

Myhill-Nerode Theorem for (Nominal) G -Automata

Cárolos Laméris

April 18, 2024

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	8
1.3	Basic (G)-Automata Theory	19
1.4	Syntactic Automaton	38
1.5	Proving the Myhill-Nerode Theorem for G -Automata	53
1.6	Proving the standard Myhill-Nerode Theorem	65
2	Myhill-Nerode Theorem for Nominal G-Automata	69
2.1	Data Symmetries, Supports and Nominal Actions	69
2.2	Proving the Myhill-Nerode Theorem for Nominal G -Automata	75

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-Library.Adhoc-Overloading
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 (-* [1000] 999)

adhoc-overloading

star 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** (\odot) 70)
where (\odot φ) \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*:

$\llbracket S \subseteq T; x \in T \wedge f \in \text{Bij } T; (\text{restrict } f \ S) \ \text{' } S = S; f \ x \in S \rrbracket \implies x \in S$
apply (*clarsimp simp add: extensional-def subset-eq Bij-def bij-betw-def restrict-def inj-on-def*)
by (*metis imageE*)

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) monoid-scheme and
  X :: 'X set (structure) and
   $\varphi$ 
begin
```

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

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

proof –

```
assume
  A-0:  $x \in X$ 
show  $\mathbf{1}_G \odot_{\varphi} x = x$ 
using group-action-axioms
apply (simp add: group-action-def BijGroup-def)
by (metis A-0 id-eq-one restrict-apply')
next
assume
  A-0:  $g \in \text{carrier } G$  and
  A-1:  $h \in \text{carrier } G$  and
  A-2:  $x \in X$ 
show  $g \otimes_G h \odot_{\varphi} x = g \odot_{\varphi} (h \odot_{\varphi} x)$ 
using group-action-axioms
apply (simp add: group-action-def group-hom-def group-hom-axioms-def hom-def BijGroup-def)
using composition-rule A-0 A-1 A-2
by auto
qed
```

definition

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

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 induced-star-map

definition

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

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

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

lemma *act-maps-n-distrib*:

$\forall g \in \text{carrier } G. \forall w \in X^*. \forall v \in X^*. (\varphi^*) g (w @ v) = ((\varphi^*) g w) @ ((\varphi^*) g v)$
by (*auto simp add: group-action-def group-hom-def group-hom-axioms-def hom-def*)

lemma *triv-act*:

$a \in X \Longrightarrow (\varphi \mathbf{1}_G) a = a$
using *group-hom.hom-one*[*of G BijGroup X phi*] *group-BijGroup*[**where** $S = X$]
apply (*clarsimp simp add: group-action-def group-hom-def group-hom-axioms-def*
BijGroup-def)
by (*metis id-eq-one restrict-apply'*)

lemma *triv-act-map*:

$\forall w \in X^*. ((\varphi^*) \mathbf{1}_G) w = w$
using *triv-act*
apply *clarsimp*
apply (*rule conjI; rule impI*)
apply *clarify*
using *map-idI*
apply *metis*
using *group.subgroup-self group-hom group-hom.axioms(1) subgroup.one-closed*
by *blast*

proposition *lists-a-Gset*:

alt-grp-act G (X) (phi*)*

proof–

have *H-0*: $\bigwedge g. g \in \text{carrier } G \Longrightarrow$
 $\text{restrict } (\text{map } (\varphi g)) (X^*) \in \text{carrier } (\text{BijGroup } (X^*))$

proof–

fix *g*

assume

A1-0: $g \in \text{carrier } G$

from *A1-0* **have** *H1-0*: *inj-on* ($\lambda x. \text{if } x \in X^* \text{ then } \text{map } (\varphi g) x \text{ else undefined}$)
 (X^*)

apply (*clarsimp simp add: inj-on-def*)

by (*metis (mono-tags, lifting) inj-onD inj-prop list.inj-map-strong*)

from *A1-0* **have** *H1-1*: $\bigwedge y z. \forall x \in \text{set } y. x \in X \Longrightarrow z \in \text{set } y \Longrightarrow \varphi g z \in X$

using *element-image*

by *blast*

```

have H1-2: (inv G g) ∈ carrier G
  by (meson A1-0 group.inv-closed group-hom group-hom.axioms(1))
have H1-3:  $\bigwedge x. x \in X^* \implies$ 
  map (comp (φ g) (φ (inv G g))) x = map (φ (g ⊗G (inv G g))) x
  using alt-grp-act-axioms
apply (simp add: alt-grp-act-def group-action-def group-hom-def group-hom-axioms-def
hom-def
  BijGroup-def)
  apply (rule meta-mp[of  $\bigwedge x. x \in \text{carrier } G \implies \varphi x \in \text{Bij } X$ ])
  apply (metis A1-0 H1-2 composition-rule in-lists-conv-set)
  by blast
from H1-2 have H1-4:  $\bigwedge x. x \in X^* \implies \text{map } (\varphi (inv \ G \ g)) \ x \in X^*$ 
  using surj-prop
  by fastforce
have H1-5:  $\bigwedge y. \forall x \in \text{set } y. x \in X \implies y \in \text{map } (\varphi \ g) \ ' \ X^*$ 
  apply (simp add: image-def)
  using H1-3 H1-4
  by (metis A1-0 group.r-inv group-hom group-hom.axioms(1) in-lists-conv-set
map-idI map-map
  triv-act)
show restrict (map (φ g)) (X*) ∈ carrier (BijGroup (X*))
  apply (clarsimp simp add: restrict-def BijGroup-def Bij-def
  extensional-def bij-betw-def)
  apply (rule conjI)
  using H1-0
  apply simp
  using H1-1 H1-5
  by (auto simp add: image-def)
qed
have H-1:  $\bigwedge x \ y. \llbracket x \in \text{carrier } G; y \in \text{carrier } G; x \otimes_G y \in \text{carrier } G \rrbracket \implies$ 
  restrict (map (φ (x ⊗G y))) (X*) =
  restrict (map (φ x)) (X*) ⊗BijGroup (X*)
  restrict (map (φ y)) (X*)
proof -
  fix x y
  assume
    A1-0: x ∈ carrier G and
    A1-1: y ∈ carrier G and
    A1-2: x ⊗G y ∈ carrier G
  have H1-0:  $\bigwedge z. z \in \text{carrier } G \implies$ 
    bij-betw (λx. if x ∈ X* then map (φ z) x else undefined) (X*) (X*)
  using ⟨λg. g ∈ carrier G ⟹ restrict (map (φ g)) (X*) ∈ carrier (BijGroup
(X*))⟩
  by (auto simp add: BijGroup-def Bij-def bij-betw-def inj-on-def)
  from A1-1 have H1-1:  $\bigwedge \text{lst}. \text{lst} \in X^* \implies (\text{map } (\varphi \ y)) \ \text{lst} \in X^*$ 
  by (metis group-action.surj-prop group-action-axioms lists-image rev-image-eqI)
  have H1-2:  $\bigwedge a. a \in X^* \implies \text{map } (\lambda x b.$ 
    if xb ∈ X
    then φ x ((φ y) xb)

```

```

else undefined) a = map (φ x) (map (φ y) a)
  by auto
have H1-3: (λxa. if xa ∈ X* then map (φ (x ⊗G y)) xa else undefined) =
compose (X*) (λxa. if xa ∈ X* then map (φ x) xa else undefined)
(λx. if x ∈ X* then map (φ y) x else undefined)
  using alt-grp-act-axioms
  apply (clarsimp simp add: compose-def alt-grp-act-def group-action-def
group-hom-def group-hom-axioms-def hom-def BijGroup-def restrict-def)
  using A1-0 A1-1 H1-2 H1-1 bij-prop0
  by auto
show restrict (map (φ (x ⊗G y))) (X*) =
restrict (map (φ x)) (X*) ⊗BijGroup (X*)
restrict (map (φ y)) (X*)
  apply (clarsimp simp add: restrict-def BijGroup-def Bij-def extensional-def)
  apply (simp add: H1-3)
  using A1-0 A1-1 H1-0
  by auto
qed
show alt-grp-act G (X*) (φ*)
  apply (clarsimp simp add: alt-grp-act-def group-action-def group-hom-def group-hom-axioms-def)
  apply (intro conjI)
  using group-hom group-hom-def
  apply (auto)[1]
  apply (simp add: group-BijGroup)
  apply (clarsimp simp add: hom-def)
  apply (intro conjI; clarify)
  apply (rule H-0)
  apply simp
  apply (rule conjI; rule impI)
  apply (rule H-1)
  apply simp+
  apply (rule meta-mp[of ∧x y. x ∈ carrier G ⇒
y ∈ carrier G ⇒ x ⊗G y ∈ carrier G])
  apply blast
  by (meson group.subgroup-self group-hom group-hom.axioms(1) subgroup.m-closed)
qed
end

```

lemma *alt-group-act-is-grp-act* [simp, GMN-simps]:
alt-grp-act = group-action
using *alt-grp-act-def*
by *blast*

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) (λg ∈ carrier G. λ(a, b) ∈ (A × B). (φ g a, ψ g b))*

apply (*simp add: alt-grp-act-def group-action-def group-hom-def*)

```

apply (intro conjI)
subgoal
  using grp-act-A grp-act-B
  by (auto simp add: alt-grp-act-def group-action-def group-hom-def)
subgoal
  using grp-act-A grp-act-B
  by (auto simp add: alt-grp-act-def group-action-def group-hom-def group-BijGroup)
apply (clarsimp simp add: group-hom-axioms-def hom-def BijGroup-def)
apply (intro conjI; clarify)
subgoal for g
  apply (clarsimp simp add: Bij-def bij-betw-def inj-on-def restrict-def exten-
sional-def)
  apply (intro conjI)
  using grp-act-A
  apply (simp add: alt-grp-act-def group-action-def group-hom-def group-hom-axioms-def
BijGroup-def hom-def Pi-def compose-def Bij-def bij-betw-def inj-on-def)
  using grp-act-B
  apply (simp add: alt-grp-act-def group-action-def group-hom-def group-hom-axioms-def
BijGroup-def hom-def Pi-def compose-def Bij-def bij-betw-def inj-on-def)
  apply (rule meta-mp[of  $\varphi$   $g \in \text{Bij } A \wedge \psi$   $g \in \text{Bij } B$ ])
  apply (clarsimp simp add: Bij-def bij-betw-def)
  using grp-act-A grp-act-B
  apply (simp add: alt-grp-act-def group-action-def group-hom-def group-hom-axioms-def
BijGroup-def hom-def Pi-def Bij-def)
  using grp-act-A grp-act-B
  apply (clarsimp simp add: compose-def restrict-def image-def alt-grp-act-def
group-action-def group-hom-def group-hom-axioms-def BijGroup-def hom-def
Pi-def Bij-def
bij-betw-def)
  apply (rule subset-antisym)
  apply blast+
  by (metis alt-group-act-is-grp-act group-action.bij-prop0 grp-act-A grp-act-B)
apply (intro conjI; intro impI)
apply (clarify)
apply (intro conjI; intro impI)
apply (rule conjI)
subgoal for x y
  apply unfold-locales
  apply (clarsimp simp add: Bij-def compose-def restrict-def bij-betw-def)
  apply (rule extensionalityI[where  $A = A \times B$ ])
  apply (clarsimp simp add: extensional-def)
  using grp-act-A grp-act-B
  apply (simp add: alt-grp-act-def group-action-def group-hom-def group-hom-axioms-def
BijGroup-def hom-def Pi-def Bij-def compose-def extensional-def)
  apply (simp add: fun-eq-iff; rule conjI; rule impI)
using group-action.composition-rule[of  $G$   $A$   $\varphi$ ] group-action.composition-rule[of
 $G$   $B$   $\psi$ ] grp-act-A
grp-act-B
apply force

```

```

by blast
apply (simp add: ⟨ $\bigwedge g. g \in \text{carrier } G \implies (\lambda(a, b) \in A \times B. (\varphi g a, \psi g b)) \in \text{Bij } (A \times B)$ ⟩)
apply (simp add: ⟨Group.group G⟩ group.subgroup-self subgroup.m-closed)
by (simp add: ⟨ $\bigwedge g. g \in \text{carrier } G \implies (\lambda(a, b) \in A \times B. (\varphi g a, \psi g b)) \in \text{Bij } (A \times B)$ ⟩)+

```

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:  $\forall g \in \text{carrier } G. (\varphi g) \text{ ` } Y = Y$ 

```

```

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

```

```

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

```

```

proof –

```

```

  fix Y

```

```

  assume

```

```

    A-0:  $Y \subseteq X$  and

```

```

    A-1:  $\forall g \in \text{carrier } G. (\varphi g) \text{ ` } Y \subseteq Y$ 

```

```

  have H-0:  $\bigwedge g. g \in \text{carrier } G \implies (\text{inv}_G g) \in \text{carrier } G$ 

```

```

    by (meson group.inv-closed group-hom group-hom.axioms(1))

```

```

  hence H-1:  $\bigwedge g y. g \in \text{carrier } G \implies y \in Y \implies (\varphi (\text{inv}_G g)) y \in Y$ 

```

```

    using A-1

```

```

    by (simp add: image-subset-iff)

```

```

  have H-2:  $\bigwedge g y. g \in \text{carrier } G \implies y \in Y \implies \varphi g ((\varphi (\text{inv}_G g)) y) = y$ 

```

```

    by (metis A-0 bij-prop1 orbit-sym-aux subsetD)

```

```

  show eq-var-subset G X  $\varphi$  Y

```

```

    apply (simp add: eq-var-subset-def eq-var-subset-axioms-def)

```

```

    apply (intro conjI)

```

```

    apply (simp add: group-action-axioms)

```

```

    apply (rule A-0)

```

```

    apply (clarify)

```

```

    apply (rule subset-antisym)

```

```

    using A-1

```

```

    apply simp

```

```

    apply (simp add: image-def)

```

```

    apply (rule subsetI)

```

```

    apply clarify

```

```

    using H-1 H-2

```

```

    by metis

```

```

qed

```


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

lemma *some-equiv-class-id*:

$\llbracket \text{equiv } X \ R; w \in X \ // \ R; x \in w \rrbracket \implies R \text{ `` } \{x\} = R \text{ `` } \{\text{SOME } z. z \in w\}$
by (*smt* (*z3*) *Eps-cong equiv-Eps-in equiv-class-eq-iff quotient-eq-iff*)

lemma *nested-somes*:

$\llbracket \text{equiv } X \ R; w \in X \ // \ R \rrbracket \implies (\text{SOME } z. z \in w) = (\text{SOME } z. z \in R \text{ `` } \{\text{SOME } z'. z' \in w\})$
by (*metis proj-Eps proj-def*)

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:

$\bigwedge a \ b. (a, b) \in R \implies \forall g \in \text{carrier } G. (g \odot_{\varphi} a, g \odot_{\varphi} b) \in R$

begin

lemma *is-eq-var-rel'* [*simp*, *GMN-simps*]:

$\bigwedge a \ b. (a, b) \in R \implies \forall g \in \text{carrier } G. ((\varphi \ g) \ a, (\varphi \ g) \ b) \in R$

using *is-eq-var-rel*

by *auto*

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 –

assume

A-0: $(g \odot_{\varphi} a, g \odot_{\varphi} b) \in R$ **and**

A-1: $a \in X$ **and**

A-2: $b \in X$ **and**

A-3: $g \in \text{carrier } G$

have *H-0*: *group-action* *G* *X* φ **and**

H-1: $R \subseteq X \times X$ **and**

H-2: $\bigwedge a \ b \ g. (a, b) \in R \implies g \in \text{carrier } G \implies (\varphi \ g \ a, \varphi \ g \ b) \in R$

by (*simp add: group-action-axioms is-subrel*) +

from *H-0* **have** *H-3*: *group* *G*

by (*auto simp add: group-action-def group-hom-def*)

have *H-4*: $(\varphi \ (\text{inv}_G \ g) \ (\varphi \ g \ a), \varphi \ (\text{inv}_G \ g) \ (\varphi \ g \ b)) \in R$

apply (*rule H-2*)

using *A-0* **apply** *simp*

by (*simp add: A-3 H-3*)

from *H-3* *A-3* **have** *H-5*: $(\text{inv}_G \ g) \in \text{carrier } G$

by *auto*

hence *H-6*: $\bigwedge e. e \in X \implies \varphi \ (\text{inv}_G \ g) \ (\varphi \ g \ e) = \varphi \ ((\text{inv}_G \ g) \otimes_G \ g) \ e$

using *H-0 A-3 group-action.composition-rule*
 by *fastforce*
 hence *H-7*: $\bigwedge e. e \in X \implies \varphi (\text{inv}_G g) (\varphi g e) = \varphi \mathbf{1}_G e$
 using *H-3 A-3 group.l-inv*
 by *fastforce*
 hence *H-8*: $\bigwedge e. e \in X \implies \varphi (\text{inv}_G g) (\varphi g e) = e$
 using *H-0*
 by (*simp add: A-3 group-action.orbit-sym-aux*)
 thus $(a, b) \in R$
 using *A-1 A-2 H-4*
 by *simp*
 qed

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 –
 obtain z where *H-z*: $w = R \text{ `` } \{z\} \wedge z \in w$
 by (*metis A-0 A-1 equiv-class-self quotientE*)
 have *H-0*: $\bigwedge x. (\varphi g z, x) \in R \implies x \in \varphi g \text{ ` } \{y. (z, y) \in R\}$
 proof –
 fix y
 assume
 A1-0: $(\varphi g z, y) \in R$
 obtain y' where *H2-y'*: $y' = \varphi (\text{inv}_G g) y \wedge y' \in X$
 using *eq-var-rel-axioms*
 apply (*clarsimp simp add: eq-var-rel-def group-action-def group-hom-def*)
 by (*meson A-0 eq-var-rel-axioms A-2 A1-0 equiv-class-eq-iff eq-var-rel.is-eq-var-rel*
 group.inv-closed element-image)
 from *A-1 A-2 H2-y'* have *H2-0*: $\varphi g y' = y$
 apply (*clarsimp simp add: eq-var-rel-def eq-var-rel-axioms-def*)
 using *A-2 A1-0 group-action.bij-prop1* [where $G = G$ and $E = X$ and $\varphi =$
 φ]
 by (*metis A-0 alt-group-act-is-grp-act alt-grp-act-axioms equiv-class-eq-iff*
 orbit-sym-aux)
 from *A-1 A-2 A1-0* have *H2-1*: $(z, y') \in R$
 by (*metis H2-0 A-0 A-2 H2-y' H-z equiv-class-eq-iff is-eq-var-rel-rev*
 quotient-eq-iff make-op-def)
 thus $y \in \varphi g \text{ ` } \{v. (z, v) \in R\}$
 using *H2-0*
 by (*auto simp add: image-def*)
 qed
 have *H-1*: $\varphi g \text{ ` } (R \text{ `` } \{z\}) = R \text{ `` } \{\varphi g z\}$
 apply (*clarsimp simp add: Relation.Image-def*)
 apply (*rule subset-antisym; simp add: Set.subset-eq; rule allI; rule impI*)
 using *eq-var-rel-axioms A-2 eq-var-rel.is-eq-var-rel*

```

    apply simp
  by (simp add: H-0)
have H-2:  $\varphi g \text{ ' } w \in X // R$ 
  using eq-var-rel-axioms A-1 A-2 H-1
  by (metis A-0 H-z equiv-class-eq-iff quotientI quotient-eq-iff element-image)
thus ( $[\varphi]_R$ )  $g w = R \text{ '' } \{SOME z'. z' \in \varphi g \text{ ' } w\}$ 
  using A-0 A-1 A-2
  apply (clarsimp simp add: Image-def)
  apply (intro subset-antisym)
  apply (clarify)
using A-0 H-z imageI insert-absorb insert-not-empty some-in-eq some-equiv-class-id

  apply (smt (z3) A-1 Eps-cong Image-singleton-iff equiv-Eps-in)
  apply (clarify)
  by (smt (z3) Eps-cong equiv-Eps-in image-iff in-quotient-imp-closed quotient-eq-iff)
qed

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 -
  obtain z where H-z:  $R \text{ '' } \{z\} = w \wedge z \in X$ 
  by (metis A-2 quotientE)
  have H-0:  $\text{equiv } X R \implies g \in \text{carrier } G \implies w \in X // R \implies$ 
     $\{y. (\varphi g z, y) \in R\} \subseteq \{y. \exists x. (z, x) \in R \wedge y = \varphi g x\}$ 
  proof (clarify)
    fix x
    assume
      A1-0: equiv X R and
      A1-1:  $g \in \text{carrier } G$  and
      A1-2:  $w \in X // R$  and
      A1-3:  $(\varphi g z, x) \in R$ 
    obtain x' where H2-x':  $x = \varphi g x' \wedge x' \in X$ 
    using group-action-axioms
    by (metis A1-1 is-subrel A1-3 SigmaD2 group-action.bij-prop1 subsetD)
    thus  $\exists y. (z, y) \in R \wedge x = \varphi g y$ 
    using is-eq-var-rel-rev[where  $g = g$  and  $a = z$  and  $b = x'$ ] A1-3
  by (auto simp add: eq-var-rel-def eq-var-rel-axioms-def A1-1 A1-2 group-action-axioms
H-z
      H2-x')
  qed
have H-1:  $\varphi g \text{ ' } R \text{ '' } \{z\} = R \text{ '' } \{\varphi g z\}$ 
  using A-0 A-1 A-2
  apply (clarsimp simp add: eq-var-rel-axioms-def eq-var-rel-def
Image-def image-def)
  apply (intro subset-antisym)

```

```

apply (auto)[1]
by (rule H-0)
thus  $\varphi g \text{ ' } w \in X // R$ 
using H-1 H-z
by (metis A-1 quotientI element-image)
qed

```

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

```

lemma quot-act-wd:
 $\llbracket \text{equiv } X R; x \in X; g \in \text{carrier } G \rrbracket \implies g \odot_{[\varphi]_R} (R \text{ `` } \{x\}) = (R \text{ `` } \{g \odot_{\varphi} x\})$ 
apply (clarsimp simp add: eq-var-rel-def eq-var-rel-axioms-def)
apply (rule conjI; rule impI)
apply (smt (verit, best) Eps-cong Image-singleton-iff eq-var-rel.is-eq-var-rel'
  eq-var-rel-axioms equiv-Eps-in equiv-class-eq)
by (simp add: quotientI)+

