The Myhill-Nerode Theorem
Based on Regular Expressions

Chunhan Wu, Xingyuan Zhang and Christian Urban

June 11, 2019

Abstract
There are many proofs of the Myhill-Nerode theorem using automata. In this library we give a proof entirely based on regular expressions, since regularity of languages can be conveniently defined using regular expressions (it is more painful in HOL to define regularity in terms of automata). We prove the first direction of the Myhill-Nerode theorem by solving equational systems that involve regular expressions. For the second direction we give two proofs: one using tagging-functions and another using partial derivatives. We also establish various closure properties of regular languages.¹

Contents

1 “Summation” for regular expressions 2

2 First direction of MN: finite partition ⇒ regular language 3
  2.1 Equational systems .............................................. 4
  2.2 Arden Operation on equations ................................. 5
  2.3 Substitution Operation on equations .......................... 5
  2.4 While-combinator and invariants .............................. 5
  2.5 Initial Equational Systems ................................. 7
  2.6 Iterations ............................................. 8
  2.7 The conclusion of the first direction ..................... 10

3 Second direction of MN: regular language ⇒ finite partition 11
  3.1 Tagging functions .............................................. 11
  3.2 Base cases: Zero, One and Atom ............................. 12
  3.3 Case for Plus ............................................. 13
  3.4 Case for Times ............................................. 13
  3.5 Case for Star ............................................. 14
  3.6 The conclusion of the second direction .................. 15

¹Most details of the theories are described in the paper [2].
1 “Summation” for regular expressions

To obtain equational system out of finite set of equivalence classes, a fold operation on finite sets folds is defined. The use of SOME makes folds more robust than the fold in the Isabelle library. The expression folds f makes sense when f is not associative and commutitive, while fold f does not.

definition
folds :: ('a ⇒ 'b ⇒ 'b) ⇒ 'b ⇒ 'a set ⇒ 'b
where
folds f z S ≡ SOME x. fold-graph f z S x

Plus-combination for a set of regular expressions

abbreviation
Setalt :: 'a rexp set ⇒ 'a rexp (⨄ - [1000] 999)
where
⨄ A ≡ folds Plus Zero A
For finite sets, $\text{Setalt}$ is preserved under $\text{lang}$.

**lemma** `folds-plus-simp [simp]`:
- **fixes** $rs :: (\text{\'a expr})\ \text{set}$
- **assumes** $a :: \text{finite} \ rs$
- **shows** $\text{lang} (\bigcup rs) = \bigcup (\text{lang} \ rs)$

`⟨proof⟩` end

**theory** `Myhill-1`
**imports** `Folds`
`HOL-Library.\ While-Combinator`

begin

2 First direction of MN: finite partition $\Rightarrow$ regular language

**notation**
- `conc (infixr \cdot 100)` and
- `star (-\star [101] 102)`

**lemma** `Pair-Collect [simp]`:
- **shows** $(x, y) \in \{(x, y). \ P \ x \ y\} \longleftrightarrow P \ x \ y$

`⟨proof⟩` end

Myhill-Nerode relation

**definition**
- `str-eq :: \'a lang $\Rightarrow$ (\'a list $\times$ \'a list) set (\approx [100] 100)`
- **where**
  
  $\approx A \equiv \{(x, y). (\forall z. x @ z \in A \longleftrightarrow y @ z \in A)\}$

**abbreviation**
- `str-eq-applied :: \'a list $\Rightarrow$ \'a lang $\Rightarrow$ \'a list $\Rightarrow$ bool (- $\approx$ -)`
- **where**
  
  $x \approx A \ y \equiv (x, y) \in \approx A$

**lemma** `str-eq-conv-Derivs`:
- `str-eq A = \{(u,v). \text{Deriv} u A = \text{Deriv} v A\}`

`⟨proof⟩` end

**definition**
- `finals :: \'a lang $\Rightarrow$ \'a lang set`
- **where**
  
  $\text{finals} A \equiv \{\approx A ^{"s"} \{s\} | s . s \in A\}$

**lemma** `lang-is-union-of-finals`:
- **shows** $A = \bigcup (\text{finals} A)$

`⟨proof⟩` end
 lemma finals-in-partitions:
  shows finals A ⊆ (UNIV // ≈A)
 ⟨proof⟩

2.1 Equational systems

The two kinds of terms in the rhs of equations.

datatype 'a trm =
  Lam 'a rexp
 | Trn 'a lang 'a rexp

fun lang-trm::'a trm ⇒ 'a lang
where
  lang-trm (Lam r) = lang r
 | lang-trm (Trn X r) = X · lang r

fun lang-rhs::('a trm) set ⇒ 'a lang
where
  lang-rhs rhs = ∪ (lang-trm ' rhs)

lemma lang-rhs-set:
  shows lang-rhs \{Trn X r | r. P r\} = ∪\{lang-trm (Trn X r) | r. P r\}
 ⟨proof⟩

lemma lang-rhs-union-distrib:
  shows lang-rhs A ∪ lang-rhs B = lang-rhs (A ∪ B)
 ⟨proof⟩

   Transitions between equivalence classes

definition transition :: 'a lang ⇒ 'a ⇒ 'a lang ⇒ bool (- |-⇒- [100,100,100] 100)
where
  Y |-c⇒ X ≡ Y · {[c]} ⊆ X

