The Myhill-Nerode Theorem Based on Regular Expressions

Chunhan Wu, Xingyuan Zhang and Christian Urban

September 13, 2023

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] Most details of the theories are described in the paper [2].

theory *Folds*
imports *Regular−Sets.Regular-Exp*
begin

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 :: $('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a\ set \Rightarrow 'b$
where
folds f z S $\equiv$ *SOME x. fold-graph f z S x*

Plus-combination for a set of regular expressions

abbreviation
Setalt :: *'a rexp set* $\Rightarrow$ *'a rexp* ($\biguplus$ - [*1000*] *999*)
where
$\biguplus A \equiv$ *folds Plus Zero A*

For finite sets, *Setalt* is preserved under *lang*.

lemma *folds-plus-simp* [*simp*]:
fixes *rs*::(*'a rexp*) *set*
assumes *a*: *finite rs*
shows *lang* ($\biguplus$ *rs*) = $\bigcup$ (*lang* ‘ *rs*)
⟨*proof*⟩

end

theory *Myhill-1*
imports *Folds*
HOL−Library.While-Combinator
begin

2 First direction of MN: *finite partition* ⇒ *regular language*

notation
conc (**infixr** · *100*) **and**
star (-⋆ [*101*] *102*)

lemma *Pair-Collect* [*simp*]:
shows (*x*, *y*) ∈ {(*x*, *y*). *P x y*} ⟷ *P x y*
⟨*proof*⟩

Myhill-Nerode relation

definition
str-eq :: *'a lang* ⇒ (*'a list* × *'a list*) *set* (≈- [*100*] *100*)
where
≈*A* ≡ {(*x*, *y*). (∀ *z*. *x* @ *z* ∈ *A* ⟷ *y* @ *z* ∈ *A*)}

abbreviation
str-eq-applied :: *'a list* ⇒ *'a lang* ⇒ *'a list* ⇒ *bool* (- ≈- -)
where
x ≈*A y* ≡ (*x*, *y*) ∈ ≈*A*

lemma *str-eq-conv-Derivs*:
str-eq A = {(*u*,*v*). *Derivs u A* = *Derivs v A*}
⟨*proof*⟩

definition
finals :: *'a lang* ⇒ *'a lang set*
where
finals A ≡ {≈*A* `` {*s*} | *s* . *s* ∈ *A*}

lemma *lang-is-union-of-finals*:
shows *A* = $\bigcup$(*finals A*)
⟨*proof*⟩

lemma *finals-in-partitions*:
shows *finals A* $\subseteq$ (*UNIV* // $\approx$*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*}

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*) *set* ⇒ (*'a lang* × (*'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*

definition
Solve X ES $\equiv$ *while Cond* (*Iter X*) *ES*

definition
distinctness ES $\equiv$
$\forall$ *X rhs rhs'*. $(X, rhs) \in ES \wedge (X, rhs') \in ES \longrightarrow rhs = rhs'$

definition
soundness ES $\equiv \forall (X, rhs) \in ES.\ X =$ *lang-rhs rhs*

definition
ardenable rhs $\equiv (\forall\ Y\ r.\ Trn\ Y\ r \in rhs \longrightarrow [] \notin lang\ r)$

definition
ardenable-all ES $\equiv \forall (X, rhs) \in ES.$ *ardenable rhs*

definition
finite-rhs ES $\equiv \forall (X, rhs) \in ES.$ *finite rhs*

lemma *finite-rhs-def2*:
finite-rhs ES $=$ $(\forall\ X\ rhs.\ (X, rhs) \in ES \longrightarrow$ *finite rhs*$)$
⟨*proof*⟩

definition
rhss rhs $\equiv \{X \mid X\ r.\ Trn\ X\ r \in rhs\}$

definition
lhss ES $\equiv \{Y \mid Y\ yrhs.\ (Y, yrhs) \in ES\}$

definition
validity ES $\equiv \forall (X, rhs) \in ES.$ *rhss rhs* $\subseteq$ *lhss ES*

