Myhill-Nerode Theorem for (Nominal) G-Automata

Cárolos Laméris

March 17, 2025

Abstract

This work formalizes the Myhill-Nerode theorems for G-automata and nominal G-automata. The Myhill-Nerode theorem for (nominal) G-automata states that given an orbit finite (nominal) alphabet A and a G-language $L \subseteq A^*$ the following are equivalent:

- The set of equivalence classes of $L/\equiv_{\mathtt{MN}}$,with respect to the Myhill-Nerode equivalence relation, $\equiv_{\mathtt{MN}}$, is orbit finite.
- L is recognized by a deterministic (nominal) G-automaton with an orbit finite set of states.

The proofs formalized are based on those from [1].

Contents

1 Myhill-Nerode Theorem for G-automata

We prove the Myhill-Nerode Theorem for G-automata / nominal G-automata following the proofs from [1] (The standard Myhill-Nerode theorem is also proved, as a special case of the G-Myhill-Nerode theorem). Concretely, we formalize the following results from [1]: lemmas: 3.4, 3.5, 3.6, 3.7, 4.8, 4.9; proposition: 5.1; theorems: 3.8 (Myhill-Nerode for G-automata), 5.2 (Myhill-Nerode for nominal G-automata).

Throughout this document, we maintain the following convention for isar proofs: If we `obtain` some term `t` for which some result holds, we name it `H_t`. An assumption which is an induction hypothesis is named `A_IH` Assumptions start with an `"A"` and intermediate results start with a `"H"`. Typically we just name them via indexes, i.e. as `A_i` and `H_j`. When encountering nested isar proofs we add an index for how nested the assumption / intermediate result is. For example if we have an isar proof in an isar proof in an isar proof, we would name assumptions of the most nested proof `A3_i`.

theory *Nominal-Myhill-Nerode*
imports
Main
HOL.Groups
HOL.Relation
HOL.Fun
HOL−Algebra.Group-Action
HOL−Algebra.Elementary-Groups

begin

`GMN_simps` will contain selection of lemmas / definitions is updated through out the document.

named-theorems *GMN-simps*
lemmas *GMN-simps*

We will use the $\star$-symbol for the set of words of elements of a set, $A^\star$, the induced group action on the set of words $\phi^\star$ and for the extended transition function $\delta^\star$, thus we introduce the map `star` and apply `adhoc_overloading` to get the notation working in all three situations:

consts *star* :: *'typ1* ⇒ *'typ2* (‹-⋆› [*1000*] *999*)

adhoc-overloading
star ⇌ *lists*

We use $\odot$ to convert between the definition of group actions via group homomoprhisms and the more standard infix group action notation. We deviate from [1] in that we consider left group actions, rather than right group actions:

definition
make-op :: (*'grp* ⇒ *'X* ⇒ *'X*) ⇒ *'grp* ⇒ *'X* ⇒ *'X* (**infixl** ‹(⊙₁)› *70*)
where (⊙ φ) ≡ (λ*g*. (λ*x*. φ *g x*))

lemmas *make-op-def* [*simp*, *GMN-simps*]

1.1 Extending Group Actions

The following lemma is used for a proof in the locale `alt_grp_act`:

lemma *pre-image-lemma*:

$[\![S \subseteq T;\ x \in T \wedge f \in Bij\ T;\ (restrict\ f\ S)\ `\ S = S;\ f\ x \in S]\!] \Longrightarrow x \in S$
⟨*proof*⟩

The locale `alt_grp_act` is just a renaming of the locale `group_action`. This was done to obtain more easy to interpret type names and context variables closer to the notation of [1]:

locale *alt-grp-act* = *group-action G X φ*
for
G :: (*'grp*, *'b*) *monoid-scheme* **and**
X :: *'X set* (**structure**) **and**
φ
begin

lemma *alt-grp-act-is-left-grp-act*:
shows $x \in X \Longrightarrow \mathbf{1}_G \odot_\varphi x = x$ **and**
$g \in carrier\ G \Longrightarrow h \in carrier\ G \Longrightarrow x \in X \Longrightarrow (g \otimes_G h) \odot_\varphi x = g \odot_\varphi (h \odot_\varphi x)$
⟨*proof*⟩

definition
induced-star-map :: (*'grp* ⇒ *'X* ⇒*'X*) ⇒ *'grp* ⇒ *'X list* ⇒ *'X list*
where *induced-star-map func* = (λ*g*∈*carrier G*. (λ*lst* ∈ $X^\star$. *map* (*func g*) *lst*))

Because the adhoc overloading is used within a locale, isues will be encountered later due to there being multiple instances of the locale `alt_grp_act` in a single context:

adhoc-overloading
star ⇌ *induced-star-map*

definition
induced-quot-map ::
'Y set ⇒ (*'grp* ⇒ *'Y* ⇒ *'Y*) ⇒ (*'Y* ×*'Y*) *set* ⇒ *'grp* ⇒ *'Y set* ⇒ *'Y set* (‹[-]_1› *60*)
where ([*func*]$_R$ $_S$) = (λ*g*∈*carrier G*. (λ*x* ∈ (*S* // *R*). *R* `` {(*func g*) (*SOME z*. *z*∈*x*)}))

lemmas *induced-star-map-def* [*simp*, *GMN-simps*]
induced-quot-map-def [*simp*, *GMN-simps*]

lemma *act-maps-n-distrib*:
∀ *g*∈*carrier G*. ∀ *w*∈$X^\star$. ∀ *v*∈$X^\star$. ($\varphi^\star$) *g* (*w* @ *v*) = (($\varphi^\star$) *g w*) @ (($\varphi^\star$) *g v*)
⟨*proof*⟩

lemma *triv-act*:
$a \in X \Longrightarrow (\varphi\ \mathbf{1}_G)\ a = a$
⟨*proof*⟩

lemma *triv-act-map*:
∀ *w*∈$X^\star$. (($\varphi^\star$) $\mathbf{1}_G$) *w* = *w*