   Initial equational system

definition Init-rhs CS X ≡
  if ([] ∈ X) then
    \{Lam One\} ∪ \{Trn Y (Atom c) | Y c. Y ∈ CS ∧ Y |-c⇒ X\}
  else
    \{Trn Y (Atom c) | Y c. Y ∈ CS ∧ Y |-c⇒ X\}

definition Init CS ≡ \{(X, Init-rhs CS X) | X. X ∈ CS\}

4
2.2 Arden Operation on equations

fun
  Append-rexp :: 'a rexp ⇒ 'a trm ⇒ 'a trm
where
  Append-rexp r (Lam rexp) = Lam (Times rexp r)
  | Append-rexp r (Trn X rexp) = Trn X (Times rexp r)

definition
  Append-rexp-rhs rhs rexp ≡ (Append-rexp rexp) ' rhs

definition
  Arden X rhs ≡
  Append-rexp-rhs (rhs − { Trn X r | r. Trn X r ∈ rhs }) (Star (⨄ { r. Trn X r ∈ rhs }))

2.3 Substitution Operation on equations

definition
  Subst rhs X xrhs ≡
  (rhs − { Trn X r | r. Trn X r ∈ rhs }) ∪ (Append-rexp-rhs xrhs (⨄ { r. Trn X r ∈ rhs }))

definition
  Subst-all :: ('a lang × ('a trm) set) set ⇒ 'a lang ⇒ ('a trm × ('a trm) set) set
where
  Subst-all ES X xrhs ≡
  { (Y, Subst yrhs X xrhs) | Y yrhs. (Y, yrhs) ∈ ES }

definition
  Remove ES X xrhs ≡
  Subst-all (ES − {(X, xrhs)}) X (Arden X xrhs)

2.4 While-combinator and invariants

definition
  Iter X ES ≡ (let (Y, yrhs) = SOME (Y, yrhs). (Y, yrhs) ∈ ES ∧ X ≠ Y in Remove ES Y yrhs)

lemma IterI2:
  assumes (Y, yrhs) ∈ ES
  and X ≠ Y
  and ∃ Y yrhs. [(Y, yrhs) ∈ ES; X ≠ Y] ⇒ Q (Remove ES Y yrhs)
  shows Q (Iter X ES)
⟨proof⟩

abbreviation
  Cond ES ≡ card ES ≠ 1
\textbf{definition} \hfill \textit{Solve} \ X \ \textit{ES} \equiv \text{while} \ \textit{Cond} \ (\textit{Iter} \ X) \ \textit{ES}

\textbf{definition} \hfill \textit{distinctness} \ \textit{ES} \equiv \\
\forall \ X \ \textit{rhs} \ \textit{rhs}'  . \ (X, \ \textit{rhs}) \in \ \textit{ES} \land (X, \ \textit{rhs}') \in \ \textit{ES} \rightarrow \ \textit{rhs} = \ \textit{rhs}'

\textbf{definition} \hfill \textit{soundness} \ \textit{ES} \equiv \forall (X, \ \textit{rhs}) \in \ \textit{ES}. \ X \ = \ \textit{lang-rhs} \ \textit{rhs}

\textbf{definition} \hfill \textit{ardenable} \ \textit{rhs} \equiv (\forall \ Y \ r. \ \textit{Trn} \ Y \ r \ \in \ \textit{rhs} \rightarrow \ [] \notin \ \textit{lang} \ r)

\textbf{definition} \hfill \textit{ardenable-all} \ \textit{ES} \equiv \forall (X, \ \textit{rhs}) \in \ \textit{ES}. \ \textit{ardenable} \ \textit{rhs}

\textbf{definition} \hfill \textit{finite-rhs} \ \textit{ES} \equiv \forall (X, \ \textit{rhs}) \in \ \textit{ES}. \ \textit{finite} \ \textit{rhs}

\textbf{lemma} \ \textit{finite-rhs-def2}:
\textit{finite-rhs} \ \textit{ES} \equiv (\forall \ X \ \textit{rhs}. \ (X, \ \textit{rhs}) \in \ \textit{ES} \rightarrow \ \textit{finite} \ \textit{rhs})
\textit{proof}

\textbf{definition} \hfill \textit{rhss} \ \textit{rhs} \equiv \{X \mid X \ r. \ \textit{Trn} \ X \ r \ \in \ \textit{rhs}\}

\textbf{definition} \hfill \textit{lhss} \ \textit{ES} \equiv \{Y \mid Y \ \textit{yrhs}. \ (Y, \ \textit{yrhs}) \in \ \textit{ES}\}

\textbf{definition} \hfill \textit{validity} \ \textit{ES} \equiv \forall (X, \ \textit{rhs}) \in \ \textit{ES}. \ \textit{rhss} \ \textit{rhs} \subseteq \ \textit{lhss} \ \textit{ES}

\textbf{lemma} \ \textit{rhss-union-distrib}:
\textbf{shows} \ \textit{rhss} (A \cup B) = \textit{rhss} A \cup \textit{rhss} B
\textit{proof}

\textbf{lemma} \ \textit{lhss-union-distrib}:
\textbf{shows} \ \textit{lhss} (A \cup B) = \textit{lhss} A \cup \textit{lhss} B
\textit{proof}