```

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 –
  assume A-0:  $\text{equiv } X R$ 
  have H-0:  $\bigwedge x. \text{Group.group } G \implies$ 
     $\text{Group.group } (\text{BijGroup } X) \implies$ 
     $R \subseteq X \times X \implies$ 
     $\varphi \in \text{carrier } G \rightarrow \text{carrier } (\text{BijGroup } X) \implies$ 
     $\forall x \in \text{carrier } G. \forall y \in \text{carrier } G. \varphi (x \otimes_G y) = \varphi x \otimes_{\text{BijGroup } X} \varphi y \implies$ 
     $x \in \text{carrier } G \implies (\lambda xa \in X // R. R \text{ `` } \{\varphi x (\text{SOME } z. z \in xa)\}) \in \text{carrier}$ 
     $(\text{BijGroup } (X // R))$ 
  proof –
  fix g
  assume
    A1-0:  $\text{Group.group } G$  and
    A1-1:  $\text{Group.group } (\text{BijGroup } X)$  and
    A1-2:  $\varphi \in \text{carrier } G \rightarrow \text{carrier } (\text{BijGroup } X)$  and
    A1-3:  $\forall x \in \text{carrier } G. \forall y \in \text{carrier } G. \varphi (x \otimes_G y) = \varphi x \otimes_{\text{BijGroup } X} \varphi y$  and
    A1-4:  $g \in \text{carrier } G$ 
  have H-0:  $\text{group-action } G X \varphi$ 
  apply (clarsimp simp add: group-action-def group-hom-def group-hom-axioms-def)
  apply (simp add: A1-0 A1-1)+
  apply (simp add: hom-def)
  apply (rule conjI)
  using A1-2
  apply blast
  by (simp add: A1-3)
  have H1-0:  $\bigwedge x y. \llbracket x \in X // R; y \in X // R; R \text{ `` } \{\varphi g (\text{SOME } z. z \in x)\} =$ 
     $R \text{ `` } \{\varphi g (\text{SOME } z. z \in y)\} \rrbracket \implies x \subseteq y$ 
  proof (clarify; rename-tac a)
  fix x y a

```

assume
A2-0: $x \in X // R$ **and**
A2-1: $y \in X // R$ **and**
A2-2: $R \text{ “ } \{\varphi g (\text{SOME } z. z \in x)\} = R \text{ “ } \{\varphi g (\text{SOME } z. z \in y)\}$ **and**
A2-3: $a \in x$
obtain b **where** *H2-b*: $R \text{ “ } \{b\} = y \wedge b \in X$
by (*metis A2-1 quotientE*)
obtain $a' b'$ **where** *H2-a'-b'*: $a' \in x \wedge b' \in y \wedge R \text{ “ } \{\varphi g a'\} = R \text{ “ } \{\varphi g b'\}$
by (*metis A-0 A2-1 A2-2 A2-3 equiv-Eps-in some-eq-imp*)
from *H2-a'-b'* **have** *H2-2*: $(\varphi g a', \varphi g b') \in R$
by (*metis A-0 A1-4 A2-1 Image-singleton-iff eq-var-rel.is-eq-var-rel' eq-var-rel-axioms quotient-eq-iff*)
hence *H2-0*: $(\varphi (\text{inv}_G g) (\varphi g a'), \varphi (\text{inv}_G g) (\varphi g b')) \in R$
by (*simp add: A1-0 is-eq-var-rel A1-4*)
have *H2-1*: $a' \in X \wedge b' \in X$
using *A-0 A2-0 A2-1 H2-a'-b' in-quotient-imp-subset*
by *blast*
hence *H2-2*: $(a', b') \in R$
using *H2-0*
by (*metis A1-4 H-0 group-action.orbit-sym-aux*)
have *H2-3*: $(a, a') \in R$
by (*meson A-0 A2-0 A2-3 H2-a'-b' quotient-eq-iff*)
hence *H2-4*: $(b', a) \in R$
using *H2-2*
by (*metis A-0 A2-0 A2-1 A2-3 H2-a'-b' quotient-eqI quotient-eq-iff*)
thus $a \in y$
by (*metis A-0 A2-1 H2-a'-b' in-quotient-imp-closed*)
qed
have *H1-1*: $\bigwedge x. x \in X // R \implies \exists xa \in X // R. x = R \text{ “ } \{\varphi g (\text{SOME } z. z \in xa)\}$
proof –
fix x
assume
A2-0: $x \in X // R$
have *H2-0*: $\bigwedge e. R \text{ “ } \{e\} \in X // R \implies R \text{ “ } \{e\} \subseteq R \text{ “ } \{\varphi g (\varphi (\text{inv}_G g) e)\}$
proof (*rule subsetI*)
fix $e y$
assume
A3-0: $R \text{ “ } \{e\} \in X // R$ **and**
A3-1: $y \in R \text{ “ } \{e\}$
have *H3-0*: $y \in X$
using *A3-1 is-subrel*
by *blast*
from *H-0* **have** *H3-1*: $\varphi g (\varphi (\text{inv}_G g) y) = y$
by (*metis (no-types, lifting) A1-0 A1-4 H3-0 group.inv-closed group.inv-inv group-action.orbit-sym-aux*)
from *A3-1* **have** *H3-2*: $(e, y) \in R$
by *simp*
hence *H3-3*: $((\varphi (\text{inv}_G g) e), (\varphi (\text{inv}_G g) y)) \in R$

```

    using is-eq-var-rel A1-4 A1-0
    by simp
  hence H3-4:  $(\varphi g (\varphi (\text{inv}_G g) e), \varphi g (\varphi (\text{inv}_G g) y)) \in R$ 
    using is-eq-var-rel A1-4 A1-0
    by simp
  hence H3-5:  $(\varphi g (\varphi (\text{inv}_G g) e), y) \in R$ 
    using H3-1
    by simp
  thus  $y \in R \text{ “ } \{\varphi g (\varphi (\text{inv}_G g) e)\}$ 
    by simp
qed
  hence H2-1:  $\bigwedge e. R \text{ “ } \{e\} \in X // R \implies R \text{ “ } \{e\} = R \text{ “ } \{\varphi g (\varphi (\text{inv}_G g) e)\}$ 
e)}}
  by (metis A-0 proj-def proj-in-iff equiv-class-eq-iff subset-equiv-class)
  have H2-2:  $\bigwedge e f. R \text{ “ } \{e\} \in X // R \implies R \text{ “ } \{f\} \in X // R \implies$ 
 $R \text{ “ } \{e\} = R \text{ “ } \{f\} \implies \forall f' \in R \text{ “ } \{f\}. R \text{ “ } \{e\} = R \text{ “ } \{f'\}$ 
  by (metis A-0 Image-singleton-iff equiv-class-eq)
  have H2-3:  $x \in X // R \implies \exists e \in X. x = R \text{ “ } \{e\}$ 
  by (meson quotientE)
  have H2-4:  $\bigwedge e. R \text{ “ } \{e\} \in X // R \implies R \text{ “ } \{e\} = R \text{ “ } \{\varphi g (\varphi (\text{inv}_G g) e)\}$ 
^
   $(\varphi (\text{inv}_G g) e) \in R \text{ “ } \{\varphi (\text{inv}_G g) e\}$ 
  by (smt (z3) A-0 A1-0 A1-4 H-0 H2-1 Image-singleton-iff equiv-class-eq-iff
    group.inv-closed group-action.element-image in-quotient-imp-non-empty
    subset-empty subset-emptyI)
  have H2-5:  $\bigwedge e. R \text{ “ } \{e\} \in X // R \implies \forall z \in R \text{ “ } \{\varphi (\text{inv}_G g) e\}. (\varphi (\text{inv}_G g) e, z) \in R$ 
  by simp
  hence H2-6:  $\bigwedge e. R \text{ “ } \{e\} \in X // R \implies$ 
 $\forall z \in R \text{ “ } \{\varphi (\text{inv}_G g) e\}. (\varphi g (\varphi (\text{inv}_G g) e), \varphi g z) \in R$ 
  using is-eq-var-rel' A1-4 A1-0
  by blast
  hence H2-7:  $\bigwedge e. R \text{ “ } \{e\} \in X // R \implies \forall z \in R \text{ “ } \{\varphi (\text{inv}_G g) e\}. (e, \varphi g z) \in R$ 
  using H2-1
  by blast
  hence H2-8:  $\bigwedge e. R \text{ “ } \{e\} \in X // R \implies \forall z \in R \text{ “ } \{\varphi (\text{inv}_G g) e\}. R \text{ “ } \{e\} = R \text{ “ } \{\varphi g z\}$ 
  by (meson A-0 equiv-class-eq-iff)
  have H2-9:  $\bigwedge e. R \text{ “ } \{e\} \in X // R \implies$ 
 $R \text{ “ } \{e\} = R \text{ “ } \{\varphi g (\text{SOME } z. z \in R \text{ “ } \{\varphi (\text{inv}_G g) e\})\}$ 
proof-
  fix e
  assume
  A3-0:  $R \text{ “ } \{e\} \in X // R$ 
  show  $R \text{ “ } \{e\} = R \text{ “ } \{\varphi g (\text{SOME } z. z \in R \text{ “ } \{\varphi (\text{inv}_G g) e\})\}$ 
  apply (rule someI2[where Q =  $\lambda z. R \text{ “ } \{e\} = R \text{ “ } \{\varphi g z\}$  and
    P =  $\lambda z. z \in R \text{ “ } \{\varphi (\text{inv}_G g) e\}$  and a =  $\varphi (\text{inv}_G g) e$ ])
  using A3-0 H2-4

```

```

    apply blast
    using A3-0 H2-8
    by auto
  qed
  have H2-10:  $\forall e. (R \text{ `` } \{e\} \in X // R \longrightarrow$ 
     $(R \text{ `` } \{e\} = R \text{ `` } \{\varphi g (\text{SOME } z. z \in R \text{ `` } \{\varphi (\text{inv}_G g) e\})\}))$ 
    using H2-9
    by auto
  hence H2-11:  $\forall e. (R \text{ `` } \{e\} \in X // R \longrightarrow$ 
     $(\exists xa \in X // R. R \text{ `` } \{e\} = R \text{ `` } \{\varphi g (\text{SOME } z. z \in xa)\}))$ 
    using H2-8
    apply clarsimp
    by (smt (verit, best) A-0 H2-3 H2-5 H2-4 equiv-Eps-in equiv-class-eq-iff
    quotientI)
  have H2-12:  $\bigwedge x. x \in X // R \implies \exists e \in X. x = R \text{ `` } \{e\}$ 
    by (meson quotientE)
  have H2-13:  $\bigwedge x. x \in X // R \implies \exists xa \in X // R. x = R \text{ `` } \{\varphi g (\text{SOME } z. z \in xa)\}$ 
    using H2-11 H2-12
    by blast
  show  $\exists xa \in X // R. x = R \text{ `` } \{\varphi g (\text{SOME } z. z \in xa)\}$ 
    by (simp add: A2-0 H2-13)
  qed
  show  $(\lambda x \in X // R. R \text{ `` } \{\varphi g (\text{SOME } z. z \in x)\}) \in \text{carrier } (\text{BijGroup } (X // R))$ 
    apply (clarsimp simp add: BijGroup-def Bij-def bij-betw-def)
    apply (clarsimp simp add: inj-on-def)
    apply (rule conjI)
    apply (clarsimp)
    apply (rule subset-antisym)
    apply (simp add: H1-0)
    apply (simp add:  $\langle \bigwedge y x. \llbracket x \in X // R; y \in X // R; R \text{ `` } \{\varphi g (\text{SOME } z. z \in x)\} = R \text{ `` } \{\varphi g (\text{SOME } z. z \in y)\} \rrbracket \implies x \subseteq y \rangle$ )
    apply (rule subset-antisym; clarify)
    subgoal for x y
    by (metis A-0 is-eq-var-rel' A1-4 Eps-cong equiv-Eps-preserves equiv-class-eq-iff
    quotientI)
    apply (clarsimp simp add: Set.image-def)
    by (simp add: H1-1)
  qed
  have H-1:  $\bigwedge x y. \llbracket \text{Group.group } G; \text{Group.group } (\text{BijGroup } X); R \subseteq X \times X;$ 
     $\varphi \in \text{carrier } G \rightarrow \text{carrier } (\text{BijGroup } X);$ 
     $\forall x \in \text{carrier } G. \forall y \in \text{carrier } G. \varphi (x \otimes_G y) = \varphi x \otimes_{\text{BijGroup } X} \varphi y;$ 
     $x \in \text{carrier } G; y \in \text{carrier } G; x \otimes_G y \in \text{carrier } G \rrbracket \implies$ 
     $(\lambda xa \in X // R. R \text{ `` } \{(\varphi x \otimes_{\text{BijGroup } X} \varphi y) (\text{SOME } z. z \in xa)\}) =$ 
     $(\lambda xa \in X // R. R \text{ `` } \{\varphi x (\text{SOME } z. z \in xa)\}) \otimes_{\text{BijGroup } (X // R)}$ 
     $(\lambda x \in X // R. R \text{ `` } \{\varphi y (\text{SOME } z. z \in x)\})$ 
  proof -

```

```

fix x y
assume
  A1-1: Group.group G and
  A1-2: Group.group (BijGroup X) and
  A1-3:  $\varphi \in \text{carrier } G \rightarrow \text{carrier } (\text{BijGroup } X)$  and
  A1-4:  $\forall x \in \text{carrier } G. \forall y \in \text{carrier } G. \varphi (x \otimes_G y) = \varphi x \otimes_{\text{BijGroup } X} \varphi y$  and
  A1-5:  $x \in \text{carrier } G$  and
  A1-6:  $y \in \text{carrier } G$  and
  A1-7:  $x \otimes_G y \in \text{carrier } G$ 
have H1-0:  $\bigwedge w :: 'X \text{ set}. w \in X // R \implies$ 
  R “  $\{(\varphi x \otimes_{\text{BijGroup } X} \varphi y) (\text{SOME } z. z \in w)\} =$ 
   $((\lambda v \in X // R. R “ \{\varphi x (\text{SOME } z. z \in v)\}) \otimes_{\text{BijGroup } (X // R)}$ 
   $(\lambda x \in X // R. R “ \{\varphi y (\text{SOME } z. z \in x)\})) w$ 
proof –
  fix w
  assume
    A2-0:  $w \in X // R$ 
  have H2-4:  $\varphi y ' w \in X // R$ 
    using ec-er-closed-under-action[where  $w = w$  and  $g = y$ ]
    by (clarsimp simp add: group-hom-axioms-def hom-def A-0 A1-1 A1-2
  is-eq-var-rel' A1-3 A1-4
    A1-6 A2-0)
  hence H2-1: R “  $\{(\varphi x \otimes_{\text{BijGroup } X} \varphi y) (\text{SOME } z. z \in w)\} =$ 
  R “  $\{\varphi (x \otimes_G y) (\text{SOME } z. z \in w)\}$ 
    using A1-4 A1-5 A1-6
    by auto
  also have H2-2:  $\dots = R “ \{\text{SOME } z. z \in \varphi (x \otimes_G y) ' w\}$ 
    using A1-7 equiv-equivar-class-some-eq[where  $w = w$  and  $g = x \otimes_G y$ ]
    by (clarsimp simp add: A1-7 A-0 A2-0 group-action-def group-hom-def
  group-hom-axioms-def
    hom-def)
  also have H2-3:  $\dots = R “ \{\text{SOME } z. z \in \varphi x ' \varphi y ' w\}$ 
    apply (rule meta-mp[of  $\neg(\exists x. x \in w \wedge x \notin X)$ ])
    using A1-1 is-eq-var-rel' A1-3 A1-4 A1-5 A1-6 A2-0
    apply (clarsimp simp add: image-def BijGroup-def restrict-def compose-def
  Pi-def)
    apply (smt (z3) Eps-cong)
    apply (clarify)
    using A-0 A2-0 in-quotient-imp-subset
    by auto
  also have H2-5:  $\dots = R “ \{\varphi x (\text{SOME } z. z \in \varphi y ' w)\}$ 
    using equiv-equivar-class-some-eq[where  $w = \varphi y ' w$  and  $g = x$ ]
    apply (clarsimp simp add: A-0 group-action-def group-hom-def group-hom-axioms-def
  hom-def)
    by (simp add: A1-1 A1-2 is-eq-var-rel' A1-3 A1-4 A1-5 H2-4)
  also have H2-6:  $\dots = R “ \{\varphi x (\text{SOME } z. z \in R “ \{(\text{SOME } z'. z' \in \varphi y ' w)\})\}$ 
    using H2-4 nested-somes[where  $w = \varphi y ' w$  and  $X = X$  and  $R = R$ ] A-0
    by presburger

```


also have *H2-7*: ... = $R \text{ “ } \{\varphi x (\text{SOME } z. z \in R \text{ “ } \{\varphi y (\text{SOME } z'. z' \in w)\})\}$
using *equiv-equivar-class-some-eq*[**where** $g = y$ **and** $w = w$] *H2-6*
by (*simp add: A-0 group-action-def*
group-hom-def group-hom-axioms-def hom-def A1-1 A1-2 is-eq-var-rel'
A1-3 A1-4 A2-0 A1-6)
also have *H2-9*: ... = $((\lambda v \in X // R. R \text{ “ } \{\varphi x (\text{SOME } z. z \in v)\}) \otimes_{\text{BijGroup}} (X // R))$
 $(\lambda x \in X // R. R \text{ “ } \{\varphi y (\text{SOME } z. z \in x)\}) w$
proof–
have *H3-0*: $\bigwedge u. R \text{ “ } \{\varphi y (\text{SOME } z. z \in w)\} \in X // R \implies u \in \text{carrier } G$
 \implies
 $(\lambda v \in X // R. R \text{ “ } \{\varphi u (\text{SOME } z. z \in v)\}) \in \text{Bij } (X // R)$
proof –
fix u
assume
A4-0: $R \text{ “ } \{\varphi y (\text{SOME } z. z \in w)\} \in X // R$ **and**
A4-1: $u \in \text{carrier } G$
have *H4-0*: $\forall g \in \text{carrier } G.$
 $(\lambda x \in X // R. R \text{ “ } \{\varphi g (\text{SOME } z. z \in x)\}) \in \text{carrier } (\text{BijGroup } (X // R))$
by (*simp add: A-0 A1-1 A1-2 A1-3 A1-4 H-0 is-subrel*)
thus $(\lambda v \in X // R. R \text{ “ } \{\varphi u (\text{SOME } z. z \in v)\}) \in \text{Bij } (X // R)$
by (*auto simp add: BijGroup-def A4-1*)
qed
have *H3-1*: $R \text{ “ } \{\varphi y (\text{SOME } z. z \in w)\} \in X // R$
proof–
have *H4-0*: $\varphi y \text{ ‘ } w \in X // R$
using *ec-er-closed-under-action*
by (*simp add: H2-4*)
hence *H4-1*: $R \text{ “ } \{(\text{SOME } z. z \in \varphi y \text{ ‘ } w)\} = \varphi y \text{ ‘ } w$
apply (*clarsimp simp add: image-def*)
apply (*rule subset-antisym*)
using *A-0 equiv-Eps-in in-quotient-imp-closed*
apply *fastforce*
using *A-0 equiv-Eps-in quotient-eq-iff*
by *fastforce*
have *H4-2*: $R \text{ “ } \{\varphi y (\text{SOME } z. z \in w)\} = R \text{ “ } \{(\text{SOME } z. z \in \varphi y \text{ ‘ } w)\}$
using *equiv-equivar-class-some-eq*[**where** $g = y$ **and** $w = w$]
by (*metis A-0 A2-0 H4-0 H4-1 equiv-Eps-in imageI some-equiv-class-id*)
from *H4-0 H4-1 H4-2* **show** $R \text{ “ } \{\varphi y (\text{SOME } z. z \in w)\} \in X // R$
by *auto*
qed
show *?thesis*
apply (*rule meta-mp*[*of* $R \text{ “ } \{\varphi y (\text{SOME } z. z \in w)\} \in X // R$])
apply (*rule meta-mp*[*of* $\forall u \in \text{carrier } G.$])
 $(\lambda v \in X // R. R \text{ “ } \{\varphi u (\text{SOME } z. z \in v)\}) \in \text{Bij } (X // R)$])
using *A2-0 A1-5 A1-6*
apply (*simp add: BijGroup-def compose-def*)
apply *clarify*
by (*simp add: H3-0 H3-1*)+

```

qed
finally show  $R \text{ “ } \{(\varphi x \otimes_{\text{BijGroup } X} \varphi y) (\text{SOME } z. z \in w)\} =$ 
 $((\lambda v \in X // R. R \text{ “ } \{\varphi x (\text{SOME } z. z \in v)\}) \otimes_{\text{BijGroup } (X // R)}$ 
 $(\lambda x \in X // R. R \text{ “ } \{\varphi y (\text{SOME } z. z \in x)\})) w$ 
by simp
qed
have  $H1-1: \bigwedge w :: 'X \text{ set. } w \notin X // R \implies$ 
 $((\lambda v \in X // R. R \text{ “ } \{\varphi x (\text{SOME } z. z \in v)\}) \otimes_{\text{BijGroup } (X // R)}$ 
 $(\lambda x \in X // R. R \text{ “ } \{\varphi y (\text{SOME } z. z \in x)\})) w = \text{undefined}$ 
proof –
fix  $w$ 
assume
 $A2-0: w \notin X // R$ 
have  $H2-0: \bigwedge u. u \in \text{carrier } G \implies (\lambda v \in X // R. R \text{ “ } \{\varphi u (\text{SOME } z. z \in v)\})$ 
 $\in \text{Bij } (X // R)$ 
using  $H-0$ 
apply ( $\text{clarsimp simp add: } A-0 \ A1-1 \ A1-2 \ \text{is-eq-var-rel}' \ A1-3 \ A1-4 \ \text{is-subrel}$ )
by ( $\text{simp add: BijGroup-def}$ )
hence  $H2-1: (\lambda x' \in X // R. R \text{ “ } \{\varphi y (\text{SOME } z. z \in x')\}) \in \text{Bij } (X // R)$ 
using  $A1-6$ 
by auto
from  $H2-0$  have  $H2-2: (\lambda x' \in X // R. R \text{ “ } \{\varphi x (\text{SOME } z. z \in x')\}) \in \text{Bij}$ 
 $(X // R)$ 
by ( $\text{simp add: } A1-5$ )
thus  $((\lambda v \in X // R. R \text{ “ } \{\varphi x (\text{SOME } z. z \in v)\}) \otimes_{\text{BijGroup } (X // R)}$ 
 $(\lambda x \in X // R. R \text{ “ } \{\varphi y (\text{SOME } z. z \in x)\})) w = \text{undefined}$ 
using  $H2-1 \ H2-2$ 
by ( $\text{auto simp add: BijGroup-def compose-def } A2-0$ )
qed
from  $H1-0 \ H1-1$  have  $\bigwedge w. (\lambda xa \in X // R. R \text{ “ } \{(\varphi x \otimes_{\text{BijGroup } X} \varphi y) (\text{SOME}$ 
 $z. z \in xa)\}) w =$ 
 $((\lambda xa \in X // R. R \text{ “ } \{\varphi x (\text{SOME } z. z \in xa)\}) \otimes_{\text{BijGroup } (X // R)}$ 
 $(\lambda x' \in X // R. R \text{ “ } \{\varphi y (\text{SOME } z. z \in x')\})) w$ 
by auto
thus  $(\lambda xa \in X // R. R \text{ “ } \{(\varphi x \otimes_{\text{BijGroup } X} \varphi y) (\text{SOME } z. z \in xa)\}) =$ 
 $(\lambda xa \in X // R. R \text{ “ } \{\varphi x (\text{SOME } z. z \in xa)\}) \otimes_{\text{BijGroup } (X // R)}$ 
 $(\lambda x \in X // R. R \text{ “ } \{\varphi y (\text{SOME } z. z \in x)\})$ 
by ( $\text{simp add: restrict-def}$ )
qed
show  $?thesis$ 
apply ( $\text{clarsimp simp add: group-action-def group-hom-def}$ )
using  $\text{eq-var-rel-axioms}$ 
apply ( $\text{clarsimp simp add: eq-var-rel-def eq-var-rel-axioms-def}$ 
 $\text{group-action-def group-hom-def}$ )
apply ( $\text{rule conjI}$ )
apply ( $\text{simp add: group-BijGroup}$ )
apply ( $\text{clarsimp simp add: group-hom-axioms-def hom-def}$ )
apply ( $\text{intro conjI}$ )

```

```

    apply (rule funcsetI; simp)
    apply (simp add: H-0)
    apply (clarify; rule conjI; intro impI)
    apply (simp add: H-1)
    by (auto simp add: group.is-monoid monoid.m-closed)
qed
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) monoid-scheme and
  X :: 'X set and
   $\varphi$  and
  Y :: 'Y set and
   $\psi$  +
fixes
  f :: 'X  $\Rightarrow$  'Y
assumes
  is-ext-func-bet:
  f  $\in$  (X  $\rightarrow_E$  Y) and
  is-eq-var-func:
   $\bigwedge a g. a \in X \implies g \in \text{carrier } G \implies f (g \odot_{\varphi} a) = g \odot_{\psi} (f a)$ 
begin

```

```

lemma is-eq-var-func' [simp]:
  a  $\in$  X  $\implies$  g  $\in$  carrier G  $\implies$  f ( $\varphi$  g a) =  $\psi$  g (f a)
using is-eq-var-func
by auto

```

end

```

lemma G-set-equiv:
  alt-grp-act G A  $\varphi \implies$  eq-var-subset G A  $\varphi$  A
by (auto simp add: eq-var-subset-def eq-var-subset-axioms-def group-action-def
  group-hom-def group-hom-axioms-def hom-def BijGroup-def Bij-def bij-betw-def)

```

1.3 Basic (G)-Automata Theory

