

# Formalising Regular Language Theory with Regular Expressions, Only

Christian Urban  
King's College London

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

# Formalising Regular Language Theory with Regular Expressions, **Only**

Christian Urban  
King's College London

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



Roy intertwined with my scientific life on many occasions, most notably:

- he admitted me for M.Phil. in St Andrews and made me like theory
- sent me to Cambridge for Ph.D.
- made me appreciate precision in proofs



Bob Harper  
(CMU)



Frank Pfenning  
(CMU)

published a proof in  
**ACM Transactions on  
Computational Logic**, 2005,  
~31pp



Bob Harper  
(CMU)



Frank Pfenning  
(CMU)

published a proof in  
**ACM Transactions on  
Computational Logic**, 2005,  
~31pp



Andrew Appel  
(Princeton)

relied on their proof in a  
**security** critical application



Bob Harper  
(CMU)



Frank Pfenning  
(CMU)

published a proof in  
**ACM Transactions on  
Computational Logic**, 2005,  
~31pp



Andrew Appel  
(Princeton)

relied on their proof in a  
**security** critical application

(I also found an **error** in my Ph.D.-thesis about cut-elimination  
examined by Henk Barendregt and Andy Pitts.)

Formal language theory...

# in Theorem Provers

e.g. Isabelle, Coq, HOL4, . . .

- automata  $\Rightarrow$  graphs, matrices, functions

Formal language theory...

# in Theorem Provers

e.g. Isabelle, Coq, HOL4, . . .

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



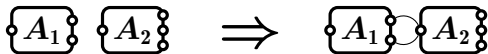


# Formal language theory...

## in Theorem Provers

e.g. Isabelle, Coq, HOL4, . . .

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

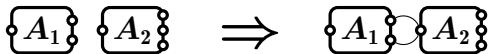


# Formal language theory...

## in Theorem Provers

e.g. Isabelle, Coq, HOL4, ...

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



disjoint union:

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

# Formal language theory...

## in Theorem Provers

e.g. Isabelle, Coq, HOL4, ...

- automata  $\Rightarrow$  graphs, matrices, functions

Problems with definition for regularity:

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

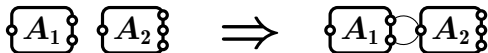
$$A_1 \uplus A_2 \stackrel{\text{def}}{=} \{(1, x) \mid x \in A_1\} \cup \{(2, y) \mid y \in A_2\}$$

# Formal language theory...

## in Theorem Provers

e.g. Isabelle, Coq, HOL4, . . .

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



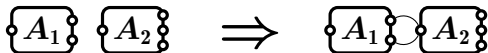
A solution: use `nats`  $\Rightarrow$  state nodes

# Formal language theory...

## in Theorem Provers

e.g. Isabelle, Coq, HOL4, . . .

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



A solution: use **nats**  $\Rightarrow$  state nodes

You have to **rename** states!

# Formal language theory...

## in Theorem Provers

e.g. Isabelle, Coq, HOL4, . . .

- 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~~ ( $\Rightarrow$  Brozowski'64, Owens et al '09)

## 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?

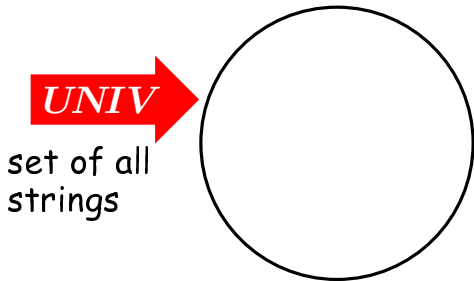
- pumping lemma
- closure under complementation
- ~~regular expression matching~~ ( $\Rightarrow$  Brozowski'64, Owens et al '09)
- most textbooks are about automata

