

The Myhill-Nerode Theorem Based on Regular Expressions

Chunhan Wu, Xingyuan Zhang and Christian Urban

December 14, 2021

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: <i>finite partition</i> \Rightarrow <i>regular language</i>	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: <i>regular language</i> \Rightarrow <i>finite partition</i>	11
3.1	Tagging functions	11
3.2	Base cases: <i>Zero</i> , <i>One</i> and <i>Atom</i>	12
3.3	Case for <i>Plus</i>	13
3.4	Case for <i>Times</i>	13
3.5	Case for <i>Star</i>	14
3.6	The conclusion of the second direction	15

¹Most details of the theories are described in the paper [2].

4	The theorem	15
4.1	Second direction proved using partial derivatives	15
5	Closure properties of regular languages	16
5.1	Closure under \cup , \cdot and \star	16
5.2	Closure under complementation	16
5.3	Closure under $-$ and \cap	16
5.4	Closure under string reversal	16
5.5	Closure under left-quotients	17
5.6	Finite and co-finite sets are regular	17
5.7	Continuation lemma for showing non-regularity of languages .	18
5.8	The language $a^n b^n$ is not regular	18
6	Closure under <i>SUBSEQ</i> and <i>SUPSEQ</i>	18
6.1	Sub- and Supersequences	19
6.2	Regular expression that recognises every character	19
6.3	Closure of <i>SUBSEQ</i> and <i>SUPSEQ</i>	21
7	Tools for showing non-regularity of a language	21
7.1	Auxiliary material	21
7.2	Non-regularity by giving an infinite set of equivalence classes	22
7.3	The Pumping Lemma	22
7.4	Examples	23

```

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 (\uplus - [1000] 999)

where

$\uplus A \equiv$ *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 ( $\bigoplus$  rs) =  $\bigcup$  (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 · 100) and  
  star (-* [101] 102)
```

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

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 =  $\{(u, v). \text{Derivs } u\ A = \text{Derivs } v\ A\}$   
<proof>
```

```
definition  
  finals :: 'a lang  $\Rightarrow$  'a lang set  
where  
  finals A  $\equiv \{\approx A\ \text{“ } \{s\} \mid s . s \in A\}$ 
```

```
lemma lang-is-union-of-finals:  
  shows A =  $\bigcup$  (finals A)  
<proof>
```

lemma *finals-in-partitions*:
shows $\text{finals } A \subseteq (\text{UNIV} // \approx A)$
<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 \Rightarrow *'a lang*
where
lang-trm (Lam r) = *lang r*
| *lang-trm (Trn X r)* = $X \cdot \text{lang } r$

fun
lang-rhs::('a trm) set \Rightarrow *'a lang*
where
lang-rhs rhs = $\bigcup (\text{lang-trm } ' \text{ rhs})$

lemma *lang-rhs-set*:
shows $\text{lang-rhs } \{\text{Trn } X \ r \mid r. P \ r\} = \bigcup \{\text{lang-trm } (\text{Trn } X \ r) \mid r. P \ r\}$
<proof>

lemma *lang-rhs-union-distrib*:
shows $\text{lang-rhs } A \cup \text{lang-rhs } B = \text{lang-rhs } (A \cup B)$
<proof>

Transitions between equivalence classes

definition
transition :: 'a lang \Rightarrow *'a* \Rightarrow *'a lang* \Rightarrow *bool* ($- \models \Rightarrow - [100,100,100] \ 100$)
where
 $Y \models c \Rightarrow X \equiv Y \cdot \{[c]\} \subseteq X$

