The Myhill-Nerode Theorem in a Theorem Prover

Christian Urban
King's College London

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

The Myhill-Nerode Theorem in a Theorem Prover

Isabelle/HOL

Christian Urban
King's College London

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

- my background is in
  - programming languages and theorem provers
  - develop Nominal Isabelle

- to formalise and mechanically check proofs from programming language research, TCS and OS

- my background is in
  - programming languages and theorem provers
  - develop Nominal Isabelle

- to formalise and mechanically check proofs from programming language research, TCS and OS

- we found out that the variable convention can lead to faulty proofs...

Variable Convention:
If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables. Henk Barendregt

Bob Harper
(CMU)

Frank Pfenning
(CMU)

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

Bob Harper
(CMU)

Frank Pfenning
(CMU)

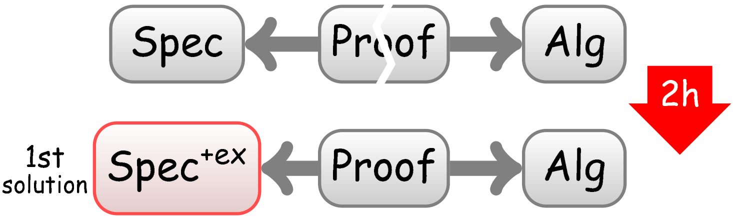
published a proof on LF in **ACM Transactions on Computational Logic**, 2005, ∼31pp

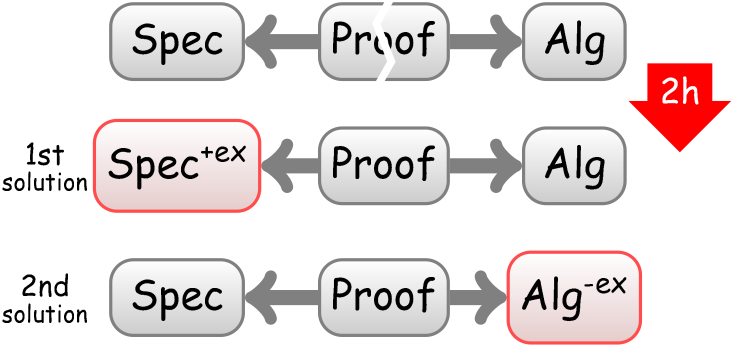
Andrew Appel
(Princeton)

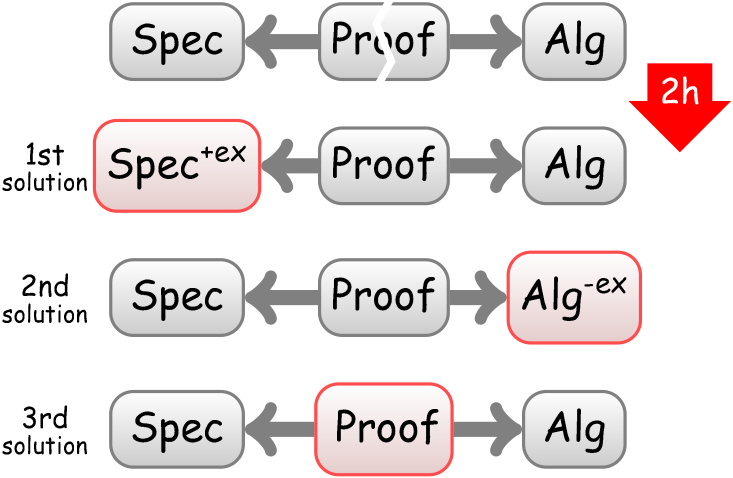
relied on their proof in a **security** critical application (proof-carrying code)

$$\boxed{\text{Spec}} \longleftarrow \boxed{\text{Proof}} \longrightarrow \boxed{\text{Alg}}$$


Spec
Proof
Alg


Spec
Proof
Alg
2h
1st solution
Spec+ex
Proof
Alg


Spec
Proof
Alg
2h
1st solution
Spec+ex
Proof
Alg
2nd solution
Spec
Proof
Alg-ex


Spec
Proof
Alg
2h
1st solution
Spec+ex
Proof
Alg
2nd solution
Spec
Proof
Alg-ex
3rd solution
Spec
Proof
Alg

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

- found flaws in a proof about a classic OS scheduling algorithm — helped us to implement it correctly and efficiently

  (the existing literature “proved” correct an incorrect algorithm; used in the Mars Pathfinder mission)

- I also found fixable errors in my Ph.D.-thesis about cut-elimination (examined by Henk Barendregt and Andy Pitts)
- found flaws in a proof about a classic OS scheduling algorithm — helped us to implement it correctly and efficiently

  (the existing literature "proved" correct an incorrect algorithm; used in the Mars Pathfinder mission)

Conclusion:

Pencil-and-paper proofs in TCS are not foolproof, not even expertproof.