lemma *rhss-union-distrib*:
shows *rhss* $(A \cup B) =$ *rhss A* $\cup$ *rhss B*
⟨*proof*⟩

lemma *lhss-union-distrib*:
shows *lhss* $(A \cup B) =$ *lhss A* $\cup$ *lhss B*
⟨*proof*⟩

definition
invariant ES $\equiv$ *finite ES*
$\wedge$ *finite-rhs ES*
$\wedge$ *soundness ES*
$\wedge$ *distinctness ES*
$\wedge$ *ardenable-all ES*
$\wedge$ *validity ES*

lemma *invariantI*:
assumes *soundness ES finite ES distinctness ES ardenable-all ES finite-rhs ES validity ES*
shows *invariant ES*
⟨*proof*⟩

declare [[*simproc add*: *finite-Collect*]]

lemma *finite-Trn*:
assumes *fin*: *finite rhs*
shows *finite* {*r*. *Trn Y r* ∈ *rhs*}
⟨*proof*⟩

lemma *finite-Lam*:
assumes *fin*: *finite rhs*
shows *finite* {*r*. *Lam r* ∈ *rhs*}
⟨*proof*⟩

lemma *trm-soundness*:
assumes *finite*:*finite rhs*
shows *lang-rhs* ({*Trn X r*| *r*. *Trn X r* ∈ *rhs*}) = *X* · (*lang* (⨄{*r*. *Trn X r* ∈ *rhs*}))
⟨*proof*⟩

lemma *lang-of-append-rexp*:
lang-trm (*Append-rexp r trm*) = *lang-trm trm* · *lang r*
⟨*proof*⟩

lemma *lang-of-append-rexp-rhs*:
lang-rhs (*Append-rexp-rhs rhs r*) = *lang-rhs rhs* · *lang r*
⟨*proof*⟩

2.5 Intial Equational Systems

lemma *defined-by-str*:
assumes *s* ∈ *X X* ∈ *UNIV* // ≈*A*
shows *X* = ≈*A* `` {*s*}
⟨*proof*⟩

lemma *every-eqclass-has-transition*:
assumes *has-str*: *s* @ [*c*] ∈ *X*
and *in-CS*: *X* ∈ *UNIV* // ≈*A*
obtains *Y* **where** *Y* ∈ *UNIV* // ≈*A* **and** *Y* · {[*c*]} ⊆ *X* **and** *s* ∈ *Y*
⟨*proof*⟩

lemma *l-eq-r-in-eqs*:

assumes *X-in-eqs*: $(X, rhs) \in Init\ (UNIV\ //\ \approx A)$
shows $X = lang\text{-}rhs\ rhs$
⟨*proof*⟩

lemma *finite-Init-rhs*:
fixes CS::$((\prime a$::*finite*$)\ lang)\ set$
assumes *finite*: *finite CS*
shows *finite* $(Init\text{-}rhs\ CS\ X)$
⟨*proof*⟩

lemma *Init-ES-satisfies-invariant*:
fixes A::$((\prime a$::*finite*$)\ lang)$
assumes *finite-CS*: *finite* $(UNIV\ //\ \approx A)$
shows *invariant* $(Init\ (UNIV\ //\ \approx A))$
⟨*proof*⟩

2.6 Interations

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

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

lemma *Arden-preserves-finite*:
finite rhs $\Longrightarrow$ *finite* $(Arden\ X\ rhs)$
⟨*proof*⟩

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

lemma *ardenable-set-sub*:
ardenable rhs $\Longrightarrow$ *ardenable* $(rhs - A)$
⟨*proof*⟩

lemma *ardenable-set-union*:
$[\![ardenable\ rhs;\ ardenable\ rhs']\!] \Longrightarrow ardenable\ (rhs \cup rhs')$
⟨*proof*⟩

lemma *Arden-preserves-ardenable*:
ardenable rhs $\Longrightarrow$ *ardenable* $(Arden\ X\ rhs)$

