

Basic First-Order Model Theory – A Translation from HOL Light

Sophie Touret and Lawrence Paulson

March 29, 2025

Abstract

This AFP entry presents a proof of compactness of first-order logic, the Löwenheim-Skolem theorem and the Uniformity lemma. It is a translation of a HOL Light formalization work by John Harrison [1]. Whenever possible, existing Isabelle/HOL theories have been used instead of direct translation.

Contents

theory *FOL-Syntax*

imports

Main

Propositional-Proof-Systems.Compactness

First-Order-Terms.Term

First-Order-Terms.Subterm-and-Context

begin

no-notation *Not* (\neg)

no-notation *And* (**infix** \wedge 68)

no-notation *Or* (**infix** \vee 68)

lemma *count-terms:*

OFCLASS('f::countable, 'v::countable) term, countable-class)

by *countable-datatype*

instance *term* :: (*countable, countable*) *countable*

using *count-terms* **by** *simp*

type-synonym $nterm = \langle (nat, nat) term \rangle$

lemma $count\text{-}nterms: OFCLASS(nterm, countable\text{-}class)$
using $count\text{-}terms$ **by** $simp$

instance $formula :: (countable) countable$
by $countable\text{-}datatype$

term $test (0, [Var 0])$

abbreviation $functions\text{-}term :: \langle nterm \Rightarrow (nat \times nat) set \rangle$ **where**
 $\langle functions\text{-}term t \equiv funas\text{-}term t \rangle$

datatype $form =$
 $Bot \langle \perp \rangle$
 $| Atom (pred: nat) (args: \langle nterm list \rangle)$
 $| Implies form form (\mathbf{infixl} \langle \longrightarrow \rangle 85)$
 $| Forall nat form (\langle \forall \cdot \cdot \rangle [0, 70] 70)$

fun $functions\text{-}form :: \langle form \Rightarrow (nat \times nat) set \rangle$ **where**
 $\langle functions\text{-}form \perp = \{ \} \rangle$
 $| \langle functions\text{-}form (Atom p ts) = (\bigcup t \in set\ ts. functions\text{-}term t) \rangle$
 $| \langle functions\text{-}form (\varphi \longrightarrow \psi) = functions\text{-}form \varphi \cup functions\text{-}form \psi \rangle$
 $| \langle functions\text{-}form (\forall x. \varphi) = functions\text{-}form \varphi \rangle$

fun $predicates\text{-}form :: \langle form \Rightarrow (nat \times nat) set \rangle$ **where**
 $\langle predicates\text{-}form \perp = \{ \} \rangle$
 $| \langle predicates\text{-}form (Atom p ts) = \{ (p, length\ ts) \} \rangle$
 $| \langle predicates\text{-}form (\varphi \longrightarrow \psi) = predicates\text{-}form \varphi \cup predicates\text{-}form \psi \rangle$
 $| \langle predicates\text{-}form (\forall x. \varphi) = predicates\text{-}form \varphi \rangle$

definition $functions\text{-}forms :: \langle form\ set \Rightarrow (nat \times nat) set \rangle$ **where**
 $\langle functions\text{-}forms fms \equiv \bigcup f \in fms. functions\text{-}form f \rangle$

definition $predicates :: \langle form\ set \Rightarrow (nat \times nat) set \rangle$ **where**
 $\langle predicates fms \equiv \bigcup f \in fms. predicates\text{-}form f \rangle$

definition $language :: \langle form\ set \Rightarrow ((nat \times nat) set \times (nat \times nat) set) \rangle$ **where**
 $\langle language fms = (functions\text{-}forms fms, predicates fms) \rangle$

lemma $lang\text{-}singleton: \langle language \{p\} = (functions\text{-}form p, predicates\text{-}form p) \rangle$
unfolding $language\text{-}def$ $functions\text{-}forms\text{-}def$ $predicates\text{-}def$ **by** $simp$

abbreviation $Not :: \langle form \Rightarrow form \rangle (\langle \neg \cdot \rangle [90] 90)$ **where**
 $\langle \neg \varphi \equiv \varphi \longrightarrow \perp \rangle$

abbreviation $Top :: \langle form \rangle (\langle \top \rangle)$ **where**
 $\langle \top \equiv \neg \perp \rangle$

abbreviation *Or* :: $\langle \text{form} \Rightarrow \text{form} \Rightarrow \text{form} \rangle$

(**infixl** $\langle \vee \rangle$ δ_4) **where**

$\langle \varphi \vee \psi \equiv (\varphi \longrightarrow \psi) \longrightarrow \psi \rangle$

abbreviation *And* :: $\langle \text{form} \Rightarrow \text{form} \Rightarrow \text{form} \rangle$

(**infixl** $\langle \wedge \rangle$ δ_4) **where**

$\langle \varphi \wedge \psi \equiv \neg (\neg \varphi \vee \neg \psi) \rangle$

abbreviation *Equiv* :: $\langle \text{form} \Rightarrow \text{form} \Rightarrow \text{form} \rangle$

(**infix** $\langle \longleftrightarrow \rangle$ η_0) **where**

$\langle \varphi \longleftrightarrow \psi \equiv (\varphi \longrightarrow \psi \wedge \psi \longrightarrow \varphi) \rangle$

abbreviation *Exists* :: $\langle \text{nat} \Rightarrow \text{form} \Rightarrow \text{form} \rangle$

($\langle \exists \text{ .} \rightarrow [0, \eta_0] \eta_0 \rangle$) **where**

$\langle \exists x. \varphi \equiv \neg (\forall x. \neg \varphi) \rangle$

lemma *ex-all-distinct*: $\langle \forall x. \varphi \neq \exists y. \psi \rangle$

by *simp*

abbreviation *FVT* :: $\langle \text{nterm} \Rightarrow \text{nat set} \rangle$ **where**

$\langle \text{FVT} \equiv \text{vars-term} \rangle$

fun *FV* :: $\langle \text{form} \Rightarrow \text{nat set} \rangle$ **where**

$\langle \text{FV} \perp = \{\} \rangle$

| $\langle \text{FV} (\text{Atom} - \text{ts}) = (\bigcup a \in \text{set ts. FVT } a) \rangle$

| $\langle \text{FV} (\varphi \longrightarrow \psi) = \text{FV } \varphi \cup \text{FV } \psi \rangle$

| $\langle \text{FV} (\forall x. \varphi) = \text{FV } \varphi - \{x\} \rangle$

lemma *FV-all-subst*: $\langle \text{FV } \varphi \subseteq \text{FV} (\forall x. \varphi) \cup \{x\} \rangle$

by *fastforce*

lemma *FV-exists*: $\langle \text{FV} (\exists x. \varphi) = \text{FV } \varphi - \{x\} \rangle$

by *simp*

lemma *finite-FV*: $\langle \text{finite} (\text{FV } \varphi) \rangle$

by (*induction* φ , *auto*)

fun *BV* :: $\langle \text{form} \Rightarrow \text{nat set} \rangle$ **where**

$\langle \text{BV} \perp = \{\} \rangle$

| $\langle \text{BV} (\text{Atom} - \text{args}') = \{\} \rangle$

| $\langle \text{BV} (\varphi \longrightarrow \psi) = \text{BV } \varphi \cup \text{BV } \psi \rangle$

| $\langle \text{BV} (\forall x. \varphi) = \text{BV } \varphi \cup \{x\} \rangle$

lemma *finite-BV*: $\langle \text{finite} (\text{BV } \varphi) \rangle$

by (*induction* φ , *auto*)

definition *variant* :: $\langle \text{nat set} \Rightarrow \text{nat} \rangle$ **where**

$\langle \text{variant } s = \text{Max } s + 1 \rangle$

lemma *variant-finite*: $\langle \text{finite } s \Longrightarrow \neg (\text{variant } s \in s) \rangle$

unfolding *variant-def* **using** *Max-ge less-le-not-le* **by** *auto*

lemma *variant-form*: $\langle \neg \text{variant } (\text{FV } \varphi) \in \text{FV } \varphi \rangle$

using *variant-finite finite-FV* **by** *blast*

fun *formsubst* :: $\langle \text{form} \Rightarrow (\text{nat}, \text{nat}) \text{ subst} \Rightarrow \text{form} \rangle$ (**infixl** $\langle \cdot_{fm} \rangle$ 75) **where**

$\langle \perp \cdot_{fm} - = \perp \rangle$
 $| \langle (\text{Atom } p \ ts) \cdot_{fm} \sigma = \text{Atom } p \ [t \cdot \sigma. t \leftarrow ts] \rangle$
 $| \langle (\varphi \longrightarrow \psi) \cdot_{fm} \sigma = (\varphi \cdot_{fm} \sigma) \longrightarrow (\psi \cdot_{fm} \sigma) \rangle$
 $| \langle (\forall x. \varphi) \cdot_{fm} \sigma =$
 $\quad (\text{let } \sigma' = \sigma(x := \text{Var } x);$
 $\quad \quad z = \text{if } \exists y. y \in \text{FV } (\forall x. \varphi) \wedge x \in \text{FVT } (\sigma' y)$
 $\quad \quad \text{then variant } (\text{FV } (\varphi \cdot_{fm} \sigma')) \text{ else } x \text{ in}$
 $\quad \forall z. (\varphi \cdot_{fm} \sigma(x := \text{Var } z))) \rangle$

fun *formsubst2* :: $\langle \text{form} \Rightarrow (\text{nat}, \text{nat}) \text{ subst} \Rightarrow \text{form} \rangle$ (**infixl** $\langle \cdot_{fm2} \rangle$ 75) **where**

$\langle \perp \cdot_{fm2} - = \perp \rangle$
 $| \langle (\text{Atom } p \ ts) \cdot_{fm2} \sigma = \text{Atom } p \ [t \cdot \sigma. t \leftarrow ts] \rangle$
 $| \langle (\varphi \longrightarrow \psi) \cdot_{fm2} \sigma = (\varphi \cdot_{fm2} \sigma) \longrightarrow (\psi \cdot_{fm2} \sigma) \rangle$
 $| \langle (\forall x. \varphi) \cdot_{fm2} \sigma = (\text{let } \sigma' = \sigma(x := \text{Var } x) \text{ in}$
 $\quad (\text{if } \exists y. y \in \text{FV } (\forall x. \varphi) \wedge x \in \text{FVT } (\sigma' y)$
 $\quad \text{then } (\text{let } z = \text{variant } (\text{FV } (\varphi \cdot_{fm2} \sigma')) \text{ in}$
 $\quad \quad \forall z. (\varphi \cdot_{fm2} \sigma(x := \text{Var } z)))$
 $\quad \text{else } \forall x. (\varphi \cdot_{fm2} \sigma')) \rangle$

lemma *formsubst-def-switch*: $\langle \varphi \cdot_{fm} \sigma = \varphi \cdot_{fm2} \sigma \rangle$

proof (*induction* φ *arbitrary*; σ *rule*: *form.induct*)

case *Bot*

then show *?case*

by *fastforce*

next

case (*Atom* $x1$ $x2$)

then show *?case*

by *fastforce*

next

case (*Implies* $x1$ $x2$)

then show *?case*

by *fastforce*

next

case (*Forall* $x1$ $x2$)

then show *?case*

by (*smt* (*verit*, *best*) *formsubst.simps(4)* *formsubst2.simps(4)*)

qed

lemma *termsubst-valuation*: $\langle \forall x \in \text{FVT } t. \sigma \ x = \sigma' \ x \Longrightarrow t \cdot \sigma = t \cdot \sigma' \rangle$

```

using eval-same-vars by blast

lemma termsetsubst-valuation:  $\langle \forall y \in T. \forall x \in FVT\ y. \sigma\ x = \sigma'\ x \implies t \in T \implies t \cdot \sigma = t \cdot \sigma' \rangle$ 
using termsubst-valuation by fast

lemma formsubst-valuation:  $\langle \forall x \in (FV\ \varphi). (Var\ x) \cdot \sigma = (Var\ x) \cdot \sigma' \implies \varphi \cdot_{fm}\ \sigma = \varphi \cdot_{fm}\ \sigma' \rangle$ 
proof (induction  $\varphi$  arbitrary:  $\sigma\ \sigma'$  rule:form.induct)
  case Bot
  then show ?case by simp
next
  case (Atom  $x1\ x2$ )
  then show ?case
    using termsetsubst-valuation
    by auto
next
  case (Implies  $x1\ x2$ )
  then show ?case by simp
next
  case (Forall  $x\ \varphi$ )
  define  $\sigma''$  where  $\sigma'' = \sigma(x := Var\ x)$ 
  define  $\sigma'''$  where  $\sigma''' = \sigma'(x := Var\ x)$ 
  have ex-var-equiv:  $\langle \exists y. y \in FV\ (\forall x. \varphi) \wedge x \in FVT\ (\sigma''\ y) \equiv \exists y. y \in FV\ (\forall x. \varphi) \wedge x \in FVT\ (\sigma'''\ y) \rangle$ 
    using  $\sigma''$ -def  $\sigma'''$ -def Forall(2)
    by (smt (verit, ccfv-threshold) eval-term.simps(1) fun-upd-other fun-upd-same)
  have sig-x-subst:  $\langle \forall y \in FV\ \varphi. Var\ y \cdot \sigma(x := Var\ z) = Var\ y \cdot \sigma'(x := Var\ z) \rangle$ 
for  $z$ 
    using Forall(2) by simp
  show ?case
  proof (cases  $\langle \exists y. y \in FV\ (\forall x. \varphi) \wedge x \in FVT\ (\sigma''\ y) \rangle$ )
    case True
    then have  $\langle (\forall x. \varphi) \cdot_{fm}\ \sigma = (let\ z = variant\ (FV\ (\varphi \cdot_{fm}\ \sigma''))\ in\ \forall z. (\varphi \cdot_{fm}\ \sigma(x := Var\ z))) \rangle$ 
      by (simp add:  $\sigma''$ -def)
    also have  $\langle \dots = (let\ z = variant\ (FV\ (\varphi \cdot_{fm}\ \sigma'''))\ in\ \forall z. (\varphi \cdot_{fm}\ \sigma'(x := Var\ z))) \rangle$ 
      using sig-x-subst  $\sigma'''$ -def by (metis Forall.IH  $\sigma''$ -def)
    also have  $\langle \dots = (\forall x. \varphi) \cdot_{fm}\ \sigma' \rangle$ 
      using True  $\sigma'''$ -def ex-var-equiv formsubst.simps(4) by presburger
    finally show ?thesis .
  next
  case False
  then show ?thesis
    using Forall.IH  $\sigma'''$ -def  $\sigma''$ -def ex-var-equiv sig-x-subst by auto
qed
qed

```

lemma $\langle \{x. \exists y. y \in (s \cup t) \wedge P x y\} = \{x. \exists y. y \in s \wedge P x y\} \cup \{x. \exists y. y \in t \wedge P x y\} \rangle$
by *blast*

lemma *formsubst-structure-bot*: $\langle \varphi \cdot_{fm} \sigma = \perp \longleftrightarrow \varphi = \perp \rangle$
by (*smt* (*verit*) *form.distinct*(5) *form.simps*(5) *form.simps*(7) *formsubst.elims*)

lemma *formsubst-structure-pred*: $\langle (\exists p ts. \varphi \cdot_{fm} \sigma = Atom p ts) \longleftrightarrow (\exists p ts. \varphi = Atom p ts) \rangle$
proof (*cases* φ)
case (*Forall* $x \psi$)
then show *?thesis*
using *formsubst-def-switch* **by** (*metis* (*no-types*, *lifting*) *form.distinct*(10) *formsubst.simps*(4))
qed *auto*

lemma *formsubst-structure-imp*: $\langle (\exists \varphi 1 \varphi 2. \varphi \cdot_{fm} \sigma = \varphi 1 \longrightarrow \varphi 2) \longleftrightarrow (\exists \psi 1 \psi 2. \varphi = \psi 1 \longrightarrow \psi 2) \rangle$
proof (*cases* φ)
case (*Forall* $x \psi$)
then show *?thesis*
using *formsubst-def-switch*
by (*metis* (*no-types*, *lifting*) *form.distinct*(11) *formsubst.simps*(4))
qed *auto*

lemma *formsubst-structure-all*: $\langle (\exists x \psi. \varphi \cdot_{fm} \sigma = (\forall x. \psi)) \longleftrightarrow (\exists x \psi. \varphi = (\forall x. \psi)) \rangle$
proof (*cases* φ)
case (*Forall* $x \psi$)
then show *?thesis*
using *formsubst-def-switch*
by (*metis* (*no-types*, *lifting*) *formsubst.simps*(4))
qed *auto*

lemma *formsubst-structure-not*: $\langle (\exists \psi. \varphi \cdot_{fm} \sigma = Not \psi) \longleftrightarrow (\exists \psi. \varphi = Not \psi) \rangle$
using *formsubst-structure-imp* *formsubst-structure-bot*
by (*metis* *form.sel*(4) *formsubst.simps*(3))

lemma *formsubst-structure-not-all-imp*:
 $\langle (\exists x \psi. \varphi \cdot_{fm} \sigma = (\forall x. \psi) \longrightarrow \perp) \longleftrightarrow (\exists x \psi. \varphi = (\forall x. \psi) \longrightarrow \perp) \rangle$
proof (*cases* φ)
case *Bot*
then show *?thesis* **by** *simp*
next
case (*Atom* $p ts$)
then show *?thesis* **by** *simp*
next
case (*Implies* $\varphi 1 \varphi 2$)

then show *?thesis*
by (*metis form.inject(2) formsubst.simps(3) formsubst-structure-all formsubst-structure-not*)
next
case (*Forall y ψ*)
then show *?thesis*
by (*metis form.distinct(11) formsubst-structure-not*)
qed

lemma *formsubst-structure-all-not:*

$\langle (\exists x \psi. \varphi \cdot_{fm} \sigma = (\forall x. \psi \longrightarrow \perp)) \longleftrightarrow (\exists x \psi. \varphi = (\forall x. \psi \longrightarrow \perp)) \rangle$

proof

show $\langle \exists x \psi. \varphi = \forall x. \psi \longrightarrow \perp \implies \exists x \psi. \varphi \cdot_{fm} \sigma = \forall x. \psi \longrightarrow \perp \rangle$

by (*smt (verit, ccfv-threshold) formsubst.simps(1) formsubst.simps(3) form-subst.simps(4)*)

next

assume $\langle \exists x \psi. \varphi \cdot_{fm} \sigma = \forall x. \psi \longrightarrow \perp \rangle$

then obtain $z \psi'$ **where** *phi-sub-is:* $\langle \varphi \cdot_{fm} \sigma = \forall z. \psi' \longrightarrow \perp \rangle$

by *blast*

then obtain $x \varphi'$ **where** *phi-is:* $\langle \varphi = \forall x. \varphi' \rangle$

using *formsubst-structure-all* **by** *blast*

then have $\langle \exists \sigma'. \varphi' \cdot_{fm} \sigma' = \psi' \longrightarrow \perp \rangle$

using *phi-sub-is*

by (*metis (no-types, lifting) form.sel(6) formsubst.simps(4)*)

then obtain σ' **where** $\langle \varphi' \cdot_{fm} \sigma' = \psi' \longrightarrow \perp \rangle$

by *blast*

then have $\langle \exists \psi. \varphi' = \psi \longrightarrow \perp \rangle$

using *formsubst-structure-imp formsubst-structure-not* **by** *blast*

then show $\langle \exists x \psi. \varphi = \forall x. \psi \longrightarrow \perp \rangle$

using *phi-is* **by** *blast*

qed

lemma *formsubst-structure-ex:* $\langle (\exists x \psi. \varphi \cdot_{fm} \sigma = (\exists x. \psi)) \longleftrightarrow (\exists x \psi. \varphi = (\exists x. \psi)) \rangle$

proof

assume $\langle \exists x \psi. \varphi \cdot_{fm} \sigma = (\exists x. \psi) \rangle$

then show $\langle \exists x \psi. \varphi = (\exists x. \psi) \rangle$

by (*metis form.inject(2) formsubst.simps(3) formsubst-structure-all-not form-subst-structure-not*)

next

assume $\langle (\exists x \psi. \varphi = (\exists x. \psi)) \rangle$

then show $\langle \exists x \psi. \varphi \cdot_{fm} \sigma = (\exists x. \psi) \rangle$

by (*smt (verit, ccfv-threshold) formsubst.simps(1) formsubst.simps(3) form-subst.simps(4)*)

qed

lemma *formsubst-structure:* $\langle (\varphi \cdot_{fm} \sigma = \perp \longleftrightarrow \varphi = \perp) \wedge$