Initial equational system

definition
Init-rhs CS X \equiv
if ($\square \in X$) *then*
 $\{\text{Lam One}\} \cup \{\text{Trn } Y \ (\text{Atom } c) \mid Y \ c. Y \in \text{CS} \wedge Y \models c \Rightarrow X\}$
else
 $\{\text{Trn } Y \ (\text{Atom } c) \mid Y \ c. Y \in \text{CS} \wedge Y \models c \Rightarrow X\}$

definition
Init CS $\equiv \{(X, \text{Init-rhs } \text{CS } X) \mid X. X \in \text{CS}\}$

2.2 Arden Operation on equations

fun

$Append\text{-}rexp :: 'a\ rexp \Rightarrow 'a\ trm \Rightarrow 'a\ trm$

where

$Append\text{-}rexp\ r\ (Lam\ rexp) = Lam\ (Times\ rexp\ r)$

$| Append\text{-}rexp\ r\ (Trn\ X\ rexp) = Trn\ X\ (Times\ rexp\ r)$

definition

$Append\text{-}rexp\text{-}rhs\ rhs\ rexp \equiv (Append\text{-}rexp\ rexp)\ 'rhs$

definition

$Arden\ X\ rhs \equiv$

$Append\text{-}rexp\text{-}rhs\ (rhs - \{Trn\ X\ r \mid r.\ Trn\ X\ r \in rhs\})\ (Star\ (\bigoplus\ \{r.\ Trn\ X\ r \in rhs\}))$

2.3 Substitution Operation on equations

definition

$Subst\ rhs\ X\ xrhs \equiv$

$(rhs - \{Trn\ X\ r \mid r.\ Trn\ X\ r \in rhs\}) \cup (Append\text{-}rexp\text{-}rhs\ xrhs\ (\bigoplus\ \{r.\ Trn\ X\ r \in rhs\}))$

definition

$Subst\text{-}all :: ('a\ lang \times ('a\ trm)\ set)\ set \Rightarrow 'a\ lang \Rightarrow ('a\ trm)\ set \Rightarrow ('a\ lang \times ('a\ trm)\ set)\ set$

where

$Subst\text{-}all\ ES\ X\ xrhs \equiv \{(Y, Subst\ yrhs\ X\ xrhs) \mid Y\ yrhs.\ (Y, yrhs) \in ES\}$

definition

$Remove\ ES\ X\ xrhs \equiv$

$Subst\text{-}all\ (ES - \{(X, xrhs)\})\ X\ (Arden\ X\ xrhs)$

2.4 While-combinator and invariants

definition

$Iter\ X\ ES \equiv (let\ (Y, yrhs) = SOME\ (Y, yrhs).\ (Y, yrhs) \in ES \wedge X \neq Y\ in\ Remove\ ES\ Y\ yrhs)$

lemma *IterI2*:

assumes $(Y, yrhs) \in ES$

and $X \neq Y$

and $\bigwedge Y\ yrhs.\ \llbracket (Y, yrhs) \in ES; X \neq Y \rrbracket \Longrightarrow Q\ (Remove\ ES\ Y\ yrhs)$

shows $Q\ (Iter\ X\ ES)$

<proof>

abbreviation

$Cond\ ES \equiv card\ ES \neq 1$

definition

Solve X $ES \equiv \text{while Cond (Iter } X) ES$

definition

distinctness $ES \equiv$

$\forall X \text{ rhs rhs}'. (X, \text{rhs}) \in ES \wedge (X, \text{rhs}') \in ES \longrightarrow \text{rhs} = \text{rhs}'$

definition

soundness $ES \equiv \forall (X, \text{rhs}) \in ES. X = \text{lang-rhs rhs}$

definition

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

definition

ardenable-all $ES \equiv \forall (X, \text{rhs}) \in ES. \text{ardenable rhs}$

definition

finite-rhs $ES \equiv \forall (X, \text{rhs}) \in ES. \text{finite rhs}$

lemma *finite-rhs-def2:*

finite-rhs $ES = (\forall X \text{ rhs}. (X, \text{rhs}) \in ES \longrightarrow \text{finite rhs})$
 ⟨proof⟩

definition

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

definition

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

definition

validity $ES \equiv \forall (X, \text{rhs}) \in ES. \text{rhss rhs} \subseteq \text{lhss } ES$

lemma *rhss-union-distrib:*

shows $\text{rhss } (A \cup B) = \text{rhss } A \cup \text{rhss } B$
 ⟨proof⟩

lemma *lhss-union-distrib:*