⟨*proof*⟩

lemma *Subst-preserves-ardenable*:
  ⟦*ardenable rhs*; *ardenable xrhs*⟧ ⟹ *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*⟧ ⟹ *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*⟧ ⟹ *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* − {*Y*}
⟨*proof*⟩

lemma *lhss-preserves-cls*:
  lhss (*Subst-all ES Y yrhs*) = *lhss ES*
⟨*proof*⟩

lemma *Subst-updates-cls*:
  X ∉ *rhss xrhs* ⟹
      rhss (*Subst rhs X xrhs*) = *rhss rhs* ∪ *rhss xrhs* − {*X*}
⟨*proof*⟩

lemma *Subst-all-preserves-validity*:
  assumes *sc*: *validity* (*ES* ∪ {(*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* $(ES \cup \{(Y, yrhs)\})$
shows *invariant* $(Subst\text{-}all\ ES\ Y\ (Arden\ Y\ yrhs))$
⟨*proof*⟩

lemma *Remove-in-card-measure*:
assumes *finite*: *finite ES*
and *in-ES*: $(X, rhs) \in ES$
shows $(Remove\ ES\ X\ rhs, ES) \in measure\ card$
⟨*proof*⟩

lemma *Subst-all-cls-remains*:
$(X, xrhs) \in ES \Longrightarrow \exists\ xrhs'.\ (X, xrhs') \in (Subst\text{-}all\ ES\ Y\ yrhs)$
⟨*proof*⟩

lemma *card-noteq-1-has-more*:
assumes *card*:*Cond ES*
and *e-in*: $(X, xrhs) \in ES$
and *finite*: *finite ES*
shows $\exists (Y, yrhs) \in ES.\ (X, xrhs) \neq (Y, yrhs)$
⟨*proof*⟩

lemma *iteration-step-measure*:
assumes *Inv-ES*: *invariant ES*
and *X-in-ES*: $(X, xrhs) \in ES$
and *Cnd*: *Cond ES*
shows $(Iter\ X\ ES, ES) \in measure\ card$
⟨*proof*⟩

lemma *iteration-step-invariant*:
assumes *Inv-ES*: *invariant ES*
and *X-in-ES*: $(X, xrhs) \in ES$
and *Cnd*: *Cond ES*
shows *invariant* $(Iter\ X\ ES)$
⟨*proof*⟩

lemma *iteration-step-ex*:
assumes *Inv-ES*: *invariant ES*
and *X-in-ES*: $(X, xrhs) \in ES$
and *Cnd*: *Cond ES*
shows $\exists xrhs'.\ (X, xrhs') \in (Iter\ X\ ES)$
⟨*proof*⟩

2.7 The conclusion of the first direction

lemma *Solve*:
fixes A::$('a$::*finite*$)\ lang$

assumes *fin*: *finite* (*UNIV* // $\approx A$)
and *X-in*: $X \in$ (*UNIV* // $\approx A$)
shows $\exists$ *rhs*. *Solve X* (*Init* (*UNIV* // $\approx A$)) = {(*X*, *rhs*)} $\wedge$ *invariant* {(*X*, *rhs*)}
⟨*proof*⟩

lemma *every-eqcl-has-reg*:
fixes *A*::('*a*::*finite*) *lang*
assumes *finite-CS*: *finite* (*UNIV* // $\approx A$)
and *X-in-CS*: $X \in$ (*UNIV* // $\approx A$)
shows $\exists\, r.\ X = lang\ r$
⟨*proof*⟩

lemma *bchoice-finite-set*:
assumes *a*: $\forall\, x \in S.\ \exists\, y.\ x = f\ y$
and *b*: *finite S*
shows $\exists\, ys.\ (\bigcup\ S) = \bigcup (f\ `\ ys) \wedge$ *finite ys*
⟨*proof*⟩

theorem *Myhill-Nerode1*:
fixes *A*::('*a*::*finite*) *lang*
assumes *finite-CS*: *finite* (*UNIV* // $\approx A$)
shows $\exists\, r.\ A = lang\ r$
⟨*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
$=tag= \equiv \{(x, y).\ tag\ x = tag\ y\}$