⟨*proof*⟩

proposition *lists-a-Gset*:
alt-grp-act G (*X**) (*φ**)
⟨*proof*⟩
end

lemma *alt-group-act-is-grp-act* [*simp*, *GMN-simps*]:
alt-grp-act = *group-action*
⟨*proof*⟩

lemma *prod-group-act*:
assumes
grp-act-A: *alt-grp-act G A φ* **and**
grp-act-B: *alt-grp-act G B ψ*
shows *alt-grp-act G* (*A*×*B*) (λ*g*∈*carrier G*. λ(*a*, *b*) ∈ (*A* × *B*). (*φ g a*, *ψ g b*))
⟨*proof*⟩

1.2 Equivariance and Quotient Actions

locale *eq-var-subset* = *alt-grp-act G X φ*
for
G :: (′*grp*, ′*b*) *monoid-scheme* **and**
X :: ′*X set* (**structure**) **and**
φ +
fixes
Y
assumes
is-subset: *Y* ⊆ *X* **and**
is-equivar: ∀ *g*∈*carrier G*. (*φ g*) ‘ *Y* = *Y*

lemma (**in** *alt-grp-act*) *eq-var-one-direction*:
⋀*Y*. *Y* ⊆ *X* ⟹ ∀ *g*∈*carrier G*. (*φ g*) ‘ *Y* ⊆ *Y* ⟹ *eq-var-subset G X φ Y*
⟨*proof*⟩

The following lemmas are used for proofs in the locale `eq_var_rel`:

lemma *some-equiv-class-id*:
⟦*equiv X R*; *w* ∈ *X* // *R*; *x* ∈ *w*⟧ ⟹ *R* ‘‘ {*x*} = *R* ‘‘ {*SOME z*. *z* ∈ *w*}
⟨*proof*⟩

lemma *nested-somes*:
⟦*equiv X R*; *w* ∈ *X* // *R*⟧ ⟹ (*SOME z*. *z* ∈ *w*) = (*SOME z*. *z* ∈ *R*‘‘{(*SOME z*′. *z*′ ∈ *w*)})
⟨*proof*⟩

locale *eq-var-rel* = *alt-grp-act G X φ*
for
G :: (′*grp*, ′*b*) *monoid-scheme* **and**
X :: ′*X set* (**structure**) **and**
φ +

fixes R
assumes
is-subrel:
$R \subseteq X \times X$ **and**
is-eq-var-rel:
$\bigwedge a\ b.\ (a,\ b) \in R \Longrightarrow \forall\, g \in \mathit{carrier}\ G.\ (g \odot_\varphi a,\ g \odot_\varphi b) \in R$
begin

lemma *is-eq-var-rel′* [*simp*, *GMN-simps*]:
$\bigwedge a\ b.\ (a,\ b) \in R \Longrightarrow \forall\, g \in \mathit{carrier}\ G.\ ((\varphi\ g)\ a,\ (\varphi\ g)\ b) \in R$
⟨*proof*⟩

lemma *is-eq-var-rel-rev*:
$a \in X \Longrightarrow b \in X \Longrightarrow g \in \mathit{carrier}\ G \Longrightarrow (g \odot_\varphi a,\ g \odot_\varphi b) \in R \Longrightarrow (a,\ b) \in R$
⟨*proof*⟩

lemma *equiv-equivar-class-some-eq*:
assumes
A-0: *equiv* $X\ R$ **and**
A-1: $w \in X\ //\ R$ **and**
A-2: $g \in \mathit{carrier}\ G$
shows $([\ \varphi\ ]_R)\ g\ w = R\ ``\ \{(\mathit{SOME}\ z'.\ z' \in \varphi\ g\ `\ w)\}$
⟨*proof*⟩

lemma *ec-er-closed-under-action*:
assumes
A-0: *equiv* $X\ R$ **and**
A-1: $g \in \mathit{carrier}\ G$ **and**
A-2: $w \in X//R$
shows $\varphi\ g\ `\ w \in X\ //\ R$
⟨*proof*⟩

The following lemma corresponds to the first part of lemma 3.5 from [1]:

lemma *quot-act-wd*:
$[\![\mathit{equiv}\ X\ R;\ x \in X;\ g \in \mathit{carrier}\ G]\!] \Longrightarrow g \odot_{[\varphi]_R} (R\ ``\ \{x\}) = (R\ ``\ \{g \odot_\varphi x\})$
⟨*proof*⟩

The following lemma corresponds to the second part of lemma 3.5 from [1]:

lemma *quot-act-is-grp-act*:
$\mathit{equiv}\ X\ R \Longrightarrow \mathit{alt\text{-}grp\text{-}act}\ G\ (X\ //\ R)\ ([\varphi]_R)$
⟨*proof*⟩
end

locale *eq-var-func* = *GA-0*: *alt-grp-act* $G\ X\ \varphi$ + *GA-1*: *alt-grp-act* $G\ Y\ \psi$
for
$G :: ('grp,\ 'b)\ \mathit{monoid\text{-}scheme}$ **and**
$X :: 'X\ \mathit{set}$ **and**
φ **and**

$Y :: 'Y\ set$ **and**
ψ +
fixes
$f :: 'X \Rightarrow 'Y$
assumes
is-ext-func-bet:
$f \in (X \rightarrow_E Y)$ **and**
is-eq-var-func:
$\bigwedge a\ g.\ a \in X \Longrightarrow g \in carrier\ G \Longrightarrow f\ (g \odot_\varphi a) = g \odot_\psi (f\ a)$
begin

lemma *is-eq-var-func'* [*simp*]:
$a \in X \Longrightarrow g \in carrier\ G \Longrightarrow f\ (\varphi\ g\ a) = \psi\ g\ (f\ a)$
⟨*proof*⟩

end

lemma *G-set-equiv*:
alt-grp-act G A $\varphi \Longrightarrow$ *eq-var-subset G A* φ *A*
⟨*proof*⟩