shows $\text{lhss } (A \cup B) = \text{lhss } A \cup \text{lhss } B$
 ⟨proof⟩

definition

invariant $ES \equiv \text{finite } ES$
 $\wedge \text{finite-rhs } ES$
 $\wedge \text{soundness } ES$
 $\wedge \text{distinctness } ES$
 $\wedge \text{ardenable-all } ES$
 $\wedge \text{validity } ES$

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-eclass-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 $X\text{-in-eqs}$: $(X, rhs) \in \text{Init } (UNIV // \approx A)$
shows $X = \text{lang-rhs } rhs$
 $\langle \text{proof} \rangle$

lemma *finite-Init-rhs*:
fixes $CS::('a::\text{finite}) \text{ lang}$ *set*
assumes *finite*: *finite* CS
shows *finite* $(\text{Init-rhs } CS X)$
 $\langle \text{proof} \rangle$

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

2.6 Iterations

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

lemma *Append-preserves-finite*:
 $\text{finite } rhs \implies \text{finite } (\text{Append-rexp-rhs } rhs r)$
 $\langle \text{proof} \rangle$

lemma *Arden-preserves-finite*:
 $\text{finite } rhs \implies \text{finite } (\text{Arden } X rhs)$
 $\langle \text{proof} \rangle$

lemma *Append-preserves-ardenable*:
 $\text{ardenable } rhs \implies \text{ardenable } (\text{Append-rexp-rhs } rhs r)$
 $\langle \text{proof} \rangle$

lemma *ardenable-set-sub*:
 $\text{ardenable } rhs \implies \text{ardenable } (rhs - A)$
 $\langle \text{proof} \rangle$

lemma *ardenable-set-union*:
 $\llbracket \text{ardenable } rhs; \text{ardenable } rhs' \rrbracket \implies \text{ardenable } (rhs \cup rhs')$
 $\langle \text{proof} \rangle$

lemma *Arden-preserves-ardenable*:
 $\text{ardenable } rhs \implies \text{ardenable } (\text{Arden } X rhs)$

<proof>

lemma *Subst-preserves-ardenable:*

$\llbracket \text{ardenable } rhs; \text{ardenable } xrhs \rrbracket \implies \text{ardenable } (\text{Subst } rhs \ X \ xrhs)$
<proof>

lemma *Subst-preserves-soundness:*

assumes *substor*: $X = \text{lang-rhs } xrhs$
and *finite*: *finite rhs*
shows $\text{lang-rhs } (\text{Subst } rhs \ X \ xrhs) = \text{lang-rhs } rhs$ (**is** *?Left = ?Right*)
<proof>

lemma *Subst-preserves-finite-rhs:*

$\llbracket \text{finite } rhs; \text{finite } yrhs \rrbracket \implies \text{finite } (\text{Subst } rhs \ Y \ yrhs)$
<proof>

lemma *Subst-all-preserves-finite:*

assumes *finite*: *finite ES*
shows *finite* (*Subst-all ES Y yrhs*)
<proof>

declare $\llbracket \text{simplproc del: finite-Collect} \rrbracket$

lemma *Subst-all-preserves-finite-rhs:*

$\llbracket \text{finite-rhs } ES; \text{finite } yrhs \rrbracket \implies \text{finite-rhs } (\text{Subst-all } ES \ Y \ yrhs)$
<proof>

lemma *append-rhs-preserves-cl:*

$\text{rhss } (\text{Append-rexp-rhs } rhs \ r) = \text{rhss } rhs$
<proof>

lemma *Arden-removes-cl:*

$\text{rhss } (\text{Arden } Y \ yrhs) = \text{rhss } yrhs - \{Y\}$
<proof>

lemma *lhss-preserves-cl:*

$\text{lhss } (\text{Subst-all } ES \ Y \ yrhs) = \text{lhss } ES$
<proof>

lemma *Subst-updates-cl:*

$X \notin \text{rhss } xrhs \implies$
 $\text{rhss } (\text{Subst } rhs \ X \ xrhs) = \text{rhss } rhs \cup \text{rhss } xrhs - \{X\}$