Scott Aaronson (Berkeley/MIT):

"I still remember having to grade hundreds of exams where the students started out by assuming what had to be proved, or filled page after page with gibberish in the hope that, somewhere in the mess, they might accidentally have said something correct. ... innumerable examples of "parrot proofs" — NP-completeness reductions done in the wrong direction, arguments that look more like LSD trips than coherent chains of logic ..."

Scott Aaronson (Berkeley/MIT):

> "I still remember having to grade hundreds of exams where the students started out by assuming what had to be proved, or filled page after page with gibberish in the hope that, somewhere in the mess, they might accidentally have said something correct. ...innumerable examples of "parrot proofs" — NP-completeness reductions done in the wrong direction, arguments that look more like LSD trips than coherent chains of logic ..."

Tobias Nipkow calls this the "London Underground Phenomenon":

students MIND THE GAP proofs

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:

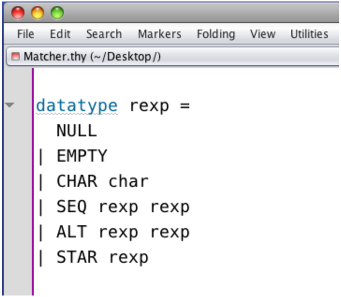
I want to teach students with theorem provers (especially for inductions).

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

Regular Expressions

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

Isabelle:



students have seen them and
can be motivated about them

$$
\begin{aligned}
\mathit{nullable}(\varnothing) &= \mathit{false}\\
\mathit{nullable}([]) &= \mathit{true}\\
\mathit{nullable}(c) &= \mathit{false}\\
\mathit{nullable}(r_1 + r_2) &= (\mathit{nullable}\ r_1) \vee (\mathit{nullable}\ r_2)\\
\mathit{nullable}(r_1 \cdot r_2) &= (\mathit{nullable}\ r_1) \wedge (\mathit{nullable}\ r_2)\\
\mathit{nullable}(r^*) &= \mathit{true}
\end{aligned}
$$

$$
\begin{array}{lcl}
\mathit{nullable}\,(\varnothing) & = & \mathit{false}\\
\mathit{nullable}\,([]) & = & \mathit{true}\\
\mathit{nullable}\,(c) & = & \mathit{false}\\
\mathit{nullable}\,(r_1 + r_2) & = & (\mathit{nullable}\ r_1) \vee (\mathit{nullable}\ r_2)\\
\mathit{nullable}\,(r_1 \cdot r_2) & = & (\mathit{nullable}\ r_1) \wedge (\mathit{nullable}\ r_2)\\
\mathit{nullable}\,(r^*) & = & \mathit{true}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{der}\ c\ (\varnothing) & = & \varnothing\\
\mathit{der}\ c\ ([]) & = & \varnothing\\
\mathit{der}\ c\ (d) & = & \mathit{if}\ c = d\ \mathit{then}\ []\ \mathit{else}\ \varnothing\\
\mathit{der}\ c\ (r_1 + r_2) & = & (\mathit{der}\ c\ r_1) + (\mathit{der}\ c\ r_2)\\
\mathit{der}\ c\ (r_1 \cdot r_2) & = & ((\mathit{der}\ c\ r_1) \cdot r_2) +\\
 & & \quad (\mathit{if}\ \mathit{nullable}\ r_1\ \mathit{then}\ \mathit{der}\ c\ r_2\ \mathit{else}\ \varnothing)\\
\mathit{der}\ c\ (r^*) & = & (\mathit{der}\ c\ r) \cdot (r^*)
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{nullable}\,(\varnothing) & = & \mathit{false} \\
\mathit{nullable}\,([]) & = & \mathit{true} \\
\mathit{nullable}\,(c) & = & \mathit{false} \\
\mathit{nullable}\,(r_1 + r_2) & = & (\mathit{nullable}\,r_1) \vee (\mathit{nullable}\,r_2) \\
\mathit{nullable}\,(r_1 \cdot r_2) & = & (\mathit{nullable}\,r_1) \wedge (\mathit{nullable}\,r_2) \\
\mathit{nullable}\,(r^*) & = & \mathit{true}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{der}\,c\,(\varnothing) & = & \varnothing \\
\mathit{der}\,c\,([]) & = & \varnothing \\
\mathit{der}\,c\,(d) & = & \mathit{if}\ c = d\ \mathit{then}\ []\ \mathit{else}\ \varnothing \\
\mathit{der}\,c\,(r_1 + r_2) & = & (\mathit{der}\,c\,r_1) + (\mathit{der}\,c\,r_2) \\
\mathit{der}\,c\,(r_1 \cdot r_2) & = & ((\mathit{der}\,c\,r_1) \cdot r_2) + \\
 & & \quad (\mathit{if}\ \mathit{nullable}\,r_1\ \mathit{then}\ \mathit{der}\,c\,r_2\ \mathit{else}\ \varnothing) \\
\mathit{der}\,c\,(r^*) & = & (\mathit{der}\,c\,r) \cdot (r^*)
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{derivative}\,[]\,r & = & r \\
\mathit{derivative}\,(c{::}s)\,r & = & \mathit{derivative}\,s\,(\mathit{der}\,c\,r)
\end{array}
$$