$((\exists p \ ts. \varphi \cdot_{fm} \sigma = \text{Atom } p \ ts) \longleftrightarrow (\exists p \ ts. \varphi = \text{Atom } p \ ts)) \wedge$

$((\exists \varphi 1 \ \varphi 2. \varphi \cdot_{fm} \sigma = \varphi 1 \longrightarrow \varphi 2) \longleftrightarrow (\exists \psi 1 \ \psi 2. \varphi = \psi 1 \longrightarrow \psi 2)) \wedge$

$((\exists x \psi. \varphi \cdot_{fm} \sigma = (\forall x. \psi)) \longleftrightarrow (\exists x \psi. \varphi = (\forall x. \psi))) \rangle$

```

using formsubst-structure-bot formsubst-structure-pred formsubst-structure-imp
formsubst-structure-all
by auto

lemma formsubst-fv:  $\langle FV (\varphi \cdot_{fm} \sigma) = \{x. \exists y. y \in (FV \varphi) \wedge x \in FVT ((Var y) \cdot \sigma)\} \rangle$ 
proof (induction  $\varphi$  arbitrary:  $\sigma$  rule:form.induct)
case (Atom  $x1\ x2$ )
have  $\langle FV (Atom\ x1\ x2 \cdot_{fm} \sigma) = (\bigcup a \in set\ x2. FVT (a \cdot \sigma)) \rangle$ 
by auto
also have  $\langle \dots = \{x. \exists y. y \in (\bigcup a \in set\ x2. FVT a) \wedge x \in FVT ((Var y) \cdot \sigma)\} \rangle$ 
proof
show  $\langle (\bigcup a \in set\ x2. FVT (a \cdot \sigma)) \subseteq \{x. \exists y. y \in (\bigcup a \in set\ x2. FVT a) \wedge x \in FVT (Var\ y \cdot \sigma)\} \rangle$ 
proof
fix  $v$ 
assume  $\langle v \in (\bigcup a \in set\ x2. FVT (a \cdot \sigma)) \rangle$ 
then obtain  $a$  where  $a$ -is:  $\langle a \in set\ x2 \rangle \langle v \in FVT (a \cdot \sigma) \rangle$ 
by auto
then obtain  $ya$  where  $\langle ya \in FVT a \rangle \langle v \in FVT (Var\ ya \cdot \sigma) \rangle$ 
using eval-term.simps(1) vars-term-subst-apply-term by force
then show  $\langle v \in \{x. \exists y. y \in (\bigcup a \in set\ x2. FVT a) \wedge x \in FVT (Var\ y \cdot \sigma)\} \rangle$ 
using  $a$ -is by auto
qed
next
show  $\langle \{x. \exists y. y \in (\bigcup a \in set\ x2. FVT a) \wedge x \in FVT (Var\ y \cdot \sigma)\} \subseteq (\bigcup a \in set\ x2. FVT (a \cdot \sigma)) \rangle$ 
proof
fix  $v$ 
assume  $\langle v \in \{x. \exists y. y \in (\bigcup a \in set\ x2. FVT a) \wedge x \in FVT (Var\ y \cdot \sigma)\} \rangle$ 
then obtain  $yv$  where  $\langle yv \in (\bigcup a \in set\ x2. FVT a) \rangle \langle v \in FVT (Var\ yv \cdot \sigma) \rangle$ 
by auto
then show  $\langle v \in (\bigcup a \in set\ x2. FVT (a \cdot \sigma)) \rangle$ 
using vars-term-subst-apply-term by fastforce
qed
qed
also have  $\langle \dots = \{x. \exists y. y \in (FV (Atom\ x1\ x2)) \wedge x \in FVT ((Var\ y) \cdot \sigma)\} \rangle$ 
by auto
finally show ?case .
next
case (Forall  $x\ \varphi$ )
define  $\sigma'$  where  $\sigma' = \sigma(x := Var\ x)$ 
show ?case
proof (cases  $\exists y. y \in FV (\forall x. \varphi) \wedge x \in FVT (\sigma' y)$ )
case True
then obtain  $y$  where  $y$ -in:  $\langle y \in FV (\forall x. \varphi) \rangle$  and  $x$ -in:  $\langle x \in FVT (\sigma' y) \rangle$ 
by blast
then have  $y$ -neg- $x$ :  $\langle y \neq x \rangle$  by simp
then have  $y$ -in2:  $\langle y \in FV \varphi \rangle$ 

```

```

using y-in by fastforce
have x-in2:  $\langle x \in FVT (Var\ y \cdot \sigma) \rangle$ 
using x-in y-neq-x unfolding  $\sigma'$ -def by simp

define z where z = variant ( $FV (\varphi \cdot_{fm} \sigma')$ )
have x-in3:  $\langle x \in FVT (Var\ y \cdot \sigma(x := Var\ z)) \rangle$ 
using x-in2 y-neq-x by simp

have  $\langle (\forall\ x.\ \varphi) \cdot_{fm} \sigma = \forall\ z.\ (\varphi \cdot_{fm} \sigma(x := Var\ z)) \rangle$ 
using z-def formsubst-def-switch
by (smt (verit, ccfv-threshold) True  $\sigma'$ -def formsubst.simps(4))
then have  $\langle FV ((\forall\ x.\ \varphi) \cdot_{fm} \sigma) = \{xa.\ \exists y.\ y \in FV\ \varphi \wedge xa \in FVT (Var\ y \cdot \sigma(x := Var\ z))\} - \{z\} \rangle$ 
using Forall[ $\sigma(x := Var\ z)$ ] using  $FV.simps(4)$  by presburger
also have  $\langle \dots = \{xa.\ \exists y.\ y \in FV\ \varphi - \{x\} \wedge xa \in FVT (Var\ y \cdot \sigma) \} \rangle$ 
proof
show  $\langle \{xa.\ \exists y.\ y \in FV\ \varphi \wedge xa \in FVT (Var\ y \cdot \sigma(x := Var\ z))\} - \{z\} \subseteq \{xa.\ \exists y.\ y \in FV\ \varphi - \{x\} \wedge xa \in FVT (Var\ y \cdot \sigma) \} \rangle$ 
proof
fix xa
assume xa-in:  $\langle xa \in \{xa.\ \exists y.\ y \in FV\ \varphi \wedge xa \in FVT (Var\ y \cdot \sigma(x := Var\ z))\} - \{z\} \rangle$ 
then obtain ya where ya-in:  $\langle ya \in FV\ \varphi \rangle$  and xa-image:  $\langle xa \in FVT (Var\ y \cdot \sigma(x := Var\ z)) \rangle$ 
by blast
have ya-neq-x:  $\langle ya \neq x \rangle$  using xa-image xa-in by fastforce
then have  $\langle xa \in FVT (Var\ ya \cdot \sigma) \rangle$  using xa-image by simp
moreover have  $\langle ya \in FV\ \varphi - \{x\} \rangle$ 
using ya-neq-x ya-in by blast
ultimately show  $\langle xa \in \{xa.\ \exists y.\ y \in FV\ \varphi - \{x\} \wedge xa \in FVT (Var\ y \cdot \sigma) \} \rangle$ 
by auto
qed
next
show  $\langle \{xa.\ \exists y.\ y \in FV\ \varphi - \{x\} \wedge xa \in FVT (Var\ y \cdot \sigma) \} \subseteq \{xa.\ \exists y.\ y \in FV\ \varphi \wedge xa \in FVT (Var\ y \cdot \sigma(x := Var\ z))\} - \{z\} \rangle$ 
proof
fix xa
assume xa-in:  $\langle xa \in \{xa.\ \exists y.\ y \in FV\ \varphi - \{x\} \wedge xa \in FVT (Var\ y \cdot \sigma) \} \rangle$ 
then obtain ya where ya-in:  $\langle ya \in FV\ \varphi - \{x\} \rangle$  and xa-image:  $\langle xa \in FVT (Var\ ya \cdot \sigma) \rangle$ 
by blast
have ya-neq:  $\langle ya \neq x \rangle$  using ya-in by blast
then have xa-in2:  $\langle xa \in FVT (Var\ ya \cdot \sigma(x := Var\ z)) \rangle$  using xa-image
by simp
then have  $\langle xa \in FV (\varphi \cdot_{fm} \sigma(x := Var\ z)) \rangle$ 
using ya-in Forall by force
then have  $\langle xa \in FV (\varphi \cdot_{fm} \sigma') \rangle$ 
using ya-neq Forall xa-image ya-in unfolding  $\sigma'$ -def by auto

```

then have $\langle xa \neq z \rangle$ **using** *z-def unfolding variant-def*
by (*metis Max-ge Suc-eq-plus1 finite-FV lessI less-le-not-le*)
moreover have $\langle ya \in FV \varphi \rangle$ **using** *ya-in* **by** *blast*
ultimately show $\langle xa \in \{xa. \exists y. y \in FV \varphi \wedge xa \in FVT (Var y \cdot \sigma(x := Var z))\} - \{z\} \rangle$
using *xa-in2* **by** *blast*
qed
qed
finally show *?thesis* **by** *simp*
next
case *False*
then have $\langle (\forall x. \varphi) \cdot_{fm} \sigma = \forall x. (\varphi \cdot_{fm} \sigma') \rangle$
using *formsubst-def-switch σ' -def* **by** *fastforce*
then have $\langle FV ((\forall x. \varphi) \cdot_{fm} \sigma) = \{z. \exists y. y \in FV \varphi \wedge z \in FVT (Var y \cdot \sigma')\} - \{x\} \rangle$
using *Forall* **by** *simp*
also have $\langle \dots = \{z. \exists y. y \in FV (\forall x. \varphi) \wedge z \in FVT (Var y \cdot \sigma)\} \rangle$
using *False unfolding σ' -def* **by** *auto*
finally show *?thesis* .
qed
qed *auto*

lemma *subst-var [simp]*: $\langle \varphi \cdot_{fm} Var = \varphi \rangle$
by (*induction φ*) *auto*

lemma *formsubst-rewrite*: $\langle FV (\varphi \cdot_{fm} (subst x (Var y))) - \{y\} = FV \varphi - \{x\} - \{y\} \rangle$

proof (*cases y = x*)

case *True*

then have $\langle subst x (Var y) = Var \rangle$

by *simp*

then have $\langle FV (\varphi \cdot_{fm} (subst x (Var y))) = FV \varphi \rangle$

using *subst-var* **by** *metis*

then show *?thesis*

using *True* **by** *simp*

next

case *False*

show *?thesis*

proof

show $\langle FV (\varphi \cdot_{fm} subst x (Var y)) - \{y\} \subseteq FV \varphi - \{x\} - \{y\} \rangle$

proof

fix *v*

assume *v-in*: $\langle v \in FV (\varphi \cdot_{fm} subst x (Var y)) - \{y\} \rangle$

moreover have $\langle v \neq x \rangle$

using *v-in*

by (*smt (verit, ccfv-threshold) DiffE Term.term.simps(17) eval-term.simps(1)*)

formsubst-fv fun-upd-other insert-iff mem-Collect-eq subst-def subst-simps(1))

moreover have $\langle v \in FV \varphi \rangle$

by (*smt (verit, del-Insts) DiffE Term.term.simps(17) eval-term.simps(1)*)

```

formsubst-fv
  fun-upd-other mem-Collect-eq subst-def subst-simps(1) subst-var v-in)
  ultimately show ⟨v ∈ FV φ - {x} - {y}⟩
    by blast
qed
next
show ⟨FV φ - {x} - {y} ⊆ FV (φ ·fm subst x (Var y)) - {y}⟩
using formsubst-fv False by (smt (verit, del-insts) Diff-iff Term.term.simps(17))

mem-Collect-eq singleton-iff subsetI subst-ident)
qed
qed

lemma termsubst-functions-term:
  ⟨functions-term (t · σ) = functions-term t ∪ {x. ∃ y. y ∈ FVT t ∧ x ∈ func-
tions-term ((Var y) · σ)}⟩
  by (induction t arbitrary: σ) auto

lemma formsubst-functions-form:
  ⟨functions-form (φ ·fm σ) = functions-form φ ∪ {x. ∃ y. y ∈ FV φ ∧ x ∈
functions-term ((Var y) · σ)}⟩
proof (induction φ arbitrary: σ)
  case Bot
  then show ?case by simp
next
  case (Atom p ts)
  show ?case
    using termsubst-functions-term by auto
next
  case (Implies φ ψ)
  then show ?case by auto
next
  case (Forall x φ)
  define σ' where ⟨σ' = σ(x := Var x)⟩
  define z where ⟨z = variant (FV (φ ·fm2 σ'))⟩
  have fun-terms-set-eq: ⟨{xa. ∃ y. y ∈ FV (∀ x. φ) ∧ xa ∈ functions-term (Var
y · σ)} =
  (if (∃ y. y ∈ FV (∀ x. φ) ∧ x ∈ FVT (σ' y))
  then {xa. ∃ y. y ∈ FV φ ∧ xa ∈ functions-term ((Var y) · σ(x := Var z))}
  else {x. ∃ y. y ∈ FV φ ∧ x ∈ functions-term ((Var y) · σ')}⟩ (is ?lhs = ?rhs)
proof
  show ⟨?lhs ⊆ ?rhs⟩
proof
  fix v
  assume v-in: v ∈ ?lhs
  have ⟨∃ y. y ∈ FV (∀ x. φ) ∧ x ∈ FVT (σ' y) ⟹ v ∈ {xa. ∃ y. y ∈ FV φ
∧ xa ∈ functions-term ((Var y) · σ(x := Var z))}⟩
  using v-in by auto
  moreover have ⟨∃ y. y ∈ FV (∀ x. φ) ∧ x ∈ FVT (σ' y) ⟹ v ∈ {x. ∃ y. y

```

```

∈ FV  $\varphi \wedge x \in \text{functions-term } ((\text{Var } y) \cdot \sigma')$ 
  using v-in  $\sigma'$ -def by auto
  ultimately show  $v \in ?rhs$ 
    by argo
  qed
next
show  $\langle ?rhs \subseteq ?lhs \rangle$ 
proof
  fix v
  assume v-in:  $\langle v \in ?rhs \rangle$ 
  have  $\langle \exists y. y \in FV (\forall x. \varphi) \wedge x \in FVT (\sigma' y) \implies v \in ?lhs \rangle$ 
    using v-in by (smt (verit, del-Insts) Diff-empty Diff-insert0 FV.simps(4)
      Term.term.simps(17) empty-iff eval-term.simps(1) eval-with-fresh-var
fun-upd-same
      funas-term.simps(1) insertE insert-Diff mem-Collect-eq)
  moreover have  $\langle \nexists y. y \in FV (\forall x. \varphi) \wedge x \in FVT (\sigma' y) \implies v \in ?lhs \rangle$ 
    using v-in by (smt (verit, ccfv-threshold) Diff-iff FV.simps(4)  $\sigma'$ -def
empty-iff
      eval-term.simps(1) fun-upd-other fun-upd-same funas-term.simps(1)
insertE
      mem-Collect-eq)
  ultimately show  $\langle v \in ?lhs \rangle$ 
    by argo
  qed
qed
have  $\langle \text{functions-form } ((\forall x. \varphi) \cdot_{fm} \sigma) = \text{functions-form } ((\forall x. \varphi) \cdot_{fm2} \sigma) \rangle$ 
  using formsubst-def-switch by simp
also have  $\langle \dots = (\text{if } (\exists y. y \in FV (\forall x. \varphi) \wedge x \in FVT (\sigma' y))$ 
  then  $\text{functions-form } (\forall z. (\varphi \cdot_{fm2} \sigma(x := \text{Var } z)))$ 
  else  $\text{functions-form } (\forall x. (\varphi \cdot_{fm2} \sigma')) \rangle$ 
  using  $\sigma'$ -def z-def by (smt (verit) formsubst2.simps(4))
also have  $\langle \dots = (\text{if } (\exists y. y \in FV (\forall x. \varphi) \wedge x \in FVT (\sigma' y))$ 
  then  $\text{functions-form } \varphi \cup \{xa. \exists y. y \in FV \varphi \wedge xa \in \text{functions-term } ((\text{Var } y) \cdot$ 
 $\sigma(x := \text{Var } z))\}$ 
  else  $\text{functions-form } \varphi \cup \{x. \exists y. y \in FV \varphi \wedge x \in \text{functions-term } ((\text{Var } y) \cdot$ 
 $\sigma')\} \rangle$ 
  using formsubst-def-switch Forall by auto
also have  $\langle \dots = \text{functions-form } \varphi \cup (\text{if } (\exists y. y \in FV (\forall x. \varphi) \wedge x \in FVT (\sigma'$ 
 $y))$ 
  then  $\{xa. \exists y. y \in FV \varphi \wedge xa \in \text{functions-term } ((\text{Var } y) \cdot \sigma(x := \text{Var } z))\}$ 
  else  $\{x. \exists y. y \in FV \varphi \wedge x \in \text{functions-term } ((\text{Var } y) \cdot \sigma')\} \rangle$ 
  by auto
  finally show ?case
    using fun-terms-set-eq by auto
qed

lemma formsubst-predicates:  $\langle \text{predicates-form } (\varphi \cdot_{fm} \sigma) = \text{predicates-form } \varphi \rangle$ 
proof (induction  $\varphi$  arbitrary:  $\sigma$  rule: predicates-form.induct)
  case ( $\lambda x \varphi$ )