<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* ($ES \cup \{(Y, yrhs)\}$)
shows *invariant* (*Subst-all* ES Y (*Arden* Y $yrhs$))
 \langle *proof* \rangle

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*
 \langle *proof* \rangle

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

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

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*
 \langle *proof* \rangle

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)
 \langle *proof* \rangle

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

2.7 The conclusion of the first direction

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

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

lemma *every-eccl-has-reg*:
fixes *A*::('a::finite) *lang*
assumes *finite-CS*: *finite* (*UNIV* // $\approx A$)
and *X-in-CS*: $X \in (\text{UNIV} // \approx A)$
shows $\exists r. X = \text{lang } r$
<proof>

lemma *bchoice-finite-set*:
assumes *a*: $\forall x \in S. \exists y. x = f y$
and *b*: *finite* *S*
shows $\exists \text{ys}. (\bigcup S) = \bigcup (f ` \text{ys}) \wedge \text{finite } \text{ys}$
<proof>

theorem *Myhill-Nerode1*:
fixes *A*::('a::finite) *lang*
assumes *finite-CS*: *finite* (*UNIV* // $\approx A$)
shows $\exists r. A = \text{lang } r$
<proof>

end

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

3 Second direction of MN: *regular language* \Rightarrow *finite partition*

3.1 Tagging functions

definition
 $\text{tag-eq} :: ('a \text{ list} \Rightarrow 'b) \Rightarrow ('a \text{ list} \times 'a \text{ list}) \text{ set} (= =)$
where
 $=\text{tag} = \equiv \{(x, y). \text{tag } x = \text{tag } y\}$

abbreviation
 $\text{tag-eq-applied} :: 'a \text{ list} \Rightarrow ('a \text{ list} \Rightarrow 'b) \Rightarrow 'a \text{ list} \Rightarrow \text{bool} (- = = -)$
where
 $x =\text{tag} = y \equiv (x, y) \in =\text{tag} =$

lemma [*simp*]:
shows $(\approx A) \text{ `` } \{x\} = (\approx A) \text{ `` } \{y\} \longleftrightarrow x \approx A y$
<proof>

lemma *refined-intro*:
 assumes $\bigwedge x y z. \llbracket x = \text{tag} = y; x @ z \in A \rrbracket \implies y @ z \in A$
 shows $=\text{tag} = \subseteq \approx A$
 $\langle \text{proof} \rangle$

lemma *finite-eq-tag-rel*:
 assumes *rng-fnt*: *finite* (*range tag*)
 shows *finite* (*UNIV // =tag=*)
 $\langle \text{proof} \rangle$

lemma *refined-partition-finite*:
 assumes *fnt*: *finite* (*UNIV // R1*)
 and *refined*: $R1 \subseteq R2$
 and *eq1*: *equiv UNIV R1* and *eq2*: *equiv UNIV R2*
 shows *finite* (*UNIV // R2*)
 $\langle \text{proof} \rangle$

lemma *tag-finite-imageD*:
 assumes *rng-fnt*: *finite* (*range tag*)
 and *refined*: $=\text{tag} = \subseteq \approx A$
 shows *finite* (*UNIV // \approx A*)
 $\langle \text{proof} \rangle$

3.2 Base cases: Zero, One and Atom

lemma *quot-zero-eq*:
 shows $UNIV // \approx \{\} = \{UNIV\}$
 $\langle \text{proof} \rangle$

lemma *quot-zero-finiteI [intro]*:
 shows *finite* (*UNIV // \approx \{\}*)
 $\langle \text{proof} \rangle$

lemma *quot-one-subset*:
 shows $UNIV // \approx \{\} \subseteq \{\{\}, UNIV - \{\}\}$
 $\langle \text{proof} \rangle$

lemma *quot-one-finiteI [intro]*:
 shows *finite* (*UNIV // \approx \{\}*)
 $\langle \text{proof} \rangle$

lemma *quot-atom-subset*:
 $UNIV // (\approx \{[c]\}) \subseteq \{\{\}, \{[c]\}, UNIV - \{\}, [c]\}$
 $\langle \text{proof} \rangle$

lemma *quot-atom-finiteI [intro]*:

shows *finite* (*UNIV* // $\approx\{c\}$)
 ⟨*proof*⟩

3.3 Case for *Plus*

definition

tag-Plus :: 'a lang \Rightarrow 'a lang \Rightarrow 'a list \Rightarrow ('a lang \times 'a lang)

where

tag-Plus *A B* \equiv $\lambda x. (\approx A \text{ `` } \{x\}, \approx B \text{ `` } \{x\})$

lemma *quot-plus-finiteI* [*intro*]:

assumes *finite1*: *finite* (*UNIV* // $\approx A$)

and *finite2*: *finite* (*UNIV* // $\approx B$)

shows *finite* (*UNIV* // $\approx(A \cup B)$)

⟨*proof*⟩

3.4 Case for *Times*

definition

Partitions *x* \equiv $\{(x_p, x_s). x_p @ x_s = x\}$

lemma *conc-partitions-elim*:

assumes $x \in A \cdot B$

shows $\exists (u, v) \in \text{Partitions } x. u \in A \wedge v \in B$

⟨*proof*⟩

lemma *conc-partitions-intro*:

assumes $(u, v) \in \text{Partitions } x \wedge u \in A \wedge v \in B$

shows $x \in A \cdot B$

⟨*proof*⟩

lemma *equiv-class-member*:

assumes $x \in A$

and $\approx A \text{ `` } \{x\} = \approx A \text{ `` } \{y\}$

shows $y \in A$

⟨*proof*⟩

definition

tag-Times :: 'a lang \Rightarrow 'a lang \Rightarrow 'a list \Rightarrow 'a lang \times 'a lang set

where

tag-Times *A B* \equiv $\lambda x. (\approx A \text{ `` } \{x\}, \{(\approx B \text{ `` } \{x_s\}) \mid x_p \ x_s. x_p \in A \wedge (x_p, x_s) \in \text{Partitions } x\})$

lemma *tag-Times-injI*:

assumes *a*: *tag-Times* *A B* $x = \text{tag-Times } A B y$

and *c*: $x @ z \in A \cdot B$

shows $y @ z \in A \cdot B$

⟨*proof*⟩

lemma *quot-conc-finiteI* [*intro*]:

assumes *fin1*: *finite* (*UNIV* // $\approx A$)
and *fin2*: *finite* (*UNIV* // $\approx B$)
shows *finite* (*UNIV* // $\approx (A \cdot B)$)
 <proof>

3.5 Case for *Star*

lemma *star-partitions-elim*:

assumes $x @ z \in A^\star$ $x \neq []$
shows $\exists (u, v) \in \text{Partitions } (x @ z)$. *strict-prefix* $u x \wedge u \in A^\star \wedge v \in A^\star$
 <proof>

lemma *finite-set-has-max2*:

$\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies \exists \text{max} \in A. \forall a \in A. \text{length } a \leq \text{length } \text{max}$
 <proof>

lemma *finite-strict-prefix-set*:

shows *finite* $\{x.a. \text{strict-prefix } xa \ (x::'a \text{ list})\}$
 <proof>

lemma *append-eq-cases*:

assumes $a: x @ y = m @ n$ $m \neq []$
shows *prefix* $x m \vee \text{strict-prefix } m x$
 <proof>

lemma *star-spartitions-elim2*:

assumes $a: x @ z \in A^\star$
and $b: x \neq []$
shows $\exists (u, v) \in \text{Partitions } x. \exists (u', v') \in \text{Partitions } z. \text{strict-prefix } u x \wedge u \in A^\star \wedge v @ u' \in A \wedge v' \in A^\star$
 <proof>

definition

tag-Star $:: 'a \text{ lang} \Rightarrow 'a \text{ list} \Rightarrow ('a \text{ lang}) \text{ set}$

where