abbreviation
tag-eq-applied :: '*a list* ⇒ ('*a list* ⇒ '*b*) ⇒ '*a list* ⇒ *bool* (- =-= -)
where
$x =tag= y \equiv (x, y) \in =tag=$

lemma [*simp*]:
shows ($\approx A$) `` {*x*} = ($\approx A$) `` {*y*} ⟷ $x \approx A\ y$
⟨*proof*⟩

lemma *refined-intro*:
assumes $\bigwedge x\ y\ z.\ [\![x =tag= y;\ x\ @\ z \in A]\!] \Longrightarrow y\ @\ z \in A$
shows $=tag= \subseteq \approx A$
⟨*proof*⟩

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

lemma *refined-partition-finite*:
assumes *fnt*: *finite* (*UNIV* // *R1*)
and *refined*: *R1* ⊆ *R2*
and *eq1*: *equiv UNIV R1* **and** *eq2*: *equiv UNIV R2*
shows *finite* (*UNIV* // *R2*)
⟨*proof*⟩

lemma *tag-finite-imageD*:
assumes *rng-fnt*: *finite* (*range tag*)
and *refined*: $=tag= \subseteq \approx A$
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 // $(\approx\{[c]\}) \subseteq \{\{[]\},\{[c]\},\ UNIV - \{[],\ [c]\}\}$
⟨*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 ≡ λ*x*. (≈*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 ≡ λ*x*. (≈*A* `` {*x*}, {(≈*B* `` {*x*$_s$}) | *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 *fin1*: *finite* (*UNIV* // ≈*A*)
and *fin2*: *finite* (*UNIV* // ≈*B*)
shows *finite* (*UNIV* // ≈(*A* · *B*))
⟨*proof*⟩

3.5 Case for *Star*

lemma *star-partitions-elim*:
assumes *x* @ *z* ∈ *A*⋆ *x* ≠ []
shows ∃(*u*, *v*) ∈ *Partitions* (*x* @ *z*). *strict-prefix u x* ∧ *u* ∈ *A*⋆ ∧ *v* ∈ *A*⋆
⟨*proof*⟩

lemma *finite-set-has-max2*:
⟦*finite A*; *A* ≠ {}⟧ ⟹ ∃ *max* ∈ *A*. ∀ *a* ∈ *A*. *length a* ≤ *length max*
⟨*proof*⟩

lemma *finite-strict-prefix-set*:
shows *finite* {*xa*. *strict-prefix xa* (*x*::'*a list*)}
⟨*proof*⟩

lemma *append-eq-cases*:
assumes *a*: *x* @ *y* = *m* @ *n* *m* ≠ []
shows *prefix x m* ∨ *strict-prefix m x*
⟨*proof*⟩

lemma *star-spartitions-elim2*:
assumes *a*: *x* @ *z* ∈ *A*⋆
and *b*: *x* ≠ []
shows ∃(*u*, *v*) ∈ *Partitions x*. ∃ (*u*′, *v*′) ∈ *Partitions z*. *strict-prefix u x* ∧ *u* ∈ *A*⋆ ∧ *v* @ *u*′ ∈ *A* ∧ *v*′ ∈ *A*⋆
⟨*proof*⟩

definition
tag-Star :: '*a lang* ⇒ '*a list* ⇒ ('*a lang*) *set*
where
tag-Star A ≡ λ*x*. {≈*A* `` {*v*} | *u v*. *strict-prefix u x* ∧ *u* ∈ *A*⋆ ∧ (*u*, *v*) ∈ *Partitions x*}

lemma *tag-Star-non-empty-injI*:
assumes *a*: *tag-Star A x* = *tag-Star A y*
and *c*: *x* @ *z* ∈ *A*⋆
and *d*: *x* ≠ []
shows *y* @ *z* ∈ *A*⋆
⟨*proof*⟩

lemma *tag-Star-empty-injI*:
assumes *a*: *tag-Star A x* = *tag-Star A y*
and *c*: *x* @ *z* ∈ *A*⋆
and *d*: *x* = []

shows $y \mathbin{@} z \in A\star$
⟨*proof*⟩

lemma *quot-star-finiteI* [*intro*]:
assumes *finite1*: *finite* (*UNIV* // $\approx A$)
shows *finite* (*UNIV* // $\approx(A\star)$)
⟨*proof*⟩