```

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

```

```

locale G-lang = alt-grp-act G A  $\varphi$  + language A L
for
  G :: ('grp, 'b) monoid-scheme and
  A :: 'alpha set (structure) and
   $\varphi$  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
  by (simp add: language-axioms)
end

sublocale G-lang ⊆ language
  by simp

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 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 ⇒ 'alpha ⇒ 'state (δ)
  assumes
    init-state-is-a-state:
    init-state ∈ states and
    fin-states-are-states:
    fin-states ⊆ states and
    trans-func-ext:
    ( $\lambda(state, label). trans-func\ state\ label) \in (states \times labels) \rightarrow_E states$ 
  begin

lemma trans-func-well-def:
   $\bigwedge state\ label. state \in states \implies label \in labels \implies (\delta\ state\ label) \in states$ 
  using trans-func-ext
  by auto

lemma give-input-closed:
   $input \in (labels^*) \implies s \in states \implies (\delta^*)\ s\ input \in states$ 
  apply (induction input arbitrary: s)
  by (auto simp add: trans-func-well-def)

lemma input-under-concat:
   $w \in labels^* \implies v \in labels^* \implies (\delta^*)\ s\ (w @ v) = (\delta^*)\ ((\delta^*)\ s\ w)\ v$ 
  apply (induction w arbitrary: s)
  by auto

```

lemma *eq-pres-under-concat*:

assumes
 $w \in \text{labels}^*$ **and**
 $w' \in \text{labels}^*$ **and**
 $s \in \text{states}$ **and**
 $(\delta^*) s w = (\delta^*) s w'$
shows $\forall v \in \text{labels}^*. (\delta^*) s (w @ v) = (\delta^*) s (w' @ v)$
using *input-under-concat* [**where** $w = w$ **and** $s = s$] *input-under-concat* [**where**
 $w = w'$ **and** $s = s$] *assms*
by *auto*

lemma *trans-to-charact*:

$\bigwedge s a w. [s \in \text{states}; a \in \text{labels}; w \in \text{labels}^*; s = (\delta^*) i w] \implies (\delta^*) i (w @ [a]) = \delta s a$

proof –

fix $s a w$

assume

A-0: $s \in \text{states}$ **and**

A-1: $a \in \text{labels}$ **and**

A-2: $w \in \text{labels}^*$ **and**

A-3: $s = (\delta^*) i w$

have *H-0*: *trans-func* $s a = (\delta^*) s [a]$

by *auto*

from *A-2* *A-3* *H-0* **have** *H-1*: $(\delta^*) s [a] = (\delta^*) ((\delta^*) i w) [a]$

by *simp*

from *A-1* *A-2* **have** *H-2*: $(\delta^*) ((\delta^*) i w) [a] = (\delta^*) i (w @ [a])$

using *input-under-concat*

by *force*

show $(\delta^*) i (w @ [a]) = \delta s a$

using *A-1* *H-0* *A-3* *H-1* *H-2*

by *force*

qed

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 **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:
    s0 ∈ S0 ⇒ a ∈ A ⇒ f (δ0 s0 a) = δ1 (f s0) a
begin

lemma hom-translation:
  input ∈ (A*) ⇒ s ∈ S0 ⇒ (f ((δ0*) s input)) = ((δ1*) (f s) input)
  apply (induction input arbitrary: s)
  by (auto simp add: Aut0.trans-func-well-def pres-trans)

lemma recognise-same-lang:
  input ∈ A* ⇒ ((δ0*) i0 input) ∈ F0 ↔ ((δ1*) i1 input) ∈ F1
  using hom-translation[where input = input and s = i0]
  apply (clarsimp simp add: Aut0.init-state-is-a-state pres-init pres-final)
  apply (induction input)
  apply (clarsimp simp add: Aut0.init-state-is-a-state)
  using Aut0.give-input-closed Aut0.init-state-is-a-state
  by blast

end

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

locale det-G-aut =
  is-aut:      det-aut A S i F δ +
  labels-a-G-set:  alt-grp-act G A φ +
  states-a-G-set:  alt-grp-act G S ψ +
  accepting-is-eq-var: eq-var-subset G S ψ F +
  init-is-eq-var:  eq-var-subset G S ψ {i} +
  trans-is-eq-var: eq-var-func G S × A
  λg∈carrier G. λ(s, a) ∈ (S × A). (ψ g s, φ g a)
  S ψ (λ(s, a) ∈ (S × A). δ s a)
  for A :: 'alpha set (structure) and
    S :: 'states set and
    i F δ and
    G :: ('grp, 'b) monoid-scheme and
    φ ψ
begin

ad hoc-overloading
  star labels-a-G-set.induced-star-map

lemma give-input-eq-var:
  eq-var-func G
  (A* × S) (λg∈carrier G. λ(w, s) ∈ (A* × S). ((φ*) g w, ψ g s))
  S ψ
  (λ(w, s) ∈ (A* × S). (δ*) s w)
proof -

```

have $H-0$: $\bigwedge a w s g.$
 $(\bigwedge s. s \in S \implies (\varphi^*) g w \in A^* \wedge \psi g s \in S \implies$
 $(\delta^*) (\psi g s) ((\varphi^*) g w) = \psi g ((\delta^*) s w)) \implies$
 $s \in S \implies$
 $g \in \text{carrier } G \implies$
 $a \in A \implies \forall x \in \text{set } w. x \in A \implies \psi g s \in S \implies \forall x \in \text{set } ((\varphi^*) g (a \# w)). x$
 $\in A \implies$
 $(\delta^*) (\psi g s) ((\varphi^*) g (a \# w)) = \psi g ((\delta^*) (\delta s a) w)$
proof –
fix $a w s g$
assume
 $A-IH$: $(\bigwedge s. s \in S \implies$
 $(\varphi^*) g w \in A^* \wedge \psi g s \in S \implies$
 $(\delta^*) (\psi g s) ((\varphi^*) g w) = \psi g ((\delta^*) s w))$ **and**
 $A-0$: $s \in S$ **and**
 $A-1$: $\psi g s \in S$ **and**
 $A-2$: $\forall x \in \text{set } ((\varphi^*) g (a \# w)). x \in A$ **and**
 $A-3$: $g \in \text{carrier } G$ **and**
 $A-4$: $a \in A$ **and**
 $A-5$: $\forall x \in \text{set } w. x \in A$
have $H-0$: $((\varphi^*) g (a \# w)) = (\varphi g a) \# (\varphi^*) g w$
using $A-4$ $A-5$ $A-3$
by *auto*
hence $H-1$: $(\delta^*) (\psi g s) ((\varphi^*) g (a \# w))$
 $= (\delta^*) (\psi g s) ((\varphi g a) \# (\varphi^*) g w)$
by *simp*
have $H-2$: $\dots = (\delta^*) ((\delta^*) (\psi g s) [\varphi g a]) ((\varphi^*) g w)$
using *is-aut.input-under-concat*
by *simp*
have $H-3$: $(\delta^*) (\psi g s) [\varphi g a] = \psi g (\delta s a)$
using *trans-is-eq-var.eq-var-func-axioms* $A-4$ $A-5$ $A-0$ $A-1$ $A-3$ **apply** (*clarsimp*
simp del:
 $GMN-simps$ *simp add: eq-var-func-def eq-var-func-axioms-def make-op-def*)
apply (*rule meta-mp*[*of* $\psi g s \in S \wedge \varphi g a \in A \wedge s \in S \wedge a \in A$])
apply *presburger*
apply (*clarify*)
using *labels-a-G-set.element-image*
by *presburger*
have $H-4$: $(\delta^*) (\psi g (\delta s a)) ((\varphi^*) g w) = \psi g ((\delta^*) (\delta s a) w)$
apply (*rule A-IH*[**where** $s1 = \delta s a$])
subgoal
using $A-4$ $A-5$ $A-0$
by (*auto simp add: is-aut.trans-func-well-def*)
using $A-4$ $A-5$ $A-0$ $A-3$ $\langle \delta s a \in S \rangle$ *states-a-G-set.element-image*
by (*metis A-2 Cons-in-lists-iff H-0 in-listsI*)
show $(\delta^*) (\psi g s) ((\varphi^*) g (a \# w)) = \psi g ((\delta^*) (\delta s a) w)$
using $H-0$ $H-1$ $H-2$ $H-3$ $H-4$
by *presburger*
qed

```

show ?thesis
  apply (subst eq-var-func-def)
  apply (subst eq-var-func-axioms-def)
  apply (rule conjI)
  apply (rule prod-group-act[where  $G = G$  and  $A = A^*$  and  $\varphi = (\varphi^*)$ 
    and  $B = S$  and  $\psi = \psi$ ])
  using labels-a-G-set.lists-a-Gset
  apply blast
  apply (simp add: states-a-G-set.group-action-axioms)
  apply (rule conjI)
  apply (simp add: states-a-G-set.group-action-axioms)
  apply (rule conjI)
  apply (subst extensional-funcset-def)
  apply (subst restrict-def)
  apply (subst Pi-def)
  apply (subst extensional-def)
  apply (auto simp add: in-listsI is-aut.give-input-closed)[1]
  apply (subst restrict-def)
  apply (clarsimp simp del: GMN-simps simp add: make-op-def)
  apply (rule conjI; intro impI)
  subgoal for  $w s g$ 
    apply (induction w arbitrary: s)
    apply simp
    apply (clarsimp simp del: GMN-simps)
    by (simp add: H-0 del: GMN-simps)
  apply clarsimp
  by (metis (no-types, lifting) image-iff in-lists-conv-set labels-a-G-set.surj-prop
    list.set-map
    states-a-G-set.element-image)
qed

```

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

```

proof –

```

  have H-0:  $\bigwedge g w. g \in \text{carrier } G \implies w \in A^* \wedge (\delta^*) i w \in F \implies \text{map } (\varphi g) w \in A^*$ 

```

```

  apply (clarsimp)
  using labels-a-G-set.element-image
  by blast

```

```

  have H-1:  $\bigwedge g w. g \in \text{carrier } G \implies w \in A^* \implies (\delta^*) i w \in F \implies (\delta^*) i (\text{map } (\varphi g) w) \in F$ 

```

proof –

```

  fix  $g w$ 
  assume
    A-0:  $g \in \text{carrier } G$  and

```



```

    A-1:  $w \in A^*$  and
    A-2:  $(\delta^*) i w \in F$ 
  have H1-0:  $\psi g ((\delta^*) i w) \in F$ 
    using accepting-is-eq-var.eq-var-subset-axioms
      A-0 A-2 accepting-is-eq-var.is-equivar
    by blast
  have H1-1:  $\psi g i = i$ 
    using init-is-eq-var.eq-var-subset-axioms A-0
      init-is-eq-var.is-equivar
    by auto
  have H1-2:  $\bigwedge w g. \llbracket g \in \text{carrier } G; w \in A^*; (\delta^*) i w \in F \rrbracket \implies (\varphi^*) g w \in A^*$ 
    using H-0
    by auto
  from A-1 have H1-3:  $w \in A^*$ 
    by auto
  show  $(\delta^*) i (\text{map } (\varphi g) w) \in F$ 
    using give-input-eq-var A-0 A-1 H1-1 H1-3
  apply (clarsimp simp del: GMN-simps simp add: eq-var-func-def eq-var-func-axioms-def
    make-op-def)
    using A-2 H1-0 is-aut.init-state-is-a-state H1-2
    by (smt (verit, best) H1-3 labels-a-G-set.induced-star-map-def restrict-apply)
qed
show ?thesis
  apply (clarsimp simp del: GMN-simps simp add: G-lang-def accepted-words-def
    G-lang-axioms-def)
  apply (rule conjI)
  using labels-a-G-set.alt-grp-act-axioms
  apply (auto)[1]
  apply (intro conjI)
  apply (simp add: language.intro)
  apply (rule alt-grp-act.eq-var-one-direction)
  using labels-a-G-set.alt-grp-act-axioms labels-a-G-set.lists-a-Gset
  apply blast
  apply (clarsimp)
  apply (clarsimp)
  by (simp add: H-0 H-1 in-listsI)
qed
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 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
  using reach-det-aut-axioms
  by simp

```

```

locale G-aut-hom = Aut0: reach-det-G-aut A S0 i0 F0 δ0 G φ ψ0 +
  Aut1: reach-det-G-aut A S1 i1 F1 δ1 G φ ψ1 +
  hom-f: aut-hom A S0 i0 F0 δ0 S1 i1 F1 δ1 f +
  eq-var-f: eq-var-func G S0 ψ0 S1 ψ1 f for
  A :: 'alpha set and
  S0 :: 'states-0 set and
  i0 and F0 and δ0 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*: $G\text{-lang } G A \varphi L$
proof –
 have $H0$: $L = \text{accepted-words}$
 apply (*simp add: accepted-words-def*)
 apply (*subst is-recognised [symmetric]*)
 by *simp*
 show $G\text{-lang } G A \varphi L$
 apply (*subst H0*)
 apply (*rule det-G-aut.induced-g-lang[of A S i F δ G φ ψ]*)
 by (*simp add: det-G-aut-axioms*)
qed

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* = $\text{reach-det-aut } A S i F \delta + \text{det-aut-rec-lang } A S i F \delta L$

for A :: '*alpha set* **and**
 S :: '*states set* **and**
 $i F \delta$ **and**
 L :: '*alpha list set*

locale *reach-det-G-aut-rec-lang* = $\text{det-G-aut-rec-lang } A S i F \delta G \varphi \psi L + \text{reach-det-G-aut } A S i F \delta G \varphi \psi$

for A :: '*alpha set* **and**
 S :: '*states set* **and**
 $i F \delta$ **and**
 G :: ('*grp*, '*b*) *monoid-scheme* **and**
 $\varphi \psi$ **and**
 L :: '*alpha list set*

sublocale *reach-det-G-aut-rec-lang* \subseteq *det-G-aut-rec-lang*

apply (*simp add: det-G-aut-rec-lang-def*)
using *reach-det-G-aut-rec-lang-axioms*
by (*simp add: det-G-aut-axioms det-aut-rec-lang-axioms*)

locale *det-G-aut-recog-G-lang* = $\text{det-G-aut-rec-lang } A S i F \delta G \varphi \psi L + G\text{-lang } G A \varphi L$

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
  apply (simp add: det-G-aut-recog-G-lang-def)
  apply (rule conjI)
  apply (simp add: det-G-aut-rec-lang-axioms)
  by (simp add: lang-is-G-lang)

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
  apply (simp add: reach-det-G-aut-rec-G-lang-def)
  apply (rule conjI)
  apply (simp add: reach-det-G-aut-rec-lang-axioms)
  by (simp add: lang-is-G-lang)

lemma (in reach-det-G-aut)
  reach-det-G-aut-rec-lang A S i F  $\delta$  G  $\varphi$   $\psi$  accepted-words
  apply (clarsimp simp del: simp add: reach-det-G-aut-rec-lang-def
    det-G-aut-rec-lang-def det-aut-rec-lang-axioms-def)
  apply (intro conjI)
  apply (simp add: det-G-aut-axioms)
  apply (clarsimp simp add: reach-det-G-aut-axioms accepted-words-def reach-det-aut-rec-lang-def)
  apply (simp add: det-aut-rec-lang-def det-aut-rec-lang-axioms.intro is-aut.det-aut-axioms
    language-def)
  by (simp add: reach-det-G-aut-axioms)

lemma (in det-G-aut) action-on-input:
   $\bigwedge g w. g \in \text{carrier } G \implies w \in A^* \implies \psi g ((\delta^*) i w) = (\delta^*) i ((\varphi^*) g w)$ 
proof–
  fix g w
  assume
    A-0: g  $\in$  carrier G and
    A-1: w  $\in$  A*
  have H-0:  $(\delta^*) (\psi g i) ((\varphi^*) g w) = (\delta^*) i ((\varphi^*) g w)$ 
  using A-0 init-is-eq-var.is-equivar
  by fastforce
  have H-1:  $\psi g ((\delta^*) i w) = (\delta^*) (\psi g i) ((\varphi^*) g w)$ 
  using A-0 A-1 give-input-eq-var
  apply (clarsimp simp del: GMN-simps simp add: eq-var-func-axioms-def eq-var-func-def
    make-op-def)
  apply (rule meta-mp[of  $((\varphi^*) g w) \in A^* \wedge \psi g i \in S$ ])
  using is-aut.init-state-is-a-state A-1
  apply presburger
  using det-G-aut-axioms

```

apply (*clarsimp simp add: det-G-aut-def*)
apply (*rule conjI; rule impI; rule conjI*)
using *labels-a-G-set.element-image*
apply *fastforce*
using *is-aut.init-state-is-a-state states-a-G-set.element-image*
by *blast+*
show $\psi g ((\delta^*) i w) = (\delta^*) i ((\varphi^*) g w)$
using *H-0 H-1*
by *simp*
qed

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

definition (*in det-G-aut*)
reachable-trans :: 'states \Rightarrow 'alpha \Rightarrow 'states (δ_{reach})
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 (ψ_{reach})
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*:
 $\bigwedge g s. g \in carrier G \Longrightarrow s \in S_{reach} \Longrightarrow \psi_{reach} g s = \psi g s$
by (*auto simp add: reachable-action-def reachable-states-def*)

lemma (*in det-G-aut-rec-lang*) *reach-det-aut-is-det-aut-rec-L*:
 $reach-det-G-aut-rec-lang A S_{reach} i (F \cap S_{reach}) \delta_{reach} G \varphi \psi_{reach} L$

proof–

have *H-0*: $(\lambda(x, y). \delta_{reach} x y) \in S_{reach} \times A \rightarrow_E S_{reach}$

proof–

have *H1-0*: $(\lambda(x, y). \delta x y) \in extensional (S \times A)$

using *is-aut.trans-func-ext*

by (*simp add: PiE-iff*)

have *H1-1*: $(\lambda(s', a') \in S_{reach} \times A. \delta s' a') \in extensional (S_{reach} \times A)$

using *H1-0*

by *simp*

have *H1-2*: $(\lambda(s', a') \in S_{reach} \times A. \delta s' a') = (\lambda(x, y). \delta_{reach} x y)$

by (*auto simp add: reachable-trans-def*)

show $(\lambda(x, y). \delta_{reach} x y) \in S_{reach} \times A \rightarrow_E S_{reach}$

apply (*clarsimp simp add: PiE-iff*)

apply (*rule conjI*)

apply (*clarify*)

using *reachable-trans-def*

apply (*simp add: reachable-states-def*)[1]

apply (*metis Cons-in-lists-iff append-Nil2 append-in-lists-conv is-aut.give-input-closed*)

is-aut.init-state-is-a-state is-aut.trans-to-charact)

using *H1-1 H1-2*

by *simp*
qed
have *H-1*: $\bigwedge g. g \in \text{carrier } G \implies$
 ($\bigwedge s. \psi_{\text{reach}} g s = (\text{if } s \in S_{\text{reach}} \text{ then case } (g, s) \text{ of } (x, xa) \Rightarrow \psi x xa \text{ else}$
undefined) \implies
 $\text{bij-betw } (\psi_{\text{reach}} g) S_{\text{reach}} S_{\text{reach}}$
proof –
fix *g*
assume
A1-0: $g \in \text{carrier } G$ **and**
A1-1: ($\bigwedge s. \psi_{\text{reach}} g s =$
 ($\text{if } s \in S_{\text{reach}}$
 then case (g, s) of $(x, xa) \Rightarrow \psi x xa$
 else *undefined*)
have *H1-0*: $\bigwedge r. r \in S_{\text{reach}} \implies (\psi_{\text{reach}} g) r \in S_{\text{reach}}$
using *A1-0*
apply (*clarsimp simp add: reachable-states-def reachable-action-def*)
apply (*rule meta-mp[of $\bigwedge w. w \in A^* \implies ((\varphi^*) g w) \in A^*$]*)
using *action-on-input[where $g = g$]*
apply (*metis in-listsI*)
by (*metis alt-group-act-is-grp-act group-action.element-image labels-a-G-set.lists-a-Gset*)
have *H1-1*: $\bigwedge f T U. \text{bij-betw } f T T \implies f ' U = U \implies U \subseteq T \implies \text{bij-betw}$
 (*restrict f U*) $U U$
apply (*clarsimp simp add: bij-betw-def inj-on-def image-def*)
by (*meson in-mono*)
have *H1-2*: $\psi_{\text{reach}} g = \text{restrict } (\psi g) S_{\text{reach}}$
using *reachable-action-def A1-0*
by (*auto simp add: restrict-def*)
have *H1-3*: $\text{bij-betw } (\psi g) S S \implies (\psi_{\text{reach}} g) ' S_{\text{reach}} = S_{\text{reach}}$
 $\implies S_{\text{reach}} \subseteq S \implies \text{bij-betw } (\psi_{\text{reach}} g) S_{\text{reach}} S_{\text{reach}}$
by (*metis H1-2 bij-betw-imp-inj-on inj-on-imp-bij-betw inj-on-restrict-eq inj-on-subset*)
have *H1-4*: $\bigwedge w s. s = (\delta^*) i w \implies$
 $\forall x \in \text{set } w. x \in A \implies$
 $\exists x. (\exists w \in A^*. (\delta^*) i w = x) \wedge (\delta^*) i w = \psi_{\text{reach}} g x$
proof –
fix *w s*
assume
A2-0: $\forall x \in \text{set } w. x \in A$ **and**
A2-1: $s = (\delta^*) i w$
have *H2-0*: $(\text{inv } G g) \in \text{carrier } G$
apply (*rule meta-mp[of group G]*)
using *A1-0*
apply *simp*
using *det-G-aut-rec-lang-axioms*
by (*auto simp add: det-G-aut-rec-lang-def*
det-aut-rec-lang-axioms-def det-G-aut-def group-action-def group-hom-def)
have *H2-1*: $\psi (\text{inv } G g) s = (\delta^*) i ((\varphi^*) (\text{inv } G g) w)$
apply (*simp del: GMN-simps add: A2-1*)
apply (*rule action-on-input[where $g = (\text{inv } G g)$ and $w = w$]*)

```

    using H2-0 A2-0
    by auto
  have H2-2:  $((\varphi^*) (inv\ G\ g)\ w) \in A^*$ 
    using A2-0 H2-0 det-G-aut-rec-lang-axioms
    apply (clarsimp)
    using labels-a-G-set.surj-prop list.set-map
    by fastforce
  have H2-3:  $\exists w \in A^*. (\delta^*)\ i\ w = \psi\ (inv\ G\ g)\ s$ 
    by (metis H2-1 H2-2)
  from H2-3 have H2-4:  $\psi\ (inv\ G\ g)\ s \in S_{reach}$ 
    by (simp add: reachable-states-def)
  have H2-5:  $\psi_{reach}\ g\ (\psi\ (inv\ G\ g)\ s) = \psi\ g\ (\psi\ (inv\ G\ g)\ s)$ 
    apply (rule reachable-action-is-restrict)
    using A1-0 H2-4
    by simp+
  have H2-6:  $(\delta^*)\ i\ w = \psi_{reach}\ g\ (\psi\ (inv\ G\ g)\ s)$ 
    apply (simp add: H2-5 A2-1)
    by (metis A1-0 A2-0 in-listsI A2-1 H2-5 is-aut.give-input-closed
      is-aut.init-state-is-a-state states-a-G-set.bij-prop1 states-a-G-set.orbit-sym-aux)
  show  $\exists x. (\exists w \in A^*. (\delta^*)\ i\ w = x) \wedge (\delta^*)\ i\ w = \psi_{reach}\ g\ x$ 
    using H2-3 H2-6
    by blast
qed
show bij-betw  $(\psi_{reach}\ g)\ S_{reach}\ S_{reach}$ 
  apply (rule H1-3)
  apply (simp add: A1-0 bij-betw-def states-a-G-set.inj-prop states-a-G-set.surj-prop)
  apply (clarsimp simp add: image-def H1-0)
  apply (rule subset-antisym; simp add: Set.subset-eq; clarify)
  using H1-0
  apply auto[1]
  subgoal for s
    apply (clarsimp simp add: reachable-states-def)
    by (simp add: H1-4)
  apply (simp add: reachable-states-def Set.subset-eq; rule allI; rule impI)
  using is-aut.give-input-closed is-aut.init-state-is-a-state
  by auto
qed
have H-2: group G
  using det-G-aut-rec-lang-axioms
  by (auto simp add: det-G-aut-rec-lang-def det-G-aut-def group-action-def
    group-hom-def)
have H-3:  $\bigwedge g. g \in carrier\ G \implies \psi_{reach}\ g \in carrier\ (BijGroup\ S_{reach})$ 
  subgoal for g
    using reachable-action-def[where g = g]
    apply (simp add: BijGroup-def Bij-def extensional-def)
    by (simp add: H-1)
  done
have H-4:  $\bigwedge x\ y. x \in carrier\ G \implies y \in carrier\ G \implies \psi_{reach}\ (x \otimes_G y) = \psi_{reach}$ 
 $x \otimes_{BijGroup\ S_{reach}}$ 

```

$\psi_{reach} y$