\textbf{definition} \hfill \textit{invariant} \ \textit{ES} \equiv \textit{finite} \ \textit{ES}
\land \ \textit{finite-rhs} \ \textit{ES}
\land \ \textit{soundness} \ \textit{ES}
\land \ \textit{distinctness} \ \textit{ES}
\land \ \textit{ardenable-all} \ \textit{ES}
\land \ \textit{validity} \ \textit{ES}
lemma invariantI:
\textbf{assumes} soundness ES finite ES distinctness ES ardenable-all ES finite-rhs ES validity ES
\textbf{shows} invariant ES
\langle proof \rangle

\textbf{declare} [\textit{simproc add: finite-Collect}]

lemma finite-Trn:
\textbf{assumes} fin: finite rhs
\textbf{shows} finite \{ r. Trn Y r \in rhs \}
\langle proof \rangle

lemma finite-Lam:
\textbf{assumes} fin: finite rhs
\textbf{shows} finite \{ r. Lam r \in rhs \}
\langle proof \rangle

lemma trm-soundness:
\textbf{assumes} finite:finite rhs
\textbf{shows} lang-rhs (\{ Trn X r| r. Trn X r \in rhs \}) = X \cdot (lang (\bigcup \{ r. Trn X r \in rhs \}))
\langle proof \rangle

lemma lang-of-append-rexp:
\textbf{lang-trm} (Append-rexp r trm) = lang-trm trm \cdot lang r
\langle proof \rangle

lemma lang-of-append-rexp-rhs:
\textbf{lang-rhs} (Append-rexp-rhs rhs r) = lang-rhs rhs \cdot lang r
\langle proof \rangle

2.5 Initial Equational Systems

lemma defined-by-str:
\textbf{assumes} s \in X X \in UNIV // \approx A
\textbf{shows} X = \approx A \ " \{s\}
\langle proof \rangle

lemma every-eqclass-has-transition:
\textbf{assumes} has-str: s \oplus [c] \in X
\textbf{and} in-CS: X \in UNIV // \approx A
\textbf{obtains} Y \textbf{where} Y \in UNIV // \approx A \textbf{and} Y \cdot \{[c]\} \subseteq X \textbf{and} s \in Y
\langle proof \rangle

lemma l-eq-r-in-eqs:
assumes \( X \in \text{eqs} : (X, \text{rhs}) \in \text{Init} (\text{UNIV} /\!\!\!\!\!\!\!\!\!\approx A) \)
shows \( X = \text{lang-rhs rhs} \)
⟨\text{proof}⟩

\text{lemma finite-Init-rhs:}
\text{fixes } CS::(\langle\text{finite} \rangle \text{lang} \set)
\text{assumes finite: finite CS}
\text{shows finite (Init-rhs CS X)}
⟨\text{proof}⟩

\text{lemma Init-ES-satisfies-invariant:}
\text{fixes } A::(\langle\text{finite} \rangle \text{lang})
\text{assumes finite-CS: finite (UNIV // \approx A)}
\text{shows invariant (Init (UNIV // \approx A))}
⟨\text{proof}⟩

\textbf{2.6 Interations}

\text{lemma Arden-preserves-soundness:}
\text{assumes l-eq-r: } X = \text{lang-rhs rhs}
\text{and not-empty: ardenable rhs}
\text{and finite: finite rhs}
\text{shows } X = \text{lang-rhs (Arden X rhs)}
⟨\text{proof}⟩

\text{lemma Append-preserves-finite:}
\text{finite rhs } \Rightarrow \text{finite (Append-rexp-rhs rhs r)}
⟨\text{proof}⟩

\text{lemma Arden-preserves-finite:}
\text{finite rhs } \Rightarrow \text{finite (Arden X rhs)}
⟨\text{proof}⟩

\text{lemma Append-preserves-ardenable:}
\text{ardenable rhs } \Rightarrow \text{ardenable (Append-rexp-rhs rhs r)}
⟨\text{proof}⟩

\text{lemma ardenable-set-sub:}
\text{ardenable rhs } \Rightarrow \text{ardenable (rhs - A)}
⟨\text{proof}⟩

\text{lemma ardenable-set-union:}
\text{[ardenable rhs; ardenable rhs'] } \Rightarrow \text{ardenable (rhs \cup rhs')} 
⟨\text{proof}⟩

\text{lemma Arden-preserves-ardenable:}
\text{ardenable rhs } \Rightarrow \text{ardenable (Arden X rhs)}
lemma Subst-preserves-ardenable:
    [ardenable rhs; ardenable xrhs] \implies ardenable (Subst rhs X xrhs)
(\proof)

lemma Subst-preserves-soundness:
    assumes substor: X = lang-rhs xrhs
    and finite: finite rhs
    shows lang-rhs (Subst rhs X xrhs) = lang-rhs rhs (is ?Left = ?Right)
(\proof)

lemma Subst-preserves-finite-rhs:
    [finite rhs; finite yrhs] \implies finite (Subst rhs Y yrhs)
(\proof)

lemma Subst-all-preserves-finite:
    assumes finite: finite ES
    shows finite (Subst-all ES Y yrhs)
(\proof)

declare [[simproc del: finite-Collect]]

lemma Subst-all-preserves-finite-rhs:
    [finite-rhs ES; finite yrhs] \implies finite-rhs (Subst-all ES Y yrhs)
(\proof)

lemma append-rhs-preserves-cls:
    rhss (Append-rexp-rhs rhs r) = rhss rhs
(\proof)

lemma Arden-removes-cl:
    rhss (Arden Y yrhs) = rhss yrhs \setminus \{ Y \}
(\proof)