```

```

then show ?case
  by (metis (no-types, lifting) formsubst.simps(4) predicates-form.simps(4))
qed auto

lemma formsubst-language-rewrite: ⟨language { $\varphi \cdot_{f_m} \text{subst } x (\text{Var } y)$ }⟩ = language { $\varphi$ }⟩
  using lang-singleton formsubst-predicates formsubst-functions-form by (simp add: subst-def)

end

theory FOL-Semantics
  imports FOL-Syntax
begin

locale struct =
  fixes
     $M :: \langle 'm \text{ set} \rangle$  and
     $FN :: \langle \text{nat} \Rightarrow 'm \text{ list} \Rightarrow 'm \rangle$  and
     $REL :: \langle \text{nat} \Rightarrow 'm \text{ list set} \rangle$ 
  assumes
     $M\text{-nonempty}: \langle M \neq \{\} \rangle$ 

typedef 'm intrp =
  ⟨{  $(M :: 'm \text{ set}, FN :: \text{nat} \Rightarrow 'm \text{ list} \Rightarrow 'm, REL :: \text{nat} \Rightarrow 'm \text{ list set}). \text{struct } M$ ⟩
  using struct.intro
  by fastforce

declare Abs-intrp-inverse [simp] Rep-intrp-inverse [simp]

setup-lifting type-definition-intrp

lift-definition dom :: ⟨'m intrp  $\Rightarrow$  'm set⟩ is fst .
lift-definition intrp-fn :: ⟨'m intrp  $\Rightarrow$  (nat  $\Rightarrow$  'm list  $\Rightarrow$  'm)⟩ is ⟨fst  $\circ$  snd⟩ .
lift-definition intrp-rel :: ⟨'m intrp  $\Rightarrow$  (nat  $\Rightarrow$  'm list set)⟩ is ⟨snd  $\circ$  snd⟩ .

lemma intrp-is-struct [iff]: ⟨struct (dom  $\mathcal{M}$ )⟩
  by transfer auto

lemma dom-Abs-is-fst [simp]: ⟨struct  $M \Longrightarrow$  dom (Abs-intrp ( $M, FN, REL$ )) =  $M$ ⟩
  by (simp add: dom.rep-eq)

lemma intrp-fn-Abs-is-fst-snd [simp]: ⟨struct  $M \Longrightarrow$  intrp-fn (Abs-intrp ( $M, FN, REL$ )) =  $FN$ ⟩
  by (simp add: intrp-fn.rep-eq)

lemma intrp-rel-Abs-is-snd-snd [simp]:

```

$\langle \text{struct } M \implies \text{intrp-rel } (\text{Abs-intrp } (M, FN, REL)) = REL \rangle$
by (*simp add: intrp-rel.rep-eq*)

definition *is-valuation* :: $\langle 'm \text{ intrp} \Rightarrow (\text{nat} \Rightarrow 'm) \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{is-valuation } \mathcal{M} \beta \longleftrightarrow (\forall v. \beta v \in \text{dom } \mathcal{M}) \rangle$

lemma *valuation-valmod*: $\langle \llbracket \text{is-valuation } \mathcal{M} \beta; a \in \text{dom } \mathcal{M} \rrbracket \implies \text{is-valuation } \mathcal{M} (\beta(x := a)) \rangle$

by (*simp add: is-valuation-def*)

fun *eval*

$:: \langle \text{nterm} \Rightarrow 'm \text{ intrp} \Rightarrow (\text{nat} \Rightarrow 'm) \Rightarrow 'm \rangle$

$\langle \llbracket \cdot \rrbracket \cdot \rangle [50, 0, 0] 70) \text{ where}$

$\langle \llbracket \text{Var } v \rrbracket \cdot, \beta = \beta v \rangle$

$| \langle \llbracket \text{Fun } f \text{ ts} \rrbracket \cdot, \beta = \text{intrp-fn } \mathcal{M} f \llbracket \llbracket t \rrbracket \cdot \rrbracket \cdot, \beta. t \leftarrow \text{ts} \rangle$

definition *list-all* :: $\langle 'a \Rightarrow \text{bool} \rangle \Rightarrow 'a \text{ list} \Rightarrow \text{bool} \rangle$ **where**

$\llbracket \text{simp} \rrbracket: \langle \text{list-all } P \text{ ls} \longleftrightarrow (\text{fold } (\lambda l b. b \wedge P l) \text{ ls } \text{True}) \rangle$

lemma *term-subst-eval*: $\langle \text{intrp-fn } M = \text{Fun} \implies t \cdot v = \text{eval } t M v \rangle$

by (*induction t*) *auto*

lemma *term-eval-triv* $\llbracket \text{simp} \rrbracket: \langle \text{intrp-fn } M = \text{Fun} \implies \text{eval } t M \text{Var} = t \rangle$

by (*metis subst-apply-term-empty term-subst-eval*)

lemma *fold-bool-prop*: $\langle (\text{fold } (\lambda l b. b \wedge P l) \text{ ls } b) = (b \wedge (\forall l \in \text{set } \text{ls}. P l)) \rangle$

by (*induction ls arbitrary: b*) *auto*

lemma *list-all-set*: $\langle \text{list-all } P \text{ ls} = (\forall l \in \text{set } \text{ls}. P l) \rangle$

unfolding *list-all-def* **using** *fold-bool-prop* **by** *auto*

hide-const *lang*

definition *is-interpretation* **where**

$\langle \text{is-interpretation } \text{lang } \mathcal{M} \longleftrightarrow$

$((\forall f l. (f, \text{length}(l)) \in \text{fst } \text{lang} \wedge \text{set } l \subseteq \text{dom } \mathcal{M} \longrightarrow \text{intrp-fn } \mathcal{M} f l \in \text{dom } \mathcal{M})) \rangle$

lemma *interpretation-sublanguage*: $\langle \text{funs2} \subseteq \text{funs1} \implies \text{is-interpretation } (\text{funs1}, \text{pred1}) \mathcal{M}$

$\implies \text{is-interpretation } (\text{funs2}, \text{preds2}) \mathcal{M} \rangle$

unfolding *is-interpretation-def* **by** *auto*

lemma *interpretation-eval*:

assumes \mathcal{M} : $\langle \text{is-interpretation } (\text{functions-term } t, \text{any}) \mathcal{M} \rangle$ **and** *val*: $\langle \text{is-valuation}$

```

 $\mathcal{M} \beta$ 
shows  $\langle \llbracket t \rrbracket^{\mathcal{M}, \beta} \in \text{dom } \mathcal{M} \rangle$ 
using  $\mathcal{M}$ 
proof (induction t)
  case (Var x)
    with val show ?case
    by (simp add: is-valuation-def)
next
  case (Fun f ts)
  then have  $\langle \llbracket t \rrbracket^{\mathcal{M}, \beta} \in \text{dom } \mathcal{M} \rangle$  if  $\langle t \in \text{set } ts \rangle$  for  $t$ 
    by (meson interpretation-sublanguage supt.arg supt-imp-funas-term-subset that)
  then show ?case
    using Fun by (simp add: is-interpretation-def image-subsetI)
qed

```

```

fun holds
  ::  $\langle 'm \text{ intrp} \Rightarrow (\text{nat} \Rightarrow 'm) \Rightarrow \text{form} \Rightarrow \text{bool} \rangle$   $\langle \langle -, \models \rightarrow [30, 30, 80] 80 \rangle$  where
   $\langle \mathcal{M}, \beta \models \perp \longleftrightarrow \text{False} \rangle$ 
  |  $\langle \mathcal{M}, \beta \models \text{Atom } p \text{ ts} \longleftrightarrow \llbracket t \rrbracket^{\mathcal{M}, \beta}. t \leftarrow ts \in \text{intrp-rel } \mathcal{M} \ p \rangle$ 
  |  $\langle \mathcal{M}, \beta \models \varphi \longrightarrow \psi \longleftrightarrow ((\mathcal{M}, \beta \models \varphi) \longrightarrow (\mathcal{M}, \beta \models \psi)) \rangle$ 
  |  $\langle \mathcal{M}, \beta \models (\forall x. \varphi) \longleftrightarrow (\forall a \in \text{dom } \mathcal{M}. \mathcal{M}, \beta(x := a) \models \varphi) \rangle$ 

```

```

lemma holds-exists:  $\langle \mathcal{M}, \beta \models (\exists x. \varphi) \longleftrightarrow (\exists a \in \text{dom } \mathcal{M}. \mathcal{M}, \beta(x := a) \models \varphi) \rangle$ 
  by simp

```

```

lemma holds-indep-beta-if:
   $\langle \forall v \in FV \varphi. \beta_1 v = \beta_2 v \implies \mathcal{M}, \beta_1 \models \varphi \longleftrightarrow \mathcal{M}, \beta_2 \models \varphi \rangle$ 
proof (induction  $\varphi$  arbitrary:  $\beta_1 \beta_2$ )
  case Bot
  then show ?case
    by simp
next
  case (Atom p ts)
  then have  $\langle \forall t \in \text{set } ts. \forall v \in FVT \ t. \beta_1 v = \beta_2 v \rangle$ 
    by simp
  then have  $\langle \llbracket t \rrbracket^{\mathcal{M}, \beta_1}. t \leftarrow ts = \llbracket t \rrbracket^{\mathcal{M}, \beta_2}. t \leftarrow ts \rangle$ 
  proof (induction ts)
    case Nil
    then show ?case
      by simp
  next
  case (Cons a ts)
  then show ?case
  proof (induction a)
    case (Var x)
    then show ?case
      by simp
  next

```

```

case (Fun f ts')
then have  $\langle \forall t \in \text{set } ts'. \forall v \in FVT t. \beta_1 v = \beta_2 v \rangle$ 
  by simp
then have  $\langle [[t]]^{\mathcal{M}, \beta_1}. t \leftarrow ts^\wedge = [[t]]^{\mathcal{M}, \beta_2}. t \leftarrow ts^\wedge \rangle$ 
  using Cons.IH Fun.IH Fun.premis(2)
  by force
then have  $\langle \text{intrp-fn } \mathcal{M} f [[t]]^{\mathcal{M}, \beta_1}. t \leftarrow ts^\wedge = \text{intrp-fn } \mathcal{M} f [[t]]^{\mathcal{M}, \beta_2}. t \leftarrow$ 
 $ts^\wedge \rangle$ 
  by argo
then show ?case
  using Cons.IH Fun.premis(2)
  by force
qed
qed
then have  $\langle [[t]]^{\mathcal{M}, \beta_1}. t \leftarrow ts \in \text{intrp-rel } \mathcal{M} p \longleftrightarrow [[t]]^{\mathcal{M}, \beta_2}. t \leftarrow ts \in \text{intrp-rel}$ 
 $\mathcal{M} p \rangle$ 
  by argo
then show ?case
  by simp
next
case (Implies  $\varphi \psi$ )
then have
   $\langle \forall v \in FV \varphi. \beta_1 v = \beta_2 v \rangle$  and
   $\langle \forall v \in FV \psi. \beta_1 v = \beta_2 v \rangle$ 
  by auto
then have
   $\langle \mathcal{M}, \beta_1 \models \varphi \longleftrightarrow \mathcal{M}, \beta_2 \models \varphi \rangle$  and
   $\langle \mathcal{M}, \beta_1 \models \psi \longleftrightarrow \mathcal{M}, \beta_2 \models \psi \rangle$ 
  using Implies.IH by auto
then show ?case
  by simp
next
case (Forall x  $\varphi$ )
then have  $\langle \forall a \in \text{dom } \mathcal{M}. (\mathcal{M}, \beta_1(x := a) \models \varphi) = (\mathcal{M}, \beta_2(x := a) \models \varphi) \rangle$ 
  by simp
then show ?case
  by simp
qed

```

lemma *holds-indep-intrp-if:*

```

fixes
   $\varphi :: \text{form}$  and
   $\mathcal{M} \mathcal{M}' :: \langle 'm \text{ intrp} \rangle$ 
assumes
  dom-eq:  $\langle \text{dom } \mathcal{M} = \text{dom } \mathcal{M}' \rangle$  and
  rel-eq:  $\langle \forall p. \text{intrp-rel } \mathcal{M} p = \text{intrp-rel } \mathcal{M}' p \rangle$  and
  fn-eq:  $\langle \forall f ts. (f, \text{length } ts) \in \text{functions-form } \varphi \longrightarrow \text{intrp-fn } \mathcal{M} f ts = \text{intrp-fn}$ 
 $\mathcal{M}' f ts \rangle$ 
shows

```

$\langle \forall \beta. \mathcal{M}, \beta \models \varphi \longleftrightarrow \mathcal{M}', \beta \models \varphi \rangle$
using *fn-eq*
proof (*intro allI impI, induction φ*)
case (*Atom p ts*)

have *all-fn-sym-in*: $\langle \bigcup t \in \text{set } ts. \text{functions-term } t \subseteq \text{functions-form } (\text{Atom } p \text{ } ts) \rangle$ (*is $\langle ?A \subseteq - \rangle$*)
by *simp*

have *eval-tm-eq*: $\langle \llbracket t \rrbracket^{\mathcal{M}, \beta} = \llbracket t \rrbracket^{\mathcal{M}', \beta} \rangle$
if $\langle \text{functions-term } t \subseteq \text{functions-form } (\text{Atom } p \text{ } ts) \rangle$
for *t*
using *that*
proof (*induction t*)
case (*Fun f ts'*)

have $\langle \forall t' \in \text{set } ts'. \text{functions-term } t' \subseteq \text{functions-form } (\text{Atom } p \text{ } ts) \rangle$
using *Fun.prem*
by *auto*
moreover have $\langle (f, \text{length } \llbracket t \rrbracket^{\mathcal{M}, \beta}. t' \leftarrow ts') \in \text{functions-form } (\text{Atom } p \text{ } ts) \rangle$
using *Fun.prem*
by *fastforce*
ultimately show *?case*
using *Fun.IH Atom.prem(1)[rule-format, of f $\langle \llbracket t \rrbracket^{\mathcal{M}, \beta}. t' \leftarrow ts' \rangle$]*
by (*smt (verit, del-Insts) eval.simps(2) map-eq-conv*)
qed *auto*

have $\langle \mathcal{M}, \beta \models \text{Atom } p \text{ } ts \longleftrightarrow \llbracket t \rrbracket^{\mathcal{M}, \beta}. t \leftarrow ts \in \text{intrap-rel } \mathcal{M} \text{ } p \rangle$
by *simp*
also have $\langle \dots \longleftrightarrow \llbracket t \rrbracket^{\mathcal{M}, \beta}. t \leftarrow ts \in \text{intrap-rel } \mathcal{M}' \text{ } p \rangle$
by (*simp add: rel-eq*)
also have $\langle \dots \longleftrightarrow \llbracket t \rrbracket^{\mathcal{M}', \beta}. t \leftarrow ts \in \text{intrap-rel } \mathcal{M}' \text{ } p \rangle$
using *eval-tm-eq all-fn-sym-in*
by (*metis (mono-tags, lifting) UN-subset-iff map-eq-conv*)
also have $\langle \dots \longleftrightarrow \mathcal{M}', \beta \models \text{Atom } p \text{ } ts \rangle$
by *auto*
finally show *?case .*

next
case (*Forall x φ*)

have $\langle \mathcal{M}, \beta \models (\forall x. \varphi) \longleftrightarrow (\forall a \in \text{dom } \mathcal{M}. \mathcal{M}, \beta(x := a) \models \varphi) \rangle$
by *simp*
also have $\langle \dots = (\forall a \in \text{dom } \mathcal{M}. \mathcal{M}', \beta(x := a) \models \varphi) \rangle$
using *Forall.IH Forall.prem by simp*
also have $\langle \dots = (\forall a \in \text{dom } \mathcal{M}'. \mathcal{M}', \beta(x := a) \models \varphi) \rangle$
by (*simp add: dom-eq*)
also have $\langle \dots = (\mathcal{M}', \beta \models (\forall x. \varphi)) \rangle$
by *auto*
finally show *?case .*

qed *auto*

the above in a more idiomatic form (it is a congruence rule)

corollary *holds-cong*:

assumes

$\langle \text{dom } \mathcal{M} = \text{dom } \mathcal{M}' \rangle$

$\langle \bigwedge p. \text{intrp-rel } \mathcal{M} p = \text{intrp-rel } \mathcal{M}' p \rangle$

$\langle \bigwedge f ts. (f, \text{length } ts) \in \text{functions-form } \varphi \implies \text{intrp-fn } \mathcal{M} f ts = \text{intrp-fn } \mathcal{M}' f ts \rangle$

shows $\langle \mathcal{M}, \beta \models \varphi \longleftrightarrow \mathcal{M}', \beta \models \varphi \rangle$

using *assms holds-indep-intrp-if* **by** *blast*

abbreviation (*input*) $\langle \text{termsubst } \mathcal{M} \beta \sigma v \equiv \llbracket \sigma v \rrbracket^{\mathcal{M}, \beta} \rangle$

lemma *subst-lemma-terms*: $\langle \llbracket t \cdot \sigma \rrbracket^{\mathcal{M}, \beta} = \llbracket t \rrbracket^{\mathcal{M}, \text{termsubst } \mathcal{M} \beta \sigma} \rangle$

proof (*induction t*)

case (*Var v*)

then show *?case*

by *auto*

next

case (*Fun f ts*)

have $\langle \llbracket \text{Fun } f ts \cdot \sigma \rrbracket^{\mathcal{M}, \beta} = \llbracket \text{Fun } f [t \cdot \sigma. t \leftarrow ts] \rrbracket^{\mathcal{M}, \beta} \rangle$

by *auto*

also have $\langle \dots = \text{intrp-fn } \mathcal{M} f \llbracket t \rrbracket^{\mathcal{M}, \beta}. t \leftarrow [t \cdot \sigma. t \leftarrow ts] \rangle$

by *auto*

also have $\langle \dots = \text{intrp-fn } \mathcal{M} f \llbracket t \cdot \sigma \rrbracket^{\mathcal{M}, \beta}. t \leftarrow ts \rangle$

unfolding *map-map*

by (*meson comp-apply*)

also have $\langle \dots = \text{intrp-fn } \mathcal{M} f \llbracket t \rrbracket^{\mathcal{M}, \text{termsubst } \mathcal{M} \beta \sigma}. t \leftarrow ts \rangle$

using *Fun.IH*

by (*smt (verit, best) map-eq-conv*)

also have $\langle \dots = \llbracket \text{Fun } f ts \rrbracket^{\mathcal{M}, \text{termsubst } \mathcal{M} \beta \sigma} \rangle$

by *auto*

finally show *?case* .