$$
\mathit{matches}\,r\,s = \mathit{nullable}\,(\mathit{derivative}\,s\,r)
$$

Regular Expression Matching in Education

- Harper in JFP'99: "Functional Pearl: Proof-Directed Debugging"
- Yi in JFP'06: "Educational Pearl: 'Proof-Directed Debugging' revisited for a first-order version"

Regular Expression Matching in Education

- 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."

Formal language theory...

in Theorem Provers

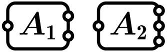
e.g. Isabelle, Coq, HOL4, . . .

- automata $\Rightarrow$ graphs, matrices, functions

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:

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

in Theorem Provers

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

- automata $\Rightarrow$ graphs, matrices, functions

Problems with definition for regularity:

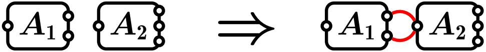
$$\text{is\_regular}(A) \overset{\text{def}}{=} \exists M.\ \text{is\_dfa}(M) \wedge \mathcal{L}(M) = A$$

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

in Theorem Provers

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

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

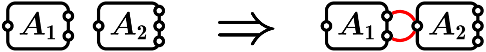


<u>A solution</u>: use nats $\Rightarrow$ state nodes

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!

in Theorem Provers

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

- Kozen's paper-proof of Myhill-Nerode: requires absence of inaccessible states
- complementation of automata only works for complete automata (need sink 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~~ (⇒Brzozowski'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~~ (⇒Brzozowski'64, Owens et al '09)
- most textbooks are about automata

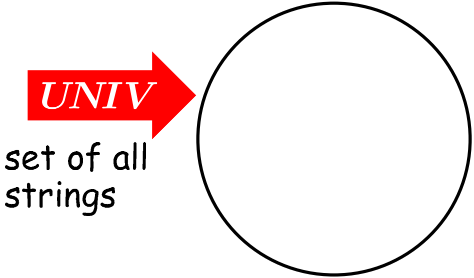
The Myhill-Nerode Theorem

- provides necessary and sufficient conditions for a language being regular
  (pumping lemma only necessary)

- key is the equivalence relation:

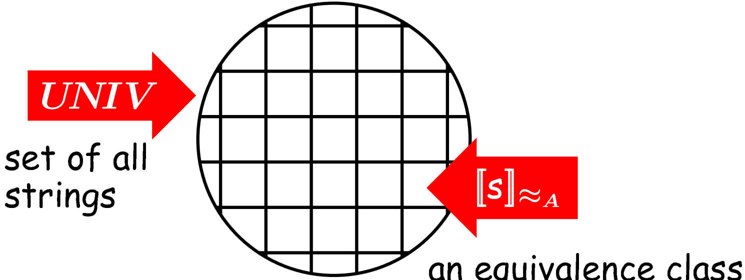
$$x \approx_A y \stackrel{\text{def}}{=} \forall z.\ x@z \in A \Leftrightarrow y@z \in A$$

The Myhill-Nerode Theorem



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

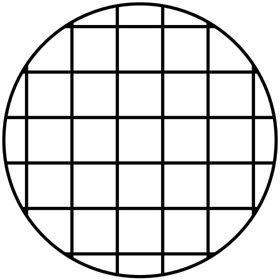
2.) regular $\Rightarrow$ finite

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

an equivalence class

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

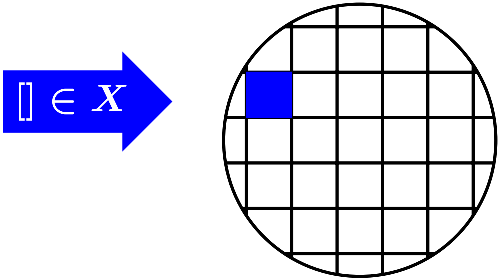
Initial and Final ~~States~~ Equivalence Classes



