

Myhill-Nerode Theorem for (Nominal) G -Automata

Cárolos Laméris

February 6, 2026

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_{MN} , with respect to the Myhill-Nerode equivalence relation, \equiv_{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	1
1.1	Extending Group Actions	2
1.2	Equivariance and Quotient Actions	4
1.3	Basic (G)-Automata Theory	6
1.4	Syntactic Automaton	12
1.5	Proving the Myhill-Nerode Theorem for G -Automata	14
1.6	Proving the standard Myhill-Nerode Theorem	17
2	Myhill-Nerode Theorem for Nominal G-Automata	18
2.1	Data Symmetries, Supports and Nominal Actions	18
2.2	Proving the Myhill-Nerode Theorem for Nominal G -Automata	19

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 ϕ^\star and for the extended transition function δ^\star , thus we introduce the map **star** and apply **adhoc_overloading** to get the notation working in all three situations:

consts *star* :: 'typ1 \Rightarrow 'typ2 $\langle \langle \star \rangle [1000] 999 \rangle$

adhoc-overloading

star \Rightarrow lists

We use \odot to convert between the definition of group actions via group homomorphisms 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 \Rightarrow 'X \Rightarrow 'X) \Rightarrow 'grp \Rightarrow 'X \Rightarrow 'X (**infixl** $\langle \langle \odot \rangle 70 \rangle$)

where $\langle \odot \rangle \varphi \equiv (\lambda g. (\lambda x. \varphi g x))$

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*:

$\llbracket S \subseteq T; x \in T \wedge f \in \text{Bij } T; (\text{restrict } f S) \text{ ‘ } S = S; f x \in S \rrbracket \implies x \in S$
 $\langle \text{proof} \rangle$

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* $G X \varphi$
for
 $G :: ('grp, 'b) \text{ monoid-scheme}$ **and**
 $X :: 'X \text{ set}$ (**structure**) **and**
 φ
begin

lemma *alt-grp-act-is-left-grp-act*:

shows $x \in X \implies \mathbf{1}_G \odot_\varphi x = x$ **and**
 $g \in \text{carrier } G \implies h \in \text{carrier } G \implies x \in X \implies (g \otimes_G h) \odot_\varphi x = g \odot_\varphi (h \odot_\varphi x)$
 $\langle \text{proof} \rangle$

definition

induced-star-map $:: ('grp \Rightarrow 'X \Rightarrow 'X) \Rightarrow 'grp \Rightarrow 'X \text{ list} \Rightarrow 'X \text{ list}$
where *induced-star-map* *func* = $(\lambda g \in \text{carrier } G. (\lambda lst \in X^*. \text{map } (\text{func } g) lst))$

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

adhoc-overloading

$star \rightleftharpoons \text{induced-star-map}$

definition

induced-quot-map $::$
 $'Y \text{ set} \Rightarrow ('grp \Rightarrow 'Y \Rightarrow 'Y) \Rightarrow ('Y \times 'Y) \text{ set} \Rightarrow 'grp \Rightarrow 'Y \text{ set} \Rightarrow 'Y \text{ set} (\langle [-]_{-1} \rangle$
 $60)$
where $([\text{func}]_R S) = (\lambda g \in \text{carrier } G. (\lambda x \in (S // R). R \text{ “ } \{(\text{func } g) (SOME z. z \in x) \}))$

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

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

lemma *act-maps-n-distrib*:

$\forall g \in \text{carrier } G. \forall w \in X^*. \forall v \in X^*. (\varphi^*) g (w @ v) = ((\varphi^*) g w) @ ((\varphi^*) g v)$
 $\langle \text{proof} \rangle$

lemma *triv-act*:

$a \in X \implies (\varphi \mathbf{1}_G) a = a$
 $\langle \text{proof} \rangle$

lemma *triv-act-map*:

$\forall w \in X^*. ((\varphi^*) \mathbf{1}_G) w = w$

<proof>

proposition *lists-a-Gset:*

alt-grp-act $G (X^*) (\varphi^*)$

<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 $G A \varphi$ **and**

grp-act-B: alt-grp-act $G B \psi$

shows *alt-grp-act* $G (A \times B) (\lambda g \in \text{carrier } G. \lambda(a, b) \in (A \times B). (\varphi g a, \psi g b))$

<proof>

1.2 Equivariance and Quotient Actions

locale *eq-var-subset* = *alt-grp-act* $G X \varphi$

for

$G :: ('grp, 'b)$ *monoid-scheme* **and**

$X :: 'X$ *set* (**structure**) **and**

$\varphi +$

fixes

Y

assumes

is-subset: Y $\subseteq X$ **and**

is-equivar: $\forall g \in \text{carrier } G. (\varphi g) ' Y = Y$

lemma (**in** *alt-grp-act*) *eq-var-one-direction:*

$\bigwedge Y. Y \subseteq X \implies \forall g \in \text{carrier } G. (\varphi g) ' Y \subseteq Y \implies \text{eq-var-subset } G X \varphi Y$

<proof>

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

lemma *some-equiv-class-id:*

$\llbracket \text{equiv } X R; w \in X // R; x \in w \rrbracket \implies R '' \{x\} = R '' \{\text{SOME } z. z \in w\}$

<proof>

lemma *nested-somes:*

$\llbracket \text{equiv } X R; w \in X // R \rrbracket \implies (\text{SOME } z. z \in w) = (\text{SOME } z. z \in R '' \{(\text{SOME } z'. z' \in w)\})$

<proof>

locale *eq-var-rel* = *alt-grp-act* $G X \varphi$

for

$G :: ('grp, 'b)$ *monoid-scheme* **and**

$X :: 'X$ *set* (**structure**) **and**

$\varphi +$

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

lemma *is-eq-var-rel'* [*simp, GMN-simps*]:
 $\bigwedge a b. (a, b) \in R \implies \forall g \in \text{carrier } G. ((\varphi g) a, (\varphi g) b) \in R$
<proof>

lemma *is-eq-var-rel-rev:*
 $a \in X \implies b \in X \implies g \in \text{carrier } G \implies (g \odot_{\varphi} a, g \odot_{\varphi} b) \in R \implies (a, b) \in R$
<proof>

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

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

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

lemma *quot-act-wd:*
 $\llbracket \text{equiv } X R; x \in X; g \in \text{carrier } G \rrbracket \implies g \odot_{[\varphi]_R} (R \text{ `` } \{x\}) = (R \text{ `` } \{g \odot_{\varphi} x\})$
<proof>

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

lemma *quot-act-is-grp-act:*
 $\text{equiv } X R \implies \text{alt-grp-act } G (X // R) ([\varphi]_R)$
<proof>
end

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

```

    Y :: 'Y set and
    ψ +
fixes
    f :: 'X ⇒ 'Y
assumes
    is-ext-func-bet:
    f ∈ (X →E Y) and
    is-eq-var-func:
    ∧ a g. a ∈ X ⇒ g ∈ carrier G ⇒ f (g ⊙φ a) = g ⊙ψ (f a)
begin

```

```

lemma is-eq-var-func' [simp]:
    a ∈ X ⇒ g ∈ carrier G ⇒ f (φ g a) = ψ g (f a)
    ⟨proof⟩

```

end

```

lemma G-set-equiv:
    alt-grp-act G A φ ⇒ 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 ⊆ A*

```

```

locale G-lang = alt-grp-act G A φ + language A L
for
    G :: ('grp, 'b) monoid-scheme and
    A :: 'alpha set (structure) and
    φ L +
assumes
    L-is-equivar:
    eq-var-subset G (A*) (induced-star-map φ) L