qed

lemma *eval-indep-β-if*:

assumes $\langle \forall v \in FVT. \beta v = \beta' v \rangle$

shows $\langle \llbracket t \rrbracket^{\mathcal{M}, \beta} = \llbracket t \rrbracket^{\mathcal{M}, \beta'} \rangle$

using *assms*

proof (*induction t*)

case (*Var v*)

then show *?case*

by *auto*

next

case (*Fun f ts*)

then show *?case*

by (smt (verit, ccfv-SIG) eval.simps(2) map-eq-conv term.set-intros(4))
qed

lemma concat-map: $\langle [f t. t \leftarrow [g t. t \leftarrow ts]] = [f (g t). t \leftarrow ts] \rangle$ by simp

lemma swap-subst-eval: $\langle \mathcal{M}, \beta \models (\varphi \cdot_{fm} \sigma) \longleftrightarrow \mathcal{M}, (\lambda v. \text{termsubst } \mathcal{M} \beta \sigma v) \models \varphi \rangle$

proof (induction φ arbitrary: σ β)

case (Atom p ts)

have $\langle \mathcal{M}, \beta \models (\text{Atom } p \text{ ts} \cdot_{fm} \sigma) \longleftrightarrow \mathcal{M}, \beta \models (\text{Atom } p [t \cdot \sigma. t \leftarrow ts]) \rangle$

by auto

also have $\langle \dots \longleftrightarrow \llbracket [t] \rrbracket^{\mathcal{M}, \beta}. t \leftarrow [t \cdot \sigma. t \leftarrow ts] \in \text{intrap-rel } \mathcal{M} p \rangle$

by auto

also have $\langle \dots \longleftrightarrow \llbracket [t \cdot \sigma] \rrbracket^{\mathcal{M}, \beta}. t \leftarrow ts \in \text{intrap-rel } \mathcal{M} p \rangle$

using concat-map[of $\lambda t. \llbracket [t] \rrbracket^{\mathcal{M}, \beta} \lambda t. t \cdot \sigma$] by presburger

also have $\langle \dots \longleftrightarrow \llbracket [t] \rrbracket^{\mathcal{M}, \text{termsubst } \mathcal{M} \beta \sigma}. t \leftarrow ts \in \text{intrap-rel } \mathcal{M} p \rangle$

using subst-lemma-terms[of - σ $\mathcal{M} \beta$] by auto

finally show ?case

by simp

next

case (Forall x φ)

define σ' where $\sigma' = \sigma(x := \text{Var } x)$

show ?case

proof (cases $\langle \exists y. y \in FV (\forall x. \varphi) \wedge x \in FVT (\sigma' y) \rangle$)

case False

then have $\langle (\forall x. \varphi) \cdot_{fm} \sigma = \forall x. (\varphi \cdot_{fm} \sigma') \rangle$

using formsubst-def-switch σ' -def by fastforce

then have $\langle \mathcal{M}, \beta \models ((\forall x. \varphi) \cdot_{fm} \sigma) = (\forall a \in \text{dom } \mathcal{M}. \mathcal{M}, \beta(x := a) \models (\varphi \cdot_{fm} \sigma')) \rangle$

by auto

also have $\langle \dots = (\forall a \in \text{dom } \mathcal{M}. \mathcal{M}, (\lambda v. \llbracket [\sigma' v] \rrbracket^{\mathcal{M}, \beta}(x := a)) \models \varphi) \rangle$

using Forall by blast

also have $\langle \dots = (\forall a \in \text{dom } \mathcal{M}. \mathcal{M}, (\lambda v. \llbracket [\sigma' v] \rrbracket^{\mathcal{M}, \beta})(x := a) \models \varphi) \rangle$

proof

assume forward: $\langle \forall a \in \text{dom } \mathcal{M}. \mathcal{M}, (\lambda v. \llbracket [\sigma' v] \rrbracket^{\mathcal{M}, \beta}(x := a)) \models \varphi \rangle$

show $\langle \forall a \in \text{dom } \mathcal{M}. \mathcal{M}, (\lambda v. \llbracket [\sigma' v] \rrbracket^{\mathcal{M}, \beta})(x := a) \models \varphi \rangle$

proof

fix a

assume $\langle a \in \text{dom } \mathcal{M} \rangle$

then have $\langle \mathcal{M}, (\lambda v. \llbracket [\sigma' v] \rrbracket^{\mathcal{M}, \beta}(x := a)) \models \varphi \rangle$

using forward by blast

moreover have $\langle \forall v \in FV \varphi. (\lambda v. \llbracket [\sigma' v] \rrbracket^{\mathcal{M}, \beta}(x := a)) v = ((\lambda v. \llbracket [\sigma' v] \rrbracket^{\mathcal{M}, \beta})(x := a)) v \rangle$

proof

fix v

assume $\langle v \in FV \varphi \rangle$

then have $\langle v \neq x \implies \llbracket [\sigma' v] \rrbracket^{\mathcal{M}, \beta}(x := a) = ((\lambda v. \llbracket [\sigma' v] \rrbracket^{\mathcal{M}, \beta})(x := a)) v \rangle$

by (*metis (mono-tags, lifting) DiffI FV.simps(4) False eval-indep-β-if fun-upd-other singletonD*)
 moreover have $\langle v = x \implies \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta}(x := a) = ((\lambda v. \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta})(x := a)) v \rangle$
 using σ' -def by auto
 ultimately show $\langle \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta}(x := a) = ((\lambda v. \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta})(x := a)) v \rangle$
 by simp
 qed
 ultimately show $\langle \mathcal{M}, (\lambda v. \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta})(x := a) \models \varphi \rangle$
 using holds-indep-β-if by fast
 qed
 next
 assume backward: $\langle \forall a \in \text{dom } \mathcal{M}. \mathcal{M}, (\lambda v. \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta})(x := a) \models \varphi \rangle$
 show $\langle \forall a \in \text{dom } \mathcal{M}. \mathcal{M}, (\lambda v. \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta}(x := a)) \models \varphi \rangle$
 proof
 fix a
 assume $\langle a \in \text{dom } \mathcal{M} \rangle$
 then have $\langle \mathcal{M}, (\lambda v. \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta})(x := a) \models \varphi \rangle$
 using backward by blast
 moreover have $\langle \forall v \in FV \varphi. (\lambda v. \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta}(x := a)) v = ((\lambda v. \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta})(x := a)) v \rangle$
 proof
 fix v
 assume $\langle v \in FV \varphi \rangle$
 then have $\langle v \neq x \implies \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta}(x := a) = ((\lambda v. \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta})(x := a)) v \rangle$
 by (*metis (mono-tags, lifting) DiffI FV.simps(4) False eval-indep-β-if fun-upd-other singletonD*)
 moreover have $\langle v = x \implies \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta}(x := a) = ((\lambda v. \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta})(x := a)) v \rangle$
 using σ' -def by auto
 ultimately show $\langle \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta}(x := a) = ((\lambda v. \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta})(x := a)) v \rangle$
 by simp
 qed
 ultimately show $\langle \mathcal{M}, (\lambda v. \llbracket \sigma' v \rrbracket^{\mathcal{M}, \beta}(x := a)) \models \varphi \rangle$
 using holds-indep-β-if by fast
 qed
 qed
 also have $\langle \dots = (\mathcal{M}, (\lambda v. \llbracket \sigma v \rrbracket^{\mathcal{M}, \beta}) \models (\forall x. \varphi)) \rangle$
 by (*smt (verit, ccfv-SIG) σ' -def fun-upd-apply holds.simps(4) holds-indep-β-if*)
 finally show ?thesis .
 next
 case True
 then have x-ex: $\langle \exists y. y \in FV \varphi - \{x\} \wedge x \in FVT (\sigma' y) \rangle$
 by simp
 then have x-in: $\langle x \in FV (\varphi \cdot_{fm} \sigma') \rangle$
 using formsubst-fv
 by auto
 define z where $\langle z = \text{variant } (FV (\varphi \cdot_{fm} \sigma')) \rangle$

then have $\langle z \neq x \rangle$
using *x-in variant-form by auto*
have $\langle (\forall x. \varphi) \cdot_{fm} \sigma = \forall z. (\varphi \cdot_{fm} \sigma(x := Var z)) \rangle$
using *z-def True formsubst-def-switch σ' -def by (smt (verit, best) form-subst.simps(4))*
then have $\langle \mathcal{M}, \beta \models ((\forall x. \varphi) \cdot_{fm} \sigma) = (\forall a \in dom \mathcal{M}. \mathcal{M}, \beta(z := a) \models (\varphi \cdot_{fm} \sigma(x := Var z))) \rangle$
by auto
also have $\langle \dots = (\forall a \in dom \mathcal{M}. \mathcal{M}, (\lambda v. \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta}(z := a)) \models \varphi) \rangle$
using *Forall by blast*
also have $\langle \dots = (\forall a \in dom \mathcal{M}. \mathcal{M}, (\lambda v. \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta})(x := a) \models \varphi) \rangle$
proof
assume forward: $\langle \forall a \in dom \mathcal{M}. \mathcal{M}, (\lambda v. \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta}(z := a)) \models \varphi \rangle$
show $\langle \forall a \in dom \mathcal{M}. \mathcal{M}, (\lambda v. \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta})(x := a) \models \varphi \rangle$
proof
fix a
assume $\langle a \in dom \mathcal{M} \rangle$
then have $\langle \mathcal{M}, (\lambda v. \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta}(z := a)) \models \varphi \rangle$
using *forward by blast*
moreover have $\langle \forall v \in FV \varphi. (\lambda v. \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta}(z := a)) v = ((\lambda v. \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta})(x := a)) v \rangle$
proof
fix v
assume *v-in:* $\langle v \in FV \varphi \rangle$
then have $\langle v \neq x \implies z \notin FVT(\sigma v) \rangle$
using *z-def variant-form by (smt (verit, ccfv-threshold) σ' -def eval-term.simps(1) formsubst-fv fun-upd-other mem-Collect-eq)*
then have $\langle v \neq x \implies \llbracket \sigma v \rrbracket^{\mathcal{M}, \beta}(z := a) = \llbracket \sigma v \rrbracket^{\mathcal{M}, \beta} \rangle$
by *(simp add: eval-indep- β -if)*
then have $\langle v \neq x \implies (\lambda v. \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta}(z := a)) v = ((\lambda v. \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta})(x := a)) v \rangle$
by auto
moreover have $\langle v = x \implies (\lambda v. \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta}(z := a)) v = ((\lambda v. \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta})(x := a)) v \rangle$
by auto
ultimately show
 $\langle \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta}(z := a) = ((\lambda v. \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta})(x := a)) v \rangle$
by auto
qed
ultimately show $\langle \mathcal{M}, (\lambda v. \llbracket (\sigma(x := Var z)) v \rrbracket^{\mathcal{M}, \beta})(x := a) \models \varphi \rangle$
using *holds-indep- β -if by fast*

```

qed
next
  assume backward:  $\langle \forall a \in \text{dom } \mathcal{M}. \mathcal{M}, (\lambda v. \llbracket (\sigma(x := \text{Var } z)) v \rrbracket^{\mathcal{M}, \beta})(x := a) \models \varphi \rangle$ 
show  $\langle \forall a \in \text{dom } \mathcal{M}. \mathcal{M}, (\lambda v. \llbracket (\sigma(x := \text{Var } z)) v \rrbracket^{\mathcal{M}, \beta}(z := a)) \models \varphi \rangle$ 
proof
  fix a
  assume  $\langle a \in \text{dom } \mathcal{M} \rangle$ 
  then have  $\langle \mathcal{M}, (\lambda v. \llbracket (\sigma(x := \text{Var } z)) v \rrbracket^{\mathcal{M}, \beta})(x := a) \models \varphi \rangle$ 
    using backward by auto
  moreover have  $\langle \forall v \in \text{FV } \varphi. (\lambda v. \llbracket (\sigma(x := \text{Var } z)) v \rrbracket^{\mathcal{M}, \beta}(z := a)) v =$ 
     $((\lambda v. \llbracket (\sigma(x := \text{Var } z)) v \rrbracket^{\mathcal{M}, \beta})(x := a)) v \rangle$ 
  proof
    fix v
    assume v-in:  $\langle v \in \text{FV } \varphi \rangle$ 
    then have  $\langle v \neq x \implies z \notin \text{FVT } (\sigma v) \rangle$ 
    using z-def variant-form by (smt (verit, ccfv-threshold)  $\sigma'$ -def eval-term.simps(1)
        
formsubst-fv fun-upd-other mem-Collect-eq)
    then have  $\langle v \neq x \implies \llbracket \sigma v \rrbracket^{\mathcal{M}, \beta}(z := a) = \llbracket \sigma v \rrbracket^{\mathcal{M}, \beta}$ 
        
by (simp add: eval-indep- $\beta$ -if)
    then have  $\langle v \neq x \implies$ 
     $(\lambda v. \llbracket (\sigma(x := \text{Var } z)) v \rrbracket^{\mathcal{M}, \beta}(z := a)) v = ((\lambda v. \llbracket (\sigma(x := \text{Var } z)) v \rrbracket^{\mathcal{M}, \beta})(x$ 
     $:= a)) v \rangle$ 
    by auto
    moreover have  $\langle v = x \implies$ 
     $(\lambda v. \llbracket (\sigma(x := \text{Var } z)) v \rrbracket^{\mathcal{M}, \beta}(z := a)) v = ((\lambda v. \llbracket (\sigma(x := \text{Var } z)) v \rrbracket^{\mathcal{M}, \beta})(x$ 
     $:= a)) v \rangle$ 
    by auto
    ultimately show
     $\langle \llbracket (\sigma(x := \text{Var } z)) v \rrbracket^{\mathcal{M}, \beta}(z := a) = ((\lambda v. \llbracket (\sigma(x := \text{Var } z)) v \rrbracket^{\mathcal{M}, \beta})(x :=$ 
     $a)) v \rangle$ 
    by auto
  qed
  ultimately show  $\langle \mathcal{M}, (\lambda v. \llbracket (\sigma(x := \text{Var } z)) v \rrbracket^{\mathcal{M}, \beta}(z := a)) \models \varphi \rangle$ 
    using holds-indep- $\beta$ -if by fast
qed
qed
also have  $\langle \dots = (\forall a \in \text{dom } \mathcal{M}. \mathcal{M}, (\lambda v. \llbracket \sigma v \rrbracket^{\mathcal{M}, \beta})(x := a) \models \varphi) \rangle$ 
  by (smt (verit, ccfv-SIG) fun-upd-apply holds-indep- $\beta$ -if)
also have  $\langle \dots = (\mathcal{M}, (\lambda v. \llbracket \sigma v \rrbracket^{\mathcal{M}, \beta}) \models (\forall x. \varphi)) \rangle$ 
  by auto
finally show ?thesis .
qed
qed auto

```

definition *satisfies* :: $\langle 'm \text{ intrp} \Rightarrow \text{form set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{satisfies } \mathcal{M} S \equiv (\forall \beta \varphi. \text{is-valuation } \mathcal{M} \beta \longrightarrow \varphi \in S \longrightarrow \mathcal{M}, \beta \models \varphi) \rangle$

lemma *satisfies-iff-satisfies-sing*: $\langle \text{satisfies } M S \longleftrightarrow (\forall \varphi \in S. \text{satisfies } M \{\varphi\}) \rangle$
by (*auto simp: satisfies-def*)

end

theory *Ground-FOL-Compactness*

imports

FOL-Semantics

begin

fun *qfree* :: $\langle \text{form} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{qfree } \perp = \text{True} \rangle$
 $| \langle \text{qfree } (\text{Atom } p \text{ } ts) = \text{True} \rangle$
 $| \langle \text{qfree } (\varphi \longrightarrow \psi) = (\text{qfree } \varphi \wedge \text{qfree } \psi) \rangle$
 $| \langle \text{qfree } (\forall x. \varphi) = \text{False} \rangle$

lemma *qfree-iff-BV-empty*: $\text{qfree } \varphi \longleftrightarrow \text{BV } \varphi = \{\}$
by (*induction* φ *) auto*

lemma *qfree-no-quantif*: $\langle \text{qfree } r \Longrightarrow \neg(\exists x p. r = \forall x. p) \wedge \neg(\exists x p. r = \exists x. p) \rangle$
using *qfree.simps(3) qfree.simps(4)* **by** *blast*

lemma *qfree-formsubst*: $\langle \text{qfree } \varphi \equiv \text{qfree } (\varphi \cdot_{fm} \sigma) \rangle$

proof (*induction* φ *)*

case (*Forall* x φ *)*

then show *?case*

using *formsubst-def-switch* **by** (*metis (no-types, lifting) formsubst.simps(4)*)

qfree-no-quantif)

qed *simp+*

fun *form-to-formula* :: $\text{form} \Rightarrow (\text{nat} \times \text{nterm list}) \text{ formula}$ **where**

$\langle \text{form-to-formula } \perp = \perp \rangle$
 $| \langle \text{form-to-formula } (\text{Atom } p \text{ } ts) = \text{formula.Atom } (p, ts) \rangle$
 $| \langle \text{form-to-formula } (\varphi \longrightarrow \psi) = \text{Imp } (\text{form-to-formula } \varphi) (\text{form-to-formula } \psi) \rangle$
 $| \langle \text{form-to-formula } (\forall x. \varphi) = \perp \rangle$

fun *pholds* :: $\langle (\text{form} \Rightarrow \text{bool}) \Rightarrow \text{form} \Rightarrow \text{bool} \rangle$ (*-* \models_p *-> [30, 80] 80*) **where**

$\langle I \models_p \perp \longleftrightarrow \text{False} \rangle$
 $| \langle I \models_p \text{Atom } p \text{ } ts \longleftrightarrow I (\text{Atom } p \text{ } ts) \rangle$
 $| \langle I \models_p \varphi \longrightarrow \psi \longleftrightarrow ((I \models_p \varphi) \longrightarrow (I \models_p \psi)) \rangle$
 $| \langle I \models_p (\forall x. \varphi) \longleftrightarrow I (\forall x. \varphi) \rangle$

definition *psatisfiable* :: $\text{form set} \Rightarrow \text{bool}$ **where**

⟨*psatisfiable* $S \equiv \exists I. \forall \varphi \in S. I \models_p \varphi$ ⟩

abbreviation *psatisfies where* ⟨*psatisfies* $I \Phi \equiv \forall \varphi \in \Phi. \text{pholds } I \varphi$ ⟩

definition *val-to-prop-val* :: $(\text{form} \Rightarrow \text{bool}) \Rightarrow ((\text{nat} \times \text{nterm list}) \Rightarrow \text{bool})$ **where**
 ⟨*val-to-prop-val* $I = (\lambda x. I (\text{Atom } (\text{fst } x) (\text{snd } x)))$ ⟩

lemma *pholds-Not*: ⟨ $I \models_p \text{Not } \varphi \longleftrightarrow \neg (I \models_p \varphi)$ ⟩
by *simp*

lemma *pentails-equiv*: ⟨ $\text{qfree } \varphi \Longrightarrow (I \models_p \varphi \equiv (\text{val-to-prop-val } I) \models (\text{form-to-formula } \varphi))$ ⟩

proof (*induction* φ)

case *Bot*

then show *?case*

unfolding *val-to-prop-val-def* **by** *simp*

next

case (*Atom* $x1\ x2$)

then show *?case*

unfolding *val-to-prop-val-def* **by** *simp*

next

case (*Implies* $\varphi1\ \varphi2$)

then have ⟨*qfree* $\varphi1$ ⟩ **and** ⟨*qfree* $\varphi2$ ⟩ **by** *simp+*

then have ⟨ $I \models_p \varphi1 = \text{val-to-prop-val } I \models \text{form-to-formula } \varphi1$ ⟩ **and**

 ⟨ $I \models_p \varphi2 = \text{val-to-prop-val } I \models \text{form-to-formula } \varphi2$ ⟩

using *Implies(1)* *Implies(2)* **by** *simp+*

then show *?case* **by** *simp*

next

case (*Forall* $x1\ \varphi$)

then have *False* **by** *simp*

then show *?case* **by** *simp*

qed

lemma *pentails-equiv-set*:

assumes *all-qfree*: ⟨ $\forall \varphi \in S. \text{qfree } \varphi$ ⟩

shows ⟨*psatisfiable* $S \equiv \text{sat } (\text{form-to-formula } 'S)$ ⟩

proof –

{

assume *psat-s*: ⟨*psatisfiable* S ⟩

then obtain I **where** *I-is*: ⟨ $\forall \varphi \in S. I \models_p \varphi$ ⟩

unfolding *psatisfiable-def* **by** *blast*

define \mathcal{A} **where** ⟨ $\mathcal{A} = \text{val-to-prop-val } I$ ⟩

then have ⟨ $\forall \varphi \in S. \mathcal{A} \models (\text{form-to-formula } \varphi)$ ⟩

using *pentails-equiv all-qfree I-is* **by** *blast*

then have ⟨*sat* $(\text{form-to-formula } 'S)$ ⟩

unfolding *sat-def* **by** *blast*

}

moreover {

assume ⟨*sat* $(\text{form-to-formula } 'S)$ ⟩

```

then obtain  $\mathcal{A}$  where  $a$ -is:  $\langle \forall \varphi \in S. \mathcal{A} \models \text{form-to-formula } \varphi \rangle$ 
  by (meson image-eqI sat-def)
define  $I$  where  $i$ -is:  $\langle I = (\lambda x. \mathcal{A} (\text{pred } x, \text{args } x)) \rangle$ 
then have  $\langle \mathcal{A} = \text{val-to-prop-val } I \rangle$ 
  unfolding val-to-prop-val-def by simp
then have  $\langle \forall \varphi \in S. I \models_p \varphi \rangle$ 
  using pentails-equiv all-qfree  $a$ -is by blast
then have  $\langle \text{psatisfiable } S \rangle$ 
  unfolding psatisfiable-def by auto
}
ultimately show  $\langle \text{psatisfiable } S \equiv \text{sat } (\text{form-to-formula } 'S) \rangle$ 
  by argo
qed