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

Initial and Final ~~States~~

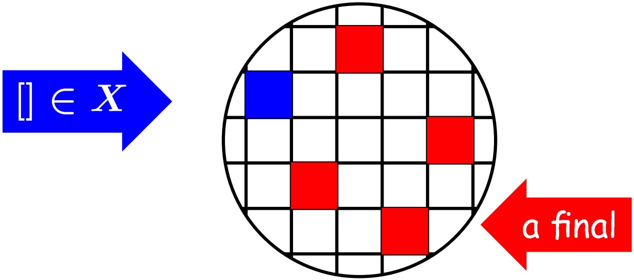
Equivalence Classes



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

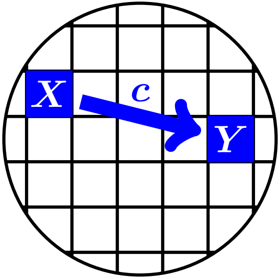
Initial and Final ~~States~~

Equivalence Classes



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

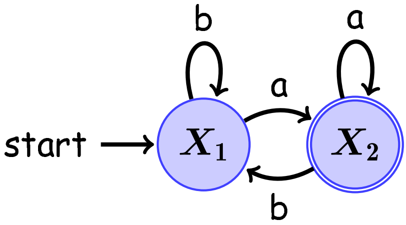
Transitions between Eq-Classes



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

Systems of Equations

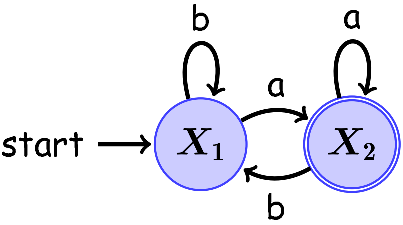
Inspired by a method of Brzozowski '64:



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

Systems of Equations

Inspired by a method of Brzozowski '64:



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

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

A Variant of Arden's Lemma

Arden's Lemma:

If $[] \not\in A$ then

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

has the (unique) solution

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

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

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

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

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

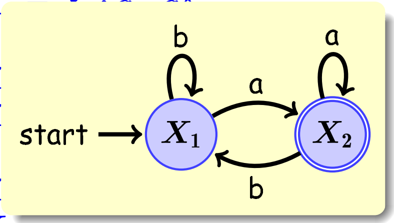
by substitution

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

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

by Arden

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



by Arden

by substitution

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

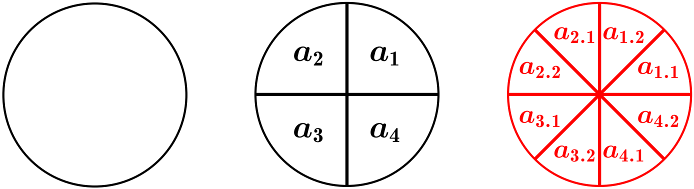
$$X_1 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star$$
$$X_2 = \lambda; b^\star \cdot (a \cdot a^\star \cdot b \cdot b^\star)^\star \cdot a \cdot a^\star$$

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:



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

Derivatives of RExps

- introduced by Brzozowski '64
- produces a regular expression after a character has been "parsed"

$$
\begin{array}{lcl}
\text{der } c\ \varnothing & \stackrel{\text{def}}{=} & \varnothing \\
\text{der } c\ [] & \stackrel{\text{def}}{=} & \varnothing \\
\text{der } c\ d & \stackrel{\text{def}}{=} & \text{if } c = d \text{ then } [] \text{ else } \varnothing \\
\text{der } c\ (r_1 + r_2) & \stackrel{\text{def}}{=} & (\text{der } c\ r_1) + (\text{der } c\ r_2) \\
\text{der } c\ (r^*) & \stackrel{\text{def}}{=} & (\text{der } c\ r) \cdot (r^*) \\
\text{der } c\ (r_1 \cdot r_2) & \stackrel{\text{def}}{=} & ((\text{der } c\ r_1) \cdot r_2) + \\
 & & (\text{if nullable } r_1 \text{ then der } c\ r_2 \text{ else } \varnothing)
\end{array}
$$

Derivatives of RExps

- introduced by Brzozowski '64
- produces a regular expression after a character has been "parsed"

derivatives refine $x \approx_{\mathcal{L}(r)} y$

$$\mathcal{L}(\text{ders}\ x\ r) = \mathcal{L}(\text{ders}\ y\ r) \iff x \approx_{\mathcal{L}(r)} y$$

