A Formalisation of the Myhill-Nerode Theorem based on Regular Expressions (Proof Pearl)

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

Christian Urban
TU Munich

A Formalisation of the Myhill-Nerode Theorem based on Regular Expressions (Proof Pearl)

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

Christian Urban
TU Munich

Motivation:

I want to teach students with theorem provers (especially for inductions).

Motivation:

I want to teach students with theorem provers (especially for inductions).

- fib, even and odd

Motivation:

I want to teach students with theorem provers (especially for inductions).

- ~~fib, even and odd~~
- formal language theory
  $\Rightarrow$ nice textbooks: Kozen, Hopcroft & Ullman...

in Nuprl

- Constable, Jackson, Naumov, Uribe
- 18 months for automata theory from Hopcroft & Ullman chapters 1–11 (including Myhill-Nerode)

in Coq

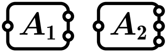
- Filliâtre, Briais, Braibant and others
- multi-year effort; a number of results in automata theory, e.g.
  - Kleene's thm. by Filliâtre ("rather big")
  - automata theory by Briais (5400 loc)
  - Braibant ATBR library, including Myhill-Nerode ($\gg$2000 loc)
  - Mirkin's partial derivative automaton construction (10600 loc)

in HOL

- automata $\Rightarrow$ graphs, matrices, functions

in HOL

- automata $\Rightarrow$ graphs, matrices, functions
- combining automata/graphs



in HOL

- automata ⇒ graphs, matrices, functions
- combining automata/graphs

in HOL

- automata $\Rightarrow$ graphs, matrices, functions
- combining automata/graphs

disjoint union:

$$A_1 \uplus A_2 \stackrel{\text{def}}{=} \{(1, x) \,|\, x \in A_1\} \cup \{(2, y) \,|\, y \in A_2\}$$

in HOL

- automata $\Rightarrow$ graphs, matrices, functions

Problems with definition for regularity (Slind):

$$\text{is\_regular}(A) \stackrel{\text{def}}{=} \exists M.\ \text{is\_dfa}(M) \wedge \mathcal{L}(M) = A$$

$$A_1 \uplus A_2 \stackrel{\text{def}}{=} \{(1, x) \,|\, x \in A_1\} \cup \{(2, y) \,|\, y \in A_2\}$$

in HOL

- automata ⇒ graphs, matrices, functions
- combining automata/graphs

A solution: use nat ⇒ state nodes

in HOL

- automata ⇒ graphs, matrices, functions
- combining automata/graphs

A solution: use nat ⇒ state nodes

You have to rename states!

in HOL

- Kozen's "paper" proof of Myhill-Nerode:
  requires absence of inaccessible states

$$\text{is\_regular}(A) \stackrel{\text{def}}{=} \exists M.\ \text{is\_dfa}(M) \wedge \mathcal{L}(M) = A$$

Definition:

A language A is regular, provided there exists a regular expression that matches all strings of A.

Definition:

A language A is regular, provided there exists a regular expression that matches all strings of A.

. . . and forget about automata

Definition:

A language A is regular, provided there exists a regular expression that matches all strings of A.

. . . and forget about automata

Infrastructure for free. But do we lose anything?

Definition:

A language A is regular, provided there exists a regular expression that matches all strings of A.

. . . and forget about automata

Infrastructure for free. But do we lose anything?

- pumping lemma

Definition:

A language A is regular, provided there exists a regular expression that matches all strings of A.

. . . and forget about automata

Infrastructure for free. But do we lose anything?

- pumping lemma
- closure under complementation

Definition:

A language A is regular, provided there exists a regular expression that matches all strings of A.

... and forget about automata

Infrastructure for free. But do we lose anything?

- pumping lemma
- closure under complementation
- regular expression matching

Definition:

A language A is regular, provided there exists a regular expression that matches all strings of A.

. . . and forget about automata

Infrastructure for free. But do we lose anything?

- pumping lemma
- closure under complementation
- ~~regular expression matching~~ (⇒Owens et al)

Definition:

A language A is regular, provided there exists a regular expression that matches all strings of A.

. . . and forget about automata

Infrastructure for free. But do we lose anything?