1.3 Basic (G)-Automata Theory

locale *language* =
fixes $A :: 'alpha\ set$ **and**
L
assumes
is-lang: $L \subseteq A^\star$

locale *G-lang* = *alt-grp-act G A* φ + *language A L*
for
$G :: ('grp, 'b)\ monoid\text{-}scheme$ **and**
$A :: 'alpha\ set$ (**structure**) **and**
$\varphi\ L$ +
assumes
L-is-equivar:
eq-var-subset G $(A^\star)$ (*induced-star-map* φ) *L*
begin
lemma *G-lang-is-lang*[*simp*]: *language A L*
⟨*proof*⟩
end

sublocale *G-lang* $\subseteq$ *language*
⟨*proof*⟩

fun *give-input* :: $('state \Rightarrow 'alpha \Rightarrow 'state) \Rightarrow 'state \Rightarrow 'alpha\ list \Rightarrow 'state$
where *give-input trans-func s Nil* = *s*
| *give-input trans-func s* (*a*#*as*) = *give-input trans-func* (*trans-func s a*) *as*

adhoc-overloading
star $\rightleftharpoons$ *give-input*

locale *det-aut* =
fixes
labels :: *'alpha set* **and**
states :: *'state set* **and**
init-state :: *'state* **and**
fin-states :: *'state set* **and**
trans-func :: *'state* $\Rightarrow$ *'alpha* $\Rightarrow$ *'state* (‹δ›)
assumes
init-state-is-a-state:
init-state $\in$ *states* **and**
fin-states-are-states:
fin-states $\subseteq$ *states* **and**
trans-func-ext:
(λ(*state*, *label*). *trans-func state label*) $\in$ (*states* $\times$ *labels*) $\rightarrow_E$ *states*
begin

lemma *trans-func-well-def*:
$\bigwedge$*state label*. *state* $\in$ *states* $\Longrightarrow$ *label* $\in$ *labels* $\Longrightarrow$ (δ *state label*) $\in$ *states*
⟨*proof*⟩

lemma *give-input-closed*:
input $\in$ (*labels**) $\Longrightarrow$ *s* $\in$ *states* $\Longrightarrow$ ($\delta^\star$) *s input* $\in$ *states*
⟨*proof*⟩

lemma *input-under-concat*:
w $\in$ *labels** $\Longrightarrow$ *v* $\in$ *labels** $\Longrightarrow$ ($\delta^\star$) *s* (*w* @ *v*) = ($\delta^\star$) (($\delta^\star$) *s w*) *v*
⟨*proof*⟩

lemma *eq-pres-under-concat*:
assumes
w $\in$ *labels** **and**
w' $\in$ *labels** **and**
s $\in$ *states* **and**
($\delta^\star$) *s w* = ($\delta^\star$) *s w'*
shows $\forall$ *v* $\in$ *labels**. ($\delta^\star$) *s* (*w* @ *v*) = ($\delta^\star$) *s* (*w'* @ *v*)
⟨*proof*⟩

lemma *trans-to-charact*:
$\bigwedge$*s a w*. ⟦*s* $\in$ *states*; *a* $\in$ *labels*; *w* $\in$ *labels**; *s* = ($\delta^\star$) *i w*⟧ $\Longrightarrow$ ($\delta^\star$) *i* (*w* @ [*a*]) = δ *s a*
⟨*proof*⟩

end

locale *aut-hom* = *Aut0*: *det-aut A* S_0 i_0 F_0 δ_0 + *Aut1*: *det-aut A* S_1 i_1 F_1 δ_1 **for**
A :: *'alpha set* **and**

S_0 :: *'states-0 set* **and**
i_0 **and** F_0 **and** δ_0 **and**
S_1 :: *'states-1 set* **and**
i_1 **and** F_1 **and** δ_1 +
fixes f :: *'states-0* $\Rightarrow$ *'states-1*
assumes
hom-is-ext:
$f \in S_0 \rightarrow_E S_1$ **and**
pres-init:
$f\ i_0 = i_1$ **and**
pres-final:
$s \in F_0 \longleftrightarrow f\ s \in F_1 \wedge s \in S_0$ **and**
pres-trans:
$s_0 \in S_0 \Longrightarrow a \in A \Longrightarrow f\ (\delta_0\ s_0\ a) = \delta_1\ (f\ s_0)\ a$
begin

lemma *hom-translation*:
$input \in (A^\star) \Longrightarrow s \in S_0 \Longrightarrow (f\ ((\delta_0{}^\star)\ s\ input)) = ((\delta_1{}^\star)\ (f\ s)\ input)$
⟨*proof*⟩

lemma *recognise-same-lang*:
$input \in A^\star \Longrightarrow ((\delta_0{}^\star)\ i_0\ input) \in F_0 \longleftrightarrow ((\delta_1{}^\star)\ i_1\ input) \in F_1$
⟨*proof*⟩

end

locale *aut-epi* = *aut-hom* +
assumes
is-epi: f ‘ $S_0 = S_1$

locale *det-G-aut* =
is-aut: *det-aut* $A\ S\ i\ F\ \delta$ +
labels-a-G-set: *alt-grp-act* $G\ A\ \varphi$ +
states-a-G-set: *alt-grp-act* $G\ S\ \psi$ +
accepting-is-eq-var: *eq-var-subset* $G\ S\ \psi\ F$ +
init-is-eq-var: *eq-var-subset* $G\ S\ \psi\ \{i\}$ +
trans-is-eq-var: *eq-var-func* $G\ S \times A$
$\lambda g \in carrier\ G.\ \lambda(s,\ a) \in (S \times A).\ (\psi\ g\ s,\ \varphi\ g\ a)$
$S\ \psi\ (\lambda(s,\ a) \in (S \times A).\ \delta\ s\ a)$
for A :: *'alpha set* (**structure**) **and**
S :: *'states set* **and**
$i\ F\ \delta$ **and**
G :: (*'grp*, *'b*) *monoid-scheme* **and**
$\varphi\ \psi$
begin