lemma lhss-preserves-cls:
    lhss (Subst-all ES Y yrhs) = lhss ES
(\proof)

lemma Subst-updates-cls:
    X \notin rhss xrhs \implies
    rhss (Subst rhs X xrhs) = rhss rhs \cup rhss xrhs \setminus \{ X \}
(\proof)

lemma Subst-all-preserves-validity:
    assumes sc: validity (ES \cup \{(Y, yrhs)\}) (is validity ?A)
    shows validity (Subst-all ES Y (Arden Y yrhs)) (is validity ?B)
(\proof)
lemma Subst-all-satisfies-invariant:
assumes invariant-ES: invariant (ES ∪ {(Y, yrhs)>)
shows invariant (Subst-all ES Y (Arden Y yrhs))
⟨proof⟩
lemma Remove-in-card-measure:
assumes finite: finite ES
and  in-ES: (X, rhs) ∈ ES
shows (Remove ES X rhs, ES) ∈ measure card
⟨proof⟩
lemma Subst-all-cls-remains:
(X, xrhs) ∈ ES ⇒ ∃ xrhs'. (X, xrhs') ∈ (Subst-all ES Y yrhs)
⟨proof⟩
lemma card-noteq-1-has-more:
assumes card:Cond ES
and e-in: (X, xrhs) ∈ ES
and finite: finite ES
shows ∃ (Y, yrhs) ∈ ES. (X, xrhs) ≠ (Y, yrhs)
⟨proof⟩
lemma iteration-step-measure:
assumes Inv-ES: invariant ES
and X-in-ES: (X, xrhs) ∈ ES
and Cnd: Cond ES
shows (Iter X ES, ES) ∈ measure card
⟨proof⟩
lemma iteration-step-invariant:
assumes Inv-ES: invariant ES
and X-in-ES: (X, xrhs) ∈ ES
and Cnd: Cond ES
shows invariant (Iter X ES)
⟨proof⟩
lemma iteration-step-ex:
assumes Inv-ES: invariant ES
and X-in-ES: (X, xrhs) ∈ ES
and Cnd: Cond ES
shows ∃ xrhs'. (X, xrhs') ∈ (Iter X ES)
⟨proof⟩

2.7 The conclusion of the first direction

lemma Solve:
fixes A::('a::finite) lang
assumes fin: finite (UNIV // ≈A)
and X-in: X ∈ (UNIV // ≈A)
shows ∃ rhs. Solve X (Init (UNIV // ≈A)) = {(X, rhs)} ∧ invariant {(X, rhs)}
⟨proof⟩

lemma every-eqcl-has-reg:
  fixes A::('a::finite) lang
  assumes finite-CS: finite (UNIV // ≈A)
  and X-in-CS: X ∈ (UNIV // ≈A)
  shows ∃ r. X = lang r
⟨proof⟩

lemma bchoice-finite-set:
  assumes a: ∀ x ∈ S. ∃ y. x = f y
  and b: finite S
  shows ∃ ys. (∪ S) = ∪ (f ' ys) ∧ finite ys
⟨proof⟩

theorem Myhill-Nerode1:
  fixes A::('a::finite) lang
  assumes finite-CS: finite (UNIV // ≈A)
  shows ∃ r. A = lang r
⟨proof⟩

end

theory Myhill-2
  imports Myhill-1 HOL-Library.Sublist
begin

3 Second direction of MN: regular language ⇒ finite partition

3.1 Tagging functions

definition tag-eq :: ('a list ⇒ 'b) ⇒ ('a list × 'a list) set (=−=)
where =tag= ≡ {(x, y). tag x = tag y}

abbreviation
  tag-eq-applied :: 'a list ⇒ ('a list ⇒ 'b) ⇒ 'a list ⇒ bool (=−−−−)
where x =tag= y ≡ (x, y) ∈ =tag=

lemma [simp]:
  shows (≈A) " {x} = (≈A) " {y} ⟷ x ≈A y
⟨proof⟩
lemma refined-intro:
  assumes $\forall x y z. \{ x \mathbin{=} y; x @ z \in A \} \Rightarrow y @ z \in A$
  shows $\tag\mathbin{=} \subseteq \approx A$
 ⟨proof⟩

lemma finite-eq-tag-rel:
  assumes rng-fnt: finite (range tag)
  shows finite (UNIV // $\tag\mathbin{=}$)
 ⟨proof⟩

lemma refined-partition-finite:
  assumes fnt: finite (UNIV // $R_1$)
  and refined: $R_1 \subseteq R_2$
  and eq1: equiv UNIV $R_1$ and eq2: equiv UNIV $R_2$
  shows finite (UNIV // $R_2$)
 ⟨proof⟩

lemma tag-finite-imageD:
  assumes rng-fnt: finite (range tag)
  and refined: $\tag\mathbin{=} \subseteq \approx A$
  shows finite (UNIV // $\approx A$)
 ⟨proof⟩