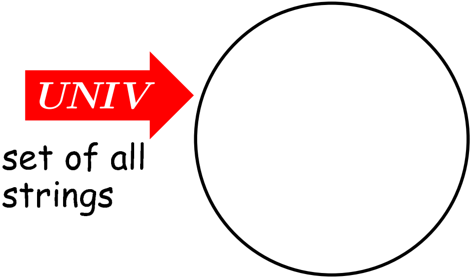
- pumping lemma
- closure under complementation
- ~~regular expression matching~~ ($\Rightarrow$Owens et al)
- most textbooks are about automata

The Myhill-Nerode Theorem

- provides necessary and sufficient conditions for a language being regular (pumping lemma only necessary)
- key is the equivalence relation:

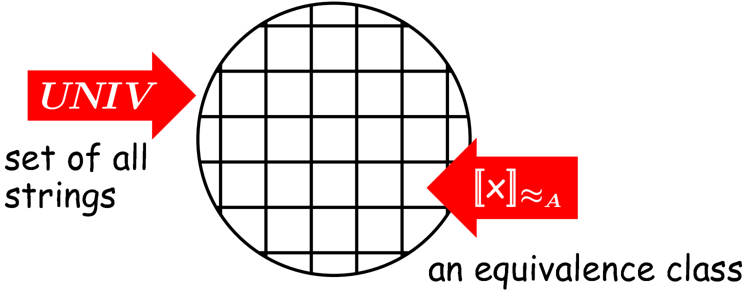
$$x \approx_A y \stackrel{\text{def}}{=} \forall z.\ x@z \in A \Leftrightarrow y@z \in A$$

The Myhill-Nerode Theorem



- finite $(UNIV // \approx_A) \;\Leftrightarrow\; A$ is regular

The Myhill-Nerode Theorem



- finite $(UNIV // \approx_A) \;\Leftrightarrow\; A$ is regular

The Myhill-Nerode Theorem

Two directions:

1.) finite $\Rightarrow$ regular

finite $(UNIV // \approx_A) \Rightarrow \exists r.\ A = \mathcal{L}(r)$

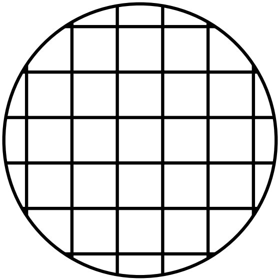
2.) regular $\Rightarrow$ finite

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

- finite $(UNIV // \approx_A) \;\Leftrightarrow\; A$ is regular

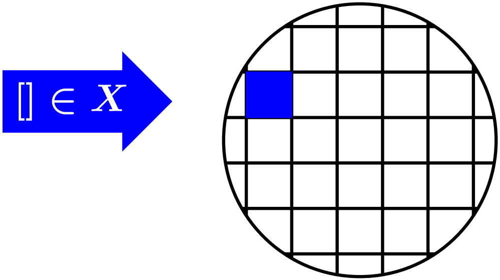
Initial and Final ~~States~~

Equivalence Classes



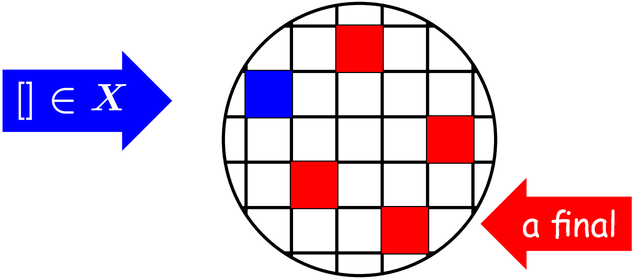
- finals $A \stackrel{\text{def}}{=} \{\|x\|_{\approx_A} \mid x \in A\}$
- we can prove: $A = \bigcup \text{finals}\ A$

Initial and Final ~~States~~ Equivalence Classes



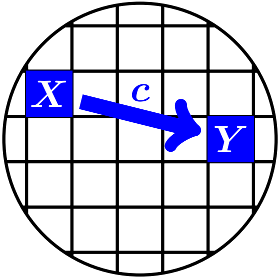
- finals $A \stackrel{\text{def}}{=} \{\|x\|_{\approx_A} \mid x \in A\}$
- we can prove: $A = \bigcup \text{finals } A$