adhoc-overloading
star $\rightleftharpoons$ *labels-a-G-set.induced-star-map*

lemma *give-input-eq-var*:
eq-var-func G
$(A^\star \times S)$ $(\lambda g{\in}carrier\ G.\ \lambda(w,\ s) \in (A^\star \times S).\ ((\varphi^\star)\ g\ w,\ \psi\ g\ s))$
$S\ \psi$
$(\lambda(w,\ s) \in (A^\star \times S).\ (\delta^\star)\ s\ w)$
⟨*proof*⟩

definition
accepted-words :: *'alpha list set*
where *accepted-words* = $\{w.\ w \in A^\star \wedge ((\delta^\star)\ i\ w) \in F\}$

lemma *induced-g-lang*:
G-lang G A φ *accepted-words*
⟨*proof*⟩
end

locale *reach-det-aut* =
det-aut A S i F δ
for *A* :: *'alpha set* (**structure**) **and**
S :: *'states set* **and**
i F δ +
assumes
is-reachable:
$s \in S \Longrightarrow \exists\, input \in A^\star.\ (\delta^\star)\ i\ input = s$

locale *reach-det-G-aut* =
det-G-aut A S i F δ *G* φ ψ + *reach-det-aut A S i F* δ
for *A* :: *'alpha set* (**structure**) **and**
S :: *'states set* **and**
i **and** *F* **and** δ **and**
G :: (*'grp*, *'b*) *monoid-scheme* **and**
φ ψ
begin

To avoid duplicate variant of "star":

no-adhoc-overloading
star ⇌ *labels-a-G-set.induced-star-map*
end

sublocale *reach-det-G-aut* ⊆ *reach-det-aut*
⟨*proof*⟩

locale *G-aut-hom* = *Aut0*: *reach-det-G-aut A* S_0 i_0 F_0 δ_0 *G* φ ψ_0 +
Aut1: *reach-det-G-aut A* S_1 i_1 F_1 δ_1 *G* φ ψ_1 +
hom-f: *aut-hom A* S_0 i_0 F_0 δ_0 S_1 i_1 F_1 δ_1 *f* +
eq-var-f: *eq-var-func G* S_0 ψ_0 S_1 ψ_1 *f* **for**
A :: *'alpha set* **and**
S_0 :: *'states-0 set* **and**
i_0 **and** F_0 **and** δ_0 **and**

S_1 :: *'states-1 set* **and**
i_1 **and** F_1 **and** δ_1 **and**
G :: (*'grp*, *'b*) *monoid-scheme* **and**
φ ψ_0 ψ_1 f

locale *G-aut-epi* = *G-aut-hom* +
assumes
is-epi: f ‘ $S_0 = S_1$

locale *det-aut-rec-lang* = *det-aut A S i F δ* + *language A L*
for A :: *'alpha set* (**structure**) **and**
S :: *'states set* **and**
i F δ L +
assumes
is-recognised:
$w \in L \longleftrightarrow w \in A^\star \wedge ((\delta^\star)\ i\ w) \in F$

locale *det-G-aut-rec-lang* = *det-G-aut A S i F δ G φ ψ* + *det-aut-rec-lang A S i F δ L*
for A :: *'alpha set* (**structure**) **and**
S :: *'states set* **and**
i F δ **and**
G :: (*'grp*, *'b*) *monoid-scheme* **and**
φ ψ L
begin

lemma *lang-is-G-lang*: *G-lang G A φ L*
⟨*proof*⟩

To avoid ambiguous parse trees:

no-notation *trans-is-eq-var.GA-0.induced-quot-map* (‹[-]_1› *60*)
no-notation *states-a-G-set.induced-quot-map* (‹[-]_1› *60*)

end

locale *reach-det-aut-rec-lang* = *reach-det-aut A S i F δ* + *det-aut-rec-lang A S i F δ L*
for A :: *'alpha set* **and**
S :: *'states set* **and**
i F δ **and**
L :: *'alpha list set*

locale *reach-det-G-aut-rec-lang* = *det-G-aut-rec-lang A S i F δ G φ ψ L* +
reach-det-G-aut A S i F δ G φ ψ
for A :: *'alpha set* **and**
S :: *'states set* **and**
i F δ **and**
G :: (*'grp*, *'b*) *monoid-scheme* **and**
φ ψ **and**

$L :: 'alpha\ list\ set$

sublocale *reach-det-G-aut-rec-lang* $\subseteq$ *det-G-aut-rec-lang*
⟨*proof*⟩

locale *det-G-aut-recog-G-lang* = *det-G-aut-rec-lang* $A\ S\ i\ F\ \delta\ G\ \varphi\ \psi\ L$ + *G-lang* $G\ A\ \varphi\ L$
for $A :: 'alpha\ set$ (**structure**) **and**
$S :: 'states\ set$ **and**
$i\ F\ \delta$ **and**
$G :: ('grp, 'b)\ monoid\text{-}scheme$ **and**
$\varphi\ \psi$ **and**
$L :: 'alpha\ list\ set$

sublocale *det-G-aut-rec-lang* $\subseteq$ *det-G-aut-recog-G-lang*
⟨*proof*⟩

locale *reach-det-G-aut-rec-G-lang* = *reach-det-G-aut-rec-lang* $A\ S\ i\ F\ \delta\ G\ \varphi\ \psi\ L$ + *G-lang* $G\ A\ \varphi\ L$
for $A :: 'alpha\ set$ (**structure**) **and**
$S :: 'states\ set$ **and**
$i\ F\ \delta$ **and**
$G :: ('grp, 'b)\ monoid\text{-}scheme$ **and**
$\varphi\ \psi\ L$