# The Myhill-Nerode Theorem

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

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

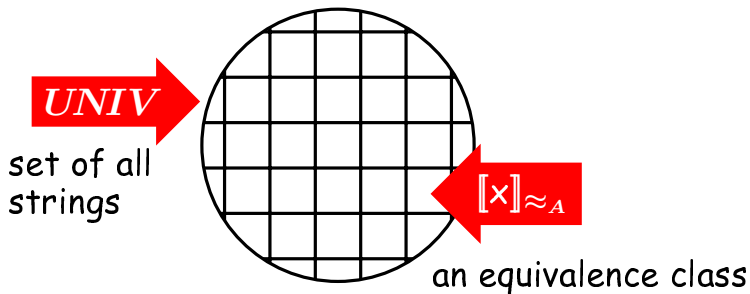
# The Myhill-Nerode Theorem



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



# The Myhill-Nerode Theorem



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

# The Myhill-Nerode Theorem

Two directions:

1.) finite  $\Rightarrow$  regular

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

2.) regular  $\Rightarrow$  finite

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

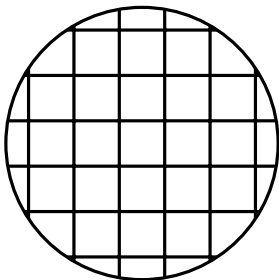


an equivalence class

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

# Initial and Final ~~States~~

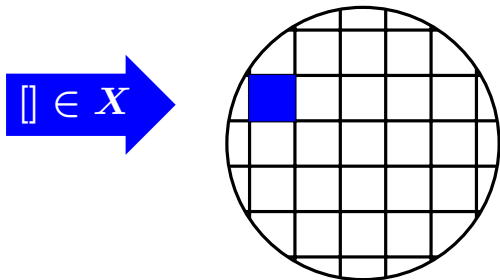
## Equivalence Classes



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

# Initial and Final ~~States~~

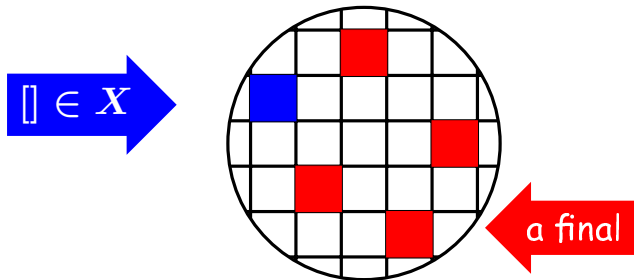
Equivalence Classes



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

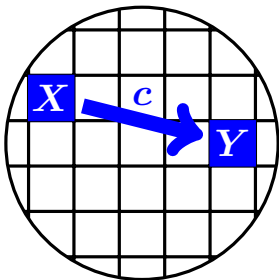
# Initial and Final States

Equivalence Classes



- finals  $A \stackrel{\text{def}}{=} \{[]x \mid x \in A\}$
- we can prove:  $A = \bigcup \text{finals } A$

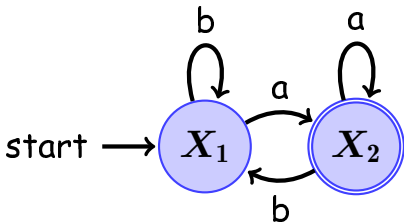
# Transitions between Eq-Classes



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

# Systems of Equations

Inspired by a method of Brzozowski '64:

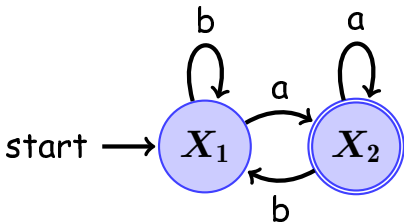


$$X_1 = X_1; b + X_2; b$$

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

# Systems of Equations

Inspired by a method of Brzozowski '64:



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

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





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

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



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

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



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

$$X_2 = X_1; a \cdot a^*$$

by Arden

$$\begin{aligned} X_1 &= X_1; b + X_2; b + \lambda; [] \\ X_2 &= X_1; a + X_2; a \end{aligned}$$

by Arden



$$\begin{aligned} X_1 &= X_1; b + X_2; b + \lambda; [] \\ X_2 &= X_1; a \cdot a^* \end{aligned}$$

by Arden




$$\begin{aligned} X_1 &= X_2; b \cdot b^* + \lambda; b^* \\ X_2 &= X_1; a \cdot a^* \end{aligned}$$


$$\begin{aligned} X_1 &= X_1; b + X_2; b + \lambda; [] \\ X_2 &= X_1; a + X_2; a \end{aligned}$$

by Arden

$$\begin{aligned} X_1 &= X_1; b + X_2; b + \lambda; [] \\ X_2 &= X_1; a \cdot a^* \end{aligned}$$

by Arden


$$\begin{aligned} X_1 &= X_2; b \cdot b^* + \lambda; b^* \\ X_2 &= X_1; a \cdot a^* \end{aligned}$$


$$\begin{aligned} X_1 &= X_1; a \cdot a^* \cdot b \cdot b^* + \lambda; b^* \\ X_2 &= X_1; a \cdot a^* \end{aligned}$$

by substitution

$$\begin{aligned} X_1 &= X_1; b + X_2; b + \lambda; [] \\ X_2 &= X_1; a + X_2; a \end{aligned}$$

by Arden

$$\begin{aligned} X_1 &= X_1; b + X_2; b + \lambda; [] \\ X_2 &= X_1; a \cdot a^* \end{aligned}$$

by Arden

$$\begin{aligned} X_1 &= X_2; b \cdot b^* + \lambda; b^* \\ X_2 &= X_1; a \cdot a^* \end{aligned}$$

by substitution



$$\begin{aligned} X_1 &= X_1; a \cdot a^* \cdot b \cdot b^* + \lambda; b^* \\ X_2 &= X_1; a \cdot a^* \end{aligned}$$

by Arden



$$\begin{aligned} X_1 &= \lambda; b^* \cdot (a \cdot a^* \cdot b \cdot b^*)^* \\ X_2 &= X_1; a \cdot a^* \end{aligned}$$

$$\begin{aligned} X_1 &= X_1; b + X_2; b + \lambda; [] \\ X_2 &= X_1; a + X_2; a \end{aligned}$$

by Arden

$$\begin{aligned} X_1 &= X_1; b + X_2; b + \lambda; [] \\ X_2 &= X_1; a \cdot a^* \end{aligned}$$

by Arden

$$\begin{aligned} X_1 &= X_2; b \cdot b^* + \lambda; b^* \\ X_2 &= X_1; a \cdot a^* \end{aligned}$$

by substitution

$$\begin{aligned} X_1 &= X_1; a \cdot a^* \cdot b \cdot b^* + \lambda; b^* \\ X_2 &= X_1; a \cdot a^* \end{aligned}$$

by Arden



$$\begin{aligned} X_1 &= \lambda; b^* \cdot (a \cdot a^* \cdot b \cdot b^*)^* \\ X_2 &= X_1; a \cdot a^* \end{aligned}$$

by substitution

$$\begin{aligned} X_1 &= \lambda; b^* \cdot (a \cdot a^* \cdot b \cdot b^*)^* \\ X_2 &= \lambda; b^* \cdot (a \cdot a^* \cdot b \cdot b^*)^* \cdot a \cdot a^* \end{aligned}$$

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

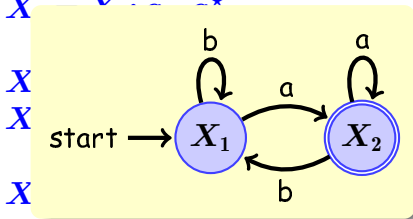
$$X_2 = X_1; a + X_2; a$$

by Arden

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

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

by Arden



by substitution

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

$$X_2 = X_1; a \cdot a^*$$

by Arden

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

$$X_2 = X_1; a \cdot a^*$$

by substitution

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

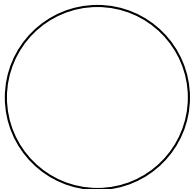
$$X_2 = \lambda; b^* \cdot (a \cdot a^* \cdot b \cdot b^*)^* \cdot a \cdot a^*$$