Initial and Final ~~States~~ Equivalence Classes



- finals $A \stackrel{\text{def}}{=} \{\|x\|_{\approx_A} \mid x \in A\}$
- we can prove: $A = \bigcup$ finals A

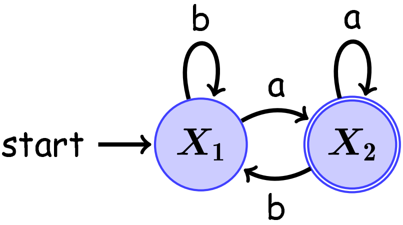
Transitions between Eq-Classes



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

Systems of Equations

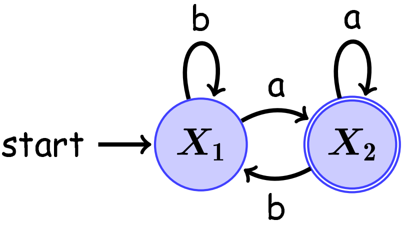
Inspired by a method of Brzozowski '64:



$$X_1 = X_1; b + X_2; b$$
$$X_2 = X_1; a + X_2; a$$

Systems of Equations

Inspired by a method of Brzozowski '64:



$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a + X_2; a$$

$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a + X_2; a$$

$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a + X_2; a$$

by Arden

$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a \cdot a^\star$$

$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a + X_2; a$$

by Arden

$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a \cdot a^\star$$

by Arden

$$X_1 = X_2; b \cdot b^\star + \lambda; b^\star$$
$$X_2 = X_1; a \cdot a^\star$$

$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a + X_2; a$$

by Arden

$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a \cdot a^\star$$

by Arden

$$X_1 = X_2; b \cdot b^\star + \lambda; b^\star$$
$$X_2 = X_1; a \cdot a^\star$$

by substitution

$$X_1 = X_1; a \cdot a^\star \cdot b \cdot b^\star + \lambda; b^\star$$
$$X_2 = X_1; a \cdot a^\star$$

$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a + X_2; a$$

by Arden

$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a \cdot a^\star$$

by Arden

$$X_1 = X_2; b \cdot b^\star + \lambda; b^\star$$
$$X_2 = X_1; a \cdot a^\star$$

by substitution

$$X_1 = X_1; a \cdot a^\star \cdot b \cdot b^\star + \lambda; b^\star$$
$$X_2 = X_1; a \cdot a^\star$$

by Arden

$$X_1 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star$$
$$X_2 = X_1; a \cdot a^\star$$

$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a + X_2; a$$

by Arden

$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a \cdot a^\star$$

by Arden

$$X_1 = X_2; b \cdot b^\star + \lambda; b^\star$$
$$X_2 = X_1; a \cdot a^\star$$

by substitution

$$X_1 = X_1; a \cdot a^\star \cdot b \cdot b^\star + \lambda; b^\star$$
$$X_2 = X_1; a \cdot a^\star$$

by Arden

$$X_1 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star$$
$$X_2 = X_1; a \cdot a^\star$$

by substitution

$$X_1 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star$$
$$X_2 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star \cdot a \cdot a^\star$$

$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a + X_2; a$$

by Arden

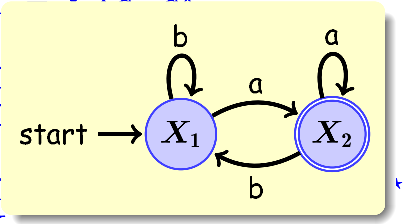
$$X_1 = X_1; b + X_2; b + \lambda; []$$
$$X_2 = X_1; a \cdot a^\star$$

by Arden

X
X

by substitution

X
$$X_2 = X_1; a \cdot a$$



by Arden

$$X_1 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star$$
$$X_2 = X_1; a \cdot a^\star$$

by substitution

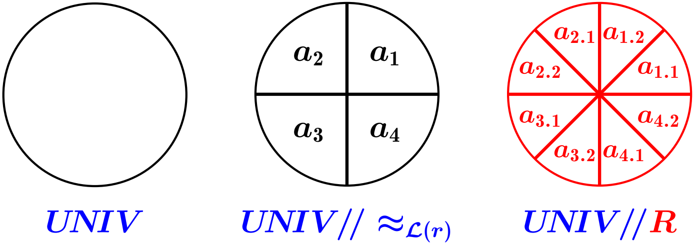
$$X_1 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star$$
$$X_2 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star \cdot a \cdot a^\star$$