```

```

definition finsat :: form set  $\Rightarrow$  bool where
   $\langle \text{finsat } S \equiv \forall T \subseteq S. \text{finite } T \longrightarrow \text{psatisfiable } T \rangle$ 

```

```

lemma finsat-fin-sat-eq:

```

```

  assumes all-qfree:  $\langle \forall \varphi \in S. \text{qfree } \varphi \rangle$ 
  shows  $\langle \text{finsat } S \longleftrightarrow \text{fin-sat } (\text{form-to-formula } 'S) \rangle$ 

```

```

proof

```

```

  assume finsat-s:  $\langle \text{finsat } S \rangle$ 
  then show  $\langle \text{fin-sat } (\text{form-to-formula } 'S) \rangle$ 
    unfolding fin-sat-def finsat-def
    by (metis (no-types, opaque-lifting) assms finite-subset-image pentails-equiv-set
subset-eq)

```

```

next

```

```

  assume fin-sat-s:  $\langle \text{fin-sat } (\text{form-to-formula } 'S) \rangle$ 
  show  $\langle \text{finsat } S \rangle$ 
    unfolding finsat-def
    by (meson assms compactness fin-sat-s pentails-equiv-set psatisfiable-def subsetD)
qed

```

```

lemma psatisfiable-mono:  $\langle \text{psatisfiable } S \Longrightarrow T \subseteq S \Longrightarrow \text{psatisfiable } T \rangle$ 
  unfolding psatisfiable-def by blast

```

```

lemma finsat-mono:  $\langle \text{finsat } S \Longrightarrow T \subseteq S \Longrightarrow \text{finsat } T \rangle$ 
  unfolding finsat-def by blast

```

```

lemma finsat-satisfiable:  $\langle \text{psatisfiable } S \Longrightarrow \text{finsat } S \rangle$ 
  unfolding psatisfiable-def finsat-def by blast

```

```

lemma prop-compactness:  $\langle (\forall \varphi \in S. \text{qfree } \varphi) \Longrightarrow \text{finsat } S = \text{psatisfiable } S \rangle$ 
  by (simp add: compactness finsat-fin-sat-eq finsat-satisfiable pentails-equiv-set)

```

as above, more in the style of HOL Light

```

lemma compact-prop:

```

```

  assumes  $\langle \bigwedge B. \llbracket \text{finite } B; B \subseteq A \rrbracket \Longrightarrow \exists I. \text{psatisfies } I B \rangle$  and  $\langle \bigwedge \varphi. \varphi \in A \longrightarrow$ 

```

qfree φ
shows $\langle \exists I. \text{psatisfies } I \ A \rangle$
by (*metis assms finsat-def prop-compactness psatisfiable-def*)

Three results required for the FOL uniformity theorem

lemma *compact-prop-alt:*

assumes $\langle \bigwedge I. \exists \varphi \in A. I \models_p \varphi \rangle$ $\langle \bigwedge \varphi. \varphi \in A \longrightarrow \text{qfree } \varphi \rangle$
obtains B **where** $\langle \text{finite } B \rangle$ $\langle B \subseteq A \rangle$ $\langle \bigwedge I. \exists \varphi \in B. I \models_p \varphi \rangle$

proof –

have $\langle \bigwedge \varphi. \varphi \in \text{FOL-Syntax.Not } 'A \longrightarrow \text{qfree } \varphi \rangle$
using *assms* **by** *force*

moreover

have $\langle \nexists I. \text{psatisfies } I \ (\text{FOL-Syntax.Not } 'A) \rangle$
by (*simp add: assms*)

ultimately obtain B **where** $B: \langle \text{finite } B \rangle$ $\langle B \subseteq (\text{FOL-Syntax.Not } 'A) \rangle$ $\langle \bigwedge I.$

$\exists r \in B. \neg (I \models_p r) \rangle$

using *compact-prop* [*of* $\langle \text{FOL-Syntax.Not } 'A \rangle$] **by** *fastforce*

show *thesis*

proof

show $\langle \text{finite } (\text{Not } -'B) \rangle$

using *form.inject* **by** (*metis* $\langle \text{finite } B \rangle$ *finite-vimageI injI*)

show $\langle \text{Not } -'B \subseteq A \rangle$

using B **by** *auto*

show $\langle \exists \varphi \in \text{Not } -'B. I \models_p \varphi \rangle$ **for** I

using B **by** *force*

qed

qed

lemma *finite-disj-lemma:*

assumes $\langle \text{finite } A \rangle$

shows $\langle \exists \Phi. \text{set } \Phi \subseteq A \wedge (\forall I. I \models_p \text{foldr } (\vee) \Phi \perp \longleftrightarrow (\exists \varphi \in A. I \models_p \varphi)) \rangle$

using *assms*

proof (*induction* A)

case *empty*

then show *?case*

by *auto*

next

case (*insert* φ A)

then obtain Φ **where** $\langle \text{set } \Phi \subseteq A \rangle$ $\langle \forall I. I \models_p \text{foldr } (\vee) \Phi \perp \longleftrightarrow (\exists \varphi \in A. I \models_p \varphi) \rangle$

by *blast*

then show *?case*

by (*force simp: intro!: exI* [**where** $x = \varphi \# \Phi$])

qed

lemma *compact-disj:*

assumes $\langle \bigwedge I. \exists \varphi \in A. I \models_p \varphi \rangle$ $\langle \bigwedge \varphi. \varphi \in A \longrightarrow \text{qfree } \varphi \rangle$

obtains Φ **where** $\langle \text{set } \Phi \subseteq A \rangle$ $\langle \bigwedge I. I \models_p \text{foldr } (\vee) \Phi \perp \rangle$

by (*smt (verit, best) assms compact-prop-alt finite-disj-lemma order-trans*)

end

theory *Prenex-Normal-Form*
imports
 Ground-FOL-Compactness
begin

inductive *is-prenex* :: *form* \Rightarrow *bool* **where**
 \langle *qfree* $\varphi \Rightarrow$ *is-prenex* φ \rangle
| \langle *is-prenex* $\varphi \Rightarrow$ *is-prenex* $(\forall x. \varphi)$ \rangle
| \langle *is-prenex* $\varphi \Rightarrow$ *is-prenex* $(\exists x. \varphi)$ \rangle

inductive-simps *is-prenex-simps* [*simp*]:
 is-prenex Bot
 is-prenex (Atom p ts)
 is-prenex ($\varphi \longrightarrow \psi$)
 is-prenex ($\forall x. \varphi$)

lemma *prenex-formsubst1*: \langle *is-prenex* $\varphi \Rightarrow$ *is-prenex* $(\varphi \cdot_{fm} \sigma)$ \rangle

proof (*induction* φ *arbitrary*: σ *rule*: *is-prenex.induct*)

case 1

then show ?*case* **using** *is-prenex.intros(1)* *qfree-formsubst*
 by *blast*

next

case (2 φx)

then show ?*case*

using *formsubst-def-switch* **by** (*metis* (*no-types*, *lifting*) *formsubst.simps(4)*)

is-prenex.intros(2)

next

case (3 φx)

then show ?*case*

using *formsubst-def-switch* *is-prenex.intros(3)*

by (*smt* (*verit*, *del-insts*) *formsubst.simps(1)* *formsubst.simps(3)* *formsubst.simps(4)*)

qed

lemma *prenex-formsubst2*: \langle *is-prenex* $(\varphi \cdot_{fm} \sigma) \Rightarrow$ *is-prenex* φ \rangle

proof (*induction* \langle $\varphi \cdot_{fm} \sigma$ \rangle *arbitrary*: φ σ *rule*: *is-prenex.induct*)

case 1

then show ?*case*

using *is-prenex.intros(1)* *qfree-formsubst* **by** *auto*

next

case (2 $\psi x \varphi$)

then obtain $y \varphi'$ **where** *phi-is*: \langle $\varphi = \forall y. \varphi'$ \rangle

using *formsubst-structure-all* **by** *metis*

then have \langle $\exists \sigma'. \psi = \varphi' \cdot_{fm} \sigma'$ \rangle

using 2(3) **by** (*metis* (*no-types*, *lifting*) *form.sel(6)* *formsubst.simps(4)*)

```

then obtain  $\sigma'$  where  $\langle \psi = \varphi'._{fm} \sigma' \rangle$ 
  by blast
then have  $\langle is\text{-prenex } \varphi' \rangle$ 
  using 2(2) by blast
then show ?case
  using phi-is by (simp add: is-prenex.intros(2))
next
case ( $\exists \psi x \varphi$ )
then obtain  $y \varphi'$  where phi-is:  $\langle \varphi = \exists y. \varphi' \rangle$ 
  using formsubst-structure-ex by metis
then have  $\langle \exists \sigma'. \psi = \varphi'._{fm} \sigma' \rangle$ 
  using 3(3) by (smt (verit, ccfv-threshold) form.inject(2) form.inject(3) form-
subst.simps(3)
  formsubst.simps(4))
then obtain  $\sigma'$  where  $\langle \psi = \varphi'._{fm} \sigma' \rangle$ 
  by blast
then have  $\langle is\text{-prenex } \varphi' \rangle$ 
  using 3(2) by blast
then show ?case
  using phi-is by (simp add: is-prenex.intros(3))
qed

```

```

lemma prenex-formsubst:  $\langle is\text{-prenex } (\varphi \cdot_{fm} \sigma) \equiv is\text{-prenex } \varphi \rangle$ 
  using prenex-formsubst1 prenex-formsubst2 by (smt (verit, ccfv-threshold))

```

```

lemma prenex-imp:  $\langle is\text{-prenex } (\varphi \longrightarrow \psi) \implies$ 
   $qfree (\varphi \longrightarrow \psi) \vee (\psi = \perp \wedge (\exists x \varphi'. is\text{-prenex } \varphi' \wedge \varphi = (\forall x. \varphi' \longrightarrow \perp))) \rangle$ 
  by (metis form.distinct(11) form.inject(2) is-prenex.cases)

```

```

inductive universal :: form  $\Rightarrow$  bool where
   $\langle qfree \varphi \implies universal \varphi \rangle$ 
|  $\langle universal \varphi \implies universal (\forall x. \varphi) \rangle$ 

```

```

inductive-simps universal-simps [simp]:
  universal Bot
  universal (Atom p ts)
  universal ( $\varphi \longrightarrow \psi$ )
  universal ( $\forall x. \varphi$ )

```

```

fun size :: form  $\Rightarrow$  nat where
   $\langle size \perp = 1 \rangle$ 
|  $\langle size (Atom p ts) = 1 \rangle$ 
|  $\langle size (\varphi \longrightarrow \psi) = size \varphi + size \psi \rangle$ 
|  $\langle size (\forall x. \varphi) = 1 + size \varphi \rangle$ 

```

```

lemma wf-size:  $\langle wfP (\lambda \varphi \psi. size \varphi < size \psi) \rangle$ 
  by (simp add: wfp-if-convertible-to-nat)

```

```

lemma size-indep-subst:  $\langle size (\varphi \cdot_{fm} \sigma) = size \varphi \rangle$ 

```

proof (*induction φ arbitrary: σ*)
case (*Forall $x \varphi$*)
have $\langle \exists z \sigma'. (\forall x. \varphi) \cdot_{fm} \sigma = \forall z. (\varphi \cdot_{fm} \sigma') \rangle$
by (*meson formsubst.simps(4)*)
then obtain $z \sigma'$ **where** $\langle (\forall x. \varphi) \cdot_{fm} \sigma = \forall z. (\varphi \cdot_{fm} \sigma') \rangle$
by *blast*
then have $\langle size ((\forall x. \varphi) \cdot_{fm} \sigma) = size (\forall z. (\varphi \cdot_{fm} \sigma')) \rangle$
by *argo*
also have $\langle \dots = size (\forall x. \varphi) \rangle$
using *Forall by auto*
finally show *?case .*
qed *auto*

lemma *prenex-distinct*: $\langle (\forall x. \varphi) \neq (\exists y. \psi) \rangle$
by *auto*

lemma *uniq-all-x*: $Uniq (\lambda x. \exists p. r = \forall x. p)$
using *Uniq-def by blast*

lemma *uniq-all-p*: $\langle Uniq ((\lambda p. r = \forall (THE x. \exists p. r = \forall x. p). p)) \rangle$
using *uniq-all-x Uniq-def*
by (*smt (verit, ccfv-threshold) form.inject(3)*)

lemma *uniq-ex-x*: $Uniq (\lambda x. \exists p. r = \exists x. p)$
using *Uniq-def by blast*

lemma *uniq-ex-p*: $\langle Uniq ((\lambda p. r = \exists (THE x. \exists p. r = \exists x. p). p)) \rangle$
using *uniq-ex-x Uniq-def*
by (*smt (verit, best) form.inject(2) form.inject(3)*)

definition *ppat* :: $(nat \Rightarrow form \Rightarrow form) \Rightarrow (nat \Rightarrow form \Rightarrow form) \Rightarrow (form \Rightarrow form) \Rightarrow form \Rightarrow form$ **where**
 $\langle ppat A B C r = (if (\exists x p. r = \forall x. p) then$
 $A (THE x. \exists p. r = \forall x. p) (THE p. r = \forall (THE x. \exists p. r = \forall x. p). p)$
 $else (if \exists x p. r = \exists x. p then$
 $B (THE x. \exists p. r = \exists x. p) (THE p. r = \exists (THE x. \exists p. r = \exists x. p). p)$
 $else C r) \rangle$

lemma *ppat-simpA*: $\langle \forall x p. ppat A B C (\forall x. p) = A x p \rangle$
unfolding *ppat-def by simp*

lemma *ppat-simpB*: $\langle \forall x p. ppat A B C (\exists x. p) = B x p \rangle$
unfolding *ppat-def by simp*

lemma *ppat-last*: $\langle (\forall r. \neg(\exists x p. r = \forall x. p) \wedge \neg(\exists x p. r = \exists x. p)) \implies ppat A B C r = C r \rangle$
by *blast*

lemma *ppat-last-qfree*: $\langle \text{qfree } r \implies \text{ppat } A \ B \ C \ r = C \ r \rangle$
using *qfree-no-quantif ppat-last* **by** (*simp add: ppat-def*)

lemma *ppat-to-ex-qfree*:

$\langle (\exists f. (\forall x \ p \ q. f \ p \ (\forall x. q) = ((A :: \text{form} \implies \text{nat} \implies \text{form} \implies \text{form}) \ p) \ x \ q) \wedge$
 $(\forall x \ p \ q. f \ p \ (\exists x. q) = (B \ p) \ x \ q) \wedge$
 $(\forall p \ q. \text{qfree } q \longrightarrow f \ p \ q = (C \ p) \ q)) \rangle$

proof

define *f* **where** $\langle f = (\lambda p \ q. \text{ppat } (A \ p) \ (B \ p) \ (C \ p) \ q) \rangle$

have *A-eq*: $\langle (\forall x \ p \ q. \text{ppat } (A \ p) \ (B \ p) \ (C \ p) \ (\forall x. q) = (A \ p) \ x \ q) \rangle$ **and**

B-eq: $\langle (\forall x \ p \ q. \text{ppat } (A \ p) \ (B \ p) \ (C \ p) \ (\exists x. q) = (B \ p) \ x \ q) \rangle$

unfolding *ppat-def* **by** *simp+*

have *C-eq*: $\langle (\forall p \ q. \text{qfree } q \longrightarrow \text{ppat } (A \ p) \ (B \ p) \ (C \ p) \ q = (C \ p) \ q) \rangle$

using *ppat-last-qfree* **by** *blast*

show $\langle (\forall x \ p \ q. f \ p \ (\forall x. q) = A \ p \ x \ q) \wedge (\forall x \ p \ q. f \ p \ (\exists x. q) = B \ p \ x \ q) \wedge (\forall p \ q. \text{qfree } q \longrightarrow f \ p \ q = (C \ p) \ q) \rangle$

using *A-eq B-eq C-eq* **unfolding** *f-def* **by** *blast*

qed

lemma *size-rec*:

$\langle \forall f \ g \ x. (\forall (z::\text{form}). \text{size } z < \text{size } x \longrightarrow (f \ z = g \ z)) \longrightarrow (H \ f \ x = H \ g \ x) \implies$
 $(\exists f. \forall x. f \ x = H \ f \ x) \rangle$

using *wfrec [of measure size H]* **by** (*metis cut-apply in-measure wf-measure*)

abbreviation *prenex-right-forall* :: $(\text{form} \implies \text{form} \implies \text{form}) \implies \text{form} \implies \text{nat} \implies \text{form} \implies \text{form}$ **where**

$\langle \text{prenex-right-forall} \equiv$

$(\lambda p \ \varphi \ x \ \psi. (\text{let } y = \text{variant}(FV \ \varphi \cup FV \ (\forall x. \psi)) \text{ in } (\forall y. p \ \varphi \ (\psi \cdot_{fm} (\text{subst } x \ (\text{Var } y)))))) \rangle$

abbreviation *prenex-right-exists* :: $(\text{form} \implies \text{form} \implies \text{form}) \implies \text{form} \implies \text{nat} \implies \text{form} \implies \text{form}$ **where**

$\langle \text{prenex-right-exists} \equiv$

$(\lambda p \ \varphi \ x \ \psi. (\text{let } y = \text{variant}(FV \ \varphi \cup FV \ (\exists x. \psi)) \text{ in } (\exists y. p \ \varphi \ (\psi \cdot_{fm} (\text{subst } x \ (\text{Var } y)))))) \rangle$

lemma *prenex-right-ex*:

$\langle \exists \text{prenex-right}. (\forall \varphi \ x \ \psi. \text{prenex-right } \varphi \ (\forall x. \psi) = \text{prenex-right-forall } \text{prenex-right } \varphi \ x \ \psi)$

$\wedge (\forall \varphi \ x \ \psi. \text{prenex-right } \varphi \ (\exists x. \psi) = \text{prenex-right-exists } \text{prenex-right } \varphi \ x \ \psi)$

$\wedge (\forall \varphi \ \psi. \text{qfree } \psi \longrightarrow \text{prenex-right } \varphi \ \psi = (\varphi \longrightarrow \psi)) \rangle$

proof –

have $\langle \forall \varphi. \exists \text{prenex-right-only}. \forall \psi. \text{prenex-right-only } \psi = \text{ppat}$

$(\lambda x \ \psi. (\text{let } y = \text{variant}(FV \ \varphi \cup FV \ (\forall x. \psi)) \text{ in } (\forall y. \text{prenex-right-only } (\psi \cdot_{fm} (\text{subst } x \ (\text{Var } y)))))) \rangle$

$(\lambda x \ \psi. (\text{let } y = \text{variant}(FV \ \varphi \cup FV \ (\exists x. \psi)) \text{ in } (\exists y. \text{prenex-right-only } (\psi \cdot_{fm}$

```

(subst x (Var y))))))
  (λψ. (φ → ψ)) ψ)
proof
  fix φ
  define A where ⟨A = (λg x ψ. (let y = variant(FV φ ∪ FV (∀ x. ψ)) in (∀ y.
g (ψ ·fm (subst x (Var y))))))⟩
  define B where ⟨B = (λp x ψ. (let y = variant(FV φ ∪ FV (∃ x. ψ)) in (∃ y.
p (ψ ·fm (subst x (Var y))))))⟩
  show ⟨∃ prenex-right-only. ∀ ψ. prenex-right-only ψ =
  ppat (A prenex-right-only) (B prenex-right-only) (λψ. (φ → ψ)) ψ⟩
proof (rule size-rec, (rule allI)+, (rule impI))
  fix prenex-right-only g:: form ⇒ form and ψ
  assume IH: ⟨∀ z. size z < size ψ → prenex-right-only z = g z⟩
  show ⟨ppat (A prenex-right-only) (B prenex-right-only) (λψ. (φ → ψ)) ψ =
  ppat (A g) (B g) (λψ. (φ → ψ)) ψ⟩
  proof (cases ∃ x ψ'. ψ = ∀ x. ψ')
    case True
      then obtain x ψ' where psi-is: ψ = ∀ x. ψ'
        by blast
      then have smaller: ⟨size (ψ' ·fm σ) < size ψ⟩ for σ
        using size-indep-subst by simp
      have ⟨ppat (A prenex-right-only) (B prenex-right-only) (λψ. (φ → ψ)) ψ
=
  A prenex-right-only x ψ'⟩
      unfolding ppat-def by (simp add: psi-is)
      also have ⟨... = A g x ψ'⟩
      unfolding A-def using IH smaller by presburger
      also have ⟨... = ppat (A g) (B g) (λψ. (φ → ψ)) ψ⟩
      unfolding ppat-def by (simp add: psi-is)
      finally show ?thesis .
    next
      case False
      assume falseAll: ⟨¬(∃ x ψ'. ψ = ∀ x. ψ')⟩
      then show ?thesis
      proof (cases ∃ x ψ'. ψ = ∃ x. ψ')
        case True
          then obtain x ψ' where psi-is: ψ = ∃ x. ψ'
            by blast
          then have smaller: ⟨size (ψ' ·fm σ) < size ψ⟩ for σ
            using size-indep-subst by simp
          have ⟨ppat (A prenex-right-only) (B prenex-right-only) (λψ. (φ → ψ)) ψ
=
  B prenex-right-only x ψ'⟩
          unfolding ppat-def by (simp add: psi-is)
          also have ⟨... = B g x ψ'⟩
          unfolding B-def using IH smaller by presburger
          also have ⟨... = ppat (A g) (B g) (λψ. (φ → ψ)) ψ⟩
          unfolding ppat-def by (simp add: psi-is)
          finally show ?thesis .