sublocale *reach-det-G-aut-rec-lang* $\subseteq$ *reach-det-G-aut-rec-G-lang*
⟨*proof*⟩

lemma (**in** *reach-det-G-aut*)
reach-det-G-aut-rec-lang $A\ S\ i\ F\ \delta\ G\ \varphi\ \psi$ *accepted-words*
⟨*proof*⟩

lemma (**in** *det-G-aut*) *action-on-input*:
$\bigwedge g\ w.\ g \in carrier\ G \Longrightarrow w \in A^\star \Longrightarrow \psi\ g\ ((\delta^\star)\ i\ w) = (\delta^\star)\ i\ ((\varphi^\star)\ g\ w)$
⟨*proof*⟩

definition (**in** *det-G-aut*)
reachable-states $:: 'states\ set$ $(\langle S_{reach} \rangle)$
where $S_{reach} = \{s\ .\ \exists\ w \in A^\star.\ (\delta^\star)\ i\ w = s\}$

definition (**in** *det-G-aut*)
reachable-trans $:: 'states \Rightarrow 'alpha \Rightarrow 'states$ $(\langle \delta_{reach} \rangle)$
where $\delta_{reach}\ s\ a = (\lambda(s', a') \in S_{reach} \times A.\ \delta\ s'\ a')\ (s, a)$

definition (**in** *det-G-aut*)
reachable-action $:: 'grp \Rightarrow 'states \Rightarrow 'states$ $(\langle \psi_{reach} \rangle)$
where $\psi_{reach}\ g\ s = (\lambda(g', s') \in carrier\ G \times S_{reach}.\ \psi\ g'\ s')\ (g, s)$

lemma (**in** *det-G-aut*) *reachable-action-is-restict*:

$\bigwedge g\ s.\ g \in carrier\ G \Longrightarrow s \in S_{reach} \Longrightarrow \psi_{reach}\ g\ s = \psi\ g\ s$
⟨*proof*⟩

lemma (**in** *det-G-aut-rec-lang*) *reach-det-aut-is-det-aut-rec-L*:
reach-det-G-aut-rec-lang A S_{reach} *i* $(F \cap S_{reach})$ δ_{reach} *G* φ ψ_{reach} *L*
⟨*proof*⟩

1.4 Syntactic Automaton

context *language* **begin**

definition
rel-MN :: (*'alpha list* × *'alpha list*) *set* (‹$\equiv_{MN}$›)
where *rel-MN* = {(*w*, *w'*) ∈ $(A^\star)\times(A^\star)$. (∀ *v* ∈ $A^\star$. (*w* @ *v*) ∈ *L* ⟷ (*w'* @ *v*) ∈ *L*)}

lemma *MN-rel-equival*:
equiv ($A^\star$) *rel-MN*
⟨*proof*⟩

abbreviation
MN-equiv
where *MN-equiv* ≡ $A^\star$ // *rel-MN*

definition
alt-natural-map-MN :: *'alpha list* ⇒ *'alpha list set* (‹$[\text{-}]_{MN}$›)
where $[w]_{MN}$ = *rel-MN* `` {*w*}

definition
MN-trans-func :: (*'alpha list set*) ⇒ *'alpha* ⇒ *'alpha list set* (‹δ_{MN}›)
where *MN-trans-func W' a'* =
(λ(*W*,*a*) ∈ *MN-equiv* × *A*. *rel-MN* `` {(*SOME w*. *w* ∈ *W*) @ [*a*]}) (*W'*, *a'*)

abbreviation
MN-init-state
where *MN-init-state* ≡ $[Nil::'alpha\ list]_{MN}$

abbreviation
MN-fin-states
where *MN-fin-states* ≡ {*v*. ∃ *w*∈*L*. *v* = $[w]_{MN}$}

lemmas
alt-natural-map-MN-def [*simp*, *GMN-simps*]
MN-trans-func-def [*simp*, *GMN-simps*]
end

context *G-lang* **begin**
adhoc-overloading
star ⇌ *induced-star-map*

lemma *MN-quot-act-wd*:
$w' \in [w]_{MN} \Longrightarrow \forall\, g \in$ *carrier* $G.\ (g \odot_{\varphi^\star}\ w') \in [g \odot_{\varphi^\star}\ w]_{MN}$
⟨*proof*⟩

The following lemma corresponds to lemma 3.4 from [1]:

lemma *MN-rel-eq-var*:
eq-var-rel $G\ (A^\star)\ (\varphi^\star) \equiv_{MN}$
⟨*proof*⟩

lemma *quot-act-wd-alt-notation*:
$w \in A^\star \Longrightarrow g \in$ *carrier* $G \Longrightarrow g \odot_{[\varphi^\star]_{\equiv MN}\ A^\star}\ ([w]_{MN}) = ([g \odot_{\varphi^\star}\ w]_{MN})$
⟨*proof*⟩

lemma *MN-trans-func-characterization*:
$v \in (A^\star) \Longrightarrow a \in A \Longrightarrow \delta_{MN}\ [v]_{MN}\ a = [v\ @\ [a]]_{MN}$
⟨*proof*⟩

lemma *MN-trans-eq-var-func* :
eq-var-func G
(*MN-equiv* $\times$ A) ($\lambda g \in$*carrier* $G.\ \lambda(W,\ a) \in$ (*MN-equiv* $\times$ A). (([$(\varphi^\star)$]$_{\equiv MN\ A^\star}$) g $W,\ \varphi\ g\ a$))
MN-equiv ([$(\varphi^\star)$]$_{\equiv MN\ A^\star}$)
($\lambda(w,\ a) \in$ *MN-equiv* $\times$ $A.\ \delta_{MN}\ w\ a$)
⟨*proof*⟩