```
proof –
  fix g h
  assume
    A1-0:  $g \in carrier\ G$  and
    A1-1:  $h \in carrier\ G$ 
  have H1-0:  $\bigwedge g . g \in carrier\ G \implies \psi_{reach} g = restrict\ (\psi\ g)\ S_{reach}$ 
    using reachable-action-def
    by (auto simp add: restrict-def)
  from H1-0 have H1-1:  $\psi_{reach} (g \otimes_G h) = restrict\ (\psi\ (g \otimes_G h))\ S_{reach}$ 
    by (simp add: A1-0 A1-1 H-2 group.subgroup-self subgroup.m-closed)
  have H1-2:  $\psi_{reach} g \otimes_{BijGroup\ S_{reach}} \psi_{reach} h =$ 
    ( $restrict\ (\psi\ g)\ S_{reach}$ )  $\otimes_{BijGroup\ S_{reach}}$ 
    ( $restrict\ (\psi\ h)\ S_{reach}$ )
    using A1-0 A1-1 H1-0
    by simp
  have H1-3:  $\bigwedge g . g \in carrier\ G \implies \psi_{reach} g \in carrier\ (BijGroup\ S_{reach})$ 
    by (simp add: H-3)
  have H1-4:  $\bigwedge x y . x \in carrier\ G \implies y \in carrier\ G \implies \psi\ (x \otimes_G y) = \psi\ x$ 
 $\otimes_{BijGroup\ S}\ \psi\ y$ 
    using det-G-aut-axioms
    by (simp add: det-G-aut-def group-action-def group-hom-def group-hom-axioms-def
    hom-def)
  hence H1-5:  $\psi\ (g \otimes_G h) = \psi\ g \otimes_{BijGroup\ S}\ \psi\ h$ 
    using A1-0 A1-1
    by simp
  have H1-6: ( $\lambda x .$  if  $x \in S_{reach}$ 
    then if (if  $x \in S_{reach}$ 
      then  $\psi\ h\ x$ 
      else undefined)  $\in S_{reach}$ 
    then  $\psi\ g$  (if  $x \in S_{reach}$ 
      then  $\psi\ h\ x$ 
      else undefined)
    else undefined
    else undefined) =
    ( $\lambda x .$  if  $x \in S_{reach}$ 
      then  $\psi\ g$  ( $\psi\ h\ x$ )
      else undefined)
  apply (rule meta-mp[of  $\bigwedge x . x \in S_{reach} \implies (\psi\ h\ x) \in S_{reach}$ ])
  using H1-3[where  $g1 = h$ ] A1-1 H1-0
  by (auto simp add: A1-1 BijGroup-def Bij-def bij-betw-def)
  have H1-7: ... = ( $\lambda x .$  if  $x \in S_{reach}$ 
    then if  $x \in S$ 
      then  $\psi\ g$  ( $\psi\ h\ x$ )
      else undefined
    else undefined)
  apply (clarsimp simp add: reachable-states-def)
  by (metis is-aut.give-input-closed is-aut.init-state-is-a-state)
  have H1-8: ( $restrict\ (\psi\ g)\ S_{reach}$ )  $\otimes_{BijGroup\ S_{reach}}$  ( $restrict\ (\psi\ h)\ S_{reach}$ ) =
```



```

    restrict ( $\psi (g \otimes_G h)$ )  $S_{reach}$ 
    apply (rule meta-mp[of  $\bigwedge g. g \in carrier\ G \implies restrict (\psi\ g)\ S_{reach} \in Bij$ 
 $S_{reach} \wedge$ 
 $\psi\ g \in Bij\ S$ ])
    apply (clarsimp simp add: H1-5 BijGroup-def; intro conjI; intro impI)
  subgoal
    using A1-0 A1-1
    apply (clarsimp simp add: compose-def restrict-def)
    by (simp add: H1-6 H1-7)
    apply (simp add: A1-0 A1-1)+
  subgoal for  $g$ 
    using H1-3[where  $g1 = g$ ] H1-0[of  $g$ ]
    by (simp add: BijGroup-def states-a-G-set.bij-prop0)
  done
  show  $\psi_{reach} (g \otimes_G h) =$ 
     $\psi_{reach}\ g \otimes_{BijGroup\ S_{reach}} \psi_{reach}\ h$ 
    by (simp add: H1-1 H1-2 H1-8)
qed
have H-5:  $\bigwedge w' w g. g \in carrier\ G \implies$ 
   $(\delta^*)\ i\ w \in F \implies \forall x \in set\ w. x \in A \implies (\delta^*)\ i\ w' = (\delta^*)\ i\ w \implies \forall x \in set$ 
 $w'. x \in A \implies$ 
   $\exists w' \in A^*. (\delta^*)\ i\ w' = \psi\ g\ ((\delta^*)\ i\ w)$ 
proof -
  fix  $w' w g$ 
  assume
    A1-0:  $g \in carrier\ G$  and
    A1-1:  $(\delta^*)\ i\ w \in F$  and
    A1-2:  $\forall x \in set\ w. x \in A$  and
    A1-3:  $(\delta^*)\ i\ w' = (\delta^*)\ i\ w$  and
    A1-4:  $\forall x \in set\ w. x \in A$ 
  from A1-1 A1-2 have H1-0:  $((\delta^*)\ i\ w) \in S_{reach}$ 
    using reachable-states-def
    by auto
  have H1-1:  $\psi\ g\ ((\delta^*)\ i\ w) = ((\delta^*)\ i\ ((\varphi^*)\ g\ w))$ 
    using give-input-eq-var
    apply (clarsimp simp add: eq-var-func-def eq-var-func-axioms-def simp del:
  GMN-simps)
    using A1-0 A1-2 action-on-input
    by blast
  have H1-2:  $(\varphi^*)\ g\ w \in A^*$ 
    using A1-0 A1-2
    by (metis in-listsI alt-group-act-is-grp-act group-action.element-image
  labels-a-G-set.lists-a-Gset)
  show  $\exists wa \in A^*. (\delta^*)\ i\ wa = \psi\ g\ ((\delta^*)\ i\ w)$ 
    by (metis H1-1 H1-2)
qed
have H-6: alt-grp-act  $G\ S_{reach}\ \psi_{reach}$ 
  apply (clarsimp simp add: group-action-def group-hom-def group-hom-axioms-def
  hom-def)

```

```

apply (intro conjI)
  apply (simp add: H-2)
subgoal
  by (simp add: group-BijGroup)
  apply clarify
  apply (simp add: H-3)
  by (simp add: H-4)
have H-7:  $\bigwedge g w. g \in \text{carrier } G \implies (\delta^*) i w \in F \implies \forall x \in \text{set } w. x \in A \implies$ 
   $\exists x. x \in F \wedge (\exists w \in A^*. (\delta^*) i w = x) \wedge (\delta^*) i w = \psi g x$ 
proof -
  fix g w
  assume
    A1-0:  $g \in \text{carrier } G$  and
    A1-1:  $(\delta^*) i w \in F$  and
    A1-2:  $\forall x \in \text{set } w. x \in A$ 
  have H1-0:  $(\text{inv } G g) \in \text{carrier } G$ 
  by (meson A1-0 group.inv-closed group-hom.axioms(1) labels-a-G-set.group-hom)
  have H1-1:  $((\delta^*) i w) \in S_{\text{reach}}$ 
  using A1-1 A1-2 reachable-states-def
  by auto
  have H1-2:  $\psi_{\text{reach}} (\text{inv } G g) ((\delta^*) i w) = \psi (\text{inv } G g) ((\delta^*) i w)$ 
  apply (rule reachable-action-is-restrict)
  using H1-0 H1-1
  by auto
  have H1-3:  $\psi_{\text{reach}} g (\psi (\text{inv } G g) ((\delta^*) i w)) = ((\delta^*) i w)$ 
  by (smt (verit) A1-0 H1-1 H-6 H1-2
    alt-group-act-is-grp-act group-action.bij-prop1 group-action.orbit-sym-aux)
  have H1-4:  $\psi (\text{inv } G g) ((\delta^*) i w) \in F$ 
  using A1-1 H1-0 accepting-is-eq-var.is-equivar
  by blast
  have H1-5:  $\psi (\text{inv } G g) ((\delta^*) i w) \in F \wedge (\delta^*) i w = \psi g (\psi (\text{inv } G g) ((\delta^*) i$ 
w))
  using H1-4 H1-3 A1-0 A1-1 H1-0 H1-1 reachable-action-is-restrict
  by (metis H-6 alt-group-act-is-grp-act
    group-action.element-image)
  have H1-6:  $\psi (\text{inv } G g) ((\delta^*) i w) = ((\delta^*) i ((\varphi^*) (\text{inv } G g) w))$ 
  using give-input-eq-var
  apply (clarsimp simp add: eq-var-func-def eq-var-func-axioms-def simp del:
GMN-simps)
  using A1-2 H1-0 action-on-input
  by blast
  have H1-7:  $(\varphi^*) (\text{inv } G g) w \in A^*$ 
  by (metis A1-2 in-listsI H1-0 alt-group-act-is-grp-act group-action.element-image
    labels-a-G-set.lists-a-Gset)
  thus  $\exists x. x \in F \wedge (\exists w \in A^*. (\delta^*) i w = x) \wedge (\delta^*) i w = \psi g x$ 
  using H1-5 H1-6 H1-7
  by metis
qed
have H-8:  $\bigwedge r a g. r \in S_{\text{reach}} \implies a \in A \implies \psi_{\text{reach}} g r \in S_{\text{reach}} \wedge \varphi g a \in$ 

```

$A \implies g \in \text{carrier } G \implies$
 $\delta_{\text{reach}} (\psi_{\text{reach}} g r) (\varphi g a) = \psi_{\text{reach}} g (\delta_{\text{reach}} r a)$

proof –

fix $r a g$

assume

$A1-0$: $r \in S_{\text{reach}}$ **and**

$A1-1$: $a \in A$ **and**

$A1-2$: $\psi_{\text{reach}} g r \in S_{\text{reach}} \wedge \varphi g a \in A$ **and**

$A1-3$: $g \in \text{carrier } G$

have $H1-0$: $r \in S \wedge \psi g r \in S$

apply (*rule conjI*)

subgoal

using $A1-0$

apply (*clarsimp simp add: reachable-states-def*)

by (*simp add: in-listsI is-aut.give-input-closed is-aut.init-state-is-a-state*)

using $\langle r \in S \rangle A1-3$ *states-a-G-set.element-image*

by *blast*

have $H1-1$: $\bigwedge a b g . a \in S \wedge b \in A \implies g \in \text{carrier } G \implies$
(if $\psi g a \in S \wedge \varphi g b \in A$ then $\delta (\psi g a) (\varphi g b)$ else undefined) =
 $\psi g (\delta a b)$

using *det-G-aut-axioms A1-0 A1-1 A1-3*

apply (*clarsimp simp add: det-G-aut-def eq-var-func-def eq-var-func-axioms-def*)

by *presburger+*

hence $H1-2$: $\psi g (\delta r a) = (\delta (\psi g r) (\varphi g a))$

using $H1-1$ [**where** $a1 = r$ **and** $b1 = a$ **and** $g1 = g$] $H1-0 A1-1 A1-2 A1-3$

by *simp*

have $H1-3$: $\bigwedge a w . a \in A \implies w \in A^* \implies \exists w' \in A^* . (\delta^*) i w' = \delta ((\delta^*) i w) a$

proof –

fix $a w$

assume

$A2-0$: $a \in A$ **and**

$A2-1$: $w \in A^*$

have $H2-0$: $(w @ [a]) \in A^* \wedge (w @ [a]) \in A^* \implies (\delta^*) i (w @ [a]) = \delta ((\delta^*)$
 $i w) a$

by (*simp add: is-aut.give-input-closed is-aut.trans-to-charact*
is-aut.init-state-is-a-state)

show $\exists w' \in A^* . (\delta^*) i w' = \delta ((\delta^*) i w) a$

using $H2-0$

apply *clarsimp*

by (*metis A2-0 A2-1 append-in-lists-conv lists.Cons lists.Nil*)

qed

have $H1-4$: $\psi_{\text{reach}} g (\delta_{\text{reach}} r a) = \psi g (\delta r a)$

apply (*clarsimp simp add: reachable-action-def reachable-trans-def*)

using $A1-0 A1-1 A1-3 H1-0 H1-3$

using *reachable-states-def* **by** *fastforce*

have $H1-5$: $\psi g r = \psi_{\text{reach}} g r$

using $A1-0 A1-3$

by (*auto simp add: reachable-action-def*)

hence *H1-6*: $\psi \ g \ r \in S_{reach}$
 using *A1-2*
 by *simp*
 have *H1-7*: $\delta_{reach} (\psi_{reach} \ g \ r) (\varphi \ g \ a) = \delta (\psi \ g \ r) (\varphi \ g \ a)$
 using *A1-0 A1-1 A1-2 A1-3*
 by (*auto simp del: simp add: reachable-trans-def reachable-action-def*)
 show $\delta_{reach} (\psi_{reach} \ g \ r) (\varphi \ g \ a) = \psi_{reach} \ g (\delta_{reach} \ r \ a)$
 using *H1-2 H1-4 H1-7*
 by *auto*
qed
 have *H-9*: $\bigwedge a \ w \ s. \llbracket (\bigwedge s. s \in S_{reach} \implies (\delta^*) \ s \ w = (\delta_{reach}^*) \ s \ w);$
 $a \in A \wedge (\forall x \in set \ w. x \in A); s \in S_{reach} \rrbracket \implies (\delta^*) (\delta \ s \ a) \ w = (\delta_{reach}^*)$
 $(\delta_{reach} \ s \ a) \ w$
proof–
 fix $a \ w \ s$
 assume
A1-IH: $(\bigwedge s. s \in S_{reach} \implies (\delta^*) \ s \ w = (\delta_{reach}^*) \ s \ w)$ **and**
A1-0: $a \in A \wedge (\forall x \in set \ w. x \in A)$ **and**
A1-1: $s \in S_{reach}$
 have *H1-0*: $\delta_{reach} \ s \ a = \delta \ s \ a$
 using *A1-1*
 apply (*clarsimp simp add: reachable-trans-def*)
 apply (*rule meta-mp[of det-aut A S i F δ]*)
 using *det-aut.trans-func-ext[where labels = A and states = S and*
init-state = i and fin-states = F and trans-func = δ]
 apply (*simp add: extensional-def*)
 by (*auto simp add: A1-0*)
 show $(\delta^*) (\delta \ s \ a) \ w = (\delta_{reach}^*) (\delta_{reach} \ s \ a) \ w$
 apply (*simp add: H1-0*)
 apply (*rule A1-IH[where s1 = $\delta \ s \ a$]*)
 using *A1-0 A1-1*
 apply (*simp add: reachable-states-def*)
 by (*metis Cons-in-lists-iff append-Nil2 append-in-lists-conv is-aut.give-input-closed*
is-aut.init-state-is-a-state is-aut.trans-to-charact)
qed
 show *?thesis*
 apply (*clarsimp simp del: GMN-simps simp add: reach-det-G-aut-rec-lang-def*
det-G-aut-rec-lang-def det-G-aut-def det-aut-def)
 apply (*intro conjI*)
subgoal
 apply (*simp add: reachable-states-def*)
 by (*meson give-input.simps(1) lists.Nil*)
 apply (*simp add: H-0*)
 using *labels-a-G-set.alt-grp-act-axioms*
 apply (*auto*)[1]
 apply (*rule H-6*)
subgoal
 apply (*clarsimp simp add: eq-var-subset-def eq-var-subset-axioms-def*)
 apply (*rule conjI*)

```

using H-6
  apply (auto)[1]
  apply (simp del: add: reachable-states-def)[1]
  apply (clarify; rule subset-antisym; simp add: Set.subset-eq; clarify)
  apply (rule conjI)
subgoal for  $g - w$ 
  apply (clarsimp simp add: reachable-action-def reachable-states-def)
  using accepting-is-eq-var.is-equivar
  by blast
subgoal for  $g - w$ 
  apply (clarsimp simp add: reachable-action-def reachable-states-def)
  apply (rule conjI; clarify)
  apply (auto)[2]
  by (simp add: H-5)
  apply (clarsimp simp add: reachable-states-def Int-def reachable-action-def )
  apply (clarsimp simp add: image-def)
  by (simp add: H-7)
subgoal
  apply (clarsimp simp add: eq-var-subset-def)
  apply (rule conjI)
  using H-6
  apply (auto)[1]
  apply (clarsimp simp add: eq-var-subset-axioms-def)
  apply (simp add:  $\langle i \in S_{reach} \rangle$ )
  apply (simp add: reachable-action-def)
  using  $\langle i \in S_{reach} \rangle$  init-is-eq-var.is-equivar
  by fastforce
subgoal
  apply (clarsimp simp add: eq-var-func-def eq-var-func-axioms-def)
  apply (intro conjI)
  using H-6 alt-grp-act.axioms
  labels-a-G-set.group-action-axioms prod-group-act labels-a-G-set.alt-grp-act-axioms
  apply blast
  using H-6
  apply (auto)[1]
  apply (rule funcsetI; clarsimp)
subgoal for  $s a$ 
  apply (clarsimp simp add: reachable-states-def reachable-trans-def)
  by (metis Cons-in-lists-iff append-Nil2 append-in-lists-conv in-listsI
      is-aut.give-input-closed is-aut.init-state-is-a-state is-aut.trans-to-charact)
  apply (intro allI; clarify; rule conjI; intro impI)
  apply (simp add: H-8)
  using G-set-equiv H-6 eq-var-subset.is-equivar
  labels-a-G-set.element-image
  by fastforce
  apply (rule meta-mp[ $\text{of } \bigwedge w s. w \in A^* \implies s \in S_{reach} \implies (\delta^*) s w = (\delta_{reach}^*)$ 
 $s w$ ])
subgoal
  using det-G-aut-rec-lang-axioms

```

```

apply (clarsimp simp add: det-aut-rec-lang-axioms-def det-aut-rec-lang-def
  det-G-aut-rec-lang-def det-aut-def)
apply (intro conjI)
using ⟨i ∈ S_reach⟩
  apply blast
using H-0
  apply blast
by (metis (mono-tags, lifting) ⟨i ∈ S_reach⟩ mem-Collect-eq reachable-states-def)
subgoal for w s
  apply (induction w arbitrary: s)
  apply (clarsimp)
  apply (simp add: in-lists-conv-set)
  by (simp add: H-9)
apply (clarsimp simp add: reach-det-G-aut-def det-G-aut-def det-aut-def)
apply (intro conjI)
  apply (simp add: ⟨i ∈ S_reach⟩)
  apply (simp add: H-0)
  apply (simp add: labels-a-G-set.group-action-axioms)
using ⟨alt-grp-act G S_reach ψ_reach⟩
  apply (auto)[1]
  apply (simp add: ⟨eq-var-subset G S_reach ψ_reach (F ∩ S_reach)⟩)
  apply (simp add: ⟨eq-var-subset G S_reach ψ_reach {i}⟩)
using ⟨eq-var-func G (S_reach × A) (λg∈carrier G. λ(s, a)∈S_reach × A. (ψ_reach
g s, φ g a))
  S_reach ψ_reach (λ(x, y)∈S_reach × A. δ_reach x y)⟩
  apply blast
apply (simp add: reach-det-aut-axioms-def reach-det-aut-def reachable-states-def)
apply (rule meta-mp[of ∧s input. s ∈ S_reach ⇒ input ∈ A* ⇒
(δ_reach*) s input = (δ*) s input])
using ⟨i ∈ S_reach⟩
  apply (metis (no-types, lifting) ⟨(∧w s. [[w ∈ A*; s ∈ S_reach]] ⇒
(δ*) s w = (δ_reach*) s w) ⇒ det-aut-rec-lang A S_reach i (F ∩ S_reach) δ_reach
L⟩ det-aut-rec-lang-def
  reachable-states-def)
  by (simp add: ⟨(∧w s. [[w ∈ A*; s ∈ S_reach]] ⇒ (δ*) s w = (δ_reach*) s w)⟩)
qed

```

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-equival*:

equiv (A*) *rel-MN*

by (auto simp add: *rel-MN-def equiv-def refl-on-def sym-def trans-def*)

abbreviation*MN-equiv***where** $MN-equiv \equiv A^* // rel-MN$ **definition***alt-natural-map-MN* :: 'alpha list \Rightarrow 'alpha list set $([-]_{MN})$ **where** $[w]_{MN} = rel-MN \text{ `` } \{w\}$ **definition***MN-trans-func* :: ('alpha list set) \Rightarrow 'alpha \Rightarrow 'alpha list set (δ_{MN}) **where** *MN-trans-func* $W' a' =$ $(\lambda(W, a) \in MN-equiv \times A. rel-MN \text{ `` } \{(SOME w. w \in W) @ [a]\}) (W', a')$ **abbreviation***MN-init-state***where** $MN-init-state \equiv [Nil::'alpha list]_{MN}$ **abbreviation***MN-fin-states***where** $MN-fin-states \equiv \{v. \exists w \in L. v = [w]_{MN}\}$ **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*: $w' \in [w]_{MN} \implies \forall g \in carrier\ G. (g \odot \varphi^* w') \in [g \odot \varphi^* w]_{MN}$ **proof**–**assume** *A-0*: $w' \in [w]_{MN}$ **have** *H-0*: $\bigwedge g. \llbracket (w, w') \in \equiv_{MN}; g \in carrier\ G; group-hom\ G\ (BijGroup\ A)\ \varphi;$
group-hom $G\ (BijGroup\ (A^*))\ (\lambda g \in carrier\ G. restrict\ (map\ (\varphi\ g))\ (A^*)); L \subseteq A^*;$ $\forall g \in carrier\ G. map\ (\varphi\ g)\ \text{' } (L \cap A^*) \cup (\lambda x. undefined)\ \text{' } (L \cap \{x. x \notin A^*\}) = L;$ $\forall x \in set\ w. x \in A; w' \in A^* \rrbracket \implies (map\ (\varphi\ g)\ w, map\ (\varphi\ g)\ w') \in \equiv_{MN}$ **proof**–**fix** g **assume***A1-0*: $(w, w') \in \equiv_{MN}$ **and***A1-1*: $g \in carrier\ G$ **and***A1-2*: *group-hom* $G\ (BijGroup\ A)\ \varphi$ **and***A1-3*: *group-hom* $G\ (BijGroup\ (A^*))\ (\lambda g \in carrier\ G. restrict\ (map\ (\varphi\ g)))$ (A^*) **and***A1-4*: $L \subseteq A^*$ **and**

A1-5: $\forall g \in \text{carrier } G$.
 $\text{map } (\varphi g) \text{ ' } (L \cap A^*) \cup (\lambda x. \text{undefined}) \text{ ' } (L \cap \{x. x \notin A^*\}) = L$ **and**
 A1-6: $\forall x \in \text{set } w. x \in A$ **and**
 A1-7: $w' \in A^*$
have H1-0: $\bigwedge v w w'. \llbracket g \in \text{carrier } G; \text{group-hom } G \text{ (BijGroup } A) \varphi;$
 $\text{group-hom } G \text{ (BijGroup } (A^*)) (\lambda g \in \text{carrier } G. \text{restrict } (\text{map } (\varphi g)) (A^*));$
 $L \subseteq A^*; \forall g \in \text{carrier } G.$
 $\{y. \exists x \in L \cap A^*. y = \text{map } (\varphi g) x\} \cup \{y. y = \text{undefined} \wedge (\exists x. x \in L \wedge x \notin$
 $A^*)\} = L;$
 $\forall x \in \text{set } w. x \in A; \forall v \in A^*. (w @ v \in L) = (w' @ v \in L); \forall x \in \text{set } w'. x \in A;$
 $\forall x \in \text{set } v. x \in A;$
 $\text{map } (\varphi g) w @ v \in L \rrbracket \implies \text{map } (\varphi g) w' @ v \in L$
proof –
fix $v w w'$
assume
 A2-0: $g \in \text{carrier } G$ **and**
 A2-1: $L \subseteq A^*$ **and**
 A2-2: $\text{group-hom } G \text{ (BijGroup } A) \varphi$ **and**
 A2-3: $\text{group-hom } G \text{ (BijGroup } (A^*)) (\lambda g \in \text{carrier } G. \text{restrict } (\text{map } (\varphi g)))$
 (A^*) **and**
 A2-4: $\forall g \in \text{carrier } G. \{y. \exists x \in L \cap A^*. y = \text{map } (\varphi g) x\} \cup$
 $\{y. y = \text{undefined} \wedge (\exists x. x \in L \wedge x \notin A^*)\} = L$ **and**
 A2-5: $\forall x \in \text{set } w. x \in A$ **and**
 A2-6: $\forall x \in \text{set } w'. x \in A$ **and**
 A2-7: $\forall v \in A^*. (w @ v \in L) = (w' @ v \in L)$ **and**
 A2-8: $\forall x \in \text{set } v. x \in A$ **and**
 A2-9: $\text{map } (\varphi g) w @ v \in L$
have H2-0: $\forall g \in \text{carrier } G. \{y. \exists x \in L \cap A^*. y = \text{map } (\varphi g) x\} = L$
using A2-1 A2-4 *subset-eq*
by (*smt (verit, ccfv-SIG) Collect-mono-iff sup.orderE*)
hence H2-1: $\forall g \in \text{carrier } G. \{y. \exists x \in L. y = \text{map } (\varphi g) x\} = L$
using A2-1 *inf.absorb-iff1*
by (*smt (verit, ccfv-SIG) Collect-cong*)
hence H2-2: $\forall g \in \text{carrier } G. \forall x \in L. \text{map } (\varphi g) x \in L$
by *auto*
from A2-2 **have** H2-3: $\forall h \in \text{carrier } G. \forall a \in A. (\varphi h) a \in A$
by (*auto simp add: group-hom-def BijGroup-def group-hom-axioms-def*
hom-def Bij-def
bij-betw-def)
from A2-8 **have** H2-4: $v \in A^*$
by (*simp add: in-listsI*)
hence H2-5: $\forall h \in \text{carrier } G. \text{map } (\varphi h) v \in A^*$
using H2-3
by *fastforce*
hence H2-6: $\forall h \in \text{carrier } G. (w @ (\text{map } (\varphi h) v) \in L) = (w' @ (\text{map } (\varphi h)$
 $v) \in L)$
using A2-7
by *force*
hence H2-7: $(w @ (\text{map } (\varphi (\text{inv}_G g)) v) \in L) = (w' @ (\text{map } (\varphi (\text{inv}_G g)) v)$

$\in L$)
using *A2-0*
by (*meson A2-7 A2-1 append-in-lists-conv in-mono*)
have ($\text{map } (\varphi g) w \in (A^*)$)
using *A2-0 A2-2 A2-5 H2-3*
by (*auto simp add: group-hom-def group-hom-axioms-def hom-def Bij-Group-def Bij-def bij-betw-def*)
hence *H2-8*: $\forall w \in A^*. \forall g \in \text{carrier } G. \text{map } (\varphi (\text{inv}_G g)) ((\text{map } (\varphi g) w) @ v)$
 $=$
 $w @ (\text{map } (\varphi (\text{inv}_G g)) v)$
using *act-maps-n-distrib triv-act-map A2-0 A2-2 A2-3 H2-4*
apply (*clarsimp*)
by (*smt (verit, del-insts) comp-apply group-action.intro group-action.orbit-sym-aux map-idI*)
have *H2-9*: $\text{map } (\varphi (\text{inv}_G g)) ((\text{map } (\varphi g) w) @ v) \in L$
using *A2-9 H2-1 H2-2 A2-1*
apply *clarsimp*
by (*metis A2-0 A2-2 group.inv-closed group-hom.axioms(1) list.map-comp map-append*)
hence *H2-10*: $w @ (\text{map } (\varphi (\text{inv}_G g)) v) \in L$
using *H2-8 A2-0*
by (*auto simp add: A2-5 in-listsI*)
hence *H2-11*: $w' @ (\text{map } (\varphi (\text{inv}_G g)) v) \in L$
using *H2-7*
by *simp*
hence *H2-12*: $\text{map } (\varphi (\text{inv}_G g)) ((\text{map } (\varphi g) w') @ v) \in L$
using *A2-0 H2-8 A2-1 subsetD*
by (*metis append-in-lists-conv*)
have *H2-13*: $\forall g \in \text{carrier } G. \text{restrict } (\text{map } (\varphi g)) (A^*) \in \text{Bij } (A^*)$
using *alt-grp-act.lists-a-Gset* **where** $G = G$ **and** $X = A$ **and** $\varphi = \varphi$ *A1-3*
by (*auto simp add: group-action-def group-hom-def group-hom-axioms-def Pi-def hom-def BijGroup-def*)
have *H2-14*: $\forall g \in \text{carrier } G. \text{restrict } (\text{map } (\varphi g)) L ' L = L$
using *H2-2*
apply (*clarsimp simp add: Set.image-def*)
using *H2-1*
by *blast*
have *H2-15*: $\text{map } (\varphi g) w' \in A^*$
using *A2-0 A2-1 H2-13 H2-2*
by (*metis H2-11 append-in-lists-conv image-eqI lists-image subset-eq surj-prop*)
have *H2-16*: $\text{inv}_G g \in \text{carrier } G$
by (*metis A2-0 A2-2 group.inv-closed group-hom.axioms(1)*)
thus $\text{map } (\varphi g) w' @ v \in L$
using *A2-0 A2-1 A2-2 H2-4 H2-12 H2-13 H2-14 H2-15 H2-16 group.inv-closed group-hom.axioms(1)*
alt-grp-act.lists-a-Gset **where** $G = G$ **and** $X = A$ **and** $\varphi = \varphi$
pre-image-lemma **where** $S = L$ **and** $T = A^*$ **and** $f = \text{map } (\varphi (\text{inv}_G g))$

and