$\text{finite}(\text{ders}\ A\ r)$, but only modulo ACI

$$\begin{array}{rcl}(r_1 + r_2) + r_3 & \equiv & r_1 + (r_2 + r_3)\\ r_1 + r_2 & \equiv & r_2 + r_1\\ r + r & \equiv & r\end{array}$$

$\varnothing)$

Derivatives of RExps

- introduced by Brzozowski '64
- produces a regular expression after a character has been "parsed"

derivatives refine $x \approx_{\mathcal{L}(r)} y$

$$\text{ders}\ x\ r = \text{ders}\ y\ r \Longrightarrow x \approx_{\mathcal{L}(r)} y$$

$\text{finite}(\text{ders}\ A\ r)$, but only modulo ACI

$$\begin{array}{rcl} (r_1 + r_2) + r_3 & \equiv & r_1 + (r_2 + r_3) \\ r_1 + r_2 & \equiv & r_2 + r_1 \\ r + r & \equiv & r \end{array}$$

$\varnothing)$

Partial Derivatives of RExps

- partial derivatives
- by Antimirov '95

$$\begin{array}{lcl}
\text{pder c } \varnothing & \stackrel{\text{def}}{=} & \{\} \\
\text{pder c } [] & \stackrel{\text{def}}{=} & \{\} \\
\text{pder c d} & \stackrel{\text{def}}{=} & \text{if c} = \text{d then } \{[]\} \text{ else } \{\} \\
\text{pder c } (r_1 + r_2) & \stackrel{\text{def}}{=} & (\text{pder c } r_1) \cup (\text{der c } r_2) \\
\text{pder c } (r^\star) & \stackrel{\text{def}}{=} & (\text{pder c } r) \cdot r^\star \\
\text{pder c } (r_1 \cdot r_2) & \stackrel{\text{def}}{=} & (\text{pder c } r_1) \cdot r_2 \cup \\
 & & \text{if nullable } r_1 \text{ then } (\text{pder c } r_2) \text{ else } \varnothing
\end{array}$$

Partial Derivatives

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

Partial Derivatives

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

$$\Downarrow$$ Antimirov '95

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

Partial Derivatives

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

$\Downarrow$ Antimirov '95

- $\text{finite}(\mathit{UNIV} // R)$
- Therefore $\text{finite}(\mathit{UNIV} // \approx_{\mathcal{L}(r)})$. 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
$$UNIV // \approx_A \;=\; UNIV // \approx_{\overline{A}}$$

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

What Have We Achieved?

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

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

- 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

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

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

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 $(a^n b^n)$
- 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
$$UNIV // \approx_A \;=\; UNIV // \approx_{\overline{A}}$$
- non-regularity $(a^n b^n)$
- take **any** language
build the language of substrings
then this language **is** regular $\;(a^n b^n \Rightarrow a^\star b^\star)$

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 ($>$7000 loc)
  - Mirkin's partial derivative automaton construction (10600 loc)

Conclusion

- we have never seen a proof of Myhill-Nerode based on regular expressions only

Conclusion

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

Conclusion

- we have never seen a proof of Myhill-Nerode based on regular expressions only
- great source of examples (inductions)
- no need to fight the theorem prover:
  - first direction (790 loc)
  - second direction (400 / 390 loc)
- I am not saying automata are bad; just formal proofs about them are quite difficult

Conclusion

- we have never seen a proof of Myhill-Nerode based on regular expressions only
- great source of examples (inductions)
- no need to fight the theorem prover:
  - first direction (790 loc)
  - second direction (400 / 390 loc)
- I am not saying automata are bad; just formal proofs about them are quite difficult

- parsing with derivatives of grammars (Matt Might ICFP'11)

An Apology

- This should all of course be done co-inductively

From: Jasmin Christian Blanchette
To: isabelle-dev@mailbroy.informatik.tu-muenchen.de
Subject: [isabelle-dev] NEWS
Date: Tue, 28 Aug 2012 17:40:55 +0200

* **HOL/Codatatype**: New (co)datatype package with support for mixed, nested recursion and interesting non-free datatypes.

* HOL/Ordinals_and_Cardinals: Theories of ordinals and cardinals (supersedes the AFP entry of the same name).

Kudos to Andrei and Dmitriy!

Jasmin
____
isabelle-dev mailing list
isabelle-dev@in.tum.de

Thank you very much!

Questions?