lemma *MN-quot-act-on-empty-str*:
$\bigwedge g.\ [\![g \in$ *carrier* $G;\ ([],\ x) \in \equiv_{MN}]\!] \Longrightarrow x \in$ *map* $(\varphi\ g)$ ‘ $\equiv_{MN}$ ‘‘ $\{[]\}$
⟨*proof*⟩

lemma *MN-init-state-equivar*:
eq-var-subset $G\ (A^\star)\ (\varphi^\star)$ *MN-init-state*
⟨*proof*⟩

lemma *MN-init-state-equivar-v2*:
eq-var-subset G (*MN-equiv*) ([$\varphi^\star$]$_{\equiv MN\ A^\star}$) {*MN-init-state*}
⟨*proof*⟩

lemma *MN-final-state-equiv*:
eq-var-subset G (*MN-equiv*) ([$\varphi^\star$]$_{\equiv MN\ A^\star}$) *MN-fin-states*
⟨*proof*⟩

interpretation *syntac-aut* :
det-aut A MN-equiv MN-init-state MN-fin-states MN-trans-func
⟨*proof*⟩

corollary *syth-aut-is-det-aut*:
det-aut A MN-equiv MN-init-state MN-fin-states δ_{MN}
⟨*proof*⟩

lemma *give-input-transition-func*:
$w \in (A^\star) \Longrightarrow \forall\, v \in (A^\star).\ [v\ @\ w]_{MN} = (\delta_{MN}{}^\star)\ [v]_{MN}\ w$
⟨*proof*⟩

lemma *MN-unique-init-state*:
$w \in (A^\star) \Longrightarrow [w]_{MN} = (\delta_{MN}{}^\star)\ [Nil]_{MN}\ w$
⟨*proof*⟩

lemma *fin-states-rep-by-lang*:
$w \in A^\star \Longrightarrow [w]_{MN} \in$ *MN-fin-states* $\Longrightarrow w \in L$
⟨*proof*⟩

The following lemma corresponds to lemma 3.6 from [1]:

lemma *syntactic-aut-det-G-aut*:
det-G-aut A MN-equiv MN-init-state MN-fin-states MN-trans-func G φ $([\varphi^\star]_{\equiv MN}$ $A^\star)$
⟨*proof*⟩

lemma *syntactic-aut-det-G-aut-rec-L*:
det-G-aut-rec-lang A MN-equiv MN-init-state MN-fin-states MN-trans-func G φ $([\varphi^\star]_{\equiv MN}\ A^\star)\ L$
⟨*proof*⟩

lemma *syntact-aut-is-reach-aut-rec-lang*:
reach-det-G-aut-rec-lang A MN-equiv MN-init-state MN-fin-states MN-trans-func G φ $([\varphi^\star]_{\equiv MN} A^\star)\ L$
⟨*proof*⟩
end

1.5 Proving the Myhill-Nerode Theorem for G-Automata

context *det-G-aut* **begin**
no-adhoc-overloading
star $\rightleftharpoons$ *labels-a-G-set.induced-star-map*
end

context *reach-det-G-aut-rec-lang* **begin**
adhoc-overloading
star $\rightleftharpoons$ *labels-a-G-set.induced-star-map*

definition
states-to-words :: *'states* ⇒ *'alpha list*
where *states-to-words* $= (\lambda s \in S.\ SOME\ w.\ w \in A^\star \wedge ((\delta^\star)\ i\ w = s))$

definition
words-to-syth-states :: *'alpha list* ⇒ *'alpha list set*
where *words-to-syth-states* $w = [w]_{MN}$

definition
induced-epi:: *'states* ⇒ *'alpha list set*
where *induced-epi* = *compose S words-to-syth-states states-to-words*

lemma *induced-epi-wd1*:
$s \in S \Longrightarrow \exists\, w.\ w \in A^\star \wedge ((\delta^\star)\ i\ w = s)$
⟨*proof*⟩

lemma *induced-epi-wd2*:
$w \in A^\star \Longrightarrow w' \in A^\star \Longrightarrow (\delta^\star)\ i\ w = (\delta^\star)\ i\ w' \Longrightarrow [w]_{MN} = [w']_{MN}$
⟨*proof*⟩

lemma *states-to-words-on-final*:
states-to-words ∈ $(F \to L)$
⟨*proof*⟩

lemma *induced-epi-eq-var*:
eq-var-func G S ψ MN-equiv $([(\varphi^\star)]_{\equiv MN\ A^\star})$ *induced-epi*
⟨*proof*⟩

The following lemma corresponds to lemma 3.7 from [1]:

lemma *reach-det-G-aut-rec-lang*:
G-aut-epi A S i F δ MN-equiv MN-init-state MN-fin-states δ_{MN} *G φ ψ* $([(\varphi^\star)]_{\equiv MN\ A^\star})$ *induced-epi*
⟨*proof*⟩

end

lemma (**in** *det-G-aut*) *finite-reachable*:
finite (*orbits G S ψ*) ⟹ *finite* (*orbits G* S_{reach} ψ_{reach})
⟨*proof*⟩

lemma (**in** *det-G-aut*)
orbs-pos-card: *finite* (*orbits G S ψ*) ⟹ *card* (*orbits G S ψ*) > *0*
⟨*proof*⟩

lemma (**in** *reach-det-G-aut-rec-lang*) *MN-B2T*:
assumes
Fin: *finite* (*orbits G S ψ*)
shows
finite (*orbits G* (*language.MN-equiv A L*) $(([(\varphi^\star)]_{\equiv MN\ A^\star}))$
⟨*proof*⟩

context *det-G-aut-rec-lang* **begin**

To avoid duplicate variant of "star":