```

```

begin
lemma G-lang-is-lang[simp]: language A L
    ⟨proof⟩
end

```

```

sublocale G-lang ⊆ language
    ⟨proof⟩

```

```

fun give-input :: ('state ⇒ 'alpha ⇒ 'state) ⇒ 'state ⇒ 'alpha list ⇒ '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 $star \Rightarrow give-input$ **locale** *det-aut* =**fixes** $labels :: 'alpha\ set\ \mathbf{and}$ $states :: 'state\ set\ \mathbf{and}$ $init-state :: 'state\ \mathbf{and}$ $fin-states :: 'state\ set\ \mathbf{and}$ $trans-func :: 'state \Rightarrow 'alpha \Rightarrow 'state\ (\iota\delta)$ **assumes***init-state-is-a-state:* $init-state \in states\ \mathbf{and}$ *fin-states-are-states:* $fin-states \subseteq states\ \mathbf{and}$ *trans-func-ext:* $(\lambda(state, label). trans-func\ state\ label) \in (states \times labels) \rightarrow_E states$ **begin****lemma** *trans-func-well-def:* $\bigwedge state\ label. state \in states \Longrightarrow label \in labels \Longrightarrow (\delta\ state\ label) \in states$ *<proof>***lemma** *give-input-closed:* $input \in (labels^*) \Longrightarrow s \in states \Longrightarrow (\delta^*)\ s\ input \in states$ *<proof>***lemma** *input-under-concat:* $w \in labels^* \Longrightarrow v \in labels^* \Longrightarrow (\delta^*)\ s\ (w\ @\ v) = (\delta^*)\ ((\delta^*)\ s\ w)\ v$ *<proof>***lemma** *eq-pres-under-concat:***assumes** $w \in labels^*\ \mathbf{and}$ $w' \in labels^*\ \mathbf{and}$ $s \in states\ \mathbf{and}$ $(\delta^*)\ s\ w = (\delta^*)\ s\ w'$ **shows** $\forall v \in labels^*. (\delta^*)\ s\ (w\ @\ v) = (\delta^*)\ s\ (w'\ @\ v)$ *<proof>***lemma** *trans-to-charact:* $\bigwedge s\ a\ w. \llbracket s \in states; a \in labels; w \in labels^*; s = (\delta^*)\ i\ w \rrbracket \Longrightarrow (\delta^*)\ i\ (w\ @\ [a]) = \delta\ s\ a$ *<proof>***end****locale** *aut-hom* = *Aut0: det-aut* $A\ S_0\ i_0\ F_0\ \delta_0$ + *Aut1: det-aut* $A\ S_1\ i_1\ F_1\ \delta_1$ **for** $A :: 'alpha\ set\ \mathbf{and}$

$S_0 :: \text{'states-0 set and}$
 $i_0 \text{ and } F_0 \text{ and } \delta_0 \text{ and}$
 $S_1 :: \text{'states-1 set and}$
 $i_1 \text{ and } F_1 \text{ and } \delta_1 +$
fixes $f :: \text{'states-0} \Rightarrow \text{'states-1}$
assumes
hom-is-ext:
 $f \in S_0 \rightarrow_E S_1 \text{ and}$
pres-init:
 $f i_0 = i_1 \text{ and}$
pres-final:
 $s \in F_0 \longleftrightarrow f s \in F_1 \wedge s \in S_0 \text{ and}$
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:*
 $input \in (A^*) \Longrightarrow s \in S_0 \Longrightarrow (f ((\delta_0^*) s input)) = ((\delta_1^*) (f s) input)$
<proof>

lemma *recognise-same-lang:*
 $input \in A^* \Longrightarrow ((\delta_0^*) i_0 input) \in F_0 \longleftrightarrow ((\delta_1^*) i_1 input) \in F_1$
<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 $A :: \text{'alpha set (structure) and}$
 $S :: \text{'states set and}$
 $i F \delta \text{ and}$
 $G :: (\text{'grp, 'b) monoid-scheme and}$
 $\varphi \psi$
begin

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

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

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

lemma *induced-g-lang*:
G-lang $G A \varphi$ *accepted-words*
 $\langle \text{proof} \rangle$
end

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

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

To avoid duplicate variant of "star":

no-adhoc-overloading
 $\text{star} \rightleftharpoons \text{labels-a-G-set.induced-star-map}$
end

sublocale *reach-det-G-aut* \subseteq *reach-det-aut*
 $\langle \text{proof} \rangle$

locale *G-aut-hom* = *Aut0*: *reach-det-G-aut* $A S_0 i_0 F_0 \delta_0 G \varphi \psi_0$ +
Aut1: *reach-det-G-aut* $A S_1 i_1 F_1 \delta_1 G \varphi \psi_1$ +
hom-f: *aut-hom* $A S_0 i_0 F_0 \delta_0 S_1 i_1 F_1 \delta_1 f$ +
eq-var-f: *eq-var-func* $G S_0 \psi_0 S_1 \psi_1 f$ **for**
 A :: 'alpha set **and**
 S_0 :: 'states-0 set **and**
 i_0 **and** F_0 **and** δ_0 **and**