tag-Star $A \equiv \lambda x. \{\approx A \text{ “}\{v\} \mid u v. \text{strict-prefix } u x \wedge u \in A^\star \wedge (u, v) \in \text{Partitions } x\}$

lemma *tag-Star-non-empty-injI*:

assumes $a: \text{tag-Star } A \ x = \text{tag-Star } A \ y$
and $c: x @ z \in A^\star$
and $d: x \neq []$
shows $y @ z \in A^\star$
 <proof>

lemma *tag-Star-empty-injI*:

assumes $a: \text{tag-Star } A \ x = \text{tag-Star } A \ y$
and $c: x @ z \in A^\star$
and $d: x = []$

shows $y @ z \in A^*$
<proof>

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

3.6 The conclusion of the second direction

lemma *Myhill-Nerode2*:
fixes $r::'a \text{ rexp}$
shows *finite* (*UNIV* // $\approx(\text{lang } r)$)
<proof>

end

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

4 The theorem

theorem *Myhill-Nerode*:
fixes $A::('a::\text{finite}) \text{ lang}$
shows $(\exists r. A = \text{lang } r) \longleftrightarrow \text{finite } (\text{UNIV} // \approx A)$
<proof>

4.1 Second direction proved using partial derivatives

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

lemma *MN-Rel-Derivs*:
shows $x \approx A y \longleftrightarrow \text{Derivs } x A = \text{Derivs } y A$
<proof>

lemma *Myhill-Nerode3*:
fixes $r::'a \text{ rexp}$
shows *finite* (*UNIV* // $\approx(\text{lang } r)$)
<proof>

end

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

5 Closure properties of regular languages

abbreviation

regular :: 'a lang \Rightarrow bool

where

regular A $\equiv \exists r. A = \text{lang } r$

5.1 Closure under \cup , \cdot and \star

lemma *closure-union* [intro]:

assumes *regular* A *regular* B

shows *regular* (A \cup B)

<proof>

lemma *closure-seq* [intro]:

assumes *regular* A *regular* B

shows *regular* (A \cdot B)

<proof>

lemma *closure-star* [intro]:

assumes *regular* A

shows *regular* (A \star)

<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 \cap

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 \cap B)

<proof>

5.4 Closure under string reversal

fun

$Rev :: 'a \text{ rexp} \Rightarrow 'a \text{ rexp}$
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 $rev \ ' (B \cdot A) = (rev \ ' A) \cdot (rev \ ' B)$
 $\langle proof \rangle$

lemma *rev-star1*:
assumes $a: s \in (rev \ ' A)^\star$
shows $s \in rev \ ' (A^\star)$
 $\langle proof \rangle$

lemma *rev-star2*:
assumes $a: s \in A^\star$
shows $rev \ s \in (rev \ ' A)^\star$
 $\langle proof \rangle$

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

lemma *rev-lang*:
shows $rev \ ' (lang \ r) = lang \ (Rev \ r)$
 $\langle proof \rangle$

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

5.5 Closure under left-quotients

abbreviation

$$Deriv\text{-}lang \ A \ B \equiv \bigcup x \in A. \ Derivs \ x \ B$$

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

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 $\exists x \in B. \exists y \in B. x \neq y \wedge x \approx_A y$

<proof>

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

abbreviation

replicate-rev (- $\widetilde{\quad}$ - [100, 100] 100)

where