```

```

next
  case False
  then show ?thesis
    using falseAll ppat-last unfolding ppat-def by argo
  qed
qed
qed
qed
then have  $\langle \exists \text{prenex-right}. \forall \varphi \psi. \text{prenex-right } \varphi \psi = \text{ppat}$ 
   $(\text{prenex-right-forall prenex-right } \varphi)$ 
   $(\text{prenex-right-exists prenex-right } \varphi)$ 
   $((\longrightarrow) \varphi) \psi \rangle$ 
  using choice[of  $\lambda \varphi p. \forall \psi. p \psi =$ 
  ppat ( $\lambda x \psi. \text{let } y = \text{variant } (FV \varphi \cup FV (\forall x. \psi)) \text{ in } \forall y. p (\psi \cdot_{fm} \text{subst}$ 
 $x (Var y))$ )
   $(\lambda x \psi. \text{let } y = \text{variant } (FV \varphi \cup FV (\exists x. \psi)) \text{ in } (\exists y. p (\psi \cdot_{fm} \text{subst } x$ 
   $(Var y)))]$ 
   $((\longrightarrow) \varphi) \psi]$  by blast
  then obtain prenex-right where prenex-right-is:  $\langle \forall \varphi \psi. \text{prenex-right } \varphi \psi =$ 
   $\text{ppat } (\text{prenex-right-forall prenex-right } \varphi) (\text{prenex-right-exists prenex-right } \varphi)$ 
   $((\longrightarrow) \varphi) \psi \rangle$ 
  by blast

  have  $\langle \forall \varphi x \psi. \text{prenex-right } \varphi (\forall x. \psi) = \text{prenex-right-forall prenex-right } \varphi x \psi \rangle$ 
  using prenex-right-is unfolding ppat-def by simp
  moreover have  $\langle \forall \varphi x \psi. \text{prenex-right } \varphi (\exists x. \psi) = \text{prenex-right-exists prenex-right}$ 
   $\varphi x \psi \rangle$ 
  using prenex-right-is unfolding ppat-def by simp
  moreover have  $\langle \forall \varphi \psi. \text{qfree } \psi \longrightarrow \text{prenex-right } \varphi \psi = (\varphi \longrightarrow \psi) \rangle$ 
  using prenex-right-is by (metis (no-types, lifting) ppat-last-qfree)
  ultimately show ?thesis
  by blast
qed

```

consts *prenex-right* :: *form* \Rightarrow *form* \Rightarrow *form*

specification (*prenex-right*) \langle

$(\forall \varphi x \psi. \text{prenex-right } \varphi (\forall x. \psi) = \text{prenex-right-forall prenex-right } \varphi x \psi) \wedge$

$(\forall \varphi x \psi. \text{prenex-right } \varphi (\exists x. \psi) = \text{prenex-right-exists prenex-right } \varphi x \psi) \wedge$

$(\forall \varphi \psi. \text{qfree } \psi \longrightarrow \text{prenex-right } \varphi \psi = (\varphi \longrightarrow \psi)) \rangle$

using *prenex-right-ex by blast*

lemma *prenex-right-qfree-case:* $\langle \text{qfree } \psi \Longrightarrow \text{prenex-right } \varphi \psi = (\varphi \longrightarrow \psi) \rangle$

proof –

assume *qfree-psi:* *qfree* ψ

have $\langle ((\forall \varphi x \psi. p \varphi (\forall x. \psi) = \text{prenex-right-forall } p \varphi x \psi) \wedge$

$(\forall \varphi x \psi. p \varphi (\exists x. \psi) = \text{prenex-right-exists } p \varphi x \psi) \wedge$

$(\forall \varphi \psi. \text{qfree } \psi \longrightarrow p \varphi \psi = (\varphi \longrightarrow \psi))) \Longrightarrow (\forall \varphi \psi. \text{qfree } \psi \longrightarrow p \varphi \psi = (\varphi \longrightarrow \psi)) \rangle$ **(is** *?P p* \Longrightarrow *?Q p*) **for** *p*

by argo
 then have $\langle (\forall \varphi \psi. \text{qfree } \psi \longrightarrow \text{prenex-right } \varphi \psi = (\varphi \longrightarrow \psi)) \rangle$
 using someI2-ex[of ?P ?Q] prenex-right-def prenex-right-ex by presburger
 then show ?thesis
 using qfree-psi by blast
 qed

lemma prenex-right-all-case: $\langle \text{prenex-right } \varphi (\forall x. \psi) = \text{prenex-right-forall } \text{prenex-right } \varphi x \psi \rangle$
proof –
 have all-cases-imp-all-case: $\langle ((\forall \varphi x \psi. p \varphi (\forall x. \psi) = \text{prenex-right-forall } p \varphi x \psi) \wedge$
 $(\forall \varphi x \psi. p \varphi (\exists x. \psi) = \text{prenex-right-exists } p \varphi x \psi) \wedge$
 $(\forall \varphi \psi. \text{qfree } \psi \longrightarrow p \varphi \psi = (\varphi \longrightarrow \psi))) \implies$
 $(p \varphi (\forall x. \psi) = \text{prenex-right-forall } p \varphi x \psi) \rangle$ (is ?P p \implies ?Q p) for p
 by meson
 then have $\langle \text{prenex-right } \varphi (\forall x. \psi) = \text{prenex-right-forall } \text{prenex-right } \varphi x \psi \rangle$
 using someI2-ex[of ?P ?Q] prenex-right-def prenex-right-ex by presburger
 then show ?thesis .
 qed

lemma prenex-right-exist-case: $\langle \text{prenex-right } \varphi (\exists x. \psi) = \text{prenex-right-exists } \text{prenex-right } \varphi x \psi \rangle$
proof –
 have ex-cases-imp-ex-case: $\langle ((\forall \varphi x \psi. p \varphi (\forall x. \psi) = \text{prenex-right-forall } p \varphi x \psi) \wedge$
 $(\forall \varphi x \psi. p \varphi (\exists x. \psi) = \text{prenex-right-exists } p \varphi x \psi) \wedge$
 $(\forall \varphi \psi. \text{qfree } \psi \longrightarrow p \varphi \psi = (\varphi \longrightarrow \psi))) \implies$
 $(p \varphi (\exists x. \psi) = \text{prenex-right-exists } p \varphi x \psi) \rangle$ (is ?P p \implies ?Q p) for p
 by meson
 then have $\langle \text{prenex-right } \varphi (\exists x. \psi) = \text{prenex-right-exists } \text{prenex-right } \varphi x \psi \rangle$
 using someI2-ex[of ?P ?Q] prenex-right-def prenex-right-ex by presburger
 then show ?thesis .
 qed

lemma prenex-right-exists-shape-case:
 $\langle \exists x2 \sigma. \text{prenex-right } \varphi (\exists x. \psi) = \exists x2. \text{prenex-right } \varphi (\psi \cdot_{fm} \sigma) \rangle$
proof –
 have all-cases-imp-all-case: $\langle ((\forall \varphi x \psi. p \varphi (\forall x. \psi) = \text{prenex-right-forall } p \varphi x \psi) \wedge$
 $(\forall \varphi x \psi. p \varphi (\exists x. \psi) = \text{prenex-right-exists } p \varphi x \psi) \wedge$
 $(\forall \varphi \psi. \text{qfree } \psi \longrightarrow p \varphi \psi = (\varphi \longrightarrow \psi))) \implies$
 $(\exists x2 \sigma. p \varphi (\exists x. \psi) = \exists x2. p \varphi (\psi \cdot_{fm} \sigma)) \rangle$ (is ?P p \implies ?Q p) for p
 by meson
 then have $\langle \exists x2 \sigma. \text{prenex-right } \varphi (\exists x. \psi) = \exists x2. \text{prenex-right } \varphi (\psi \cdot_{fm} \sigma) \rangle$
 using someI2-ex[of ?P ?Q] prenex-right-def prenex-right-ex by presburger
 then show ?thesis .
 qed

abbreviation *prenex-left-forall* :: (form \Rightarrow form \Rightarrow form) \Rightarrow form \Rightarrow nat \Rightarrow form \Rightarrow form **where**

\langle prenex-left-forall \equiv
 $(\lambda p \varphi x \psi. (\text{let } y = \text{variant}(FV (\forall x. \varphi) \cup FV \psi) \text{ in } (\exists y. p (\varphi \cdot_{fm} (\text{subst } x (\text{Var } y)))) \psi))) \rangle$

abbreviation *prenex-left-exists* :: (form \Rightarrow form \Rightarrow form) \Rightarrow form \Rightarrow nat \Rightarrow form \Rightarrow form **where**

\langle prenex-left-exists \equiv
 $(\lambda p \varphi x \psi. (\text{let } y = \text{variant}(FV (\exists x. \varphi) \cup FV \psi) \text{ in } (\forall y. p (\varphi \cdot_{fm} (\text{subst } x (\text{Var } y)))) \psi))) \rangle$

lemma *prenex-left-ex*:

$\langle \exists$ prenex-left. $(\forall \varphi x \psi. \text{prenex-left } (\forall x. \varphi) \psi = \text{prenex-left-forall } \text{prenex-left } \varphi x \psi)$

$\wedge (\forall \varphi x \psi. \text{prenex-left } (\exists x. \varphi) \psi = \text{prenex-left-exists } \text{prenex-left } \varphi x \psi)$

$\wedge (\forall \varphi \psi. \text{qfree } \varphi \longrightarrow \text{prenex-left } \varphi \psi = \text{prenex-right } \varphi \psi) \rangle$

proof –

have $\langle \forall \psi. \exists$ prenex-left-only. $\forall \varphi. \text{prenex-left-only } \varphi = \text{ppat}$

$(\lambda x \varphi. (\text{let } y = \text{variant}(FV (\forall x. \varphi) \cup FV \psi) \text{ in } (\exists y. \text{prenex-left-only } (\varphi \cdot_{fm} (\text{subst } x (\text{Var } y))))))$

$(\lambda x \varphi. (\text{let } y = \text{variant}(FV (\exists x. \varphi) \cup FV \psi) \text{ in } (\forall y. \text{prenex-left-only } (\varphi \cdot_{fm} (\text{subst } x (\text{Var } y))))))$

$(\lambda \varphi. \text{prenex-right } \varphi \psi) \varphi \rangle$

proof

fix ψ

define *A* **where** $\langle A = (\lambda g x \varphi. (\text{let } y = \text{variant}(FV (\forall x. \varphi) \cup FV \psi) \text{ in } (\exists y. g (\varphi \cdot_{fm} (\text{subst } x (\text{Var } y)))))) \rangle$

define *B* **where** $\langle B = (\lambda p x \varphi. (\text{let } y = \text{variant}(FV (\exists x. \varphi) \cup FV \psi) \text{ in } (\forall y. p (\varphi \cdot_{fm} (\text{subst } x (\text{Var } y)))))) \rangle$

show $\langle \exists$ prenex-left-only. $\forall \varphi. \text{prenex-left-only } \varphi =$

$\text{ppat } (A \text{ prenex-left-only}) (B \text{ prenex-left-only}) (\lambda \varphi. \text{prenex-right } \varphi \psi) \varphi \rangle$

proof (rule size-rec, (rule allI)+, (rule impI))

fix prenex-left-only *g*:: form \Rightarrow form **and** φ

assume *IH*: $\langle \forall z. \text{size } z < \text{size } \varphi \longrightarrow \text{prenex-left-only } z = g z \rangle$

show $\langle \text{ppat } (A \text{ prenex-left-only}) (B \text{ prenex-left-only}) (\lambda \varphi. \text{prenex-right } \varphi \psi) \varphi =$

$\varphi =$

$\text{ppat } (A g) (B g) (\lambda \varphi. \text{prenex-right } \varphi \psi) \varphi \rangle$

proof (cases $\exists x \varphi'. \varphi = \forall x. \varphi'$)

case *True*

then obtain $x \varphi'$ **where** *phi-is*: $\varphi = \forall x. \varphi'$

by *blast*

then have *smaller*: $\langle \text{size } (\varphi' \cdot_{fm} \sigma) < \text{size } \varphi \rangle$ **for** σ

using *size-indep-subst* **by** *simp*

have $\langle \text{ppat } (A \text{ prenex-left-only}) (B \text{ prenex-left-only}) (\lambda \varphi. \text{prenex-right } \varphi \psi) \varphi =$

$\varphi =$

$A \text{ prenex-left-only } x \varphi' \rangle$

unfolding *ppat-def* **by** (*simp add: phi-is*)

also have $\langle \dots = A \ g \ x \ \varphi' \rangle$
unfolding *A-def* **using** *IH smaller by presburger*
also have $\langle \dots = \text{ppat} (A \ g) (B \ g) (\lambda\varphi. \text{prenex-right } \varphi \ \psi) \ \varphi \rangle$
unfolding *ppat-def* **by** (*simp add: phi-is*)
finally show *?thesis* .
next
case *False*
assume *falseAll*: $\langle \neg(\exists x \ \varphi'. \ \varphi = \forall x. \ \varphi') \rangle$
then show *?thesis*
proof (*cases* $\exists x \ \varphi'. \ \varphi = \exists x. \ \varphi'$)
case *True*
then obtain $x \ \varphi'$ **where** *phi-is*: $\varphi = \exists x. \ \varphi'$
by *blast*
then have *smaller*: $\langle \text{size} (\varphi' \cdot_{fm} \sigma) < \text{size } \varphi \rangle$ **for** σ
using *size-indep-subst* **by** *simp*
have $\langle \text{ppat} (A \ \text{prenex-left-only}) (B \ \text{prenex-left-only}) (\lambda\varphi. \ \text{prenex-right } \varphi \ \psi) \ \varphi =$
 $B \ \text{prenex-left-only} \ x \ \varphi' \rangle$
unfolding *ppat-def* **by** (*simp add: phi-is*)
also have $\langle \dots = B \ g \ x \ \varphi' \rangle$
unfolding *B-def* **using** *IH smaller by presburger*
also have $\langle \dots = \text{ppat} (A \ g) (B \ g) (\lambda\varphi. \ \text{prenex-right } \varphi \ \psi) \ \varphi \rangle$
unfolding *ppat-def* **by** (*simp add: phi-is*)
finally show *?thesis* .
next
case *False*
then show *?thesis*
using *falseAll ppat-last* **unfolding** *ppat-def* **by** *argo*
qed
qed
qed
qed
then have $\langle \exists \text{prenex-left-argswap}. \ \forall \psi \ \varphi. \ \text{prenex-left-argswap } \psi \ \varphi = \text{ppat}$
 $(\lambda x \ \varphi. \ \text{let } y = \text{variant} (FV (\forall x. \ \varphi) \cup FV \ \psi) \ \text{in } (\exists y. \ \text{prenex-left-argswap } \psi \ (\varphi$
 $\cdot_{fm} \ \text{subst } x \ (\text{Var } y))))$
 $(\lambda x \ \varphi. \ \text{let } y = \text{variant} (FV (\exists x. \ \varphi) \cup FV \ \psi) \ \text{in } \forall y. \ \text{prenex-left-argswap } \psi \ (\varphi$
 $\cdot_{fm} \ \text{subst } x \ (\text{Var } y)))$
 $(\lambda\varphi. \ \text{prenex-right } \varphi \ \psi) \ \varphi \rangle$
using *choice*[*of* $\lambda\psi \ p. \ \forall\varphi. \ p \ \varphi =$
 $\text{ppat} (\lambda x \ \varphi. \ \text{let } y = \text{variant} (FV (\forall x. \ \varphi) \cup FV \ \psi) \ \text{in } (\exists y. \ p \ (\varphi \cdot_{fm}$
 $\text{subst } x \ (\text{Var } y))))$
 $(\lambda x \ \varphi. \ \text{let } y = \text{variant} (FV (\exists x. \ \varphi) \cup FV \ \psi) \ \text{in } \forall y. \ p \ (\varphi \cdot_{fm} \ \text{subst } x$
 $(\text{Var } y)))$
 $(\lambda\varphi. \ \text{prenex-right } \varphi \ \psi) \ \varphi]$ **by** *blast*
then have $\langle \exists \text{prenex-left}. \ \forall \varphi \ \psi. \ \text{prenex-left } \varphi \ \psi = \text{ppat}$
 $(\lambda x \ \varphi. \ \text{let } y = \text{variant} (FV (\forall x. \ \varphi) \cup FV \ \psi) \ \text{in } (\exists y. \ \text{prenex-left} (\varphi \cdot_{fm} \ \text{subst}$
 $x \ (\text{Var } y)) \ \psi))$
 $(\lambda x \ \varphi. \ \text{let } y = \text{variant} (FV (\exists x. \ \varphi) \cup FV \ \psi) \ \text{in } \forall y. \ \text{prenex-left} (\varphi \cdot_{fm} \ \text{subst}$
 $x \ (\text{Var } y)) \ \psi) \rangle$