```

    S1 :: 'states-1 set and
    i1 and F1 and δ1 and
    G :: ('grp, 'b) monoid-scheme and
    φ ψ0 ψ1 f

locale G-aut-epi = G-aut-hom +
assumes
    is-epi: f ' S0 = S1

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 ∈ L ⟷ w ∈ A* ∧ ((δ*) i w) ∈ 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**
 $G :: ('grp, 'b)\ monoid-scheme$ **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**
 $G :: ('grp, 'b)\ monoid-scheme$ **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*:
 $\bigwedge g\ w. g \in carrier\ G \implies w \in A^* \implies \psi\ g\ ((\delta^*)\ i\ w) = (\delta^*)\ i\ ((\varphi^*)\ g\ w)$
 ⟨proof⟩

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

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-restrict*:

$\bigwedge g s. g \in \text{carrier } G \implies s \in S_{\text{reach}} \implies \psi_{\text{reach}} g s = \psi g s$
 ⟨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 ∩ 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 (≡_{MN})
where *rel-MN* = {(w, w') ∈ (A*) × (A*). (∀ v ∈ A*. (w @ v) ∈ L ↔ (w' @ v) ∈ L)}

lemma *MN-rel-equiv*:

equiv (A) rel-MN*
 ⟨proof⟩

abbreviation

MN-equiv
where *MN-equiv* ≡ A* // *rel-MN*

definition

alt-natural-map-MN :: 'alpha list ⇒ 'alpha list set (≡_{MN})
where [w]_{MN} = *rel-MN* “ {w}

definition

MN-trans-func :: ('alpha list set) ⇒ 'alpha ⇒ 'alpha list set (≡_{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:*

$w' \in [w]_{MN} \implies \forall g \in \text{carrier } G. (g \odot_{\varphi^*} w') \in [g \odot_{\varphi^*} w]_{MN}$
 ⟨proof⟩

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

lemma *MN-rel-eq-var:*

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

lemma *quot-act-wd-alt-notation:*

$w \in A^* \implies g \in \text{carrier } G \implies g \odot_{[\varphi^*]_{\equiv_{MN} A^*}} ([w]_{MN}) = ([g \odot_{\varphi^*} w]_{MN})$
 ⟨proof⟩

lemma *MN-trans-func-characterization:*

$v \in (A^*) \implies a \in A \implies \delta_{MN} [v]_{MN} a = [v @ [a]]_{MN}$
 ⟨proof⟩

lemma *MN-trans-eq-var-func :*

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

lemma *MN-quot-act-on-empty-str:*

$\bigwedge g. \llbracket g \in \text{carrier } G; ([], x) \in \equiv_{MN} \rrbracket \implies x \in \text{map } (\varphi g) ' \equiv_{MN} '' \{\} \}$
 ⟨proof⟩

lemma *MN-init-state-equivar:*

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

lemma *MN-init-state-equivar-v2:*

eq-var-subset $G (MN\text{-equiv}) ([\varphi^*]_{\equiv_{MN} A^*}) \{MN\text{-init-state}\}$
 ⟨proof⟩

lemma *MN-final-state-equiv:*

eq-var-subset $G (MN\text{-equiv}) ([\varphi^*]_{\equiv_{MN} A^*})$ *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*:

$w \in (A^*) \implies \forall v \in (A^*). [v @ w]_{MN} = (\delta_{MN}^*) [v]_{MN} w$
<proof>

lemma *MN-unique-init-state*:

$w \in (A^*) \implies [w]_{MN} = (\delta_{MN}^*) [Nil]_{MN} w$
<proof>

lemma *fin-states-rep-by-lang*:

$w \in A^* \implies [w]_{MN} \in MN\text{-fin-states} \implies w \in L$
<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 ($[\varphi^]_{\equiv MN}$ A^*)*
<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 ($[\varphi^]_{\equiv MN} A^*$) 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 ($[\varphi^]_{\equiv MN} A^*$) L*
<proof>

end

1.5 Proving the Myhill-Nerode Theorem for G -Automata

context *det-G-aut begin*

no-adhoc-overloading

star \Rightarrow labels-a-G-set.induced-star-map

end

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

adhoc-overloading

star \Rightarrow labels-a-G-set.induced-star-map

definition

states-to-words :: 'states \Rightarrow 'alpha list

where *states-to-words = ($\lambda s \in S. \text{SOME } w. w \in A^* \wedge ((\delta^*) i w = s)$)*

definition

words-to-syth-states :: 'alpha list \Rightarrow 'alpha list set

where *words-to-syth-states w = $[w]_{MN}$*

definition*induced-epi*: 'states \Rightarrow 'alpha list set**where** *induced-epi* = compose *S* words-to-syth-states states-to-words**lemma** *induced-epi-wd1*: $s \in S \implies \exists w. w \in A^* \wedge ((\delta^*) i w = s)$ *<proof>***lemma** *induced-epi-wd2*: $w \in A^* \implies w' \in A^* \implies (\delta^*) i w = (\delta^*) i w' \implies [w]_{MN} = [w']_{MN}$ *<proof>***lemma** *states-to-words-on-final*:*states-to-words* $\in (F \rightarrow L)$ *<proof>***lemma** *induced-epi-eg-var*:*eg-var-func* $G S \psi$ *MN-equiv* $([(\varphi^*)]_{\equiv MN} A^*)$ *induced-epi**<proof>*

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

lemma *reach-det-G-aut-rec-lang*:*G-aut-epi* $A S i F \delta$ *MN-equiv* *MN-init-state* *MN-fin-states* $\delta_{MN} G \varphi \psi$ $([(\varphi^*)]_{\equiv MN} A^*)$ *induced-epi**<proof>***end****lemma** (**in** *det-G-aut*) *finite-reachable*:*finite* (*orbits* $G S \psi$) \implies *finite* (*orbits* $G S_{reach} \psi_{reach}$)*<proof>***lemma** (**in** *det-G-aut*)*orbs-pos-card*: *finite* (*orbits* $G S \psi$) \implies *card* (*orbits* $G S \psi$) > 0 *<proof>***lemma** (**in** *reach-det-G-aut-rec-lang*) *MN-B2T*:**assumes***Fin*: *finite* (*orbits* $G S \psi$)**shows***finite* (*orbits* G (*language.MN-equiv* $A L$) $([(\varphi^*)]_{\equiv MN} A^*)$)*<proof>***context** *det-G-aut-rec-lang* **begin**

To avoid duplicate variant of "star":

no-adhoc-overloading

star \Rightarrow *labels-a-G-set.induced-star-map*

end

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

adhoc-overloading

star \Rightarrow *labels-a-G-set.induced-star-map*

end

lemma (**in** *det-G-aut-rec-lang*) *MN-prep*:

$\exists S'. \exists \delta'. \exists F'. \exists \psi'.$

$(\text{reach-det-G-aut-rec-lang } A \ S' \ i \ F' \ \delta' \ G \ \varphi \ \psi' \ L \wedge$

$(\text{finite } (\text{orbits } G \ S \ \psi) \longrightarrow \text{finite } (\text{orbits } G \ S' \ \psi'))$)