no-adhoc-overloading

star $\rightleftharpoons$ *labels-a-G-set.induced-star-map*

end

context *det-G-aut-rec-lang* **begin**
adhoc-overloading
star $\rightleftharpoons$ *labels-a-G-set.induced-star-map*
end

lemma (**in** *det-G-aut-rec-lang*) *MN-prep*:
$\exists\, S'.\ \exists\, \delta'.\ \exists\, F'.\ \exists\, \psi'.$
(*reach-det-G-aut-rec-lang A S' i F' δ' G φ ψ' L* $\wedge$
(*finite* (*orbits G S ψ*) $\longrightarrow$ *finite* (*orbits G S' ψ'*)))
⟨*proof*⟩

lemma (**in** *det-G-aut-rec-lang*) *MN-fin-orbs-imp-fin-states*:
assumes
Fin: *finite* (*orbits G S ψ*)
shows
finite (*orbits G* (*language.MN-equiv A L*) (([$(\varphi^\star)$]$_{\equiv MN\ A^\star}$)))
⟨*proof*⟩

The following theorem corresponds to theorem 3.8 from [1], i.e. the Myhill-Nerode theorem for G-automata. The left to right direction (see statement below) of the typical Myhill-Nerode theorem would qantify over types (if some condition holds, then there exists some automaton accepting the language). As it is not possible to qantify over types in this way, the equivalence is spit into two directions. In the left to right direction, the explicit type of the syntactic automaton is used. In the right to left direction some type, 's, is fixed. As the two directions are split, the opertunity was taken to strengthen the right to left direction: We do not assume the given automaton is reachable.

This splitting of the directions will be present in all other Myhill-Nerode theorems that will be proved in this document.

theorem (**in** *G-lang*) *G-Myhill-Nerode* :
assumes
finite (*orbits G A φ*)
shows
G-Myhill-Nerode-LR: *finite* (*orbits G MN-equiv* ([$(\varphi^\star)$]$_{\equiv MN\ A^\star}$)) $\Longrightarrow$
($\exists\, S\ F$ :: *'alpha list set set*. $\exists\, i$ :: *'alpha list set*. $\exists\, \delta$. $\exists\, \psi$.
reach-det-G-aut-rec-lang A S i F δ G φ ψ L $\wedge$ *finite* (*orbits G S ψ*)) **and**
G-Myhill-Nerode-RL: ($\exists\, S\ F$ :: *'s set*. $\exists\, i$ :: *'s*. $\exists\, \delta$. $\exists\, \psi$.
det-G-aut-rec-lang A S i F δ G φ ψ L $\wedge$ *finite* (*orbits G S ψ*))
$\Longrightarrow$ *finite* (*orbits G MN-equiv* ([$(\varphi^\star)$]$_{\equiv MN\ A^\star}$))
⟨*proof*⟩

1.6 Proving the standard Myhill-Nerode Theorem

Any automaton is a G-automaton with respect to the trivial group and action, hence the standard Myhill-Nerode theorem is a special case of the G-Myhill-Nerode theorem.

interpretation *triv-act*:
alt-grp-act singleton-group (*undefined*) *X* (*λx* ∈ {*undefined*}. *one* (*BijGroup X*))
⟨*proof*⟩

lemma (**in** *det-aut*) *triv-G-aut*:
fixes *triv-G*
assumes *H-triv-G*: *triv-G* = (*singleton-group* (*undefined*))
shows *det-G-aut labels states init-state fin-states δ*
triv-G (*λx* ∈ {*undefined*}. *one* (*BijGroup labels*)) (*λx* ∈ {*undefined*}. *one* (*BijGroup states*))
⟨*proof*⟩

lemma *triv-orbits*:
orbits (*singleton-group* (*undefined*)) *S* (*λx* ∈ {*undefined*}. *one* (*BijGroup S*)) =
{{*s*} |*s*. *s* ∈ *S*}
⟨*proof*⟩

lemma *fin-triv-orbs*:
finite (*orbits* (*singleton-group* (*undefined*)) *S* (*λx* ∈ {*undefined*}. *one* (*BijGroup S*))) = *finite S*
⟨*proof*⟩

context *language* **begin**

interpretation *triv-G-lang*:
G-lang singleton-group (*undefined*) *A* (*λx* ∈ {*undefined*}. *one* (*BijGroup A*)) *L*
⟨*proof*⟩

definition *triv-G* :: *'grp monoid*
where *triv-G* = (*singleton-group* (*undefined*))

definition *triv-act* :: *'grp* ⇒ *'alpha* ⇒ *'alpha*
where *triv-act* = (*λx* ∈ {*undefined*}. $\mathbf{1}_{BijGroup\ A}$)

corollary *standard-Myhill-Nerode*:
assumes
H-fin-alph: *finite A*
shows
standard-Myhill-Nerode-LR: *finite MN-equiv* ⟹
(∃ *S F* :: *'alpha list set set*. ∃ *i* :: *'alpha list set*. ∃ *δ*.
reach-det-aut-rec-lang A S i F δ L ∧ *finite S*) **and**
standard-Myhill-Nerode-RL: (∃ *S F* :: *'s set*. ∃ *i* :: *'s*. ∃ *δ*.
det-aut-rec-lang A S i F δ L ∧ *finite S*) ⟹ *finite MN-equiv*
⟨*proof*⟩

end

2 Myhill-Nerode Theorem for Nominal G-Automata

2.1 Data Symmetries, Supports and Nominal Actions

The following locale corresponds to the definition 2.2 from [1]. Note that we let G be an arbitrary group instead of a subgroup of `BijGroup` D, but assume there is a homomoprhism $\pi : G \to$ `BijGroup`D. By `group_hom.img_is_subgroup` this is an equivalent definition:

locale *data-symm* = *group-action* G D π
  for
    G :: ('*grp*, '*b*) *monoid-scheme* **and**
    D :: '*D set* (‹𝔻›) **and**
    π