$\langle \lambda \varphi. \text{prenex-right } \varphi \ \psi \rangle \varphi$
by force
then obtain *prenex-left where prenex-left-is*: $\langle \forall \varphi \ \psi. \text{prenex-left } \varphi \ \psi = \text{ppat}$
 $\langle \lambda x \ \varphi. \text{prenex-left-forall } \text{prenex-left } \varphi \ x \ \psi \rangle$
 $\langle \lambda x \ \varphi. \text{prenex-left-exists } \text{prenex-left } \varphi \ x \ \psi \rangle$
 $\langle \lambda \varphi. \text{prenex-right } \varphi \ \psi \rangle \varphi$
by blast
have $\langle \forall \varphi \ x \ \psi. \text{prenex-left } (\forall x. \varphi) \ \psi = \text{prenex-left-forall } \text{prenex-left } \varphi \ x \ \psi \rangle$
using *prenex-left-is unfolding ppat-def by simp*
moreover have $\langle \forall \varphi \ x \ \psi. \text{prenex-left } (\exists x. \varphi) \ \psi = \text{prenex-left-exists } \text{prenex-left}$
 $\varphi \ x \ \psi \rangle$
using *prenex-left-is unfolding ppat-def by simp*
moreover have $\langle \forall \varphi \ \psi. \text{qfree } \varphi \longrightarrow \text{prenex-left } \varphi \ \psi = \text{prenex-right } \varphi \ \psi \rangle$
using *prenex-left-is by (metis (no-types, lifting) ppat-last-qfree)*
ultimately show *?thesis*
by blast
qed

definition *prenex-left where* $\langle \text{prenex-left} = (\text{SOME } \text{prenex-left.}$
 $\langle \forall \varphi \ x \ \psi. \text{prenex-left } (\forall x. \varphi) \ \psi = \text{prenex-left-forall } \text{prenex-left } \varphi \ x \ \psi \rangle \wedge$
 $\langle \forall \varphi \ x \ \psi. \text{prenex-left } (\exists x. \varphi) \ \psi = \text{prenex-left-exists } \text{prenex-left } \varphi \ x \ \psi \rangle \wedge$
 $\langle \forall \varphi \ \psi. \text{qfree } \varphi \longrightarrow \text{prenex-left } \varphi \ \psi = \text{prenex-right } \varphi \ \psi \rangle \rangle$

lemma *prenex-left-forall-case*: $\langle \text{prenex-left } (\forall x. \varphi) \ \psi = \text{prenex-left-forall } \text{prenex-left}$
 $\varphi \ x \ \psi \rangle$
unfolding *prenex-left-def by (smt (verit, del-insts) prenex-left-ex some-eq-ex)*

lemma *prenex-left-qfree-case*: $\langle \text{qfree } \varphi \implies \text{prenex-left } \varphi \ \psi = \text{prenex-right } \varphi \ \psi \rangle$
unfolding *prenex-left-def by (smt (verit, del-insts) prenex-left-ex some-eq-ex)*

lemma *prenex-left-exists-case*: $\langle \text{prenex-left } (\exists x. \varphi) \ \psi = \text{prenex-left-exists } \text{prenex-left}$
 $\varphi \ x \ \psi \rangle$
unfolding *prenex-left-def by (smt (verit, del-insts) prenex-left-ex some-eq-ex)*

lemma *prenex-left-exists-shape-case*:
 $\langle \exists x2 \ \sigma. \text{prenex-left } (\exists x. \varphi) \ \psi = \forall x2. \text{prenex-left } (\varphi \cdot_{fm} \sigma) \ \psi \rangle$
using *prenex-left-exists-case by metis*

fun *prenex where*
 $\langle \text{prenex } \perp = \perp \rangle$
 $| \langle \text{prenex } (\text{Atom } p \ ts) = \text{Atom } p \ ts \rangle$
 $| \langle \text{prenex } (\varphi \longrightarrow \psi) = \text{prenex-left } (\text{prenex } \varphi) (\text{prenex } \psi) \rangle$
 $| \langle \text{prenex } (\forall x. \varphi) = \forall x. (\text{prenex } \varphi) \rangle$

lemma *holds-indep-forall*:
assumes *y-notin*: $\langle y \notin FV (\forall x. \varphi) \rangle$
shows $\langle I, \beta \models (\forall x. \varphi) \longleftrightarrow I, \beta \models (\forall y. \varphi \cdot_{fm} (\text{subst } x (\text{Var } y))) \rangle$
proof (*cases* $\langle y = x \rangle$)

case *False*
then have *y-notin-phi*: $\langle y \notin FV \varphi \rangle$ **using** *y-notin* **by** *simp*
have *beta-equiv*: $\langle \forall w \in FV \varphi. (\lambda v. \text{termsubst } I (\beta(y := a)) (\text{subst } x (\text{Var } y)) v) w = (\beta(x := a)) w \rangle$ **for** *a*
proof
fix *w*
assume *w-in*: $\langle w \in FV \varphi \rangle$
have $\langle w = x \implies (\lambda v. \text{termsubst } I (\beta(y := a)) (\text{subst } x (\text{Var } y)) v) w = (\beta(x := a)) w \rangle$
by *simp*
moreover have $\langle w \neq x \implies (\lambda v. \text{termsubst } I (\beta(y := a)) (\text{subst } x (\text{Var } y)) v) w = (\beta(x := a)) w \rangle$
using *y-notin-phi* **by** (*metis w-in eval.simps(1) fun-upd-other subst-def*)
ultimately show $\langle (\lambda v. \text{termsubst } I (\beta(y := a)) (\text{subst } x (\text{Var } y)) v) w = (\beta(x := a)) w \rangle$
by *argo*
qed
have $\langle I, \beta \models (\forall x. \varphi) \equiv (\forall a \in \text{dom } I. I, \beta(x := a) \models \varphi) \rangle$
by *simp*
also have $\langle \dots \equiv (\forall a \in \text{dom } I. I, (\lambda v. \text{termsubst } I (\beta(y := a)) (\text{subst } x (\text{Var } y)) v) \models \varphi) \rangle$
using *holds-indep-beta-if[OF beta-equiv]* **by** *presburger*
also have $\langle \dots \equiv (\forall a \in \text{dom } I. I, \beta(y := a) \models (\varphi \cdot_{fm} (\text{subst } x (\text{Var } y)))) \rangle$
using *swap-subst-eval[of I - φ subst x (Var y)]* **by** *presburger*
also have $\langle \dots \equiv (I, \beta \models (\forall y. \varphi \cdot_{fm} (\text{subst } x (\text{Var } y)))) \rangle$
by *simp*
finally show *?thesis*
by *argo*
qed *auto*

lemma *forall-imp-commute*:

assumes *y-notin*: $\langle y \notin FV \varphi \rangle$
shows $\langle ((I :: 'a \text{ intrp}), \beta \models (\varphi \longrightarrow (\forall y. \psi))) \longleftrightarrow I, \beta \models (\forall y. \varphi \longrightarrow \psi) \rangle$
proof –
have $\langle ((I, \beta \models \varphi) \longrightarrow (\forall a \in \text{dom } I. I, \beta(y := a) \models \psi)) \longleftrightarrow (\forall a \in \text{dom } I. (I, \beta(y := a) \models \varphi \longrightarrow I, \beta(y := a) \models \psi)) \rangle$
by (*smt (verit, del-insts) fun-upd-other holds-indep-beta-if assms*)
then show *?thesis*
by *simp*
qed

lemma *forall-imp-exists*:

assumes *y-notin*: $\langle y \notin FV \psi \rangle$
shows $\langle ((I :: 'a \text{ intrp}), \beta \models ((\forall y. \varphi) \longrightarrow \psi)) \longleftrightarrow I, \beta \models (\exists y. (\varphi \longrightarrow \psi)) \rangle$
proof –
have $\langle ((\forall a \in \text{dom } I. I, \beta(y := a) \models \varphi) \longrightarrow (I, \beta \models \psi)) \longleftrightarrow (\exists a \in \text{dom } I. (I, \beta(y := a) \models \varphi \longrightarrow I, \beta \models \psi)) \rangle$
using *empty-iff list.set(1)*
by (*smt (verit, best) equalsOI intrp-is-struct struct-def*)

also have $\langle \dots \longleftrightarrow (\exists a \in \text{dom } I. (I, \beta(y := a) \models \varphi \longrightarrow I, \beta(y := a) \models \psi)) \rangle$
using *holds-indep- β -if* **by** (*smt (verit, del-insts) fun-upd-other y-notin*)
finally show *?thesis*
by *simp*
qed

lemma *exists-imp-forall*:
assumes *y-notin*: $\langle y \notin \text{FV } \psi \rangle$
shows $\langle (I, \beta \models ((\exists y. \varphi) \longrightarrow \psi) \longleftrightarrow I, \beta \models (\forall y. (\varphi \longrightarrow \psi))) \rangle$
proof –
have $\langle (\exists a \in \text{dom } I. I, \beta(y := a) \models \varphi) \longrightarrow (I, \beta \models \psi) \equiv$
 $(\forall a \in \text{dom } I. (I, \beta(y := a) \models \varphi \longrightarrow I, \beta \models \psi)) \rangle$
using *empty-iff list.set(1)* **by** (*smt (verit, ccfv-threshold)*)
also have $\langle \dots \equiv (\forall a \in \text{dom } I. (I, \beta(y := a) \models \varphi \longrightarrow I, \beta(y := a) \models \psi)) \rangle$
using *holds-indep- β -if* **by** (*smt (verit, del-insts) fun-upd-other y-notin*)
finally show *?thesis*
by *simp*
qed

lemma *exists-imp-commute*:
assumes *y-notin*: $\langle y \notin \text{FV } \varphi \rangle$
shows $\langle ((I :: 'a \text{ intrp}), \beta \models (\varphi \longrightarrow (\exists y. \psi)) \longleftrightarrow I, \beta \models (\exists y. \varphi \longrightarrow \psi)) \rangle$
proof –
have $\langle ((I, \beta \models \varphi) \longrightarrow (\exists a \in \text{dom } I. I, \beta(y := a) \models \psi)) \longleftrightarrow$
 $(\exists a \in \text{dom } I. (I, \beta \models \varphi \longrightarrow (I, \beta(y := a) \models \psi))) \rangle$
by (*smt (verit) equalsOI intrp-is-struct struct-def*)
also have $\langle \dots \longleftrightarrow (\exists a \in \text{dom } I. (I, \beta(y := a) \models \varphi \longrightarrow I, \beta(y := a) \models \psi)) \rangle$
using *y-notin* **by** (*smt (verit, ccfv-threshold) fun-upd-other holds-indep- β -if*)
finally show *?thesis*
using *holds-exists* **by** *simp*
qed

lemma *holds-indep-exists*:
 $\langle y \notin \text{FV } (\exists x. \varphi) \implies (I, \beta \models (\exists x. \varphi) \longleftrightarrow I, \beta \models (\exists y. \varphi \cdot_{fm} (\text{subst } x (\text{Var } y)))) \rangle$
by (*metis FV.simps(1,3) formsubst.simps(1,3) holds.simps(3) holds-indep-forall sup-bot.right-neutral*)

lemma *prenex-right-forall-is*:
assumes $\langle \text{dom } I \neq \{\} \rangle$
shows $\langle ((I, \beta \models \varphi \longrightarrow (\forall x. \psi)) \longleftrightarrow$
 $(I, \beta \models (\forall (\text{variant } (\text{FV } \varphi \cup \text{FV } (\forall x. \psi))))$
 $(\varphi \longrightarrow (\psi \cdot_{fm} (\text{subst } x (\text{Var } (\text{variant } (\text{FV } \varphi \cup \text{FV } (\forall x. \psi)))))))))) \rangle$
(is ?lhs = ?rhs)
proof –

define y **where** $\langle y = \text{variant} (FV \varphi \cup FV (\forall x. \psi)) \rangle$
then have $y\text{-notin1}$: $\langle y \notin FV \varphi \rangle$ **and** $y\text{-notin2}$: $\langle y \notin FV (\forall x. \psi) \rangle$
using *variant-finite finite-FV* **by** (*meson UnCI finite-UnI*)
have $\langle ?lhs \longleftrightarrow I, \beta \models (\varphi \longrightarrow (\forall y. \psi \cdot_{fm} (\text{subst } x (\text{Var } y)))) \rangle$
using *holds-indep-forall y-notin2*
by (*smt (verit, ccfv-SIG) holds.simps(3)*)
also have $\langle \dots \longleftrightarrow I, \beta \models (\forall y. \varphi \longrightarrow (\psi \cdot_{fm} (\text{subst } x (\text{Var } y)))) \rangle$
using *forall-imp-commute[OF y-notin1, of I beta psi .fm (subst x (Var y))]*
finally show $?thesis$
unfolding $y\text{-def}$.
qed

lemma *prenex-right-exists-is*:

assumes $\langle \text{dom } I \neq \{\} \rangle$
shows $\langle (I, \beta \models \varphi \longrightarrow (\exists x. \psi)) \longleftrightarrow$
 $(I, \beta \models (\exists (\text{variant} (FV \varphi \cup FV (\exists x. \psi))))$
 $(\varphi \longrightarrow (\psi \cdot_{fm} (\text{subst } x (\text{Var} (\text{variant} (FV \varphi \cup FV (\exists x. \psi)))))))) \rangle$
(is $?lhs = ?rhs$ **)**
proof –

define y **where** $\langle y = \text{variant} (FV \varphi \cup FV (\exists x. \psi)) \rangle$
then have $y\text{-notin1}$: $\langle y \notin FV \varphi \rangle$ **and** $y\text{-notin2}$: $\langle y \notin FV (\exists x. \psi) \rangle$
using *variant-finite finite-FV* **by** (*meson UnCI finite-UnI*)
have $\langle ?lhs \longleftrightarrow I, \beta \models (\varphi \longrightarrow (\exists y. \psi \cdot_{fm} (\text{subst } x (\text{Var } y)))) \rangle$
using *holds-indep-exists y-notin2 holds-exists* **by** (*smt (verit) holds.simps(3)*)
also have $\langle \dots \longleftrightarrow I, \beta \models (\exists y. \varphi \longrightarrow (\psi \cdot_{fm} (\text{subst } x (\text{Var } y)))) \rangle$
using *exists-imp-commute[OF y-notin1, of I beta psi .fm (subst x (Var y))]*
finally show $?thesis$
unfolding $y\text{-def}$.
qed

lemma *prenex-left-forall-is*:

assumes $\langle \text{dom } I \neq \{\} \rangle$
shows $\langle (I, \beta \models ((\forall x. \varphi) \longrightarrow \psi)) \equiv (I, \beta \models (\exists (\text{variant} (FV (\forall x. \varphi) \cup FV \psi)))$
 $((\varphi \cdot_{fm} (\text{subst } x (\text{Var} (\text{variant} (FV (\forall x. \varphi) \cup FV \psi)))) \longrightarrow \psi))) \rangle$
using *forall-imp-exists holds-indep-forall holds.simps(3)*
by (*smt (verit, del-insts) FV.simps(3) UnI2 sup commute variant-form*)

lemma *prenex-left-exists-is*:

assumes $\langle \text{dom } I \neq \{\} \rangle$
shows $\langle (I, \beta \models ((\exists x. \varphi) \longrightarrow \psi)) \equiv (I, \beta \models (\forall (\text{variant} (FV (\exists x. \varphi) \cup FV \psi)))$
 $((\varphi \cdot_{fm} (\text{subst } x (\text{Var} (\text{variant} (FV (\exists x. \varphi) \cup FV \psi)))) \longrightarrow \psi))) \rangle$
using *exists-imp-forall holds-indep-exists holds.simps(3)*
by (*smt (verit, ccfv-SIG) FV.simps(3) UnCI finite-FV variant-finite*)

lemma *prenex-right-forall-FV*: $\langle FV (\varphi \longrightarrow (\forall x. \psi)) =$

$FV (\forall (\text{variant} (FV \varphi \cup FV (\forall x. \psi))))$. $(\varphi \longrightarrow (\psi \cdot_{fm} (\text{subst } x (\text{Var} (\text{variant}$
 $(FV \varphi \cup FV (\forall x. \psi)))))) \rangle$
using *formsubst-rename*

by (*metis Diff-empty Diff-insert0 FV.simps(3) FV.simps(4) Un-Diff finite-FV variant-finite*)

lemma *prenex-right-exists-FV*: $\langle FV (\varphi \longrightarrow (\exists x. \psi)) = FV (\forall (\text{variant } (FV \varphi \cup FV (\exists x. \psi))). (\varphi \longrightarrow (\psi \cdot_{fm} (\text{subst } x (\text{Var } (\text{variant } (FV \varphi \cup FV (\exists x. \psi)))))))) \rangle$
using *formsubst-rewrite prenex-right-forall-FV by force*

lemma *prenex-left-forall-FV*: $\langle FV ((\forall x. \varphi) \longrightarrow \psi) = FV (\exists (\text{variant } (FV (\forall x. \varphi) \cup FV \psi)). ((\varphi \cdot_{fm} (\text{subst } x (\text{Var } (\text{variant } (FV (\forall x. \varphi) \cup FV \psi)))) \longrightarrow \psi)) \rangle$
using *formsubst-rewrite*
by (*metis Diff-idemp Diff-insert-absorb FV.simps(3) FV.simps(4) Un-Diff finite-FV variant-finite FV-exists*)

lemma *prenex-left-exists-FV*: $\langle FV ((\exists x. \varphi) \longrightarrow \psi) = FV (\forall (\text{variant } (FV (\exists x. \varphi) \cup FV \psi)). ((\varphi \cdot_{fm} (\text{subst } x (\text{Var } (\text{variant } (FV (\exists x. \varphi) \cup FV \psi)))) \longrightarrow \psi)) \rangle$
using *formsubst-rewrite FV-exists prenex-left-forall-FV by auto*

lemma *prenex-right-forall-language*: $\langle \text{language } \{\varphi \longrightarrow (\forall x. \psi)\} = \text{language } \{\forall (\text{variant } (FV \varphi \cup FV (\forall x. \psi))). (\varphi \longrightarrow (\psi \cdot_{fm} (\text{subst } x (\text{Var } (\text{variant } (FV \varphi \cup FV (\forall x. \psi))))))\} \rangle$
using *lang-singleton formsubst-functions-form formsubst-predicates formsubst-language-rewrite by auto*

lemma *prenex-right-exists-language*: $\langle \text{language } \{\varphi \longrightarrow (\exists x. \psi)\} = \text{language } \{\exists (\text{variant } (FV \varphi \cup FV (\exists x. \psi))). (\varphi \longrightarrow (\psi \cdot_{fm} (\text{subst } x (\text{Var } (\text{variant } (FV \varphi \cup FV (\exists x. \psi))))))\} \rangle$
using *lang-singleton formsubst-functions-form formsubst-predicates formsubst-language-rewrite by auto*

lemma *prenex-left-forall-language*: $\langle \text{language } \{(\forall x. \varphi) \longrightarrow \psi\} = \text{language } \{\exists (\text{variant } (FV (\forall x. \varphi) \cup FV \psi)). ((\varphi \cdot_{fm} (\text{subst } x (\text{Var } (\text{variant } (FV (\forall x. \varphi) \cup FV \psi)))) \longrightarrow \psi)\} \rangle$
using *lang-singleton formsubst-functions-form formsubst-predicates formsubst-language-rewrite by auto*

lemma *prenex-left-exists-language*: $\langle \text{language } \{(\exists x. \varphi) \longrightarrow \psi\} = \text{language } \{\forall (\text{variant } (FV (\exists x. \varphi) \cup FV \psi)). ((\varphi \cdot_{fm} (\text{subst } x (\text{Var } (\text{variant } (FV (\exists x. \varphi) \cup FV \psi)))) \longrightarrow \psi)\} \rangle$
using *lang-singleton formsubst-functions-form formsubst-predicates formsubst-language-rewrite by auto*

lemma *prenex-props-forall*: $\langle P \wedge FV \varphi = FV \psi \wedge \text{language } \{\varphi\} = \text{language } \{\psi\} \wedge (\forall (I :: 'a \text{ intrp}) \beta. \text{dom } I \neq \{\} \longrightarrow (I, \beta \models \varphi \longleftrightarrow I, \beta \models \psi)) \implies$