3.2 Base cases: Zero, One and Atom

lemma quot-zero-eq:
  shows UNIV // $\approx \{\}$ = $\{\}\{UNIV\}$
 ⟨proof⟩

lemma quot-zero-finiteI [intro]:
  shows finite (UNIV // $\approx \{\}$)
 ⟨proof⟩

lemma quot-one-subset:
  shows UNIV // $\approx \{\}$ $\subseteq$ $\{\\}$.$\{\}$,$\{UNIV - \{\}\}$
 ⟨proof⟩

lemma quot-one-finiteI [intro]:
  shows finite (UNIV // $\approx \{\}$)
 ⟨proof⟩

lemma quot-atom-subset:
  $\{UNIV // (\approx [e])\} \subseteq$ $\{\\}$.$\{\}$,$\{[e]\}$,$\{UNIV - \{\}, [e]\}$
 ⟨proof⟩

lemma quot-atom-finiteI [intro]:
shows finite (UNIV // ≈{[c]}))

⟨proof⟩

3.3 Case for Plus
definition
tag-Plus :: 'a lang ⇒ 'a lang ⇒ 'a list ⇒ ('a lang × 'a lang)
where
tag-Plus A B ≡ λx. (≈A "x", ≈B "x")

lemma quot-plus-finiteI [intro]:
assumes finite1: finite (UNIV // ≈A)
and finite2: finite (UNIV // ≈B)
shows finite (UNIV // ≈(A ∪ B))
⟨proof⟩

3.4 Case for Times
definition
Partitions x ≡ {(x_p, x_s). x_p @ x_s = x}

lemma conc-partitions-elim:
assumes x ∈ A · B
shows ∃(u, v) ∈ Partitions x. u ∈ A ∧ v ∈ B
⟨proof⟩

lemma conc-partitions-intro:
assumes (u, v) ∈ Partitions x ∧ u ∈ A ∧ v ∈ B
shows x ∈ A · B
⟨proof⟩

lemma equiv-class-member:
assumes x ∈ A
and ≈A "x" = ≈A "y"
shows y ∈ A
⟨proof⟩

definition
tag-Times :: 'a lang ⇒ 'a lang ⇒ 'a list ⇒ 'a lang × 'a lang set
where
tag-Times A B ≡ λx. (≈A "x", {(≈B "x") | x_p x_s. x_p ∈ A ∧ (x_p, x_s) ∈ Partitions x})

lemma tag-Times-injI:
assumes a: tag-Times A B x = tag-Times A B y
and c: x @ z ∈ A · B
shows y @ z ∈ A · B
⟨proof⟩

lemma quot-conc-finiteI [intro]:
assumes \( \text{fin1}: \text{finite} (\text{UNIV} // \approx A) \)
and \( \text{fin2}: \text{finite} (\text{UNIV} // \approx B) \)
shows \( \text{finite} (\text{UNIV} // \approx (A \cdot B)) \)

\langle proof \rangle

3.5 Case for Star

\textbf{lemma star-partitions-elim:}
ainsumes \( x @ z \in A^* \)
shows \( \exists (u, v) \in \text{Partitions} (x @ z). \text{strict-prefix } u x \land u \in A^* \land v \in A^* \)

\langle proof \rangle

\textbf{lemma finite-set-has-max2:}
\[
[ \text{finite } A; A \neq \{\} ] \implies \exists \text{ max } \in A. \forall a \in A. \text{ length } a \leq \text{ length max}
\]

\langle proof \rangle

\textbf{lemma finite-strict-prefix-set:}
sshows finite \( \{xa. \text{strict-prefix } xa \ (x::'a list)\} \)

\langle proof \rangle

\textbf{lemma append-eq-cases:}
\textbf{assumes} \( a: x @ y = m @ n m \neq [] \)
\textbf{shows} prefix \( x m \lor \text{strict-prefix } m x \)

\langle proof \rangle

\textbf{lemma star-partitions-elim2:}
\textbf{assumes} \( a: x @ z \in A^* \)
\textbf{and} \( b: x \neq [] \)
\textbf{shows} \( \exists (u, v) \in \text{Partitions } x. \exists (u', v') \in \text{Partitions } z. \text{strict-prefix } u x \land u \in A^* \land v @ u' \in A \land v' \in A^* \)

\langle proof \rangle

\textbf{definition}
\texttt{tag-Star :: 'a lang \Rightarrow 'a list \Rightarrow ('a lang) set}
\textbf{where}
\texttt{tag-Star } A \equiv \lambda x. \{ \approx A "v" | u v. \text{strict-prefix } u x \land u \in A^* \land (u, v) \in \text{Partitions } x \} \)

\textbf{lemma tag-Star-non-empty-injI:}
\textbf{assumes} \( a: \text{tag-Star } A x = \text{tag-Star } A y \)
\textbf{and} \( c: x @ z \in A^* \)
\textbf{and} \( d: x \neq [] \)
\textbf{shows} \( y @ z \in A^* \)

\langle proof \rangle

\textbf{lemma tag-Star-empty-injI:}
\textbf{assumes} \( a: \text{tag-Star } A x = \text{tag-Star } A y \)
\textbf{and} \( c: x @ z \in A^* \)
\textbf{and} \( d: x = [] \)

\langle proof \rangle
shows \( y \otimes z \in A^* \)

(\text{proof})

\textbf{lemma} \textit{quot-star-finiteI} [intro]:
\begin{itemize}
  \item \textbf{assumes} \textit{finite1}: finite (UNIV // ≈}\( A)\)
  \item \textbf{shows} finite (UNIV // ≈}(A^*)\)
\end{itemize}
(\text{proof})