$\langle \text{proof} \rangle$

lemma (**in** *det-G-aut-rec-lang*) *MN-fin-orbs-imp-fin-states*:

assumes

Fin: $\text{finite } (\text{orbits } G \ S \ \psi)$

shows

$\text{finite } (\text{orbits } G \ (\text{language.MN-equiv } A \ L) \ (([\varphi^*]_{\equiv MN} \ A^*)))$

$\langle \text{proof} \rangle$

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 quantify over types (if some condition holds, then there exists some automaton accepting the language). As it is not possible to quantify over types in this way, the equivalence is split 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 opportunity 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

$\text{finite } (\text{orbits } G \ A \ \varphi)$

shows

G-Myhill-Nerode-LR: $\text{finite } (\text{orbits } G \ \text{MN-equiv } ([\varphi^*]_{\equiv MN} \ A^*)) \implies$

$(\exists S \ F :: \text{'alpha list set set. } \exists i :: \text{'alpha list set. } \exists \delta. \exists \psi.$

$\text{reach-det-G-aut-rec-lang } A \ S \ i \ F \ \delta \ G \ \varphi \ \psi \ L \wedge \text{finite } (\text{orbits } G \ S \ \psi))$ **and**

G-Myhill-Nerode-RL: $(\exists S \ F :: \text{'s set. } \exists i :: \text{'s. } \exists \delta. \exists \psi.$

$\text{det-G-aut-rec-lang } A \ S \ i \ F \ \delta \ G \ \varphi \ \psi \ L \wedge \text{finite } (\text{orbits } G \ S \ \psi))$

$\implies \text{finite } (\text{orbits } G \ \text{MN-equiv } ([\varphi^*]_{\equiv MN} \ A^*))$

$\langle \text{proof} \rangle$

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 ($\lambda x \in \{\text{undefined}\}$). *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 ($\lambda x \in \{\text{undefined}\}$). *one (BijGroup labels)*) ($\lambda x \in \{\text{undefined}\}$). *one (BijGroup states)*)

<proof>

lemma *triv-orbits*:

orbits (singleton-group (undefined)) S ($\lambda x \in \{\text{undefined}\}$). *one (BijGroup S)*) =
 $\{\{s\} \mid s. s \in S\}$

<proof>

lemma *fin-triv-orbs*:

finite (orbits (singleton-group (undefined)) S ($\lambda x \in \{\text{undefined}\}$. *one (BijGroup S)*))) = *finite S*

<proof>

context *language begin*

interpretation *triv-G-lang*:

G-lang singleton-group (undefined) A ($\lambda x \in \{\text{undefined}\}$. *one (BijGroup A)*) *L*
<proof>

definition *triv-G* :: *'grp monoid*

where *triv-G* = (*singleton-group (undefined)*)

definition *triv-act* :: *'grp \Rightarrow 'alpha \Rightarrow 'alpha*

where *triv-act* = ($\lambda x \in \{\text{undefined}\}$. $\mathbf{1}_{\text{BijGroup } A}$)

corollary *standard-Myhill-Nerode*:

assumes

H-fin-alph: *finite A*

shows

standard-Myhill-Nerode-LR: *finite MN-equiv \implies*

$(\exists S F :: \text{'alpha list set set. } \exists i :: \text{'alpha list set. } \exists \delta.$

reach-det-aut-rec-lang A S i F $\delta L \wedge$ finite S) **and**

standard-Myhill-Nerode-RL: $(\exists S F :: \text{'s set. } \exists i :: \text{'s. } \exists \delta.$

det-aut-rec-lang A S i F $\delta L \wedge$ finite S) \implies 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 $\text{BijGroup } D$, but assume there is a homomorphism $\pi : G \rightarrow \text{BijGroup } D$. By `group_hom.img_is_subgroup` this is an equivalent definition:

```
locale data-symm = group-action G D  $\pi$ 
for
  G :: ('grp, 'b) monoid-scheme and
  D :: 'D set ( $\langle \mathbb{D} \rangle$ ) and
   $\pi$ 
```