```

      x = ((map (φ g) w') @ v)]
    apply (clarsimp simp add: group-action-def)
  by (smt (verit, best) A2-1 FuncSet.restrict-restrict H2-14 H2-15 H2-16 H2-4
      append-in-lists-conv inf.absorb-iff2 map-append map-map pre-image-lemma
restrict-apply'
      restrict-apply')

```

```

qed
show (map (φ g) w, map (φ g) w') ∈ ≡MN
  apply (clarsimp simp add: rel-MN-def Set.image-def)
  apply (intro conjI)
  using A1-1 A1-6 group-action.surj-prop group-action-axioms
    apply fastforce
  using A1-1 A1-7 image-iff surj-prop
    apply fastforce
  apply (clarify; rule iffI)
  subgoal for v
    apply (rule H1-0[where v1 = v and w1 = w and w'1 = w'])
    using A1-0 A1-1 A1-2 A1-3 A1-4 A1-5 A1-6 A1-7
    by (auto simp add: rel-MN-def Set.image-def)
  apply (rule H1-0[where w1 = w' and w'1 = w])
  using A1-0 A1-1 A1-2 A1-3 A1-4 A1-5 A1-6 A1-7
  by (auto simp add: rel-MN-def Set.image-def)

```

```

qed
show ?thesis
  using G-lang-axioms A-0
  apply (clarsimp simp add: G-lang-def G-lang-axioms-def eq-var-subset-def
      eq-var-subset-axioms-def alt-grp-act-def group-action-def)
  apply (intro conjI; clarify)
  apply (rule conjI; rule impI)
  apply (simp add: H-0)
  by (auto simp add: rel-MN-def)

```

qed

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

```

lemma MN-rel-eq-var:
  eq-var-rel G (A*) (φ*) ≡MN
  apply (clarsimp simp add: eq-var-rel-def alt-grp-act-def eq-var-rel-axioms-def)
  apply (intro conjI)
  apply (metis L-is-equivar alt-grp-act.axioms eq-var-subset.axioms(1) induced-star-map-def)
  using L-is-equivar
  apply (simp add: rel-MN-def eq-var-subset-def eq-var-subset-axioms-def)
  apply fastforce
  apply (clarify)
  apply (intro conjI; rule impI; rule conjI; rule impI)
    apply (simp add: in-lists-conv-set)
    apply (clarsimp simp add: rel-MN-def)
    apply (intro conjI)
    apply (clarsimp simp add: rel-MN-def)
  subgoal for w v g w'

```

```

using L-is-equivar
apply (clarsimp simp add: restrict-def eq-var-subset-def eq-var-subset-axioms-def)
by (meson element-image)
apply(metis image-mono in-listsI in-mono list.set-map lists-mono subset-code(1))
surj-prop)
apply (clarify; rule iffI)
subgoal for w v g u
using G-lang-axioms MN-quot-act-wd[where w = w and w' = v]
by (auto simp add: rel-MN-def G-lang-def G-lang-axioms-def
eq-var-subset-def eq-var-subset-axioms-def Set.subset-eq element-image)
subgoal for w v g u
using G-lang-axioms MN-quot-act-wd[where w = w and w' = v]
by (auto simp add: rel-MN-def G-lang-def G-lang-axioms-def
eq-var-subset-def eq-var-subset-axioms-def Set.subset-eq element-image)
using G-lang-axioms MN-quot-act-wd
by (auto simp add: rel-MN-def G-lang-def G-lang-axioms-def
eq-var-subset-def eq-var-subset-axioms-def Set.subset-eq element-image)

```

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

```

 $w \in A^* \implies g \in \text{carrier } G \implies g \odot_{[\varphi^*]_{\equiv_{MN}}} A^* ([w]_{MN}) = ([g \odot_{\varphi^*} w]_{MN})$ 
using eq-var-rel.quot-act-wd[where G = G and  $\varphi = \varphi^*$  and  $X = A^*$  and  $R =$ 
 $\equiv_{MN}$  and x = w
and g = g]
by (simp del: GMN-simps add: alt-natural-map-MN-def MN-rel-eq-var MN-rel-equival)

```

lemma *MN-trans-func-characterization:*

```

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

```

proof –

```

assume
  A-0: v ∈ (A*) and
  A-1: a ∈ A
have H-0:  $\bigwedge u. u \in [v]_{MN} \implies (u @ [a]) \in [v @ [a]]_{MN}$ 
by (auto simp add: rel-MN-def A-1 A-0)
hence H-1: (SOME w. (v, w) ∈  $\equiv_{MN}$ ) ∈ [v]_{MN}  $\implies ((SOME w. (v, w) \in \equiv_{MN})$ 
 $@ [a]) \in [v @ [a]]_{MN}$ 
by auto
from A-0 have  $(v, v) \in \equiv_{MN} \wedge v \in [v]_{MN}$ 
by (auto simp add: rel-MN-def)
hence H-2: (SOME w. (v, w) ∈  $\equiv_{MN}$ ) ∈ [v]_{MN}
apply (clarsimp simp add: rel-MN-def)
apply (rule conjI)
apply (smt (verit, ccfv-SIG) A-0 in-listsD verit-sko-ex-indirect)
by (smt (verit, del-insts) A-0 in-listsI tfl-some)
hence H-3: ((SOME w. (v, w) ∈  $\equiv_{MN}$ ) @ [a]) ∈ [v @ [a]]_{MN}
using H-1
by simp
thus  $\delta_{MN} [v]_{MN} a = [v @ [a]]_{MN}$ 
using A-0 A-1 MN-rel-equival
apply (clarsimp simp add: equiv-def)

```

```

apply (rule conjI; rule impI)
apply (metis MN-rel-equival equiv-class-eq)
by (simp add: A-0 quotientI)
qed

lemma MN-trans-eq-var-func :
  eq-var-func G
  (MN-equiv × A) (λg∈carrier G. λ(W, a) ∈ (MN-equiv × A). ((([φ*])≡MN A*) g
  W, φ g a))
  MN-equiv ((([φ*])≡MN A*)
  (λ(w, a) ∈ MN-equiv × A. δMN w a))
proof –
  have H-0: alt-grp-act G MN-equiv ([φ*]≡MN A*)
    using MN-rel-eq-var MN-rel-equival eq-var-rel.quot-act-is-grp-act
    alt-group-act-is-grp-act restrict-apply
  by fastforce
  have H-1: ∧ a b g.
    a ∈ MN-equiv ⇒
    b ∈ A ⇒
    ((([φ*]≡MN A*) g a ∈ MN-equiv ∧ φ g b ∈ A →
    g ∈ carrier G → δMN ((([φ*]≡MN A*) g a) (φ g b) = ([φ*]≡MN A*) g (δMN
  a b)) ∧
    ((([φ*]≡MN A*) g a ∈ MN-equiv → φ g b ∉ A) →
    g ∈ carrier G → undefined = ([φ*]≡MN A*) g (δMN a b))
  proof –
  fix C a g
  assume
    A1-0: C ∈ MN-equiv and
    A1-1: a ∈ A
  have H1-0: g ∈ carrier G ⇒ φ g a ∈ A
    by (meson A1-1 element-image)
  from A1-0 obtain c where H1-c: [c]MN = C ∧ c ∈ A*
    by (auto simp add: quotient-def)
  have H1-1: g ∈ carrier G ⇒ δMN ((([φ*]≡MN A*) g C) (φ g a) = ([φ*]≡MN
  A*) g (δMN [c]MN a))
  proof –
  assume
    A2-0: g ∈ carrier G
  have H2-0: φ g a ∈ A
    using H1-0 A2-0
  by simp
  have H2-1: (φ*) g ∈ Bij (A*) using G-lang-axioms lists-a-Gset A2-0
    apply (clarsimp simp add: G-lang-def G-lang-axioms-def group-action-def
    group-hom-def hom-def group-hom-axioms-def BijGroup-def image-def)
  by (meson Pi-iff restrict-Pi-cancel)
  hence H2-2: (φ*) g c ∈ (A*)
    using H1-c
  apply (clarsimp simp add: Bij-def bij-betw-def inj-on-def Image-def im-
  age-def)

```

apply (*rule conjI; rule impI; clarify*)
using *surj-prop*
apply *fastforce*
using *A2-0*
by *blast*
from *H1-c* **have** *H2-1*: $([\varphi^*]_{\equiv MN A^*}) g (\equiv_{MN} \{c\}) = ([\varphi^*]_{\equiv MN A^*}) g C$
by *auto*
also have *H2-2*: $([\varphi^*]_{\equiv MN A^*}) g C = [(\varphi^*) g c]_{MN}$
using *eq-var-rel.quot-act-wd* [**where** $R = \equiv_{MN}$ **and** $G = G$ **and** $X = A^*$]
and $\varphi = \varphi^*$ **and** $g = g$
and $x = c]$
by (*clarsimp simp del: GMN-simps simp add: alt-natural-map-MN-def*
make-op-def MN-rel-eq-var
MN-rel-equival H1-c A2-0 H2-1)
hence *H2-3*: $\delta_{MN} (([\varphi^*]_{\equiv MN A^*}) g C) (\varphi g a) = \delta_{MN} ([(\varphi^*) g c]_{MN}) (\varphi g a)$
using *H2-2*
by *simp*
also have *H2-4*: $\dots = [((\varphi^*) g c) @ [(\varphi g a)]]_{MN}$
using *MN-trans-func-characterization* [**where** $v = (\varphi^*) g c$ **and** $a = \varphi g a]$
H1-c A2-0
G-set-equiv H2-0 eq-var-subset.is-equivar insert-iff lists-a-Gset
by *blast*
also have *H2-5*: $\dots = [(\varphi^*) g (c @ [a])]_{MN}$
using *A2-0 H1-c A1-1*
by *auto*
also have *H2-6*: $\dots = ([\varphi^*]_{\equiv MN A^*}) g [(c @ [a])]_{MN}$
apply (*rule meta-mp[of c @ [a] \in A^*]*)
using *eq-var-rel.quot-act-wd* [**where** $R = \equiv_{MN}$ **and** $G = G$ **and** $X = A^*$]
and $\varphi = \varphi^*$ **and** $g = g$
and $x = c @ [a]$
apply (*clarsimp simp del: GMN-simps simp add: make-op-def MN-rel-eq-var*
MN-rel-equival H1-c
A2-0 H2-1)
using *H1-c A1-1*
by *auto*
also have *H2-7*: $\dots = ([\varphi^*]_{\equiv MN A^*}) g (\delta_{MN} [c]_{MN} a)$
using *MN-trans-func-characterization* [**where** $v = c$ **and** $a = a]$ *H1-c A1-1*
by *metis*
finally show $\delta_{MN} (([\varphi^*]_{\equiv MN A^*}) g C) (\varphi g a) = ([\varphi^*]_{\equiv MN A^*}) g (\delta_{MN} [c]_{MN} a)$
using *H2-1*
by *metis*
qed
show $(([\varphi^*]_{\equiv MN A^*}) g C \in MN\text{-equiv} \wedge \varphi g a \in A \longrightarrow$
 $g \in \text{carrier } G \longrightarrow$
 $\delta_{MN} (([\varphi^*]_{\equiv MN A^*}) g C) (\varphi g a) =$
 $([\varphi^*]_{\equiv MN A^*}) g (\delta_{MN} C a) \wedge$
 $(([\varphi^*]_{\equiv MN A^*}) g C \in MN\text{-equiv} \longrightarrow \varphi g a \notin A) \longrightarrow$

```

g ∈ carrier G → undefined = ([φ*]≡MN A*) g (δMN C a)
  apply (rule conjI; clarify)
  using H1-1 H1-c
  apply blast
  by (metis A1-0 H1-0 H-0 alt-group-act-is-grp-act
      group-action.element-image)
qed
show ?thesis
  apply (subst eq-var-func-def)
  apply (subst eq-var-func-axioms-def)
  apply (rule conjI)
  subgoal
    apply (rule prod-group-act[where G = G and A = MN-equiv and φ =
[[φ*]≡MN A*
      and B = A and ψ = φ]])
    apply (rule H-0)
    using G-lang-axioms
    by (auto simp add: G-lang-def G-lang-axioms-def)
  apply (rule conjI)
  subgoal
    using MN-rel-eq-var MN-rel-equival eq-var-rel.quot-act-is-grp-act
    using alt-group-act-is-grp-act restrict-apply
    by fastforce
  apply (rule conjI)
  subgoal
    apply (subst extensional-funcset-def)
    apply (subst restrict-def)
    apply (subst Pi-def)
    apply (subst extensional-def)
    apply (clarsimp)
    by (metis MN-rel-equival append-in-lists-conv equiv-Eps-preserves lists.Cons
lists.Nil
      quotientI)
  apply (subst restrict-def)
  apply (clarsimp simp del: GMN-simps simp add: make-op-def)
  by (simp add: H-1 del: GMN-simps)
qed

lemma MN-quot-act-on-empty-str:
  ∧g. [g ∈ carrier G; ([], x) ∈ ≡MN] ⇒ x ∈ map (φ g) ‘≡MN’ “{[]}”
proof -
  fix g
  assume
    A-0: g ∈ carrier G and
    A-1: ([], x) ∈ ≡MN
  from A-1 have H-0: x ∈ (A*)
  by (auto simp add: rel-MN-def)
  from A-0 H-0 have H-1: x = (φ*) g ((φ*) (invG g) x)
  by (smt (verit) alt-grp-act-def group-action.bij-prop1 group-action.orbit-sym-aux

```

```

lists-a-Gset)
have H-2: inv G g ∈ carrier G
  using A-0 MN-rel-eq-var
by (auto simp add: eq-var-rel-def eq-var-rel-axioms-def group-action-def group-hom-def)
have H-3: ([], (φ*) (inv G g) x) ∈ ≡MN
  using A-0 A-1 H-0 MN-rel-eq-var
  apply (clarsimp simp add: eq-var-rel-def eq-var-rel-axioms-def)
  apply (rule conjI; clarify)
  apply (smt (verit, best) H-0 list.simps(8) lists.Nil)
  using H-2
by simp
hence H-4: ∃ y ∈ ≡MN “{[]}. x = map (φ g) y
  using A-0 H-0 H-1 H-2
  applyclarsimp
  by (metis H-0 Image-singleton-iff insert-iff insert-image lists-image surj-prop)
thus x ∈ map (φ g) ‘≡MN “{[]}
  by (auto simp add: image-def)
qed

```

```

lemma MN-init-state-equivar:
  eq-var-subset G (A*) (φ*) MN-init-state
  apply (rule alt-grp-act.eq-var-one-direction)
  using lists-a-Gset
  apply (auto)[1]
  apply (clarsimp)
  subgoal for w a
    by (auto simp add: rel-MN-def)
  apply (simp add: Set.subset-eq; clarify)
  apply (clarsimp simp add: image-def Image-def Int-def)
  apply (erule disjE)
  subgoal for g w
    using MN-rel-eq-var
    apply (clarsimp simp add: eq-var-rel-def eq-var-rel-axioms-def)
    by (metis (full-types, opaque-lifting) in-listsI list.simps(8) lists.Nil)
  by (auto simp add: ⟨∧ a w. [([], w) ∈ ≡MN; a ∈ set w] ⟹ a ∈ A⟩)

```

```

lemma MN-init-state-equivar-v2:
  eq-var-subset G (MN-equiv) ([φ*]≡MN A* ) {MN-init-state}
proof -
  have H-0: ∀ g ∈ carrier G. (φ*) g ‘MN-init-state = MN-init-state ⟹
    ∀ g ∈ carrier G. ([φ*]≡MN A*) g MN-init-state = MN-init-state
  proof (clarify)
    fix g
    assume
      A-0: g ∈ carrier G
    have H-0: ∧ x. [x]MN = ≡MN “{x}
      by simp
    have H-1: ([φ*]≡MN A*) g [ ]MN = [(φ*) g [ ]]MN
      using eq-var-rel.quot-act-wd[where R = ≡MN and G = G and X = A*]

```

and $\varphi = \varphi^*$ **and** $g = g$
and $x = []$ *MN-rel-eq-var MN-rel-equival*
by (*clarsimp simp del: GMN-simps simp add: H-0 make-op-def A-0*)
from *A-0 H-1* **show** $([\varphi^*]_{\equiv MN A^*}) g []_{MN} = []_{MN}$
by *auto*
qed
show *?thesis*
using *MN-init-state-equivar*
apply (*clarsimp simp add: eq-var-subset-def simp del: GMN-simps*)
apply (*rule conjI*)
subgoal
by (*metis MN-rel-eq-var MN-rel-equival eq-var-rel.quot-act-is-grp-act*)
apply (*clarsimp del: subset-antisym simp del: GMN-simps simp add: eq-var-subset-axioms-def*)
apply (*rule conjI*)
apply (*auto simp add: quotient-def*)[1]
by (*simp add: H-0 del: GMN-simps*)
qed

lemma *MN-final-state-equiv:*
eq-var-subset G (MN-equiv) ([φ^] $_{\equiv MN A^*}$) MN-fin-states*
proof–
have *H-0*: $\bigwedge g x w. g \in \text{carrier } G \implies w \in L \implies \exists wa \in L. ([\varphi^*]_{\equiv MN A^*}) g [w]_{MN} = [wa]_{MN}$
proof–
fix $g w$
assume
A1-0: $g \in \text{carrier } G$ **and**
A1-1: $w \in L$
have *H1-0*: $\bigwedge v. v \in L \implies (\varphi^*) g v \in L$
using *A1-0 G-lang-axioms*
apply (*clarsimp simp add: G-lang-def G-lang-axioms-def eq-var-subset-def eq-var-subset-axioms-def*)
by *blast*
hence *H1-1*: $(\varphi^*) g w \in L$
using *A1-1*
by *simp*
from *A1-1* **have** *H1-2*: $\bigwedge v. v \in [w]_{MN} \implies v \in L$
apply (*clarsimp simp add: rel-MN-def*)
by (*metis lists.simps.self-append-conv*)
have *H1-3*: $([\varphi^*]_{\equiv MN A^*}) g [w]_{MN} = [(\varphi^*) g w]_{MN}$
using *eq-var-rel.quot-act-wd* [**where** $R = \equiv_{MN}$ **and** $G = G$ **and** $X = A^*$]
and $\varphi = \varphi^*$ **and** $g = g$
and $x = w$ *MN-rel-eq-var MN-rel-equival G-lang-axioms*
by (*clarsimp simp add: A1-0 A1-1 G-lang-axioms-def G-lang-def eq-var-subset-def eq-var-subset-axioms-def subset-eq*)
show $\exists wa \in L. ([\varphi^*]_{\equiv MN A^*}) g [w]_{MN} = [wa]_{MN}$
using *H1-1 H1-3*
by *blast*
qed


