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 Textbooks...

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

In Textbooks. . .

- A regular language is one where there is DFA that recognises it.
- Pumping lemma, closure properties of regular languages (closed under “negation”) etc are all described and proved in terms of DFAs.

In Textbooks...

- A regular language is one where there is DFA that recognises it.
- Pumping lemma, closure properties of regular languages (closed under "negation") etc are all described and proved in terms of DFAs.
- Similarly the Myhill-Nerode theorem, which gives necessary and sufficient conditions for a language being regular (also describes a minimal DFA for a language).

Really Bad News!

This is bad news for formalisations in theorem provers. DFAs might be represented as

- graphs
- matrices
- partial functions

All constructions are difficult to reason about.

Really Bad News!

This is bad news for formalisations in theorem provers. DFAs might be represented as

- graphs
- matrices
- partial functions

All constructions are difficult 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!

This is bad news for formalisations in theorem provers. DFAs might be represented as

- graphs
- matrices
- partial functions

All constructions are difficult 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:

```
rexp ::= NULL
       |  EMPTY
       |  CHR c
       |  ALT rexp rexp
       |  SEQ rexp rexp
       |  STAR rexp
```

Regular Expressions

...are a simple datatype:

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

Regular Expressions

…are a simple datatype:

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

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

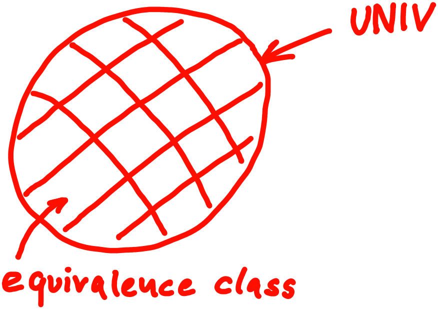
- will help with closure properties of regular languages and with the pumping lemma.
- provides necessary and sufficient conditions for a language being regular

The Myhill-Nerode Theorem

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

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

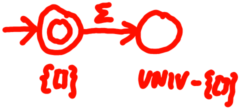
The Myhill-Nerode Theorem



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

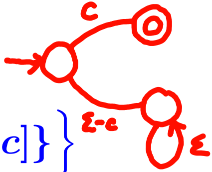
Equivalence Classes

- $L = []$



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

- $L = [c]$



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

- $L = \varnothing$

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

Regular Languages

- L is regular $\stackrel{\text{def}}{=}$ if there is an ~~automaton M~~ regular expression r such that $\mathbb{L}(\not{M}\,r) = L$

- Myhill-Nerode:

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

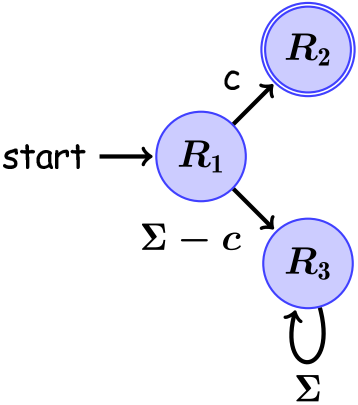
  regular $\Rightarrow$ finite
  finite $(UNIV // \approx_{\mathbb{L}(r)})$

Final States

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

Transitions between Equivalence Classes

$L = \{[c]\}$

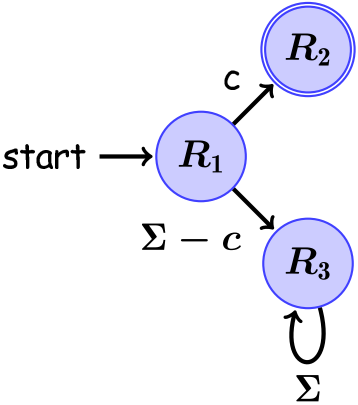


$UNIV // \approx_L$ produces

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

Transitions between Equivalence Classes

$L = \{[c]\}$



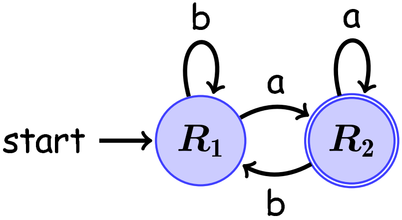
$UNIV // \approx_L$ produces

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

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

Systems of Equations

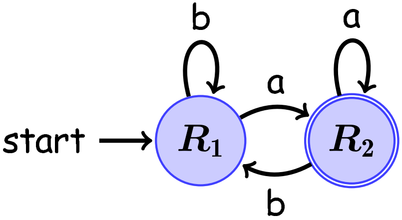
Inspired by a method of Brzozowski '64, we can build an equational system characterising the equivalence classes:



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

Systems of Equations

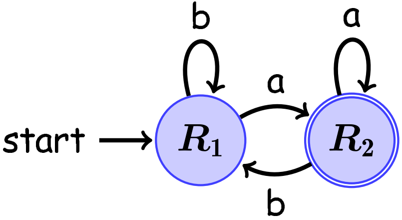
Inspired by a method of Brzozowski '64, we can build an equational system characterising the equivalence classes:



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

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

$$R_1 = R_1; b + R_2; b + \lambda; []$$
$$R_2 = \underbrace{R_1; a}_{\text{something}} + R_2; \underset{\uparrow \atop A}{a}$$

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

by Arden

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

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

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

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

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

solved form !!

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

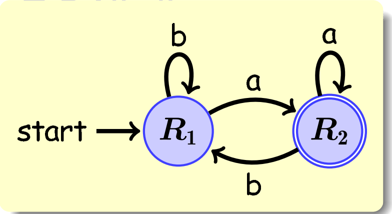
by Arden

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

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

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

solved form

The Equ's Solving Algorithm

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

- This 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

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

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

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

then

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

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 'inversion'

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

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

What Have We Achieved?

- finite $(UNIV // \approx_L) \;\Leftrightarrow\; L$ is regular
- regular languages are closed under 'inversion'

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

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

What Have We Achieved?

- finite $(UNIV // \approx_L)$ $\Leftrightarrow$ L is regular
- regular languages are closed under 'inversion'
$$UNIV // \approx_L \;=\; UNIV // \approx_{-L}$$
- regular expressions are not good if you look for a minimal one of a language (DFA have this notion)
- if you want to do regular expression matching (see Scott's paper)

Conclusion

- on balance regular expression are superior to DFAs
- 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 a lot of fun
- great source of examples