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

```
locale supports = data-symm G D  $\pi$  + alt-grp-act G X  $\varphi$ 
for
  G :: ('grp, 'b) monoid-scheme and
  D :: 'D set ( $\langle \mathbb{D} \rangle$ ) and
   $\pi$  and
  X :: 'X set (structure) and
   $\varphi$  +
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  $\implies$  ( $\forall c. c \in C \longrightarrow \pi g c = c$ )  $\implies$  g  $\odot_{\varphi}$  x = x
begin
```

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

```
lemma image-supports:
   $\bigwedge g. g \in \text{carrier } G \implies \text{supports } G D \pi X \varphi (\pi g ' C) (g \odot_{\varphi} x)$ 
  <proof>
end
```

```
locale nominal = data-symm G D  $\pi$  + alt-grp-act G X  $\varphi$ 
for
  G :: ('grp, 'b) monoid-scheme and
  D :: 'D set ( $\langle \mathbb{D} \rangle$ ) and
   $\pi$  and
  X :: 'X set (structure) and
```

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

locale *nominal-det-G-aut* = *det-G-aut* +
nominal $G \ D \ \pi \ A \ \varphi +$ *nominal* $G \ D \ \pi \ S \ \psi$
for

$D :: 'D \ \text{set} \ (\langle \mathbb{D} \rangle)$ **and**
 π

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

lemma (*in eq-var-func*) *supp-el-pres:*
 $\text{supports } G \ D \ \pi \ X \ \varphi \ C \ x \implies \text{supports } G \ D \ \pi \ Y \ \psi \ C \ (f \ x)$
<proof>

lemma (*in nominal*) *support-union-lem:*
fixes $f \ \text{sup-C} \ \text{col}$
assumes $H\text{-}f: f = (\lambda x. (\text{SOME } C. C \subseteq \mathbb{D} \wedge \text{finite } C \wedge \text{supports } G \ \mathbb{D} \ \pi \ X \ \varphi \ C \ x))$
and $H\text{-col}: \text{col} \subseteq X \wedge \text{finite } \text{col}$
and $H\text{-sup-C}: \text{sup-C} = \bigcup \{C x. C x \in f \ ' \ \text{col}\}$
shows $\bigwedge x. x \in \text{col} \implies \text{sup-C} \subseteq \mathbb{D} \wedge \text{finite } \text{sup-C} \wedge \text{supports } G \ \mathbb{D} \ \pi \ X \ \varphi \ \text{sup-C} \ x$
<proof>

lemma (*in nominal*) *set-of-list-nom:*
nominal $G \ D \ \pi \ (X^*) \ (\varphi^*)$
<proof>

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

context *det-G-aut* **begin**

adhoc-overloading

$\text{star} \rightleftharpoons \text{labels-a-G-set.induced-star-map}$

end

lemma (*in det-G-aut*) *input-to-init-eqvar:*
eq-var-func $G \ (A^*) \ (\varphi^*) \ S \ \psi \ (\lambda w \in A^*. (\delta^*) \ i \ w)$
<proof>

lemma (*in reach-det-G-aut*) *input-to-init-surj:*
 $(\lambda w \in A^*. (\delta^*) \ i \ w) \ ' \ (A^*) = S$
<proof>

context *reach-det-G-aut* **begin**

adhoc-overloading

$\text{star} \rightleftharpoons \text{labels-a-G-set.induced-star-map}$

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 \implies *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 ([[φ*]]_{≡MN} A*))* \implies

$(\exists S F :: \text{'alpha list set set. } \exists i :: \text{'alpha list set. } \exists \delta. \exists \psi.$

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: $(\exists S F :: \text{'s set. } \exists i :: \text{'s. } \exists \delta. \exists \psi.$

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*)

\implies *finite (orbits G MN-equiv ([[φ*]]_{≡MN} A*))*

⟨*proof*⟩

end

References

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