# The Other Direction

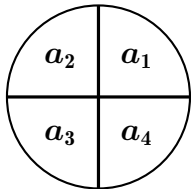
One has to prove

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

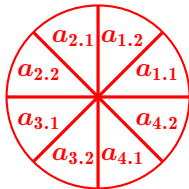
by induction on  $r$ . Not trivial, but after a bit of thinking, one can find a **refined** relation:



$UNIV$



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



$UNIV// R$



# Derivatives of RExps

- introduced by Brozowski '64
- a regular expressions after a character has been parsed

$$\text{der } c \ \emptyset \stackrel{\text{def}}{=} \emptyset$$

$$\text{der } c \ [] \stackrel{\text{def}}{=} \emptyset$$

$$\text{der } c \ d \stackrel{\text{def}}{=} \text{if } c = d \text{ then } [] \text{ else } \emptyset$$

$$\text{der } c \ (r_1 + r_2) \stackrel{\text{def}}{=} (\text{der } c \ r_1) + (\text{der } c \ r_2)$$

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

$$\text{der } c \ (r_1 \cdot r_2) \stackrel{\text{def}}{=} \begin{array}{l} \text{if nullable } r_1 \\ \text{then } (\text{der } c \ r_1) \cdot r_2 + (\text{der } c \ r_2) \\ \text{else } (\text{der } c \ r_1) \cdot r_2 \end{array}$$

# Derivatives of RExps

- introduced by Brozowski '64
- a regular expressions after a character has been parsed

- partial derivatives
- by Antimirov '95

$\text{pder } c \ \emptyset$

$\stackrel{\text{def}}{=} \{\}$

$\text{pder } c \ [ ]$

$\stackrel{\text{def}}{=} \{\}$

$\text{pder } c \ d$

$\stackrel{\text{def}}{=} \text{if } c = d \text{ then } \{ [ ] \} \text{ else } \{\}$

$\text{pder } c \ (r_1 + r_2)$

$\stackrel{\text{def}}{=} (\text{pder } c \ r_1) \cup (\text{der } c \ r_2)$

$\text{pder } c \ (r^*)$

$\stackrel{\text{def}}{=} (\text{pder } c \ r) \cdot r^*$

$\text{pder } c \ (r_1 \cdot r_2)$

$\stackrel{\text{def}}{=} \text{if nullable } r_1$   
then  $(\text{pder } c \ r_1) \cdot r_2 \cup (\text{pder } c \ r_2)$   
else  $(\text{pder } c \ r_1) \cdot r_2$

# Partial Derivatives

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

# Partial Derivatives

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



Antimirov '95

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

# Partial Derivatives

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



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

$$UNIV // \approx_A = UNIV // \approx_{\bar{A}}$$

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

# What Have We Achieved?

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

$$UNIV // \approx_A = UNIV // \approx_{\bar{A}}$$

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



# What Have We Achieved?

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

- regular languages are closed under complementation; this is now easy

$$UNIV // \approx_A = UNIV // \approx_{\bar{A}}$$

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

# What Have We Achieved?

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

$$UNIV // \approx_A = UNIV // \approx_{\bar{A}}$$

- non-regularity ( $a^n b^n$ )
- take **any** language; build the language of substrings

# What Have We Achieved?

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

- regular languages are closed under complementation; this is now easy

$$UNIV // \approx_A = UNIV // \approx_{\bar{A}}$$

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

- take **any** language; build the language of substrings

then this language **is** regular ( $a^n b^n \Rightarrow a^* b^*$ )

# 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)

**Thank you!**  
**Questions?**