```

show ?thesis
  apply (rule alt-grp-act.eq-var-one-direction)
  using MN-init-state-equivar-v2 eq-var-subset.axioms(1)
  apply blast
  apply (clarsimp)
  subgoal for w
    using G-lang-axioms
    by (auto simp add: quotient-def G-lang-axioms-def G-lang-def eq-var-subset-def
      eq-var-subset-axioms-def)
    apply (simp add: Set.subset-eq del: GMN-simps; clarify)
    by (simp add: H-0 del: GMN-simps)
qed

interpretation syntac-aut :
  det-aut A MN-equiv MN-init-state MN-fin-states MN-trans-func
proof –
  have H-0:  $\bigwedge \text{state label. state} \in \text{MN-equiv} \implies \text{label} \in A \implies \delta_{MN} \text{ state label} \in \text{MN-equiv}$ 
proof –
  fix state label
  assume
    A-0: state  $\in$  MN-equiv and
    A-1: label  $\in$  A
  obtain w where H-w: state =  $[w]_{MN} \wedge w \in A^*$ 
    by (metis A-0 alt-natural-map-MN-def quotientE)
  have H-0:  $\delta_{MN} [w]_{MN} \text{ label} = [w @ [\text{label}]]_{MN}$ 
    using MN-trans-func-characterization[where v = w and a = label] H-w A-1
    by simp
  have H-1:  $\bigwedge v. v \in A^* \implies [v]_{MN} \in \text{MN-equiv}$ 
    by (simp add: in-listsI quotientI)
  show  $\delta_{MN} \text{ state label} \in \text{MN-equiv}$ 
    using H-w H-0 H-1
    by (simp add: A-1)
qed
show det-aut A MN-equiv MN-init-state MN-fin-states  $\delta_{MN}$ 
apply (clarsimp simp del: GMN-simps simp add: det-aut-def alt-natural-map-MN-def)
apply (intro conjI)
  apply (auto simp add: quotient-def)[1]
  using G-lang-axioms
  apply (auto simp add: quotient-def G-lang-axioms-def G-lang-def
    eq-var-subset-def eq-var-subset-axioms-def)[1]
apply (auto simp add: extensional-def PiE-iff simp del: MN-trans-func-def)[1]
  apply (simp add: H-0 del: GMN-simps)
  by auto
qed

corollary syth-aut-is-det-aut:
  det-aut A MN-equiv MN-init-state MN-fin-states  $\delta_{MN}$ 
  using local.syntac-aut.det-aut-axioms

```

by *simp*

lemma *give-input-transition-func*:

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

proof –

assume

A-0: $w \in A^*$

have *H-0*: $\bigwedge a w v. [a \in A; w \in A^*; \forall v \in A^*. [v @ w]_{MN} = (\delta_{MN}^*) [v]_{MN} w; v \in A^*] \implies$

$[v @ a \# w]_{MN} = (\delta_{MN}^*) [v]_{MN} (a \# w)$

proof –

fix $a w v$

assume

A1-IH: $\forall v \in A^*. [v @ w]_{MN} = (\delta_{MN}^*) [v]_{MN} w$ **and**

A1-0: $a \in A$ **and**

A1-1: $v \in A^*$ **and**

A1-2: $w \in A^*$

from *A1-IH A1-1 A1-2* **have** *H1-1*: $[v @ w]_{MN} = (\delta_{MN}^*) [v]_{MN} w$

by *auto*

have *H1-2*: $[(v @ [a]) @ w]_{MN} = (\delta_{MN}^*) [v @ [a]]_{MN} w$

apply (*rule meta-mp*[*of* ($(v @ [a]) \in (A^*)$)])

using *A1-IH A1-2 H1-1*

apply *blast*

using *A1-0 A1-1*

by *auto*

have *H1-3*: $\delta_{MN} [v]_{MN} a = [v @ [a]]_{MN}$

using *MN-trans-func-characterization*[**where** $a = a$] *A1-0 A1-1*

by *auto*

hence *H1-4*: $[v @ a \# w]_{MN} = (\delta_{MN}^*) [v @ [a]]_{MN} w$

using *H1-2*

by *auto*

also have *H1-5*: $\dots = (\delta_{MN}^*) (\delta_{MN} [v]_{MN} a) w$

using *H1-4 H1-3 A1-1*

by *auto*

thus $[v @ a \# w]_{MN} = (\delta_{MN}^*) [v]_{MN} (a \# w)$

using *calculation*

by *auto*

qed

from *A-0* **show** *?thesis*

apply (*induction* w)

apply (*auto*)[1]

by (*simp add: H-0 del: GMN-simps*)

qed

lemma *MN-unique-init-state*:

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

using *give-input-transition-func*[**where** $w = w$]

by (*metis append-self-conv2 lists.Nil*)

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

$w \in A^* \implies [w]_{MN} \in MN\text{-fin-states} \implies w \in L$

proof –

assume

A-0: $w \in A^*$ **and**

A-1: $[w]_{MN} \in MN\text{-fin-states}$

from *A-1* **have** *H-0*: $\exists w' \in [w]_{MN}. w' \in L$

apply (*clarsimp*)

by (*metis A-0 MN-rel-equiv equiv-class-self proj-def proj-in-iff*)

from *H-0* **obtain** *w'* **where** *H-w'*: $w' \in [w]_{MN} \wedge w' \in L$

by *auto*

have *H-1*: $\bigwedge v. v \in A^* \implies w' @ v \in L \implies w @ v \in L$

using *H-w' A-1 A-0*

by (*auto simp add: rel-MN-def*)

show $w \in L$

using *H-1 H-w'*

apply *clarify*

by (*metis append-Nil2 lists.Nil*)

qed

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 ($[\varphi^]_{\equiv MN} A^*$)*

apply (*clarsimp simp add: det-G-aut-def simp del: GMN-simps*)

apply (*intro conjI*)

using *syth-aut-is-det-aut*

apply (*auto*)[1]

using *alt-grp-act-axioms*

apply (*auto*)[1]

using *MN-init-state-equivar-v2 eq-var-subset.axioms(1)*

apply *blast*

using *MN-final-state-equiv*

apply *presburger*

using *MN-init-state-equivar-v2*

apply *presburger*

using *MN-trans-eq-var-func*

by *linarith*

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*

apply (*clarsimp simp add: det-G-aut-rec-lang-def det-aut-rec-lang-axioms-def det-aut-rec-lang-def simp del: GMN-simps*)

apply (*intro conjI*)

using *syntactic-aut-det-G-aut syth-aut-is-det-aut*

apply (*auto*)[1]

using *syntactic-aut-det-G-aut syth-aut-is-det-aut*

```

apply (auto)[1]
apply (rule allI; rule iffI)
apply (rule conjI)
using L-is-equivar eq-var-subset.is-subset image-iff image-mono insert-image in-
sert-subset
  apply blast
using MN-unique-init-state L-is-equivar eq-var-subset.is-subset
  apply blast
using MN-unique-init-state fin-states-rep-by-lang in-lists-conv-set
by (smt (verit) mem-Collect-eq)

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  $\varphi$ 
  ( $[\varphi^*]_{\equiv MN A^*}$ ) L
  apply (clarsimp simp del: GMN-simps simp add: reach-det-G-aut-rec-lang-def
  det-G-aut-rec-lang-def det-aut-rec-lang-axioms-def reach-det-G-aut-def
  reach-det-aut-def reach-det-aut-axioms-def det-G-aut-def det-aut-rec-lang-def)
  apply (intro conjI)
  using syth-aut-is-det-aut
    apply blast
  using alt-grp-act-axioms
    apply (auto)[1]
  subgoal
    using MN-init-state-equivar-v2 eq-var-subset.axioms(1)
    by blast
  using MN-final-state-equiv
    apply presburger
  using MN-init-state-equivar-v2
  subgoal
    by presburger
  using MN-trans-eq-var-func
    apply linarith
  using syth-aut-is-det-aut
    apply (auto)[1]
    apply (metis (mono-tags, lifting) G-lang.MN-unique-init-state G-lang-axioms
    det-G-aut-rec-lang-def det-aut-rec-lang.is-recognised syntactic-aut-det-G-aut-rec-L)
  using syth-aut-is-det-aut
    apply (auto)[1]
  using alt-grp-act-axioms
    apply (auto)[1]
  using  $\langle$ alt-grp-act G MN-equiv ( $[\varphi^*]_{\equiv MN A^*}$ ) $\rangle$ 
    apply blast
  using  $\langle$ eq-var-subset G MN-equiv ( $[\varphi^*]_{\equiv MN A^*}$ ) MN-fin-states $\rangle$ 
    apply blast
  using  $\langle$ eq-var-subset G MN-equiv ( $[\varphi^*]_{\equiv MN A^*}$ ) {MN-init-state} $\rangle$ 
    apply blast
  using MN-trans-eq-var-func
    apply blast

```

```

using syth-aut-is-det-aut
apply auto[1]
by (metis MN-unique-init-state alt-natural-map-MN-def quotientE)
end

```

1.5 Proving the Myhill-Nerode Theorem for G -Automata

```

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

```

```

context reach-det-G-aut-rec-lang begin
adhoc-overloading
star 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:

```

 $s \in S \implies \exists w. w \in A^* \wedge ((\delta^*) i w = s)$ 
using reach-det-G-aut-rec-lang-axioms is-reachable
by auto

```

lemma induced-epi-wd2:

```

 $w \in A^* \implies w' \in A^* \implies (\delta^*) i w = (\delta^*) i w' \implies [w]_{MN} = [w']_{MN}$ 

```

proof –

assume

```

A-0:  $w \in A^*$  and
A-1:  $w' \in A^*$  and
A-2:  $(\delta^*) i w = (\delta^*) i w'$ 

```

```

have H-0:  $\bigwedge v. v \in A^* \implies w @ v \in L \iff w' @ v \in L$ 

```

apply clarify

```

by (smt (z3) A-0 A-1 A-2 append-in-lists-conv is-aut.eq-pres-under-concat
is-aut.init-state-is-a-state is-lang is-recognised subsetD)+

```

```

show  $[w]_{MN} = [w']_{MN}$ 

```

```

apply (simp add: rel-MN-def)

```

```

using H-0 A-0 A-1

```

```

by auto

```

qed

lemma *states-to-words-on-final*:

states-to-words $\in (F \rightarrow L)$

proof –

have *H-0*: $\bigwedge x. x \in F \implies x \in S \implies (\text{SOME } w. w \in A^* \wedge (\delta^*) i w = x) \in L$

proof –

fix *s*

assume

A1-0: $s \in F$

have *H1-0*: $\exists w. w \in A^* \wedge (\delta^*) i w = s$

using *A1-0 is-reachable*

by (*metis is-aut.fin-states-are-states subsetD*)

have *H1-1*: $\bigwedge w. w \in A^* \wedge (\delta^*) i w = s \implies w \in L$

using *A1-0 is-recognised*

by *auto*

show $(\text{SOME } w. w \in A^* \wedge (\delta^*) i w = s) \in L$

by (*metis (mono-tags, lifting) H1-0 H1-1 someI-ex*)

qed

show *?thesis*

apply (*clarsimp simp add: states-to-words-def*)

apply (*rule conjI; rule impI*)

apply (*simp add: H-0*)

using *is-aut.fin-states-are-states*

by *blast*

qed

lemma *induced-epi-eq-var*:

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

proof –

have *H-0*: $\bigwedge s g. [s \in S; g \in \text{carrier } G; \psi g s \in S] \implies$

words-to-syth-states (states-to-words ($\psi g s$)) =

$([(\varphi^*)]_{\equiv MN} A^*) g (\text{words-to-syth-states (states-to-words } s))$

proof –

fix *s g*

assume

A1-0: $s \in S$ **and**

A1-1: $g \in \text{carrier } G$ **and**

A1-2: $\psi g s \in S$

have *H1-0*: $([(\varphi^*)]_{\equiv MN} A^*) g (\text{words-to-syth-states (states-to-words } s)) =$

$[(\varphi^*) g (\text{SOME } w. w \in A^* \wedge (\delta^*) i w = s)]_{MN}$

apply (*clarsimp simp del: GMN-simps simp add: words-to-syth-states-def states-to-words-def A1-0*)

apply (*rule meta-mp[of (SOME w. w \in A^* \wedge (\delta^*) i w = s) \in A^*]*)

using *quot-act-wd-alt-notation[where w = (SOME w. w \in A^* \wedge (\delta^*) i w =*

s) and g = g] A1-1

apply *simp*

using *A1-0*

by (*metis (mono-tags, lifting) induced-epi-wd1 some-eq-imp*)

```

have H1-1:  $\bigwedge g s' w'. \llbracket s' \in S; w' \in A^*; g \in \text{carrier } G; (\varphi^*) g w' \in A^* \wedge \psi g s' \in S \rrbracket$ 
   $\implies (\delta^*) (\psi g s') ((\varphi^*) g w') = \psi g ((\delta^*) s' w')$ 
using give-input-eq-var
apply (clarsimp simp del: GMN-simps simp add: eq-var-func-axioms-def
eq-var-func-def
make-op-def)
by (meson in-listsI)
have H1-2:  $\{w. w \in A^* \wedge (\delta^*) i w = \psi g s\} =$ 
 $\{w'. \exists w \in A^*. (\varphi^*) g w = w' \wedge (\delta^*) i w = s\}$ 
proof (rule subset-antisym; clarify)
fix w'
assume
  A2-0:  $(\delta^*) i w' = \psi g s$  and
  A2-1:  $\forall x \in \text{set } w'. x \in A$ 
have H2-0:  $(\text{inv } G g) \in \text{carrier } G$ 
by (meson A1-1 group.inv-closed group-hom.axioms(1) states-a-G-set.group-hom)
have H2-1:  $(\varphi^*) g ((\varphi^*) (\text{inv } G g) w') = w'$ 
by (smt (verit) A1-1 A2-1 alt-group-act-is-grp-act group-action.bij-prop1
group-action.orbit-sym-aux in-listsI labels-a-G-set.lists-a-Gset)
have H2-2:  $\bigwedge g w. g \in \text{carrier } G \implies w \in A^* \implies (\delta^*) i ((\varphi^*) g w) = (\delta^*)$ 
 $(\psi g i) ((\varphi^*) g w)$ 
using init-is-eq-var.eq-var-subset-axioms init-is-eq-var.is-equivar
by auto
have H2-3:  $\bigwedge g w. g \in \text{carrier } G \implies w \in A^* \implies (\delta^*) (\psi g i) ((\varphi^*) g w) =$ 
 $\psi g ((\delta^*) i w)$ 
apply (rule H1-1[where s'1 = i])
apply (simp add: A2-1 in-lists-conv-set H2-0 is-aut.init-state-is-a-state)+
using is-aut.init-state-is-a-state labels-a-G-set.element-image
states-a-G-set.element-image
by blast
have H2-4:  $\psi (\text{inv } G g) ((\delta^*) i w') = s$ 
using A2-0 H2-0
by (simp add: A1-0 A1-1 states-a-G-set.orbit-sym-aux)
have H2-5:  $(\delta^*) i ((\varphi^*) (\text{inv } G g) w') = s$ 
apply (rule meta-mp[of w'  $\in A^*$ ])
using H2-0 H2-1 H2-4 A2-1 H2-2 H2-3
apply presburger
using A2-1
by auto
have H2-6:  $(\varphi^*) (\text{inv } G g) w' \in A^*$ 
using H2-0 A2-1
by (metis alt-group-act-is-grp-act group-action.element-image in-listsI
labels-a-G-set.lists-a-Gset)
thus  $\exists w \in A^*. (\varphi^*) g w = w' \wedge (\delta^*) i w = s$ 
using H2-1 H2-5 H2-6
by blast
next
fix x w

```

```

assume
  A2-0:  $\forall x \in \text{set } w. x \in A$  and
  A2-1:  $s = (\delta^*) i w$ 
show  $(\varphi^*) g w \in A^* \wedge (\delta^*) i ((\varphi^*) g w) = \psi g ((\delta^*) i w)$ 
apply (rule conjI)
apply (rule meta-mp[of (inv  $G$  g)  $\in$  carrier  $G$ ])
using alt-group-act-is-grp-act group-action.element-image in-listsI
  labels-a-G-set.lists-a-Gset
apply (metis A1-1 A2-0)
apply (meson A1-1 group.inv-closed group-hom.axioms(1) states-a-G-set.group-hom)
apply (rule meta-mp[of  $\psi g i = i$ ])
using H1-1[where  $s'1 = i$  and  $g1 = g$ ]
apply (metis A1-1 A2-0 action-on-input in-listsI)
using init-is-eq-var.eq-var-subset-axioms init-is-eq-var.is-equivar
by (simp add: A1-1)
qed
have H1-3:  $\exists w. w \in A^* \wedge (\delta^*) i w = s$ 
using A1-0 is-reachable
by auto
have H1-4:  $\exists w. w \in A^* \wedge (\delta^*) i w = \psi g s$ 
using A1-2 induced-epi-wd1
by auto
have H1-5:  $[(\varphi^*) g (\text{SOME } w. w \in A^* \wedge (\delta^*) i w = s)]_{MN} = [\text{SOME } w. w \in$ 
 $A^* \wedge (\delta^*) i w = \psi g s]_{MN}$ 
proof (rule subset-antisym; clarify)
fix  $w'$ 
assume
  A2-0:  $w' \in [(\varphi^*) g (\text{SOME } w. w \in A^* \wedge (\delta^*) i w = s)]_{MN}$ 
have H2-0:  $\bigwedge w. w \in A^* \wedge (\delta^*) i w = s \implies w' \in [(\varphi^*) g w]_{MN}$ 
using A2-0 H1-3 H1-2 H1-4 induced-epi-wd2 mem-Collect-eq tfl-some
by (smt (verit, best))
obtain  $w''$  where H2-w'':  $w' \in [(\varphi^*) g w'']_{MN} \wedge w'' \in A^* \wedge (\delta^*) i w'' = s$ 
using A2-0 H1-3 tfl-some
by (metis (mono-tags, lifting))
from H1-2 H2-w'' have H2-1:  $(\delta^*) i ((\varphi^*) g w'') = \psi g s$ 
by blast
have H2-2:  $\bigwedge w. w \in A^* \implies (\delta^*) i w = \psi g s \implies w' \in [w]_{MN}$ 
proof –
fix  $w''$ 
assume
  A3-0:  $w'' \in A^*$  and
  A3-1:  $(\delta^*) i w'' = \psi g s$ 
have H3-0:  $(\text{inv } G g) \in \text{carrier } G$ 
by (metis A1-1 group.inv-closed group-hom.axioms(1) states-a-G-set.group-hom)
from A3-0 H3-0 have H3-1:  $(\varphi^*) (\text{inv } G g) w'' \in A^*$ 
by (metis alt-grp-act.axioms group-action.element-image
  labels-a-G-set.lists-a-Gset)
have H3-2:  $\bigwedge g w. g \in \text{carrier } G \implies w \in A^* \implies (\delta^*) i ((\varphi^*) g w) = (\delta^*)$ 
 $(\psi g i) ((\varphi^*) g w)$ 

```



```

    using init-is-eq-var.eq-var-subset-axioms init-is-eq-var.is-equivar
    by auto
    have H3-3:  $\bigwedge g w. g \in \text{carrier } G \implies w \in A^* \implies (\delta^*) (\psi g i) ((\varphi^*) g w)$ 
=  $\psi g ((\delta^*) i w)$ 
    apply (rule H1-1[where s'1 = i])
    apply (simp add: A3-1 in-lists-conv-set H2-1 is-aut.init-state-is-a-state)+
    using is-aut.init-state-is-a-state labels-a-G-set.element-image
    states-a-G-set.element-image
    by blast
    have H3-4:  $s = (\delta^*) i ((\varphi^*) (\text{inv } G g) w')$ 
    using A3-0 A3-1 H3-0 H3-2 H3-3 A1-0 A1-1 states-a-G-set.orbit-sym-aux
    by auto
    from H3-4 show  $w' \in [w']_{MN}$ 
    by (metis (mono-tags, lifting) A1-1 G-set-equiv H2-1 H2-w''  $(\delta^*) i w'' =$ 
 $\psi g s$  A3-0
    eq-var-subset.is-equivar image-eqI induced-epi-wd2
    labels-a-G-set.lists-a-Gset)
qed
from H2-2 show  $w' \in [SOME w. w \in A^* \wedge (\delta^*) i w = \psi g s]_{MN}$ 
    by (smt (verit) H1-4 some-eq-ex)
next
fix  $w'$ 
assume
    A2-0:  $w' \in [SOME w. w \in A^* \wedge (\delta^*) i w = \psi g s]_{MN}$ 
obtain  $w''$  where H2-w'':  $w' \in [(\varphi^*) g w'']_{MN} \wedge w'' \in A^* \wedge (\delta^*) i w'' = s$ 
    using A2-0 H1-3 tfl-some
    by (smt (verit) H1-2 mem-Collect-eq)
from H1-2 H2-w'' have H2-0:  $(\delta^*) i ((\varphi^*) g w'') = \psi g s$ 
    by blast
have H2-1:  $\bigwedge w. w \in A^* \implies (\delta^*) i w = s \implies w' \in [(\varphi^*) g w]_{MN}$ 
proof –
    fix  $w''$ 
assume
    A3-0:  $w'' \in A^*$  and
    A3-1:  $(\delta^*) i w'' = s$ 
have H3-0:  $(\text{inv } G g) \in \text{carrier } G$ 
by (metis A1-1 group.inv-closed group-hom.axioms(1) states-a-G-set.group-hom)
have H3-1:  $(\varphi^*) (\text{inv } G g) w'' \in A^*$ 
    using A3-0 H3-0
by (metis alt-group-act-is-grp-act group-action.element-image labels-a-G-set.lists-a-Gset)
have H3-2:  $\bigwedge g w. g \in \text{carrier } G \implies w \in A^* \implies (\delta^*) i ((\varphi^*) g w) =$ 
 $(\delta^*) (\psi g i) ((\varphi^*) g w)$ 
    using init-is-eq-var.eq-var-subset-axioms init-is-eq-var.is-equivar
    by auto
have H3-3:  $\bigwedge g w. g \in \text{carrier } G \implies w \in A^* \implies (\delta^*) (\psi g i) ((\varphi^*) g w) =$ 
 $\psi g ((\delta^*) i w)$ 
    apply (rule H1-1[where s'1 = i])
    apply (simp add: A3-1 in-lists-conv-set H2-0 is-aut.init-state-is-a-state)+
    using is-aut.init-state-is-a-state labels-a-G-set.element-image

```

```

      states-a-G-set.element-image
    by blast
  have H3-4:  $\psi (inv_G g) s = (\delta^*) i ((\varphi^*) (inv_G g) w'')$ 
    using A3-0 A3-1 H3-0 H3-2 H3-3
    by auto
  show  $w' \in [(\varphi^*) g w']_{MN}$ 
    using H3-4 H3-1
    by (smt (verit, del-insts) A1-1 A3-0 A3-1 in-listsI H3-2 H3-3
      ⟨ $\wedge thesis. (\wedge w''. w' \in [(\varphi^*) g w']_{MN} \wedge w'' \in A^* \wedge$ 
       $(\delta^*) i w'' = s \implies thesis) \implies thesis$ ⟩
      alt-group-act-is-grp-act group-action.surj-prop image-eqI induced-epi-wd2
      labels-a-G-set.lists-a-Gset)
  qed
  show  $w' \in [(\varphi^*) g (SOME w. w \in A^* \wedge (\delta^*) i w = s)]_{MN}$ 
    using H2-1 H1-3
    by (metis (mono-tags, lifting) someI)
  qed
  show words-to-syth-states (states-to-words ( $\psi g s$ )) =
    ( $[(\varphi^*)]_{MN} A^*$ ) g (words-to-syth-states (states-to-words s))
    using H1-5
    apply (clarsimp simp del: GMN-simps simp add: words-to-syth-states-def
      states-to-words-def)
    apply (intro conjI; clarify; rule conjI)
    using H1-0
    apply (auto del: subset-antisym simp del: GMN-simps simp add: words-to-syth-states-def
      states-to-words-def)[1]
    using A1-2
    apply blast
    using A1-0
    apply blast
    using A1-0
    by blast
  qed
  show ?thesis
  apply (clarsimp del: subset-antisym simp del: GMN-simps simp add: eq-var-func-def
    eq-var-func-axioms-def)
  apply (intro conjI)
  subgoal
    using states-a-G-set.alt-grp-act-axioms
    by auto
    apply (metis MN-rel-eq-var MN-rel-equival eq-var-rel.quot-act-is-grp-act)
  apply (clarsimp simp add: FuncSet.extensional-funcset-def Pi-def)
  apply (rule conjI)
  apply (clarify)
  subgoal for s
    using is-reachable[where s = s]
    apply (clarsimp simp add: induced-epi-def compose-def states-to-words-def
      words-to-syth-states-def)
  by (smt (verit) ⟨ $s \in S \implies \exists input \in A^*. (\delta^*) i input = s$ ⟩ alt-natural-map-MN-def)