The Other Direction

One has to prove

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

by induction on r. Not trivial, but after a bit of thinking, one can find a refined relation:



Partial Derivatives

- ...(set of) regular expressions after a string has been parsed

- $\mathsf{pders}\ x\ r = \mathsf{pders}\ y\ r$ refines $x \approx_{\mathcal{L}(r)} y$

Partial Derivatives

- ...(set of) regular expressions after a string has been parsed

- $\underbrace{\text{pders x r} = \text{pders y r}}_{R}$ refines $\text{x} \approx_{\mathcal{L}(\text{r})} \text{y}$

  ⬇ Antimirov '95

- $\text{finite}(\boldsymbol{UNIV} // \boldsymbol{R})$

Partial Derivatives

- ...(set of) regular expressions after a string has been parsed

- $\underbrace{\text{pders x r} = \text{pders y r}}_{R}$ refines $\text{x} \approx_{\mathcal{L}(\text{r})} \text{y}$

Antimirov '95

- $\text{finite}(UNIV // R)$

- Therefore $\text{finite}(UNIV // \approx_{\mathcal{L}(r)})$. Qed.

What Have We Achieved?

- finite $(UNIV // \approx_A)$ $\Leftrightarrow$ A is regular

What Have We Achieved?

- finite $(UNIV // \approx_A)$ $\Leftrightarrow$ A is regular
- regular languages are closed under complementation; this is now easy

$$UNIV // \approx_A \;=\; UNIV // \approx_{\overline{A}}$$

$$x \approx_A y \overset{\text{def}}{=} \forall z.\; x@z \in A \Leftrightarrow y@z \in A$$

What Have We Achieved?

- finite $(UNIV // \approx_A) \;\Leftrightarrow\; A$ is regular
- regular languages are closed under complementation; this is now easy

$$UNIV // \approx_A \;=\; UNIV // \approx_{\overline{A}}$$

- non-regularity $(a^n b^n)$

If there exists a sufficiently large set B (for example infinitely large), such that

$$\forall x, y \in B.\; x \neq y \;\Rightarrow\; x \not\approx_A y.$$

then A is not regular.

What Have We Achieved?

- finite $(UNIV // \approx_A)$ $\Leftrightarrow$ A is regular
- regular languages are closed under complementation; this is now easy

$$UNIV // \approx_A \;=\; UNIV // \approx_{\overline{A}}$$

- non-regularity $(a^n b^n)$

> If there exists a sufficiently large set B (for example infinitely large), such that
>
> $$\forall x, y \in B.\; x \neq y \;\Rightarrow\; x \not\approx_A y.$$
>
> then A is not regular.

$(B \stackrel{\text{def}}{=} \bigcup_n a^n)$

Conclusion

- We have never seen a proof of Myhill-Nerode based on regular expressions.

Conclusion

- We have never seen a proof of Myhill-Nerode based on regular expressions.
- great source of examples (inductions)

Conclusion

- We have never seen a proof of Myhill-Nerode based on regular expressions.
- great source of examples (inductions)
- no need to fight the theorem prover:
  - first direction (790 loc)
  - second direction (400 / 390 loc)

Conclusion

- We have never seen a proof of Myhill-Nerode based on regular expressions.
- great source of examples (inductions)
- no need to fight the theorem prover:
  - first direction (790 loc)
  - second direction (400 / 390 loc)
- I have **not** yet used it in teaching for undergraduates.

Conclusion

- We have never seen a proof of Myhill-Nerode

Bold Claim: (not proved!)

95% of regular language theory can be done without automata!

...and this is much more tasteful ;o)

- I have **not** yet used it in teaching for undergraduates.

Thank you!

Questions?