3.6 The conclusion of the second direction

lemma *Myhill-Nerode2*:
fixes r::'*a rexp*
shows *finite* (*UNIV* // $\approx$(*lang r*))
⟨*proof*⟩

end

theory *Myhill*
imports *Myhill-2 Regular−Sets.Derivatives*
begin

4 The theorem

theorem *Myhill-Nerode*:
fixes A::('*a*::*finite*) *lang*
shows ($\exists\, r.\ A = lang\ r$) $\longleftrightarrow$ *finite* (*UNIV* // $\approx A$)
⟨*proof*⟩

4.1 Second direction proved using partial derivatives

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

lemma *MN-Rel-Derivs*:
shows $x \approx A\ y \longleftrightarrow Derivs\ x\ A = Derivs\ y\ A$
⟨*proof*⟩

lemma *Myhill-Nerode3*:
fixes r::'*a rexp*
shows *finite* (*UNIV* // $\approx$(*lang r*))
⟨*proof*⟩

end

theory *Closures*
imports *Myhill HOL−Library.Infinite-Set*
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

Rev :: *'a rexp* ⇒ *'a rexp*
where
Rev Zero = *Zero*
| *Rev One* = *One*
| *Rev* (*Atom c*) = *Atom c*
| *Rev* (*Plus r1 r2*) = *Plus* (*Rev r1*) (*Rev r2*)
| *Rev* (*Times r1 r2*) = *Times* (*Rev r2*) (*Rev r1*)
| *Rev* (*Star r*) = *Star* (*Rev r*)

lemma *rev-seq*[*simp*]:
shows *rev* ‘ (*B* · *A*) = (*rev* ‘ *A*) · (*rev* ‘ *B*)
⟨*proof*⟩

lemma *rev-star1*:
assumes *a*: *s* ∈ (*rev* ‘ *A*)⋆
shows *s* ∈ *rev* ‘ (*A*⋆)
⟨*proof*⟩

lemma *rev-star2*:
assumes *a*: *s* ∈ *A*⋆
shows *rev s* ∈ (*rev* ‘ *A*)⋆
⟨*proof*⟩

lemma *rev-star* [*simp*]:
shows *rev* ‘ (*A*⋆) = (*rev* ‘ *A*)⋆
⟨*proof*⟩

lemma *rev-lang*:
shows *rev* ‘ (*lang r*) = *lang* (*Rev r*)
⟨*proof*⟩

lemma *closure-reversal* [*intro*]:
assumes *regular A*
shows *regular* (*rev* ‘ *A*)
⟨*proof*⟩

5.5 Closure under left-quotients

abbreviation
Deriv-lang A B ≡ ⋃ *x* ∈ *A*. *Derivs x B*

lemma *closure-left-quotient*:
assumes *regular A*
shows *regular* (*Deriv-lang B A*)
⟨*proof*⟩

5.6 Finite and co-finite sets are regular

lemma *singleton-regular*:
shows *regular* {*s*}

⟨*proof*⟩

lemma *finite-regular*:
  assumes *finite A*
  shows *regular A*
⟨*proof*⟩

lemma *cofinite-regular*:
  fixes *A*::′*a*::*finite lang*
  assumes *finite* (− *A*)
  shows *regular A*
⟨*proof*⟩

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 ∃ *x* ∈ *B*. ∃ *y* ∈ *B*. *x* ≠ *y* ∧ *x* ≈*A y*
⟨*proof*⟩