The following locales corresponds to definition 4.3 from [1]:

locale *supports* = *data-symm* G D π + *alt-grp-act* G X φ
  for
    G :: ('*grp*, '*b*) *monoid-scheme* **and**
    D :: '*D set* (‹𝔻›) **and**
    π **and**
    X :: '*X set* (**structure**) **and**
    φ +
  fixes
    C :: '*D set* **and**
    x :: '*X*
  assumes
    is-in-set:
    $x \in X$ **and**
    is-subset:
    $C \subseteq \mathbb{D}$ **and**
    supports:
    $g \in$ *carrier* $G \Longrightarrow (\forall\, c.\ c \in C \longrightarrow \pi\ g\ c = c) \Longrightarrow g \odot_\varphi x = x$
begin

The following lemma corresponds to lemma 4.9 from [1]:

lemma *image-supports*:
  $\bigwedge g.\ g \in$ *carrier* $G \Longrightarrow$ *supports* G D π X φ $(\pi\ g\ `\ C)$ $(g \odot_\varphi x)$
⟨*proof*⟩
end

locale *nominal* = *data-symm* G D π + *alt-grp-act* G X φ
  for
    G :: ('*grp*, '*b*) *monoid-scheme* **and**
    D :: '*D set* (‹𝔻›) **and**
    π **and**
    X :: '*X set* (**structure**) **and**

φ +
assumes
is-nominal:
$\bigwedge g\ x.\ g \in carrier\ G \Longrightarrow x \in X \Longrightarrow \exists\, C.\ C \subseteq \mathbb{D} \wedge finite\ C \wedge supports\ G\ \mathbb{D}\ \pi\ X\ \varphi\ C\ x$

locale *nominal-det-G-aut* = *det-G-aut* +
nominal G D π A φ + *nominal G D π S ψ*
for
D :: *'D set* (‹𝔻›) **and**
π

The following lemma corresponds to lemma 4.8 from [1]:

lemma (**in** *eq-var-func*) *supp-el-pres*:
supports G D π X φ C x ⟹ *supports G D π Y ψ C* (*f x*)
⟨*proof*⟩

lemma (**in** *nominal*) *support-union-lem*:
fixes *f sup-C col*
assumes *H-f*: $f = (\lambda x.\ (SOME\ C.\ C \subseteq \mathbb{D} \wedge finite\ C \wedge supports\ G\ \mathbb{D}\ \pi\ X\ \varphi\ C\ x))$
and *H-col*: *col* ⊆ *X* ∧ *finite col*
and *H-sup-C*: *sup-C* = ⋃{*Cx*. *Cx* ∈ *f* ‘ *col*}
shows $\bigwedge x.\ x \in col \Longrightarrow sup\text{-}C \subseteq \mathbb{D} \wedge finite\ sup\text{-}C \wedge supports\ G\ \mathbb{D}\ \pi\ X\ \varphi\ sup\text{-}C\ x$
⟨*proof*⟩

lemma (**in** *nominal*) *set-of-list-nom*:
nominal G D π (*X*⋆) (*φ*⋆)
⟨*proof*⟩

2.2 Proving the Myhill-Nerode Theorem for Nominal G-Automata

context *det-G-aut* **begin**
adhoc-overloading
star ⇌ *labels-a-G-set.induced-star-map*
end

lemma (**in** *det-G-aut*) *input-to-init-eqvar*:
eq-var-func G (*A*⋆) (*φ*⋆) *S ψ* (*λw*∈*A*⋆. (*δ*⋆) *i w*)
⟨*proof*⟩

lemma (**in** *reach-det-G-aut*) *input-to-init-surj*:
(*λw*∈*A*⋆. (*δ*⋆) *i w*) ‘ (*A*⋆) = *S*
⟨*proof*⟩

context *reach-det-G-aut* **begin**
adhoc-overloading
star ⇌ *labels-a-G-set.induced-star-map*
end

The following lemma corresponds to proposition 5.1 from [1]:

proposition (**in** *reach-det-G-aut*) *alpha-nom-imp-states-nom*:
nominal G D π A φ $\Longrightarrow$ *nominal G D π S ψ*
⟨*proof*⟩

The following theorem corresponds to theorem 5.2 from [1]:

theorem (**in** *G-lang*) *Nom-G-Myhill-Nerode*:
assumes
orb-fin: *finite* (*orbits G A φ*) **and**
nom: *nominal G D π A φ*
shows
Nom-G-Myhill-Nerode-LR: *finite* (*orbits G MN-equiv* ($[(\varphi^\star)]_{\equiv_{M\,N}\ A^\star}$)) $\Longrightarrow$
($\exists$ *S F* :: *'alpha list set set*. $\exists$ *i* :: *'alpha list set*. $\exists\,\delta$. $\exists\,\psi$.
reach-det-G-aut-rec-lang A S i F δ G φ ψ L $\wedge$ *finite* (*orbits G S ψ*)
$\wedge$ *nominal-det-G-aut A S i F δ G φ ψ D π*) **and**
Nom-G-Myhill-Nerode-RL: ($\exists$ *S F* :: *'s set*. $\exists$ *i* :: *'s*. $\exists\,\delta$. $\exists\,\psi$.
det-G-aut-rec-lang A S i F δ G φ ψ L $\wedge$ *finite* (*orbits G S ψ*)
$\wedge$ *nominal-det-G-aut A S i F δ G φ ψ D π*)
$\Longrightarrow$ *finite* (*orbits G MN-equiv* ($[(\varphi^\star)]_{\equiv_{M\,N}\ A^\star}$))
⟨*proof*⟩
end

References

[1] M. Bojańczyk, B. Klin, and S. Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10, 2014.