$a \widetilde{\quad} n \equiv \text{replicate } n \ a$

lemma *an-bn-not-regular*:

shows $\neg \text{regular } (\bigcup n. \{ \text{CHR } "a" \widetilde{\quad} n @ \text{CHR } "b" \widetilde{\quad} n \})$

<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 \neq y*

shows *length x < length y*

<proof>

lemma *subseq-wf*:

shows $wf \{(x, y). \text{subseq } x \ y \wedge x \neq y\}$

<proof>

lemma *subseq-good*:

shows $good \text{ subseq } (f :: nat \Rightarrow ('a::finite) \text{ list})$

<proof>

lemma *subseq-Higman-antichains*:

assumes $a: \forall (x::('a::finite) \text{ list}) \in A. \forall y \in A. x \neq y \longrightarrow \neg(\text{subseq } x \ y) \wedge \neg(\text{subseq } y \ x)$

shows $finite \ A$

<proof>

6.1 Sub- and Supersequences

definition

$SUBSEQ \ A \equiv \{x::('a::finite) \text{ list}. \exists y \in A. \text{subseq } x \ y\}$

definition

$SUPSEQ \ A \equiv \{x::('a::finite) \text{ list}. \exists y \in A. \text{subseq } y \ x\}$

lemma *SUPSEQ-simps* [*simp*]:

shows $SUPSEQ \ \{\} = \{\}$

and $SUPSEQ \ \{\}\} = UNIV$

<proof>

lemma *SUPSEQ-atom* [*simp*]:

shows $SUPSEQ \ \{[c]\} = UNIV \cdot \{[c]\} \cdot UNIV$

<proof>

lemma *SUPSEQ-union* [*simp*]:

shows $SUPSEQ \ (A \cup B) = SUPSEQ \ A \cup SUPSEQ \ B$

<proof>

lemma *SUPSEQ-conc* [*simp*]:

shows $SUPSEQ \ (A \cdot B) = SUPSEQ \ A \cdot SUPSEQ \ B$

<proof>

lemma *SUPSEQ-star* [*simp*]:

shows $SUPSEQ \ (A \star) = UNIV$

<proof>

6.2 Regular expression that recognises every character

definition

$Allreg :: 'a::finite \text{ rexp}$

where

$Allreg \equiv \biguplus (Atom \text{ ' } UNIV)$

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

lemma [*simp*]:
 shows $(\bigcup a. \{[a]\})^* = UNIV$
 ⟨*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 $\exists M. \text{finite } M \wedge \text{SUPSEQ } A = \text{SUPSEQ } M$
 $\langle \text{proof} \rangle$

6.3 Closure of SUBSEQ and SUPSEQ

lemma *closure-SUPSEQ*:
fixes $A :: 'a :: \text{finite lang}$
shows *regular* ($\text{SUPSEQ } A$)
 $\langle \text{proof} \rangle$

lemma *closure-SUBSEQ*:
fixes $A :: 'a :: \text{finite lang}$
shows *regular* ($\text{SUBSEQ } A$)
 $\langle \text{proof} \rangle$

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*:
 $\text{bij-betw } (\lambda y. f -' \{y\}) (f ' A) (A // \{(a,b). f a = f b\})$
 $\langle \text{proof} \rangle$

lemma *regular-Derivs-finite*:
fixes $r :: 'a :: \text{finite rexp}$
shows *finite* ($\text{range } (\lambda w. \text{Derivs } w (\text{lang } r))$)
 $\langle \text{proof} \rangle$

lemma *Nil-in-Derivs-iff*: $\square \in \text{Derivs } w A \longleftrightarrow w \in A$
 $\langle \text{proof} \rangle$

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

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

lemma *repeat-Cons-left*: $\text{repeat } (\text{Suc } n) \text{ } xs = xs @ \text{repeat } n \text{ } xs$ $\langle \text{proof} \rangle$

lemma *repeat-Cons-right*: $\text{repeat } (\text{Suc } n) \text{ } xs = \text{repeat } n \text{ } xs @ xs$
 $\langle \text{proof} \rangle$