\textbf{3.6} \textbf{The conclusion of the second direction}

\textbf{lemma} \textit{Myhill-Nerode2}:
\begin{itemize}
  \item \textbf{fixes} \textit{r}::'a \textit{rexp}
  \item \textbf{shows} finite (UNIV // ≈}(\text{lang } r)\)
\end{itemize}
(\text{proof})

\textbf{end}

\textbf{theory} \textit{Myhill}
\begin{itemize}
  \item \textbf{imports} \textit{Myhill-2 Regular-Sets.Derivatives}
\end{itemize}
begin

\textbf{4} \textbf{The theorem}

\textbf{theorem} \textit{Myhill-Nerode}:
\begin{itemize}
  \item \textbf{fixes} \textit{A}::'(\text{n}::finite) \textit{lang}
  \item \textbf{shows} (\exists r. A = \text{lang } r) \longleftrightarrow finite (UNIV // ≈}A\)
\end{itemize}
(\text{proof})

\textbf{4.1} \textbf{Second direction proved using partial derivatives}

An alternative proof using the notion of partial derivatives for regular expressions due to Antimirov [1].

\textbf{lemma} \textit{MN-Rel-Derivs}:
\begin{itemize}
  \item \textbf{shows} \( x \approx A y \longleftrightarrow \text{Derivs } x A = \text{Derivs } y A\)
\end{itemize}
(\text{proof})

\textbf{lemma} \textit{Myhill-Nerode3}:
\begin{itemize}
  \item \textbf{fixes} \textit{r}::'a \textit{rexp}
  \item \textbf{shows} finite (UNIV // ≈}(\text{lang } r)\)
\end{itemize}
(\text{proof})

\textbf{end}

\textbf{theory} \textit{Closures}
\begin{itemize}
  \item \textbf{imports} \textit{Myhill HOL-Library.Infinite-Set}
\end{itemize}
begin
5 Closure properties of regular languages

abbreviation
  regular :: 'a lang ⇒ bool
where
  regular A ≡ ∃r. A = lang r

5.1 Closure under ∪, · and ⋆

lemma closure-union [intro]:
  assumes regular A regular B
  shows regular (A ∪ B)
⟨proof⟩

lemma closure-seq [intro]:
  assumes regular A regular B
  shows regular (A · B)
⟨proof⟩

lemma closure-star [intro]:
  assumes regular A
  shows regular (A⋆)
⟨proof⟩

5.2 Closure under complementation

Closure under complementation is proved via the Myhill-Nerode theorem

lemma closure-complement [intro]:
  fixes A::('a::finite) lang
  assumes regular A
  shows regular (− A)
⟨proof⟩

5.3 Closure under – and ∩

lemma closure-difference [intro]:
  fixes A::('a::finite) lang
  assumes regular A regular B
  shows regular (A − B)
⟨proof⟩

lemma closure-intersection [intro]:
  fixes A::('a::finite) lang
  assumes regular A regular B
  shows regular (A ∩ B)
⟨proof⟩

5.4 Closure under string reversal

fun
\[
\text{Rev} :: 'a\text{rexp} \Rightarrow 'a\text{rexp}
\]

where
\[
\begin{align*}
\text{Rev Zero} &= \text{Zero} \\
\text{Rev One} &= \text{One} \\
\text{Rev (Atom c)} &= \text{Atom c} \\
\text{Rev (Plus } r1 \text{ } r2) &= \text{Plus} (\text{Rev r1}) (\text{Rev r2}) \\
\text{Rev (Times } r1 \text{ } r2) &= \text{Times} (\text{Rev r2}) (\text{Rev r1}) \\
\text{Rev (Star } r) &= \text{Star} (\text{Rev r})
\end{align*}
\]

lemma \text{rev-seq}[\text{simp}]:
\[
\text{shows } \text{rev '}(B \cdot A) = (\text{rev '} A) \cdot (\text{rev '} B)
\]
\langle \text{proof} \rangle

lemma \text{rev-star1}:
\[
\text{assumes } a: s \in (\text{rev '} A)^* \\
\text{shows } s \in \text{rev '}(A^*)
\]
\langle \text{proof} \rangle

lemma \text{rev-star2}:
\[
\text{assumes } a: s \in A^* \\
\text{shows } \text{rev } s \in (\text{rev '} A)^*
\]
\langle \text{proof} \rangle

lemma \text{rev-star}[\text{simp}]:
\[
\text{shows } \text{rev '}(A^*) = (\text{rev '} A)^*
\]
\langle \text{proof} \rangle

lemma \text{rev-lang}:
\[
\text{shows } \text{rev '}(\text{lang } r) = \text{lang}(\text{Rev } r)
\]
\langle \text{proof} \rangle

lemma \text{closure-reversal}[\text{intro}]:
\[
\text{assumes } \text{regular } A \\
\text{shows } \text{regular } (\text{rev ' } A)
\]
\langle \text{proof} \rangle

5.5 Closure under left-quotients

abbreviation
\[
\begin{align*}
\text{Deriv-lang } A \text{ } B & \equiv \bigcup x \in A. \text{Deriv } x \text{ } B
\end{align*}
\]

lemma \text{closure-left-quotient}:
\[
\text{assumes } \text{regular } A \\
\text{shows } \text{regular } (\text{Deriv-lang } B \text{ } A)
\]
\langle \text{proof} \rangle

5.6 Finite and co-finite sets are regular

lemma \text{singleton-regular}:
\[
\text{shows } \text{regular } \{s\}
\]
lemma finite-regular:
assumes finite A
shows regular A

lemma cofinite-regular:
fixes A::'a::finite lang
assumes finite (- A)
shows regular A