```

```

    lists-eq-set quotientI rel-MN-def singleton-conv someI)
  apply (clarsimp simp del: GMN-simps simp add: induced-epi-def make-op-def
    compose-def)
  apply (clarify)
  apply (clarsimp simp del: GMN-simps simp add: induced-epi-def compose-def
    make-op-def)
  apply (rule conjI; rule impI)
  apply (simp add: H-0)
  using states-a-G-set.element-image
  by blast
qed

```

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 $([(\varphi^)]_{\equiv MN}$
 A^*) induced-epi*

proof –

have *H-0:* $\bigwedge s. s \in \text{MN-equiv} \implies \exists \text{input} \in A^*. (\delta_{MN}^*) \text{MN-init-state input} = s$

proof –

fix *s*

assume

A-0: $s \in \text{MN-equiv}$

from *A-0* **have** *H-0:* $\exists w. w \in A^* \wedge s = [w]_{MN}$

by (*auto simp add: quotient-def*)

show $\exists \text{input} \in A^*. (\delta_{MN}^*) \text{MN-init-state input} = s$

using *H-0*

by (*metis MN-unique-init-state*)

qed

have *H-1:* $\bigwedge s_0 a. s_0 \in S \implies a \in A \implies \text{induced-epi} (\delta s_0 a) = \delta_{MN} (\text{induced-epi}$
 $s_0) a$

proof –

fix *s₀ a*

assume

A1-0: $s_0 \in S$ **and**

A1-1: $a \in A$

obtain *w* **where** *H1-w:* $w \in A^* \wedge (\delta^*) i w = s_0$

using *A1-0 induced-epi-wd1*

by *auto*

have *H1-0:* $[SOME w. w \in A^* \wedge (\delta^*) i w = s_0]_{MN} = [w]_{MN}$

by (*metis (mono-tags, lifting) H1-w induced-epi-wd2 some-eq-imp*)

have *H1-1:* $(\delta^*) i (SOME w. w \in A^* \wedge (\delta^*) i w = \delta s_0 a) = (\delta^*) i (w @ [a])$

using *A1-0 A1-1 H1-w is-aut.trans-to-charact* [**where** $s = s_0$ **and** $a = a$ **and**
 $w = w$]

by (*smt (verit, del-insts) induced-epi-wd1 is-aut.trans-func-well-def tft-some*)

have *H1-2:* $w @ [a] \in A^*$ **using** *H1-w A1-1* **by** *auto*

have *H1-3:* $[(SOME w. w \in A^* \wedge (\delta^*) i w = s_0) @ [a]]_{MN} = [w @ [a]]_{MN}$

by (*metis (mono-tags, lifting) A1-1 H1-0 H1-w MN-trans-func-characterization*
someI)

have *H1-4:* $\dots = [SOME w. w \in A^* \wedge (\delta^*) i w = \delta s_0 a]_{MN}$

```

apply (rule sym)
apply (rule induced-epi-wd2[where  $w = \text{SOME } w. w \in A^* \wedge (\delta^*) i w = \delta$ 
 $s_0 a$ 
and  $w' = w @ [a]$ ])
apply (metis (mono-tags, lifting) A1-0 A1-1 H1-w some-eq-imp H1-2
is-aut.trans-to-charact)
apply (rule H1-2)
using H1-1
by simp
show induced-epi ( $\delta s_0 a$ ) =  $\delta_{MN}$  (induced-epi  $s_0$ )  $a$ 
apply (clarsimp del: subset-antisym simp del: GMN-simps simp add: in-
duced-epi-def
words-to-syth-states-def states-to-words-def compose-def is-aut.trans-func-well-def)
using A1-1 H1-w H1-0 H1-3 H1-4 MN-trans-func-characterization A1-0
is-aut.trans-func-well-def
by presburger
qed
have H-2: induced-epi ‘  $S = MN\text{-equiv}$ 
proof–
have H1-0:  $\forall s \in S. \exists v \in A^*. (\delta^*) i v = s \wedge [\text{SOME } w. w \in A^* \wedge (\delta^*) i w =$ 
 $s]_{MN} = [v]_{MN}$ 
by (smt (verit) is-reachable tft-some)
have H1-1:  $\bigwedge v. v \in A^* \implies (\delta^*) i v \in S$ 
using is-aut.give-input-closed
by (auto simp add: is-aut.init-state-is-a-state)
show ?thesis
apply (clarsimp simp del: GMN-simps simp add: induced-epi-def words-to-syth-states-def
states-to-words-def compose-def image-def)
using H1-0 H1-1
apply (clarsimp)
apply (rule subset-antisym; simp del: GMN-simps add: Set.subset-eq)
apply (metis (no-types, lifting) quotientI)
by (metis (no-types, lifting) alt-natural-map-MN-def induced-epi-wd2 quo-
tientE)
qed
show ?thesis
apply (simp del: GMN-simps add: G-aut-epi-def G-aut-epi-axioms-def)
apply (rule conjI)
subgoal
apply (clarsimp simp del: GMN-simps simp add: G-aut-hom-def aut-hom-def
reach-det-G-aut-def
is-reachable det-G-aut-def reach-det-aut-def reach-det-aut-axioms-def)
apply (intro conjI)
apply (simp add: is-aut.det-aut-axioms)
using labels-a-G-set.alt-grp-act-axioms
apply (auto)[1]
using states-a-G-set.alt-grp-act-axioms
apply blast
apply (simp add: accepting-is-eq-var.eq-var-subset-axioms)

```

```

using init-is-eq-var.eq-var-subset-axioms
  apply (auto)[1]
  apply (simp add: trans-is-eq-var.eq-var-func-axioms)
  apply (simp add: is-aut.det-aut-axioms)
using syth-aut-is-det-aut
  apply simp
using labels-a-G-set.alt-grp-act-axioms
  apply (auto)[1]
  apply (metis MN-rel-eq-var MN-rel-equival eq-var-rel.quot-act-is-grp-act)
using MN-final-state-equiv
  apply presburger
using MN-init-state-equivar-v2
  apply presburger
using MN-trans-eq-var-func
  apply blast
using syth-aut-is-det-aut
  apply auto[1]
  apply (clarify)
  apply (simp add: H-0 del: GMN-simps)
  apply (simp add: is-aut.det-aut-axioms)
using syth-aut-is-det-aut
  apply blast
apply (clarsimp del: subset-antisym simp del: GMN-simps simp add: aut-hom-axioms-def
  FuncSet.extensional-funcset-def Pi-def extensional-def)[1]
  apply (intro conjI)
  apply (clarify)
  apply (simp add: induced-epi-def)
  apply (simp add: induced-epi-def words-to-syth-states-def states-to-words-def
  compose-def)
  apply (rule meta-mp[of  $(\delta^*) i Nil = i$ ])
using induced-epi-wd2[where  $w = Nil$ ]
  apply (auto simp add: is-aut.init-state-is-a-state del: subset-antisym)[2]
subgoal for  $x$ 
  apply (rule quotientI)
  using is-reachable[where  $s = x$ ] someI[where  $P = \lambda w. w \in A^* \wedge (\delta^*) i w$ 
=  $x$ ]
  by blast
  apply (auto simp add: induced-epi-def words-to-syth-states-def states-to-words-def
  compose-def)[1]
  apply (simp add: induced-epi-def states-to-words-def compose-def
  is-aut.init-state-is-a-state)
  apply (metis (mono-tags, lifting)  $\langle \bigwedge w'. \llbracket \_ \rrbracket \in A^*; w' \in A^*;$ 
 $(\delta^*) i \llbracket \_ \rrbracket = (\delta^*) i w \rrbracket \implies MN\text{-init-state} = \llbracket w \rrbracket_{MN}$ 
alt-natural-map-MN-def give-input.simps(1) lists.Nil some-eq-imp
  words-to-syth-states-def)
  apply (clarify)
subgoal for  $s$ 
  apply (rule iffI)
  apply (smt (verit) Pi-iff compose-eq in-mono induced-epi-def is-aut.fin-states-are-states

```

```

      states-to-words-on-final words-to-syth-states-def)
apply (clarsimp simp del: GMN-simps simp add: induced-epi-def words-to-syth-states-def
      states-to-words-def compose-def)
apply (rule meta-mp[of (SOME w. w ∈ A* ∧ (δ*) i w = s) ∈ L])
apply (smt (verit) induced-epi-wd1 is-recognised someI)
using fin-states-rep-by-lang is-reachable mem-Collect-eq
by (metis (mono-tags, lifting))
apply (clarsimp simp del: GMN-simps)
apply (simp add: H-1)
using induced-epi-eq-var
by blast
by (simp add: H-2)
qed

end

```

lemma (in *det-G-aut*) *finite-reachable*:

$finite (orbits\ G\ S\ \psi) \implies finite (orbits\ G\ S_{reach}\ \psi_{reach})$

proof –

assume

A-0: $finite (orbits\ G\ S\ \psi)$

have *H-0*: $S_{reach} \subseteq S$

apply (clarsimp simp add: reachable-states-def)

by (simp add: in-listsI is-aut.give-input-closed is-aut.init-state-is-a-state)

have *H-1*: $\{\{\psi\ g\ x\ |g. g \in carrier\ G\} |x. x \in S_{reach}\} \subseteq$

$\{\{\psi\ g\ x\ |g. g \in carrier\ G\} |x. x \in S\}$

by (smt (verit, best) Collect-mono-iff H-0 subsetD)

have *H-2*: $\bigwedge x. x \in S_{reach} \implies$

$\{\psi\ g\ x\ |g. g \in carrier\ G\} = \{\psi_{reach}\ g\ x\ |g. g \in carrier\ G\}$

using reachable-action-is-restrict

by (metis)

hence *H-3*: $\{\{\psi\ g\ x\ |g. g \in carrier\ G\} |x. x \in S_{reach}\} =$

$\{\{\psi_{reach}\ g\ x\ |g. g \in carrier\ G\} |x. x \in S_{reach}\}$

by blast

show $finite (orbits\ G\ S_{reach}\ \psi_{reach})$

using *A-0* **apply** (clarsimp simp add: orbits-def orbit-def)

using Finite-Set.finite-subset H-1 H-3

by auto

qed

lemma (in *det-G-aut*)

orbs-pos-card: $finite (orbits\ G\ S\ \psi) \implies card (orbits\ G\ S\ \psi) > 0$

apply (clarsimp simp add: card-gt-0-iff orbits-def)

using is-aut.init-state-is-a-state

by auto

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 –

have *H-0*: $finite\ \{\{\psi\ g\ x\ |g.\ g \in carrier\ G\}\ |x.\ x \in S\}$
using *Fin*
by (*auto simp add: orbits-def orbit-def*)

have *H-1*: $induced-epi\ 'S = MN-equiv$
using *reach-det-G-aut-rec-lang*
by (*auto simp del: GMN-simps simp add: G-aut-epi-def G-aut-epi-axioms-def*)

have *H-2*: $\bigwedge B\ f.\ finite\ B \implies finite\ \{f\ b\ | b.\ b \in B\}$
by *auto*

have *H-3*: $finite\ \{\{\psi\ g\ x\ |g.\ g \in carrier\ G\}\ |x.\ x \in S\} \implies$
 $finite\ \{induced-epi\ 'b\ |b.\ b \in \{\{\psi\ g\ x\ |g.\ g \in carrier\ G\}\ |x.\ x \in S\}\}$
using *H-2* [**where** *f1* = $(\lambda x.\ induced-epi\ 'x)$ **and** *B1* = $\{\{\psi\ g\ x\ |g.\ g \in carrier\ G\}\ |x.\ x \in S\}$]

by *auto*

have *H-4*: $\bigwedge s.\ s \in S \implies \exists b.\ \{induced-epi\ (\psi\ g\ s)\ |g.\ g \in carrier\ G\}$
 $= \{y.\ \exists x \in b.\ y = induced-epi\ x\} \wedge (\exists x.\ b = \{\psi\ g\ x\ |g.\ g \in carrier\ G\} \wedge x \in S)$

proof –

fix *s*
assume
 $A2-0: s \in S$

have *H2-0*: $\{induced-epi\ (\psi\ g\ s)\ |g.\ g \in carrier\ G\} = \{y.\ \exists x \in \{\psi\ g\ s\ |g.\ g \in carrier\ G\}.\ y = induced-epi\ x\}$
by *blast*

have *H2-1*: $(\exists x.\ \{\psi\ g\ s\ |g.\ g \in carrier\ G\} = \{\psi\ g\ x\ |g.\ g \in carrier\ G\} \wedge x \in S)$
using *A2-0*
by *auto*

show $\exists b.\ \{induced-epi\ (\psi\ g\ s)\ |g.\ g \in carrier\ G\} = \{y.\ \exists x \in b.\ y = induced-epi\ x\} \wedge (\exists x.\ b = \{\psi\ g\ x\ |g.\ g \in carrier\ G\} \wedge x \in S)$
using *A2-0 H2-0 H2-1*
by *meson*

qed

have *H-5*: $\{induced-epi\ 'b\ |b.\ b \in \{\{\psi\ g\ x\ |g.\ g \in carrier\ G\}\ |x.\ x \in S\}\} = \{\{induced-epi\ (\psi\ g\ s)\ |g.\ g \in carrier\ G\}\ |s.\ s \in S\}$
apply (*clarsimp simp add: image-def*)
apply (*rule subset-antisym; simp add: Set.subset-eq; clarify*)
apply *auto*[1]
apply (*simp*)
by (*simp add: H-4*)

from *H-3 H-5* **have** *H-6*: $finite\ \{\{\psi\ g\ x\ |g.\ g \in carrier\ G\}\ |x.\ x \in S\} \implies$
 $finite\ \{\{induced-epi\ (\psi\ g\ s)\ |g.\ g \in carrier\ G\}\ |s.\ s \in S\}$
by *metis*

have *H-7*: $finite\ \{\{induced-epi\ (\psi\ g\ x)\ |g.\ g \in carrier\ G\}\ |x.\ x \in S\}$
apply (*rule H-6*)
by (*simp add: H-0*)

have *H-8*: $\bigwedge x. x \in S \implies \{ \text{induced-epi } (\psi \ g \ x) \mid g. g \in \text{carrier } G \} =$
 $\{ \{ [(\varphi^*)]_{\equiv MN \ A^*} \} \ g \ (\text{induced-epi } x) \mid g. g \in \text{carrier } G \}$
using *induced-epi-eq-var*
apply (*simp del: GMN-simps add: eq-var-func-def eq-var-func-axioms-def make-op-def*)
by *blast*
hence *H-9*: $\{ \{ \text{induced-epi } (\psi \ g \ x) \mid g. g \in \text{carrier } G \} \mid x. x \in S \} =$
 $\{ \{ [(\varphi^*)]_{\equiv MN \ A^*} \} \ g \ (\text{induced-epi } x) \mid g. g \in \text{carrier } G \} \mid x. x \in S \}$
by *blast*
have *H-10*: $\bigwedge f \ g \ X \ B \ C. g \ ' \ B = C \implies$
 $\{ \{ f \ x \ (g \ b) \mid x. x \in X \} \mid b. b \in B \} = \{ \{ f \ x \ c \mid x. x \in X \} \mid c. c \in C \}$
by *auto*
have *H-11*: $\{ \{ [(\varphi^*)]_{\equiv MN \ A^*} \} \ g \ (\text{induced-epi } x) \mid g. g \in \text{carrier } G \} \mid x. x \in S \} =$
 $\{ \{ [(\varphi^*)]_{\equiv MN \ A^*} \} \ g \ W \mid g. g \in \text{carrier } G \} \mid W. W \in \text{MN-equiv} \}$
apply (*rule H-10[where f2 = ([(\varphi^*)]_{\equiv MN \ A^*}) and X2 = carrier G and g2*
 $= \text{induced-epi}$
and $B2 = S$ **and** $C2 = \text{MN-equiv}$])
using *H-1*
by *simp*
have *H-12*: $\{ \{ [(\varphi^*)]_{\equiv MN \ A^*} \} \ g \ W \mid g. g \in \text{carrier } G \} \mid W. W \in \text{MN-equiv} \} =$
 $\text{orbits } G \ (\text{language.MN-equiv } A \ L) \ ([(\varphi^*)]_{\equiv MN \ A^*})$
by (*auto simp add: orbits-def orbit-def*)
show *finite* ($\text{orbits } G \ (\text{language.MN-equiv } A \ L) \ ([(\varphi^*)]_{\equiv MN \ A^*})$)
using *H-9 H-11 H-12 H-7*
by *presburger*
qed

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

To avoid duplicate variant of "star":

no-adhoc-overloading

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

end

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

adhoc-overloading

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

end

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

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

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

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

by (*meson G-lang-axioms finite-reachable reach-det-G-aut-rec-lang.intro*

reach-det-aut-is-det-aut-rec-L)

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) (((φ^) $_{\equiv MN}$ A *)))*
using *MN-prep*
by (*metis assms reach-det-G-aut-rec-lang.MN-B2T*)

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
finite (orbits G A φ)
shows
G-Myhill-Nerode-LR: finite (orbits G MN-equiv ((φ^) $_{\equiv MN}$ A *)) \implies
($\exists S F ::$ 'alpha list 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))
 \implies *finite (orbits G MN-equiv ((φ^*) $_{\equiv MN}$ A *))*
subgoal
using *syntact-aut-is-reach-aut-rec-lang*
by *blast*
by (*metis det-G-aut-rec-lang.MN-fin-orbs-imp-fin-states*)**

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 ($\lambda x \in \{undefined\}.$ one (BijGroup X))
apply (*simp add: group-action-def group-hom-def group-hom-axioms-def*)
apply (*intro conjI*)
apply (*simp add: group-BijGroup*)
using *trivial-hom*
by (*smt (verit) carrier-singleton-group group.hom-restrict group-BijGroup re-strict-apply*
singleton-group)

```

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  $\delta$ 
  triv-G ( $\lambda x \in \{\text{undefined}\}$ . one (BijGroup labels)) ( $\lambda x \in \{\text{undefined}\}$ . one (BijGroup
  states))
  apply (simp add: det-G-aut-def group-hom-def group-hom-axioms-def
    eq-var-subset-def eq-var-subset-axioms-def eq-var-func-def eq-var-func-axioms-def)
  apply (intro conjI)
    apply (rule det-aut-axioms)
    apply (simp add: assms triv-act.group-action-axioms)+
  using fin-states-are-states
    apply (auto)[1]
    apply (clarify; rule conjI; rule impI)
    apply (simp add: H-triv-G BijGroup-def image-def)
  using fin-states-are-states
    apply auto[1]
    apply (simp add: H-triv-G BijGroup-def image-def)
    apply (simp add: assms triv-act.group-action-axioms)
    apply (simp add: init-state-is-a-state)
    apply (clarify; rule conjI; rule impI)
    apply (simp add: H-triv-G BijGroup-def image-def init-state-is-a-state)+
  apply (clarsimp simp add: group-action-def BijGroup-def hom-def group-hom-def
    group-hom-axioms-def)
    apply (rule conjI)
  apply (smt (verit) BijGroup-def Bij-imp-funcset Id-compose SigmaE case-prod-conv
    group-BijGroup id-Bij restrict-ext restrict-extensional)
  apply (rule meta-mp[of undefined  $\otimes$  singleton-group undefined undefined = un-
  defined])
    apply (auto)[1]
    apply (metis carrier-singleton-group comm-groupE(1) singletonD singletonI
    singleton-abelian-group)
    apply (simp add: assms triv-act.group-action-axioms)
  apply (auto simp add: trans-func-well-def)[1]
  by (clarsimp simp add: BijGroup-def trans-func-well-def H-triv-G)

```

```

lemma triv-orbits:
  orbits (singleton-group (undefined)) S ( $\lambda x \in \{\text{undefined}\}$ . one (BijGroup S)) =
   $\{\{s\} \mid s \in S\}$ 
  apply (simp add: BijGroup-def singleton-group-def orbits-def orbit-def)
  by auto

```

```

lemma fin-triv-orbs:
  finite (orbits (singleton-group (undefined)) S ( $\lambda x \in \{\text{undefined}\}$ . one (BijGroup
  S))) = finite S
  apply (subst triv-orbits)
  apply (rule meta-mp[of bij-betw ( $\lambda s \in S$ .  $\{s\}$ ) S  $\{\{s\} \mid s \in S\}$ ])
  using bij-betw-finite

```

apply (*auto*)[1]
by (*auto simp add: bij-betw-def image-def*)

context *language* **begin**

interpretation *triv-G-lang*:

G-lang singleton-group (*undefined*) *A* ($\lambda x \in \{\text{undefined}\}$. *one* (*BijGroup A*)) *L*
apply (*simp add: G-lang-def G-lang-axioms-def eq-var-subset-def eq-var-subset-axioms-def*)
apply (*intro conjI*)
 apply (*simp add: triv-act.group-action-axioms*)
 apply (*simp add: language-axioms*)
using *triv-act.lists-a-Gset*
 apply *fastforce*
 apply (*rule is-lang*)
apply (*clarsimp simp add: BijGroup-def image-def*)
apply (*rule subset-antisym; simp add: Set.subset-eq; clarify*)
using *is-lang*
 apply (*auto simp add: map-idI*)[1]
using *is-lang map-idI*
by (*metis in-listsD in-mono inf.absorb-iff1 restrict-apply*)

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
 $(\exists S F :: 'alpha \text{ list set set. } \exists i :: 'alpha \text{ list set. } \exists \delta$.
reach-det-aut-rec-lang A S i F $\delta L \wedge$ *finite S*) **and**
standard-Myhill-Nerode-RL: $(\exists S F :: 's \text{ set. } \exists i :: 's. \exists \delta$.
det-aut-rec-lang A S i F $\delta L \wedge$ *finite S*) \implies *finite MN-equiv*

proof –

assume

A-0: finite MN-equiv

have *H-0: reach-det-aut-rec-lang A MN-equiv MN-init-state MN-fin-states* δ_{MN}

L

using *triv-G-lang.syntact-aut-is-reach-aut-rec-lang*

apply (*clarsimp simp add: reach-det-G-aut-rec-lang-def det-G-aut-rec-lang-def*
reach-det-aut-rec-lang-def reach-det-aut-def reach-det-aut-axioms-def det-G-aut-def)

by (*smt (z3) alt-natural-map-MN-def quotientE triv-G-lang.MN-unique-init-state*)