lemma *repeat-Cons-append-commute* [*simp*]: $\text{repeat } n \text{ } xs @ xs = xs @ \text{repeat } n \text{ } xs$

<proof>

lemma *repeat-Cons-add* [simp]: $\text{repeat } (m + n) \text{ } xs = \text{repeat } m \text{ } xs @ \text{repeat } n \text{ } xs$
<proof>

lemma *repeat-Nil* [simp]: $\text{repeat } n \text{ } [] = []$
<proof>

lemma *repeat-conv-replicate*: $\text{repeat } n \text{ } xs = \text{concat } (\text{replicate } n \text{ } xs)$
<proof>

lemma *nth-prefixes* [simp]: $n \leq \text{length } xs \implies \text{prefixes } xs ! n = \text{take } n \text{ } xs$
<proof>

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

lemma *length-take-prefixes*:
 assumes $xs \in \text{set } (\text{take } n \text{ } (\text{prefixes } ys))$
 shows $\text{length } xs < n$
<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 $\text{infinite } B \wedge x \ y. x \in B \implies y \in B \implies x \neq y \implies \exists w. \neg(x @ w \in A \longleftrightarrow y @ w \in A)$
 shows $\neg \text{regular-lang } (A :: 'a :: \text{finite list set})$
<proof>

lemma *not-regular-langI'*:
 assumes $\text{infinite } B \wedge x \ y. x \in B \implies y \in B \implies x \neq y \implies \exists w. \neg(f x @ w \in A \longleftrightarrow f y @ w \in A)$
 shows $\neg \text{regular-lang } (A :: 'a :: \text{finite list set})$
<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 $A :: 'a \text{ list set}$

defines $\delta \equiv \lambda w. \text{Derivs } w \ A$

defines $n \equiv \text{card } (\text{range } \delta)$

assumes $z \in A \text{ finite } (\text{range } \delta) \text{ length } z \geq n$

shows $\exists u \ v \ w. z = u @ v @ w \wedge \text{length } (u @ v) \leq n \wedge v \neq [] \wedge (\forall i. u @ \text{repeat } i \ v @ w \in A)$

<proof>

theorem *pumping-lemma*:

fixes $r :: 'a :: \text{finite rexp}$

obtains n **where**

$\bigwedge z. z \in \text{lang } r \implies \text{length } z \geq n \implies$

$\exists u \ v \ w. z = u @ v @ w \wedge \text{length } (u @ v) \leq n \wedge v \neq [] \wedge (\forall i. u @ \text{repeat } i \ v @ w \in \text{lang } r)$

<proof>

corollary *pumping-lemma-not-regular-lang*:

fixes $A :: 'a :: \text{finite list set}$

assumes $\bigwedge n. \text{length } (z \ n) \geq n$ **and** $\bigwedge n. z \ n \in A$

assumes $\bigwedge n \ u \ v \ w. z \ n = u @ v @ w \implies \text{length } (u @ v) \leq n \implies v \neq [] \implies u @ \text{repeat } (i \ n \ u \ v \ w) \ v @ w \notin A$

shows $\neg \text{regular-lang } A$

<proof>

7.4 Examples

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

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

<proof>

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

lemma *eq-replicate-iff*:

$xs = \text{replicate } n \ x \longleftrightarrow \text{set } xs \subseteq \{x\} \wedge \text{length } xs = n$

<proof>

lemma *replicate-eq-appendE*:

assumes $xs @ ys = \text{replicate } n \ x$

obtains $i \ j$ **where** $n = i + j \ xs = \text{replicate } i \ x \ ys = \text{replicate } j \ x$

<proof>

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

<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.