5.8 The language $a^n\ b^n$ is not regular

abbreviation
  replicate-rev (- $\widehat{\ \ }$ - [*100*, *100*] *100*)
where
  a $\widehat{\ \ }$ *n* ≡ *replicate n a*

lemma *an-bn-not-regular*:
  shows ¬ *regular* (⋃ *n*. {*CHR* ″*a*″ $\widehat{\ \ }$ *n* @ *CHR* ″*b*″ $\widehat{\ \ }$ *n*})
⟨*proof*⟩

end
theory *Closures2*
imports
  Closures
  Well-Quasi-Orders.Well-Quasi-Orders
begin

6 Closure under *SUBSEQ* and *SUPSEQ*

Properties about the embedding relation

lemma *subseq-strict-length*:
  assumes *a*: *subseq x y x* ≠ *y*
  shows *length x* < *length y*

⟨*proof*⟩

lemma *subseq-wf*:
shows *wf* {(*x*, *y*). *subseq x y* ∧ *x* ≠ *y*}
⟨*proof*⟩

lemma *subseq-good*:
shows *good subseq* (*f* :: *nat* ⇒ ('*a*::*finite*) *list*)
⟨*proof*⟩

lemma *subseq-Higman-antichains*:
assumes *a*: ∀ (*x*::('*a*::*finite*) *list*) ∈ *A*. ∀ *y* ∈ *A*. *x* ≠ *y* ⟶ ¬(*subseq x y*) ∧ ¬(*subseq y x*)
shows *finite A*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *SUPSEQ-atom* [*simp*]:
shows *SUPSEQ* {[*c*]} = *UNIV* · {[*c*]} · *UNIV*
⟨*proof*⟩

lemma *SUPSEQ-union* [*simp*]:
shows *SUPSEQ* (*A* ∪ *B*) = *SUPSEQ A* ∪ *SUPSEQ B*
⟨*proof*⟩

lemma *SUPSEQ-conc* [*simp*]:
shows *SUPSEQ* (*A* · *B*) = *SUPSEQ A* · *SUPSEQ B*
⟨*proof*⟩

lemma *SUPSEQ-star* [*simp*]:
shows *SUPSEQ* (*A*⋆) = *UNIV*
⟨*proof*⟩

6.2 Regular expression that recognises every character

definition
Allreg :: '*a*::*finite rexp*
where

$Allreg \equiv \biguplus (Atom \mathbin{`} UNIV)$

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

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

lemma *Star-Allreg-lang* [*simp*]:
shows $lang\ (Star\ Allreg) = UNIV$
⟨*proof*⟩

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

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

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

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

lemma *SUBSEQ-complement*:
shows $-\ (SUBSEQ\ A) = SUPSEQ\ (-\ (SUBSEQ\ A))$
⟨*proof*⟩

definition
$minimal :: 'a{::}finite\ list \Rightarrow 'a\ lang \Rightarrow bool$
where
$minimal\ x\ A \equiv (\forall\, y \in A.\ subseq\ y\ x \longrightarrow subseq\ x\ y)$

lemma *main-lemma*:
shows $\exists M.$ *finite* $M \wedge$ *SUPSEQ* $A =$ *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 −‘ {y}) (f ‘ 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*: [] ∈ *Derivs* w $A \longleftrightarrow$ $w \in A$
⟨*proof*⟩

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

primrec *repeat* :: *nat* ⇒ 'a *list* ⇒ '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*

⟨*proof*⟩

lemma *repeat-Cons-add* [*simp*]: *repeat* $(m + n)$ *xs* = *repeat m xs* @ *repeat n xs*
⟨*proof*⟩

lemma *repeat-Nil* [*simp*]: *repeat n* [] = []
⟨*proof*⟩

lemma *repeat-conv-replicate*: *repeat n xs* = *concat* (*replicate n xs*)
⟨*proof*⟩

lemma *nth-prefixes* [*simp*]: $n \leq$ *length xs* $\Longrightarrow$ *prefixes xs* ! *n* = *take n xs*
⟨*proof*⟩

lemma *nth-suffixes* [*simp*]: $n \leq$ *length xs* $\Longrightarrow$ *suffixes xs* ! *n* = *drop* (*length xs* − *n*) *xs*
⟨*proof*⟩

lemma *length-take-prefixes*:
assumes *xs* ∈ *set* (*take n* (*prefixes ys*))
shows *length xs* < *n*
⟨*proof*⟩