5.7 Continuation lemma for showing non-regularity of languages

lemma continuation-lemma:
fixes A B::'a::finite lang
assumes reg: regular A
and inf: infinite B
shows \exists x \in B. \exists y \in B. x \neq y \land x \approx_A y

5.8 The language \(a^n b^n\) is not regular

abbreviation
replicate-rev (- ``` - [100, 100] 100)
where
a ``` n \equiv replicate n a

lemma an-bn-not-regular:
shows \neg regular (\bigcup n. \{ CHR "a" ``` n @ CHR "b" ``` n})

6 Closure under \textit{SUBSEQ} and \textit{SUPSEQ}

Properties about the embedding relation

lemma subseq-strict-length:
assumes a: subseq x y x \neq y
shows length x < length y
lemma subseq-wf:
  shows wf { (x, y).
  subseq x y ∧ x ≠ y } }

lemma subseq-good:
  shows good subseq (f :: nat ⇒ ('a::finite) list)

lemma subseq-Higman-antichains:
  assumes a: ∀ (x::('a::finite) list) ∈ A. ∀ y ∈ A. x ≠ y → ¬ (subseq x y) ∧
  ¬ (subseq y x)
  shows finite A

6.1 Sub- and Supersequences

definition SUBSEQ A ≡ { x::('a::finite) list. ∃ y ∈ A. subseq x y }

definition SUPSEQ A ≡ { x::('a::finite) list. ∃ y ∈ A. subseq y x }

lemma SUPSEQ-simps [simp]:
  shows SUPSEQ {} = {}
  and SUPSEQ [] = UNIV

lemma SUPSEQ-atom [simp]:
  shows SUPSEQ {[c]} = UNIV · {[c]} · UNIV

lemma SUPSEQ-union [simp]:
  shows SUPSEQ (A ∪ B) = SUPSEQ A ∪ SUPSEQ B

lemma SUPSEQ-conc [simp]:
  shows SUPSEQ (A · B) = SUPSEQ A · SUPSEQ B

lemma SUPSEQ-star [simp]:
  shows SUPSEQ (A*) = UNIV

6.2 Regular expression that recognises every character

definition Allreg :: 'a::finite rexp

where
lemma Allreg-lang [simp]:
  shows \( \text{lang } \text{Allreg} = (\bigcup a. \{[a]\}) \)  
⟨proof⟩

lemma [simp]:
  shows \( (\bigcup a. \{[a]\})^* = \text{UNIV} \)  
⟨proof⟩

lemma Star-Allreg-lang [simp]:
  shows \( \text{lang } (\text{Star } \text{Allreg}) = \text{UNIV} \)  
⟨proof⟩

fun \( \text{UP} :: 'a::\text{finite rexp} \Rightarrow 'a \text{ rexp} \)  
where
  \( \text{UP } \text{(Zero)} = \text{Zero} \)  
| \( \text{UP } \text{(One)} = \text{Star } \text{Allreg} \)  
| \( \text{UP } \text{(Atom } c) = \text{Times } (\text{Star } \text{Allreg}) (\text{Times } (\text{Atom } c) (\text{Star } \text{Allreg})) \)  
| \( \text{UP } \text{(Plus } r1 \text{ } r2) = \text{Plus } (\text{UP } r1) (\text{UP } r2) \)  
| \( \text{UP } \text{(Times } r1 \text{ } r2) = \text{Times } (\text{UP } r1) (\text{UP } r2) \)  
| \( \text{UP } \text{(Star } r) = \text{Star } \text{Allreg} \)

lemma lang-UP:
  fixes \( r :: 'a::\text{finite rexp} \Rightarrow 'a \text{ rexp} \)
  shows \( \text{lang } (\text{UP } r) = \text{SUPSEQ } (\text{lang } r) \)  
⟨proof⟩

lemma SUPSEQ-regular:
  fixes \( A :: 'a::\text{finite lang} \)
  assumes regular \( A \)
  shows regular \( (\text{SUPSEQ } A) \)  
⟨proof⟩

lemma SUPSEQ-subset:
  fixes \( A :: 'a::\text{finite list set} \)
  shows \( A \subseteq \text{SUPSEQ } A \)  
⟨proof⟩

lemma SUBSEQ-complement:
  shows \( - (\text{SUBSEQ } A) = \text{SUPSEQ } (- (\text{SUBSEQ } A)) \)  
⟨proof⟩

definition minimal :: 'a::\text{finite list} \Rightarrow 'a lang \Rightarrow bool  
where
  minimal \( x \text{ } A \equiv (\forall y \in A. \text{subseq } y \text{ } x \rightarrow \text{subseq } x \text{ } y) \)
lemma main-lemma:
  shows \( \exists M. \text{finite } M \land \text{SUPSEQ } A = \text{SUPSEQ } M \)
⟨proof⟩

6.3 Closure of SUBSEQ and SUPSEQ

lemma closure-SUPSEQ:
  fixes A::'a::finite lang
  shows regular (SUPSEQ A)
⟨proof⟩

lemma closure-SUBSEQ:
  fixes A::'a::finite lang
  shows regular (SUBSEQ A)
⟨proof⟩
end

