

- The algorithm must terminate: Arden makes one equation smaller; substitution deletes one variable from the right-hand sides.

- We need to maintain the invariant that Arden is applicable (if $[] \not\in A$ then ...):

$$\begin{aligned} R_1 &= R_1; b + R_2; b + \lambda; [] \\ R_2 &= R_1; a + R_2; a \end{aligned}$$

by Arden

$$\begin{aligned} R_1 &= R_1; b + R_2; b + \lambda; [] \\ R_2 &= R_1; a \cdot a^\star \end{aligned}$$

The Equ's Solving Algorithm

- The algorithm is still a bit hairy to formalise because of our set-representation for equations:

$$\{(X, \{(Y_1, r_1), (Y_2, r_2), \ldots\}),$$
$$\ldots$$
$$\}$$

The Equ's Solving Algorithm

- The algorithm is still a bit hairy to formalise because of our set-representation for equations:

$$\{(X, \{(Y_1, r_1), (Y_2, r_2), \ldots\}), \\ \ldots \\ \}$$

they are generated from $UNIV // \approx_L$

Other Direction

One has to prove

$$\text{finite}(\boldsymbol{UNIV} /\!/ \approx_{\mathbb{L}(r)})$$

by induction on r. Not trivial, but after a bit of thinking (by Chunhan), one can prove that if

$$\text{finite}(\boldsymbol{UNIV} /\!/ \approx_{\mathbb{L}(r_1)}) \qquad \text{finite}(\boldsymbol{UNIV} /\!/ \approx_{\mathbb{L}(r_2)})$$

then

$$\text{finite}(\boldsymbol{UNIV} /\!/ \approx_{\mathbb{L}(r_1) \cup \mathbb{L}(r_2)})$$

What Have We Achieved?

- finite $(UNIV // \approx_L) \;\Leftrightarrow\; L$ is regular

What Have We Achieved?

- finite $(UNIV // \approx_L) \;\Leftrightarrow\; L$ is regular
- regular languages are closed under complementation; this is easy

$$UNIV // \approx_L \;=\; UNIV // \approx_{-L}$$

$$x \approx_L y \stackrel{\text{def}}{=} \forall z.\; x@z \in L \Leftrightarrow y@z \in L$$

What Have We Achieved?

- finite $(UNIV // \approx_L)$ $\Leftrightarrow$ L is regular
- regular languages are closed under complementation; this is easy
$$UNIV // \approx_L \;=\; UNIV // \approx_{-L}$$
- if you want to do regular expression matching (see Scott's paper)

What Have We Achieved?

- finite $(UNIV // \approx_L) \;\Leftrightarrow\; L$ is regular
- regular languages are closed under complementation; this is easy
$$UNIV // \approx_L \;=\; UNIV // \approx_{-L}$$
- if you want to do regular expression matching (see Scott's paper)
- I cannot yet give definite numbers

Examples

- $L \equiv \Sigma^\star 0 \Sigma$ is regular

$$\begin{aligned} A_1 &= \Sigma^\star 00 \\ A_2 &= \Sigma^\star 01 \\ A_3 &= \Sigma^\star 10 \cup \{0\} \\ A_4 &= \Sigma^\star 11 \cup \{1\} \cup \{[]\} \end{aligned}$$

- $L \equiv \{0^n 1^n \mid n \geq 0\}$ is not regular

$$\begin{aligned} B_0 &= \{0^n 1^n \mid n \geq 0\} \\ B_1 &= \{0^n 1^{(n-1)} \mid n \geq 1\} \\ B_2 &= \{0^n 1^{(n-2)} \mid n \geq 2\} \\ B_3 &= \{0^n 1^{(n-3)} \mid n \geq 3\} \\ &\vdots \end{aligned}$$

What We Have Not Achieved

- regular expressions are not good if you look for a minimal one for a language (DFAs have this notion)

What We Have Not Achieved

- regular expressions are not good if you look for a minimal one for a language (DFAs have this notion)

- Is there anything to be said about context free languages:

  A context free language is where every string can be recognised by a pushdown automaton.

Conclusion

- on balance regular expression are superior to DFAs, in my opinion
- I cannot think of a reason to not teach regular languages to students this way (!?)
- I have never ever seen a proof of Myhill-Nerode based on regular expressions
- no application, but lots of fun
- great source of examples