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

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

using *A-0 H-0*

by *auto*

```

next
  assume
    A-0:  $\exists S F :: 's \text{ set. } \exists i :: 's. \exists \delta. \text{det-aut-rec-lang } A S i F \delta L \wedge \text{finite } S$ 
  obtain  $S F :: 's \text{ set and } i :: 's \text{ and } \delta$ 
  where  $H\text{-MN}: \text{det-aut-rec-lang } A S i F \delta L \wedge \text{finite } S$ 
  using A-0
  by auto
  have H-0:  $\text{det-G-aut } A S i F \delta \text{triv-G } (\lambda x \in \{\text{undefined}\}. \mathbf{1}_{\text{BijGroup } A})$ 
     $(\lambda x \in \{\text{undefined}\}. \mathbf{1}_{\text{BijGroup } S})$ 
  apply (rule det-aut.triv-G-aut[of  $A S i F \delta \text{triv-G}$ ])
  using H-MN
  apply (simp add: det-aut-rec-lang-def)
  by (rule triv-G-def)
  have H-1:  $\text{det-G-aut-rec-lang } A S i F \delta \text{triv-G } (\lambda x \in \{\text{undefined}\}. \mathbf{1}_{\text{BijGroup } A})$ 
     $(\lambda x \in \{\text{undefined}\}. \mathbf{1}_{\text{BijGroup } S}) L$ 
  by (auto simp add: det-G-aut-rec-lang-def H-0 H-MN)
  have H-2:  $(\exists S F :: 's \text{ set. } \exists i :: 's. \exists \delta \psi.$ 
     $\text{det-G-aut-rec-lang } A S i F \delta (\text{singleton-group undefined}) (\lambda x \in \{\text{undefined}\}.$ 
 $\mathbf{1}_{\text{BijGroup } A})$ 
     $\psi L \wedge \text{finite } (\text{orbits } (\text{singleton-group undefined}) S \psi))$ 
  using H-1
  by (metis H-MN fin-triv-orbs triv-G-def)
  have H-3:  $\text{finite } (\text{orbits } \text{triv-G } A \text{triv-act})$ 
  apply (subst triv-G-def; subst triv-act-def; subst fin-triv-orbs[of  $A$ ])
  by (rule H-fin-alph)
  have H-4:  $\text{finite } (\text{orbits } \text{triv-G } \text{MN-equiv } (\text{triv-act.induced-quot-map } (A^*)$ 
     $(\text{triv-act.induced-star-map } A \text{triv-act}) \equiv_{MN}))$ 
  using H-3
  apply (simp add: triv-G-def triv-act-def del: GMN-simps)
  using triv-G-lang.G-Myhill-Nerode H-2
  by blast
  have H-5:  $\text{triv-act.induced-star-map } A \text{triv-act} = (\lambda x \in \{\text{undefined}\}. \mathbf{1}_{\text{BijGroup } (A^*)})$ 
  apply (simp add: BijGroup-def restrict-def fun-eq-iff triv-act-def)
  by (clarsimp simp add: list.map-ident-strong)
  have H-6:  $(\text{triv-act.induced-quot-map } (A^*) (\text{triv-act.induced-star-map } A$ 
     $\text{triv-act}) \equiv_{MN}) = (\lambda x \in \{\text{undefined}\}. \mathbf{1}_{\text{BijGroup } \text{MN-equiv}})$ 
  apply (subst H-5)
  apply (simp add: BijGroup-def fun-eq-iff Image-def)
  apply (rule allI; rule conjI; intro impI)
  apply (smt (verit) Collect-cong Collect-mem-eq Eps-cong MN-rel-equival equiv-Eps-in
    in-quotient-imp-closed quotient-eq-iff)
  using MN-rel-equival equiv-Eps-preserves
  by auto
  show finite MN-equiv
  apply (subst fin-triv-orbs [symmetric]; subst H-6 [symmetric]; subst triv-G-def
    [symmetric])
  by (rule H-4)
qed
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 (D) 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 (D) 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  $\subseteq$  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  $\in$  carrier G  $\implies$  supports G D  $\pi$  X  $\varphi$  ( $\pi$  g ' C) (g  $\odot_{\varphi}$  x)
proof –
  fix g
  assume
  A-0: g  $\in$  carrier G
  have H-0:  $\bigwedge$  h. data-symm G D  $\pi$   $\implies$ 
    group-action G X  $\varphi$   $\implies$ 
    x  $\in$  X  $\implies$ 
    C  $\subseteq$  D  $\implies$ 
     $\forall$  g. g  $\in$  carrier G  $\longrightarrow$  ( $\forall$  c. c  $\in$  C  $\longrightarrow$   $\pi$  g c = c)  $\longrightarrow$   $\varphi$  g x = x  $\implies$ 
    h  $\in$  carrier G  $\implies$   $\forall$  c. c  $\in$   $\pi$  g ' C  $\longrightarrow$   $\pi$  h c = c  $\implies$ 
     $\varphi$  h ( $\varphi$  g x) =  $\varphi$  g x
proof –
```

```

fix h
assume
  A1-0: data-symm G D  $\pi$  and
  A1-1: group-action G X  $\varphi$  and
  A1-2:  $\forall g. g \in \text{carrier } G \longrightarrow (\forall c. c \in C \longrightarrow \pi g c = c) \longrightarrow \varphi g x = x$  and
  A1-3:  $h \in \text{carrier } G$  and
  A1-4:  $\forall c. c \in \pi g ' C \longrightarrow \pi h c = c$ 
have H1-0:  $\bigwedge g. g \in \text{carrier } G \implies (\forall c. c \in C \longrightarrow \pi g c = c) \implies \varphi g x = x$ 
using A1-2
by auto
have H1-1:  $\forall c. c \in C \longrightarrow \pi ((\text{inv}_G g) \otimes_G h \otimes_G g) c = c \implies$ 
 $\varphi ((\text{inv}_G g) \otimes_G h \otimes_G g) x = x$ 
apply (rule H1-0[of  $((\text{inv}_G g) \otimes_G h \otimes_G g)$ ])
apply (meson A-0 A1-3 group.subgroupE(3) group.subgroup-self group-hom
group-hom.axioms(1)
subgroup.m-closed)
by simp
have H2:  $\pi (((\text{inv}_G g) \otimes_G h) \otimes_G g) = \text{compose } \mathbf{D} (\pi ((\text{inv}_G g) \otimes_G h)) (\pi g)$ 
using A1-0
apply (clarsimp simp add: data-symm-def group-action-def BijGroup-def
group-hom-def
group-hom-axioms-def hom-def restrict-def)
apply (rule meta-mp[of  $\pi g \in \text{Bij } \mathbf{D} \wedge \pi ((\text{inv}_G g) \otimes_G h) \in \text{Bij } \mathbf{D}$ ])
apply (smt (verit) A-0 A1-3 data-symm.axioms data-symm-axioms group.inv-closed
group.surj-const-mult group-action.bij-prop0 image-eqI)
apply (rule conjI)
using A-0
apply blast
by (meson A-0 A1-3 data-symm.axioms data-symm-axioms group.subgroupE(3)
group.subgroupE(4)
group.subgroup-self group-action.bij-prop0)
also have H1-3:  $\dots = \text{compose } \mathbf{D} (\text{compose } \mathbf{D} (\pi (\text{inv}_G g)) (\pi h)) (\pi g)$ 
using A1-0
apply (clarsimp simp add: data-symm-def group-action-def BijGroup-def
comp-def
group-hom-def group-hom-axioms-def hom-def restrict-def)
apply (rule meta-mp[of  $\pi (\text{inv}_G g) \in \text{Bij } \mathbf{D} \wedge \pi h \in \text{Bij } \mathbf{D}$ ])
apply (simp add: A-0 A1-3)
apply (rule conjI)
apply (simp add: A-0 Pi-iff)
using A1-3
by blast
also have H1-4:  $\dots = \text{compose } \mathbf{D} ((\pi (\text{inv}_G g)) \circ (\pi h)) (\pi g)$ 
using A1-0
apply (clarsimp simp add: data-symm-def group-action-def BijGroup-def
comp-def group-hom-def
group-hom-axioms-def hom-def restrict-def compose-def)
using A-0 A1-3
by (meson data-symm.axioms data-symm-axioms group.inv-closed group-action.element-image)

```

also have *H1-5*: $\dots = (\lambda d \in \mathbf{D}. ((\pi (\text{inv}_G g)) \circ (\pi h) \circ (\pi g)) d)$
by (*simp add: compose-def*)
have *H1-6*: $\bigwedge c. c \in C \implies ((\pi h) \circ (\pi g)) c = (\pi g) c$
using *A1-4*
by *auto*
have *H1-7*: $\bigwedge c. c \in C \implies ((\pi (\text{inv}_G g)) \circ (\pi h) \circ (\pi g)) c = c$
using *H1-6 A1-0*
apply (*simp add: data-symm-def group-action-def BijGroup-def compose-def*
group-hom-def
group-hom-axioms-def hom-def)
by (*meson A-0 data-symm.axioms data-symm-axioms group-action.orbit-sym-aux*
is-subset subsetD)
have *H1-8*: $\forall c. c \in C \longrightarrow \pi ((\text{inv}_G g) \otimes_G h \otimes_G g) c = c$
using *H1-7 H1-5*
by (*metis calculation is-subset restrict-apply' subsetD*)
have *H1-9*: $\varphi ((\text{inv}_G g) \otimes_G h \otimes_G g) x = x$
using *H1-8*
by (*simp add: H1-1*)
hence *H1-10*: $\varphi ((\text{inv}_G g) \otimes_G h \otimes_G g) x = \varphi ((\text{inv}_G g) \otimes_G (h \otimes_G g)) x$
by (*smt (verit, ccfv-SIG) A-0 A1-3 group.inv-closed group.subgroupE(4)*
group.subgroup-self
group-action.composition-rule group-action.element-image group-action-axioms
group-hom
group-hom.axioms(1) is-in-set)
have *H1-11*: $\dots = ((\varphi (\text{inv}_G g)) \circ (\varphi (h \otimes_G g))) x$
using *A-0 A1-3 group.subgroupE(4) group.subgroup-self group-action.composition-rule*
group-action-axioms group-hom group-hom.axioms(1) is-in-set
by *fastforce*
have *H1-12*: $\dots = ((\text{the-inv-into } X (\varphi g)) \circ (\varphi (h \otimes_G g))) x$
using *A1-1*
apply (*simp add: group-action-def*)
by (*smt (verit) A-0 A1-3 group.inv-closed group.subgroupE(4) group.subgroup-self*
group-action.element-image group-action.inj-prop group-action.orbit-sym-aux
group-action-axioms group-hom.axioms(1) is-in-set the-inv-into-f-f)
have *H1-13*: $((\text{the-inv-into } X (\varphi g)) \circ (\varphi (h \otimes_G g))) x = x$
using *H1-9 H1-10 H1-11 H1-12*
by *auto*
hence *H1-14*: $(\varphi (h \otimes_G g)) x = \varphi g x$
using *H1-13*
by (*metis A-0 A1-3 comp-apply composition-rule element-image f-the-inv-into-f*
inj-prop is-in-set
surj-prop)
show $\varphi h (\varphi g x) = \varphi g x$
using *A1-3 A1-2 A-0 H1-14 composition-rule*
by (*simp add: is-in-set*)
qed
show *supports* $G D \pi X \varphi (\pi g \text{ ' } C) (g \odot_\varphi x)$
using *supports-axioms*
apply (*clarsimp simp add: supports-def supports-axioms-def*)

```

apply (intro conjI)
using element-image is-in-set A-0
apply blast
apply (metis A-0 data-symm-def group-action.surj-prop image-mono is-subset)
apply (rule allI; intro impI)
apply (rename-tac h)
by (simp add: H-0)
qed
end

```

```

locale nominal = data-symm G D  $\pi$  + alt-grp-act G X  $\varphi$ 
for
  G :: ('grp, 'b) monoid-scheme and
  D :: 'D set (ID) and
   $\pi$  and
  X :: 'X set (structure) and
   $\varphi$  +
assumes
  is-nominal:
   $\bigwedge g x. g \in \text{carrier } G \implies x \in X \implies \exists C. C \subseteq \mathbf{ID} \wedge \text{finite } C \wedge \text{supports } G \mathbf{ID} \pi$ 
  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 set (ID) and
   $\pi$ 

```

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

```

lemma (in eq-var-func) supp-el-pres:
  supports G D  $\pi$  X  $\varphi$  C x  $\implies$  supports G D  $\pi$  Y  $\psi$  C (f x)
apply (clarsimp simp add: supports-def supports-axioms-def)
apply (rule conjI)
using eq-var-func-axioms
apply (simp add: eq-var-func-def eq-var-func-axioms-def)
apply (intro conjI)
using is-ext-func-bet
apply blast
apply clarify
by (metis is-eq-var-func')

```

```

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

```



```

proof –
  fix  $x$ 
  assume  $A-0: x \in col$ 
  have  $H-0: \bigwedge x. x \in X \implies \exists C. C \subseteq \mathbb{D} \wedge finite\ C \wedge supports\ G\ \mathbb{D}\ \pi\ X\ \varphi\ C\ x$ 
    using nominal-axioms
    apply (clarsimp simp add: nominal-def nominal-axioms-def)
    using stabilizer-one-closed stabilizer-subset
    by blast
  have  $H-1: \bigwedge x. x \in col \implies f\ x \subseteq \mathbb{D} \wedge finite\ (f\ x) \wedge supports\ G\ \mathbb{D}\ \pi\ X\ \varphi\ (f\ x)\ x$ 
    apply (subst H-f)
    using someI-ex H-col H-f H-0
    by (metis (no-types, lifting) in-mono)
  have  $H-2: sup-C \subseteq \mathbb{D}$ 
    using  $H-1$ 
    by (simp add: H-sup-C UN-least)
  have  $H-3: finite\ sup-C$ 
    using  $H-1\ H-col\ H-sup-C$ 
    by simp
  have  $H-4: f\ x \subseteq sup-C$ 
    using  $H-1\ H-sup-C\ A-0$ 
    by blast
  have  $H-5: \bigwedge g\ c. \llbracket g \in carrier\ G; (c \in sup-C \longrightarrow \pi\ g\ c = c); c \in (f\ x) \rrbracket \implies \pi\ g\ c = c$ 
    using  $H-4\ H-1\ A-0$ 
    by (auto simp add: image-def supports-def supports-axioms-def)
  have  $H-6: supports\ G\ \mathbb{D}\ \pi\ X\ \varphi\ sup-C\ x$ 
    apply (simp add: supports-def supports-axioms-def)
    apply (intro conjI)
    apply (simp add: data-symm-axioms)
    using  $A-0\ H-1\ supports.axioms(2)$ 
    apply fastforce
    using  $H-col\ A-0$ 
    apply blast
    apply (rule H-2)
    apply (clarify)
    using supports-axioms-def[of G D π X φ sup-C]
    apply (clarsimp)
    using  $H-1\ A-0$ 
    apply (clarsimp simp add: supports-def supports-axioms-def)
    using  $A-0\ H-5$ 
    by presburger
  show  $sup-C \subseteq \mathbb{D} \wedge finite\ sup-C \wedge supports\ G\ \mathbb{D}\ \pi\ X\ \varphi\ sup-C\ x$ 
    using  $H-2\ H-3\ H-6$  by auto
qed

```

lemma (*in nominal*) *set-of-list-nom*:

nominal G D π (X) (φ*)*

proof –

have $H-0: \bigwedge g\ x. g \in carrier\ G \implies \forall x \in set\ x. x \in X \implies$

```

       $\exists C \subseteq \mathbb{D}. \text{finite } C \wedge \text{supports } G \mathbb{D} \pi (X^*) (\varphi^*) C x$ 
proof–
  fix  $g w$ 
  assume
     $A1-0: g \in \text{carrier } G$  and
     $A1-1: \forall x \in \text{set } w. x \in X$ 
  have  $H1-0: \bigwedge x. x \in X \implies \exists C \subseteq \mathbb{D}. \text{finite } C \wedge \text{supports } G \mathbb{D} \pi X \varphi C x$ 
    using  $A1-0$  is-nominal by force
  define  $f$  where  $H1-f: f = (\lambda x. (\text{SOME } C. C \subseteq \mathbb{D} \wedge \text{finite } C \wedge \text{supports } G \mathbb{D} \pi X \varphi C x))$ 
  define  $\text{sup-C} :: 'D \text{ set}$  where  $H1\text{-sup-C}: \text{sup-C} = \bigcup \{Cx. Cx \in f \text{ 'set } w\}$ 
  have  $H1-1: \bigwedge x. x \in \text{set } w \implies \text{sup-C} \subseteq \mathbb{D} \wedge \text{finite } \text{sup-C} \wedge \text{supports } G \mathbb{D} \pi X \varphi \text{sup-C } x$ 
    apply (rule support-union-lem[where  $f = f$  and  $\text{col} = \text{set } w$ ])
    apply (rule H1-f)
    using  $A1-0$ 
    apply (simp add: A1-1 subset-code(1))
    apply (rule H1-sup-C)
    by simp
  have  $H1-2: \text{supports } G \mathbb{D} \pi (X^*) (\varphi^*) \text{sup-C } w$ 
  apply (clarsimp simp add: supports-def supports-axioms-def simp del: GMN-simps)
  apply (intro conjI)
    apply (simp add: data-symm-axioms)
  using lists-a-Gset
    apply (auto)[1]
    apply (simp add: A1-1 in-listsI)
  using  $H1-1 H1\text{-sup-C}$ 
  apply blast
  apply (rule allI; intro impI)
  apply clarsimp
  apply (rule conjI)
  using  $H1-1$ 
  by (auto simp add: supports-def supports-axioms-def map-idI)
  show  $\exists C \subseteq \mathbb{D}. \text{finite } C \wedge \text{supports } G \mathbb{D} \pi (X^*) (\varphi^*) C w$ 
    using nominal-axioms-def
    apply (clarsimp simp add: nominal-def simp del: GMN-simps)
    using  $H1-1 H1-2$ 
    by (metis Collect-empty-eq H1-sup-C Union-empty empty-iff image-empty infinite-imp-nonempty subset-empty subset-emptyI supports.is-subset)
qed
show ?thesis
  apply (clarsimp simp add: nominal-def nominal-axioms-def simp del: GMN-simps)
  apply (intro conjI)
  using group.subgroupE(2) group.subgroup-self group-hom group-hom.axioms(1)
    apply (simp add: data-symm-axioms)
    apply (rule lists-a-Gset)
  apply (clarify)
  by (simp add: H-0 del: GMN-simps)

```

qed

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 $(\lambda w \in A^*. (\delta^*) i w)$

proof –

have *H-0*: $\bigwedge a g. \llbracket \forall x \in \text{set } a. x \in A; \text{map } (\varphi g) a \in A^*; g \in \text{carrier } G \rrbracket \implies$
 $(\delta^*) i (\text{map } (\varphi g) a) = \psi g ((\delta^*) i a)$

proof –

fix $w g$

assume

A1-0: $\forall x \in \text{set } w. x \in A$ and

A1-1: $\text{map } (\varphi g) w \in A^*$ and

A1-2: $g \in \text{carrier } G$

have *H1-0*: $(\delta^*) (\psi g i) (\text{map } (\varphi g) w) = \psi g ((\delta^*) i w)$

using *give-input-eq-var*

apply (*clarsimp simp add: eq-var-func-axioms-def eq-var-func-def*)

using *A1-0 A1-1 A1-2 in-listsI is-aut.init-state-is-a-state states-a-G-set.element-image*

by (*smt (verit, del-insts)*)

have *H1-1*: $(\psi g i) = i$

using *A1-2 is-aut.init-state-is-a-state init-is-eq-var.is-equivar*

by *force*

show $(\delta^*) i (\text{map } (\varphi g) w) = \psi g ((\delta^*) i w)$

using *H1-0 H1-1*

by *simp*

qed

show *?thesis*

apply (*clarsimp simp add: eq-var-func-def eq-var-func-axioms-def*)

apply (*intro conjI*)

using *labels-a-G-set.lists-a-Gset*

apply *fastforce*

apply (*simp add: states-a-G-set.group-action-axioms del: GMN-simps*)

apply (*simp add: in-listsI is-aut.give-input-closed is-aut.init-state-is-a-state*)

apply *clarify*

apply (*rule conjI; intro impI*)

apply (*simp add: H-0*)

using *labels-a-G-set.surj-prop*

by *fastforce*

qed

lemma (in *reach-det-G-aut*) *input-to-init-surj*:

$(\lambda w \in A^*. (\delta^*) i w) ' (A^*) = S$

```

using reach-det-G-aut-axioms
apply (clarsimp simp add: image-def reach-det-G-aut-def reach-det-aut-def
        reach-det-aut-axioms-def)
using is-aut.give-input-closed is-aut.init-state-is-a-state
by blast

```

```

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:

$nominal\ G\ D\ \pi\ A\ \varphi \implies nominal\ G\ D\ \pi\ S\ \psi$

proof –

assume

$A-0$: $nominal\ G\ D\ \pi\ A\ \varphi$

have $H-0$: $\bigwedge g\ x. \llbracket g \in carrier\ G; data-symm\ G\ D\ \pi; group-action\ G\ A\ \varphi;$

$\forall x. x \in A \longrightarrow (\exists C \subseteq D. finite\ C \wedge supports\ G\ D\ \pi\ A\ \varphi\ C\ x); x \in S \rrbracket$

$\implies \exists C \subseteq D. finite\ C \wedge supports\ G\ D\ \pi\ S\ \psi\ C\ x$

proof –

fix $g\ s$

assume

$A1-0$: $g \in carrier\ G$ **and**

$A1-1$: $data-symm\ G\ D\ \pi$ **and**

$A1-2$: $group-action\ G\ A\ \varphi$ **and**

$A1-3$: $\forall x. x \in A \longrightarrow (\exists C \subseteq D. finite\ C \wedge supports\ G\ D\ \pi\ A\ \varphi\ C\ x)$ **and**

$A1-4$: $s \in S$

have $H1-0$: $\bigwedge x. x \in (A^*) \implies \exists C \subseteq D. finite\ C \wedge supports\ G\ D\ \pi\ (A^*)\ (\varphi^*)\ C$

x

using nominal.set-of-list-nom[of $G\ D\ \pi\ A\ \varphi$] $A1-2$

apply (clarsimp simp add: nominal-def)

by (metis $A1-0\ A1-1\ A1-3$ in-listsI labels-a-G-set.induced-star-map-def nominal-axioms-def)

define f **where** $H1-f$: $f = (\lambda w \in A^*. (\delta^*)\ i\ w)$

obtain w **where** $H1-w0$: $s = f\ w$ **and** $H1-w1$: $w \in (A^*)$

using input-to-init-surj $A1-4$

apply (simp add: $H1-f$ image-def)

by (metis is-reachable)

obtain C **where** $H1-C$: $finite\ C \wedge supports\ G\ D\ \pi\ (A^*)\ (labels-a-G-set.induced-star-map\ \varphi)\ C\ w$

by (meson $H1-0\ H1-w0\ H1-w1$)

have $H1-2$: $supports\ G\ D\ \pi\ S\ \psi\ C\ s$

apply (subst $H1-w0$)

apply (rule eq-var-func.supp-el-pres[**where** $X = A^*$ **and** $\varphi = \varphi^*$])

apply (subst $H1-f$)

apply (rule det-G-aut.input-to-init-eqvar[of $A\ S\ i\ F\ \delta\ G\ \varphi\ \psi$])

using reach-det-G-aut-axioms

apply (simp add: reach-det-G-aut-def)

```

    using H1-C
    by simp
  show  $\exists C \subseteq D. \text{finite } C \wedge \text{supports } G D \pi S \psi C s$ 
    using H1-2 H1-C
    by (meson supports.is-subset)
qed
show ?thesis
  apply (rule meta-mp[of ( $\exists g. g \in \text{carrier } G$ )])
  subgoal
    using A-0 apply (clarsimp simp add: nominal-def nominal-axioms-def)
    apply (rule conjI)
    subgoal for g
      by (clarsimp simp add: states-a-G-set.group-action-axioms)
    apply clarify
    by (simp add: H-0)
  by (metis bot.extremum-unique ex-in-conv is-aut.init-state-is-a-state
    states-a-G-set.stabilizer-one-closed states-a-G-set.stabilizer-subset)
qed

```

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 ($[(\varphi^*)]_{\equiv MN} A^*$))* \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 ($[(\varphi^*)]_{\equiv MN} A^*$))*

proof –

assume

A-0: *finite (orbits G MN-equiv ($[(\varphi^*)]_{\equiv MN} A^*$))*

obtain *S F* :: *'alpha list set set* **and** *i* :: *'alpha list set* **and** δ ψ

where *H-MN*: *reach-det-G-aut-rec-lang A S i F δ G φ ψ L* \wedge *finite (orbits G S ψ)*

using *A-0 orb-fin G-Myhill-Nerode-LR*

by *blast*

have *H-0*: *nominal G D π S ψ*

using *H-MN*

apply (*clarsimp simp del: GMN-simps simp add: reach-det-G-aut-rec-lang-def*)

using *nom reach-det-G-aut.alpha-nom-imp-states-nom*

by *metis*

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

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*

```

apply (simp add: nominal-det-G-aut-def reach-det-G-aut-rec-lang-def)
using nom H-MN H-0
apply (clarsimp simp add: reach-det-G-aut-rec-lang-def reach-det-G-aut-def
reach-det-aut-axioms-def)
by blast
next
assume A0:  $\exists S F i \delta \psi. \text{det-G-aut-rec-lang } A S i F \delta G \varphi \psi L \wedge \text{finite (orbits } G S \psi)$ 
 $\wedge \text{nominal-det-G-aut } A S i F \delta G \varphi \psi D \pi$ 
show finite (orbits G MN-equiv ( $[\varphi^*]_{\equiv MN A^*}$ ))
using A0 orb-fin
by (meson G-Myhill-Nerode-RL)
qed
end

```

References

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