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* $\bigwedge x\ y.\ x \in B \Longrightarrow y \in B \Longrightarrow x \neq y \Longrightarrow \exists\, w.\ \neg(x$ @ $w \in A \longleftrightarrow y$ @ $w \in A)$
shows ¬*regular-lang* (*A* :: *'a* :: *finite list set*)
⟨*proof*⟩

lemma *not-regular-langI'*:
assumes *infinite B* $\bigwedge x\ y.\ x \in B \Longrightarrow y \in B \Longrightarrow x \neq y \Longrightarrow \exists\, w.\ \neg(f\ x$ @ $w \in A \longleftrightarrow f\ y$ @ $w \in A)$
shows ¬*regular-lang* (*A* :: *'a* :: *finite list set*)
⟨*proof*⟩

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".

lemma *pumping-lemma-aux*:
fixes $A :: 'a\ list\ set$
defines $\delta \equiv \lambda w.\ Derivs\ w\ A$
defines $n \equiv card\ (range\ \delta)$
assumes $z \in A$ *finite* $(range\ \delta)$ $length\ z \geq n$
shows $\exists\, u\ v\ w.\ z = u\ @\ v\ @\ w \wedge length\ (u\ @\ v) \leq n \wedge v \neq [] \wedge (\forall\, i.\ u\ @\ repeat\ i\ v\ @\ w \in A)$
⟨*proof*⟩

theorem *pumping-lemma*:
fixes $r :: 'a :: finite\ rexp$
obtains n **where**
$\bigwedge z.\ z \in lang\ r \Longrightarrow length\ z \geq n \Longrightarrow$
$\exists\, u\ v\ w.\ z = u\ @\ v\ @\ w \wedge length\ (u\ @\ v) \leq n \wedge v \neq [] \wedge (\forall\, i.\ u\ @\ repeat\ i\ v\ @\ w \in lang\ r)$
⟨*proof*⟩

corollary *pumping-lemma-not-regular-lang*:
fixes $A :: 'a :: finite\ list\ set$
assumes $\bigwedge n.\ length\ (z\ n) \geq n$ **and** $\bigwedge n.\ z\ n \in A$
assumes $\bigwedge n\ u\ v\ w.\ z\ n = u\ @\ v\ @\ w \Longrightarrow length\ (u\ @\ v) \leq n \Longrightarrow v \neq [] \Longrightarrow$
$u\ @\ repeat\ (i\ n\ u\ v\ w)\ v\ @\ w \notin A$
shows $\neg regular\text{-}lang\ A$
⟨*proof*⟩

7.4 Examples

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

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

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

lemma *eq-replicate-iff*:
$xs = replicate\ n\ x \longleftrightarrow set\ xs \subseteq \{x\} \wedge length\ xs = n$
⟨*proof*⟩

lemma *replicate-eq-appendE*:
assumes $xs\ @\ ys = replicate\ n\ x$
obtains $i\ j$ **where** $n = i + j\ xs = replicate\ i\ x\ ys = replicate\ j\ x$
⟨*proof*⟩

lemma $\neg regular\text{-}lang\ (range\ (\lambda i.\ replicate\ i\ True\ @\ replicate\ i\ False))$ (**is** $\neg regular\text{-}lang\ ?A$)
⟨*proof*⟩

end

References

[1] V. Antimirov. Partial Derivatives of Regular Expressions and Finite Automata Constructions. *Theoretical Computer Science*, 155:291–319, 1995.

[2] C. Wu, X. Zhang, and C. Urban. A Formalisation of the Myhill-Nerode Theorem based on Regular Expressions (Proof Pearl). In *Proc. of the 2nd International Conference on Interactive Theorem Proving*, volume 6898 of *LNCS*, pages 341–356, 2011.