7 Tools for showing non-regularity of a language

theory Non-Regular-Languages
  imports Myhill
begin

7.1 Auxiliary material

lemma bij-betw-image-quotient:
  bij-betw (\lambda y. f \cdot \{y\}) (f \cdot A) (A // \{(a,b). f a = f b\})
⟨proof⟩

lemma regular-Derivs-finite:
  fixes r :: 'a :: finite rexp
  shows finite (range (\lambda w. Derivs w (lang r)))
⟨proof⟩

lemma Nil-in-Derivs-iff:
  \[ \in Derivs w A \iff w \in A \]
⟨proof⟩

The following operation repeats a list \( n \) times (usually written as \( w^n \)).

primrec repeat :: nat \Rightarrow 'a list \Rightarrow 'a list
  where repeat 0 xs = []
| repeat (Suc n) xs = xs @ repeat n xs

lemma repeat-Cons-left: repeat (Suc n) xs = xs @ repeat n xs ⟨proof⟩

lemma repeat-Cons-right: repeat (Suc n) xs = repeat n xs @ xs
⟨proof⟩

lemma repeat-Cons-append-commute [simp]: repeat n xs @ xs = xs @ repeat n xs

lemma repeat-Cons-add [simp]: repeat \((m + n)\) \(xs\) = repeat \(m\) \(xs\) @ repeat \(n\) \(xs\)

lemma repeat-Nil [simp]: repeat \(n\) \([]\) = []

lemma repeat-conv-replicate: repeat \(n\) \(xs\) = concat (replicate \(n\) \(xs\))

lemma nth-prefixes [simp]: \(n \leq \text{length} \(xs\) \implies \text{prefixes} \(xs\)! \(n\) = take \(n\) \(xs\)

lemma nth-suffixes [simp]: \(n \leq \text{length} \(xs\) \implies \text{suffixes} \(xs\)! \(n\) = drop (length \(xs\) - \(n\)) \(xs\)

lemma length-take-prefixes:
  assumes \(xs\) \(\in\) set (take \(n\) (prefixes \(ys\)))
  shows length \(xs\) \(<\) \(n\)

7.2 Non-regularity by giving an infinite set of equivalence classes

Non-regularity can be shown by giving an infinite set of non-equivalent words (w.r.t. the Myhill–Nerode relation).

lemma not-regular-langI:
  assumes infinite \(B\) \(\land\) \(x. y. x \in B \implies y \in B \implies x \neq y \implies \exists w. \neg(x @ w \in A \iff y @ w \in A)\)
  shows \(\neg\)regular-lang \((A :: 'a :: finite list set)\)

lemma not-regular-langI':
  assumes infinite \(B\) \(\land\) \(x. y. x \in B \implies y \in B \implies x \neq y \implies \exists w. \neg(f \ x @ w \in A \iff f \ y @ w \in A)\)
  shows \(\neg\)regular-lang \((A :: 'a :: finite list set)\)

7.3 The Pumping Lemma

The Pumping lemma can be shown very easily from the Myhill–Nerode theorem: if we have a word whose length is more than the (finite) number of equivalence classes, then it must have two different prefixes in the same class and the difference between these two prefixes can then be “pumped”.

22
lemma pumping-lemma-aux:

fixes A :: 'a list set
defines δ ≡ λ w. Derivs w A
defines n ≡ card (range δ)
assumes z ∈ A finite (range δ) length z ≥ n
shows ∃ u v w. z = u @ v @ w ∧ length (u @ v) ≤ n ∧ v ≠ [] ∧ (∀ i. u @ repeat i v @ w ∈ A)
⟨proof⟩

theorem pumping-lemma:

fixes r :: 'a :: finite rexp
obtains n where
    \( \forall z. z \in \text{lang } r \implies \text{length } z \geq n \implies \exists u v w. z = u \oplus v \oplus w \land \text{length } (u \oplus v) \leq n \land v \neq [] \land (\forall i. u \oplus \text{repeat } i v \oplus w \in \text{lang } r) \)
⟨proof⟩

corollary pumping-lemma-not-regular-lang:

fixes A :: 'a :: finite list set
assumes \( \forall n. \text{length } (z_n) \geq n \land \forall n. z_n \in A \)
assumes \( \forall n u v w. z_n = u \oplus v \oplus w \implies \text{length } (u \oplus v) \leq n \implies v \neq [] \implies u \oplus \text{repeat } i n u v w \oplus w \notin A \)
shows ¬regular-lang A
⟨proof⟩

7.4 Examples

The language of all words containing the same number of as and bs is not regular.

lemma ¬regular-lang { w. \text{length } (\text{filter id } w) = \text{length } (\text{filter } \text{Not } w) } (\text{is } ¬\text{regular-lang } ?A)
⟨proof⟩

The language \( \{ a^i b^i \mid i \in \mathbb{N} \} \) is not regular.

lemma eq-replicate-iff:

\( xs = \text{replicate } n x \iff \text{set } xs \subseteq \{ x \} \land \text{length } xs = n \)
⟨proof⟩

lemma replicate-eq-appendE:

assumes \( xs \oplus ys = \text{replicate } n x \)
obtains i j where \( n = i + j \land xs = \text{replicate } i x ys = \text{replicate } j x \)
⟨proof⟩

lemma ¬regular-lang (range (λ i. \text{replicate } i \text{ True } \oplus \text{replicate } i \text{ False })) (\text{is } ¬\text{regular-lang } ?A)
⟨proof⟩

end
References