$P \wedge FV (\forall x. \varphi) = FV (\forall x. \psi) \wedge \text{language } \{(\forall x. \varphi)\} = \text{language } \{(\forall x. \psi)\} \wedge$
 $(\forall (I :: 'a \text{ intrp}) \beta. \text{dom } I \neq \{\} \longrightarrow (I, \beta \models (\forall x. \varphi) \longleftrightarrow I, \beta \models (\forall x. \psi)))$

\rangle
using lang-singleton by simp

lemma prenex-props-exists: $\langle P \wedge FV \varphi = FV \psi \wedge \text{language } \{\varphi\} = \text{language } \{\psi\}$
 \wedge
 $(\forall (I :: 'a \text{ intrp}) \beta. \text{dom } I \neq \{\} \longrightarrow (I, \beta \models \varphi \longleftrightarrow I, \beta \models \psi)) \implies$
 $P \wedge FV (\exists x. \varphi) = FV (\exists x. \psi) \wedge \text{language } \{(\exists x. \varphi)\} = \text{language } \{(\exists x. \psi)\} \wedge$
 $(\forall (I :: 'a \text{ intrp}) \beta. \text{dom } I \neq \{\} \longrightarrow (I, \beta \models (\exists x. \varphi) \longleftrightarrow I, \beta \models (\exists x. \psi)))$

\rangle
using lang-singleton by simp

lemma prenex-right-props-imp0:
assumes $\langle \text{qfree } \varphi \rangle$
shows $\langle \text{is-prenex } \psi \implies \text{is-prenex } (\text{prenex-right } \varphi \psi) \rangle$
proof (*induction* ψ *rule: measure-induct-rule [of size]*)
case (*less* ψ)
show *?case*
proof (*cases rule: is-prenex.cases[OF is-prenex]*)
case (*1* ξ)
then show *?thesis*
by (*simp add: assms prenex-right-qfree-case*)
next
case (*2* ξx)
then have $\langle \text{prenex-right } \varphi \psi = \text{prenex-right-forall } \text{prenex-right } \varphi x \xi \rangle$
using *prenex-right-all-case by blast*
then show $\langle \text{is-prenex } (\text{prenex-right } \varphi \psi) \rangle$
using *less 2 by (auto simp: Let-def prenex-formsubst1 size-indep-subst)*
next
case (*3* ξx)
then have $\langle \exists y \sigma. \text{prenex-right } \varphi \psi = \exists y. \text{prenex-right } \varphi (\xi \cdot_{fm} \sigma) \rangle$
using *prenex-right-exists-shape-case by presburger*
then obtain $y \sigma$ **where** *pr-is:* $\langle \text{prenex-right } \varphi \psi = \exists y. \text{prenex-right } \varphi (\xi \cdot_{fm} \sigma) \rangle$
by *blast*
have *size-xp:* $\langle \text{size } (\xi \cdot_{fm} \sigma) < \text{size } \psi \rangle$
using *3(1) size-indep-subst by auto*
have $\langle \text{is-prenex } (\xi \cdot_{fm} \sigma) \rangle$
using *3(2) prenex-formsubst1 by blast*
then have $\langle \text{is-prenex } (\text{prenex-right } \varphi (\xi \cdot_{fm} \sigma)) \rangle$
using *less size-xp by blast*
then show *?thesis*
using *is-prenex.intros(3) pr-is by presburger*
qed
qed

lemma prenex-right-props-imp:
assumes $\langle \text{qfree } \varphi \rangle$

shows $\langle is\text{-prenex } \psi \implies$
 $is\text{-prenex } (prenex\text{-right } \varphi \psi) \wedge$
 $FV (prenex\text{-right } \varphi \psi) = FV (\varphi \longrightarrow \psi) \wedge$
 $language \{prenex\text{-right } \varphi \psi\} = language \{(\varphi \longrightarrow \psi)\} \wedge$
 $(\forall (I :: 'a \text{ intrp}) \beta. dom I \neq \{\} \longrightarrow ((I, \beta \models (prenex\text{-right } \varphi \psi)) \longleftrightarrow (I, \beta$
 $\models (\varphi \longrightarrow \psi)))) \rangle$
(is $\langle is\text{-prenex } \psi \implies ?P \psi \rangle$
proof (*induction* ψ *rule: measure-induct-rule [of size]*)
case (*less* ψ)
show $?case$
proof (*cases rule: is-prenex.cases[OF* $\langle is\text{-prenex } \psi \rangle$)
case (*1* ξ)
then show $?thesis$
using *prenex-right-qfree-case* $\langle qfree \varphi \rangle$ *is-prenex.intros(1)* *qfree.simps(3)* **by**
presburger
next
case (*2* ξx)
have *pr-is1*: $\langle prenex\text{-right } \varphi \psi = prenex\text{-right-forall } prenex\text{-right } \varphi x \xi \rangle$
using *2 prenex-right-all-case* **by** *blast*
define y **where** $\langle y = variant (FV \varphi \cup FV (\forall x. \xi)) \rangle$
then have *pr-is2*: $\langle prenex\text{-right } \varphi \psi = \forall y. prenex\text{-right } \varphi (\xi \cdot_{fm} subst x (Var$
 $y)) \rangle$
using $\langle qfree \varphi \rangle$ *2(2)* *pr-is1* **unfolding** $y\text{-def}$ **by** *meson*
have $\langle is\text{-prenex } (\xi \cdot_{fm} subst x (Var y)) \rangle$
using *prenex-formsubst1 2(2)* **by** *presburger*
then have $p\text{-xps}$: $\langle ?P (\xi \cdot_{fm} subst x (Var y)) \rangle$
using *less 2(1)* *less-Suc-eq plus-1-eq-Suc size.simps(4)* *size-indep-subst* **by**
presburger
have $\langle is\text{-prenex } (prenex\text{-right } \varphi \psi) \rangle$
using *prenex-right-props-imp0* $\langle is\text{-prenex } \psi \rangle$ $\langle qfree \varphi \rangle$ **by** *blast*
moreover have $\langle FV (prenex\text{-right } \varphi \psi) = FV (\varphi \longrightarrow \psi) \rangle$
using *prenex-right-forall-FV[of* $\varphi x \xi]$ **by** (*metis 2(1)* *FV.simps(4)* $p\text{-xps}$
pr-is1 y-def)
moreover have $\langle language \{prenex\text{-right } \varphi \psi\} = language \{\varphi \longrightarrow \psi\} \rangle$
using *prenex-right-forall-language*
by (*metis 2(1)* *functions-form.simps(4)* *lang-singleton p-xps pr-is1 predi-*
cates-form.simps(4) y-def)
moreover have $\langle (\forall (I :: 'a \text{ intrp}) \beta. dom I \neq \{\} \longrightarrow I, \beta \models prenex\text{-right } \varphi \psi$
 $= I, \beta \models \varphi \longrightarrow \psi) \rangle$
by (*metis 2(1)* *holds.simps(4)* $p\text{-xps pr-is2 prenex-right-forall-is y-def$)
ultimately show $?thesis$
by *blast*
next
case (*3* ξx)
have *pr-is1*: $\langle prenex\text{-right } \varphi \psi = prenex\text{-right-exists } prenex\text{-right } \varphi x \xi \rangle$
using *3 prenex-right-exist-case* **by** *blast*
define y **where** $\langle y = variant (FV \varphi \cup FV (\exists x. \xi)) \rangle$
then have *pr-is2*: $\langle prenex\text{-right } \varphi \psi = \exists y. prenex\text{-right } \varphi (\xi \cdot_{fm} subst x (Var$
 $y)) \rangle$

```

    using ⟨qfree φ⟩ 3(2) pr-is1 unfolding y-def by meson
  have ⟨is-prenex (ξ ·fm subst x (Var y))⟩
    using prenex-formsubst1 3(2) by presburger
  then have p-xps: ⟨?P (ξ ·fm subst x (Var y))⟩
    using less 3(1) less-Suc-eq plus-1-eq-Suc size.simps size-indep-subst by simp
  have ⟨is-prenex (prenex-right φ ψ)⟩
    using prenex-right-props-imp0 ⟨is-prenex ψ⟩ ⟨qfree φ⟩ by blast
  moreover have ⟨FV (prenex-right φ ψ) = FV (φ → ψ)⟩
    using prenex-right-exists-FV[of φ x ξ] by (metis 3(1) FV.simps(4) FV-exists
p-xps
    pr-is1 y-def)
  moreover have ⟨language {prenex-right φ ψ} = language {φ → ψ}⟩
    using prenex-right-forall-language by (smt (verit) 3(1) p-xps pr-is1
    prenex-props-exists prenex-right-exists-language y-def)
  moreover have ⟨(∀ (I :: 'a intrp) β. dom I ≠ {} → I,β ⊨ prenex-right φ ψ
= I,β ⊨ φ → ψ)⟩
    by (smt (verit, best) 3(1) p-xps pr-is2 prenex-props-exists prenex-right-exists-is
y-def)
  ultimately show ?thesis
    by blast
qed
qed

```

```

lemma prenex-right-props:
  ⟨qfree φ ∧ is-prenex ψ ⇒
  is-prenex (prenex-right φ ψ) ∧
  FV (prenex-right φ ψ) = FV (φ → ψ) ∧
  language {prenex-right φ ψ} = language {(φ → ψ)} ∧
  (∀ (I :: 'a intrp) β. dom I ≠ {} → ((I,β ⊨ (prenex-right φ ψ)) ↔ (I,β ⊨ (φ
→ ψ))))⟩
  using prenex-right-props-imp by meson

```

```

lemma prenex-left-props-imp0:
  assumes ⟨is-prenex ψ⟩
  shows ⟨is-prenex φ ⇒ is-prenex (prenex-left φ ψ)⟩
proof (induction φ rule: measure-induct-rule [of size])
  case (less φ)
  show ?case
  proof (cases rule: is-prenex.cases[OF ⟨is-prenex φ⟩])
    case (1 ξ)
    then show ?thesis
      using ⟨is-prenex φ⟩ prenex-right-props prenex-left-qfree-case ⟨is-prenex ψ⟩ by
presburger
  next
  case (2 ξ x)
  then have ⟨prenex-left φ ψ = prenex-left-forall prenex-left ξ x ψ⟩
    using prenex-left-forall-case by blast
  then show ⟨is-prenex (prenex-left φ ψ)⟩

```

```

    using less 2 by (metis is-prenex.intros(3) lessI plus-1-eq-Suc prenex-formsubst1
size.simps(4) size-indep-subst)
  next
  case (3 ξ x)
  then have ⟨∃ y σ. prenex-left φ ψ = ∀ y. prenex-left (ξ ·fm σ) ψ⟩
    using prenex-left-exists-shape-case by presburger
  then obtain y σ where pr-is: ⟨prenex-left φ ψ = ∀ y. prenex-left (ξ ·fm σ) ψ⟩
    by blast
  have size-xp: ⟨size (ξ ·fm σ) < size φ⟩
    using 3(1) size-indep-subst by auto
  have ⟨is-prenex (ξ ·fm σ)⟩
    using 3(2) prenex-formsubst1 by blast
  then have ⟨is-prenex (prenex-left (ξ ·fm σ) ψ)⟩
    using less size-xp by blast
  then show ?thesis
    using is-prenex.intros pr-is by presburger
qed
qed

lemma prenex-left-props-imp:
  assumes ⟨is-prenex ψ⟩
  shows ⟨is-prenex φ ⟹
    is-prenex (prenex-left φ ψ) ∧
    FV (prenex-left φ ψ) = FV (φ ⟶ ψ) ∧
    (language {(prenex-left φ ψ)} = language {(φ ⟶ ψ)}) ∧
    (∀ (I :: 'a intrp) β. dom I ≠ {} ⟶ (I,β ⊨ prenex-left φ ψ ⟷ I,β ⊨ φ
    ⟶ ψ))⟩
    (is ⟨is-prenex φ ⟹ ?P φ⟩)
proof (induction φ rule: measure-induct-rule [of size])
  case (less ξ)
  show ?case
  proof (cases rule: is-prenex.cases[OF ⟨is-prenex ξ⟩])
    case (1 ξ')
    then show ?thesis
      using prenex-right-qfree-case ⟨is-prenex ψ⟩
      by (simp add: prenex-left-qfree-case prenex-right-props)
  next
  case (2 ξ' x)
  have pr-is1: ⟨prenex-left ξ ψ = prenex-left-forall prenex-left ξ' x ψ⟩
    using 2 prenex-left-forall-case by blast
  define y where ⟨y = variant (FV (∀ x. ξ') ∪ FV ψ)⟩
  then have pr-is2: ⟨prenex-left ξ ψ = ∃ y. prenex-left (ξ' ·fm subst x (Var y))
  ψ⟩
    using ⟨is-prenex ψ⟩ 2(2) pr-is1 unfolding y-def by meson
  have ⟨is-prenex (ξ' ·fm subst x (Var y))⟩
    using prenex-formsubst1 2(2) by presburger
  then have p-xps: ⟨?P (ξ' ·fm subst x (Var y))⟩
    using less 2(1) less-Suc-eq plus-1-eq-Suc size.simps(4) size-indep-subst by
presburger

```

```

have ‹is-prenex (prenex-left ξ ψ)›
  using prenex-left-props-imp0 ‹is-prenex ξ› ‹is-prenex ψ› by blast
moreover have ‹FV (prenex-left ξ ψ) = FV (ξ → ψ)›
  using prenex-left-forall-FV[of x ξ' ψ] by (metis 2(1) FV-exists p-xps pr-is1
y-def)
moreover have ‹language {prenex-left ξ ψ} = language {ξ → ψ}›
  using prenex-left-forall-language
  by (smt (verit, ccfv-threshold) 2(1) p-xps pr-is1 prenex-props-exists y-def)
moreover have ‹(∧(I :: 'a intrp) β. dom I ≠ {} ⇒ I,β ⊨ prenex-left ξ ψ =
I,β ⊨ ξ → ψ)›
  by (metis 2(1) holds-exists p-xps pr-is2 prenex-left-forall-is y-def)
ultimately show ‹?P ξ›
  by blast
next
case (3 ξ' x)
have pr-is1: ‹prenex-left ξ ψ = prenex-left-exists prenex-left ξ' x ψ›
  using 3 prenex-left-exists-case by blast
define y where ‹y = variant (FV (∃ x. ξ') ∪ FV ψ)›
then have pr-is2: ‹prenex-left ξ ψ = ∀ y. prenex-left (ξ' ·fm subst x (Var y))
ψ›
  using ‹is-prenex ψ› 3(2) pr-is1 unfolding y-def by meson
have ‹is-prenex (ξ' ·fm subst x (Var y))›
  using prenex-formsubst1 3(2) by presburger
then have p-xps: ‹?P (ξ' ·fm subst x (Var y))›
  using less 3(1) less-Suc-eq plus-1-eq-Suc size.simps size-indep-subst by simp
have ‹is-prenex (prenex-left ξ ψ)›
  using prenex-left-props-imp0 ‹is-prenex ξ› ‹is-prenex ψ› by blast
moreover have ‹FV (prenex-left ξ ψ) = FV (ξ → ψ)›
  using prenex-left-exists-FV[of x ξ' ψ] by (metis 3(1) FV.simps(4) p-xps
pr-is1 y-def)
moreover have ‹language {prenex-left ξ ψ} = language {ξ → ψ}›
  using prenex-left-exists-language[of x ξ' ψ]
  by (smt (verit) 3(1) p-xps pr-is2 prenex-props-forall y-def)
moreover have ‹(∀(I :: 'a intrp) β. dom I ≠ {} →
I,β ⊨ prenex-left ξ ψ = I,β ⊨ ξ → ψ)›
  by (metis 3(1) holds.simps(4) p-xps pr-is2 prenex-left-exists-is y-def)
ultimately show ‹?P ξ›
  by blast
qed
qed

lemma prenex-left-props:
  ‹is-prenex φ ∧ is-prenex ψ ⇒
    is-prenex (prenex-left φ ψ) ∧
    FV (prenex-left φ ψ) = FV (φ → ψ) ∧
    (language {(prenex-left φ ψ)} = language {(φ → ψ)}) ∧
    (∀(I :: 'a intrp) β. dom I ≠ {} → (I,β ⊨ prenex-left φ ψ ↔ I,β ⊨ φ
→ ψ))›
  using prenex-left-props-imp by meson

```

theorem *prenex-props*: $\langle \text{is-prenex } (\text{prenex } \varphi) \wedge (FV (\text{prenex } \varphi) = FV \varphi) \wedge$
 $(\text{language } \{\text{prenex } \varphi\} = \text{language } \{\varphi\}) \wedge$
 $(\forall (I :: 'a \text{ intrp}) \beta. \text{dom } I \neq \{\} \longrightarrow (I, \beta \models (\text{prenex } \varphi)) \longleftrightarrow (I, \beta \models \varphi)) \rangle$

proof (*induction* φ *rule*: *form.induct*)

case *Bot*

then show *?case*

by (*metis is-prenex.simps prenex.simps(1) qfree.simps(1)*)

next

case (*Atom p ts*)

then show *?case*

using *is-prenex.intros(1) prenex.simps(2) qfree.simps(2)* **by** *presburger*

next

case (*Implies $\varphi \psi$*)

have $\langle \text{is-prenex } (\text{prenex } (\varphi \longrightarrow \psi)) \rangle$

using *Implies prenex-left-props prenex.simps(3)* **by** *presburger*

moreover have $\langle FV (\text{prenex } (\varphi \longrightarrow \psi)) = FV (\varphi \longrightarrow \psi) \rangle$

using *Implies prenex-left-props prenex.simps(3) FV.simps(3)* **by** *presburger*

moreover have $\langle \text{language } \{\text{prenex } (\varphi \longrightarrow \psi)\} = \text{language } \{\varphi \longrightarrow \psi\} \rangle$

using *Implies prenex-left-props prenex.simps(3) lang-singleton*
 functions-form.simps(3) predicates-form.simps(3) **by** (*metis prod.inject*)

moreover have $\langle \forall (I :: 'a \text{ intrp}) \beta. \text{FOL-Semantics.dom } I \neq \{\} \longrightarrow$
 $I, \beta \models \text{prenex } (\varphi \longrightarrow \psi) = I, \beta \models \varphi \longrightarrow \psi \rangle$

using *Implies prenex-left-props holds.simps(3) prenex.simps(3)* **by** *metis*

ultimately show *?case* **by** *blast*

next

case (*Forall x φ*)

have $\langle \text{is-prenex } (\text{prenex } (\forall x. \varphi)) \rangle$

using *Forall using is-prenex.intros(2) prenex.simps(4)* **by** *presburger*

moreover have *fv-indep-prenex*: $\langle FV (\text{prenex } (\forall x. \varphi)) = FV (\forall x. \varphi) \rangle$

using *Forall FV.simps(4) prenex.simps(4)* **by** *presburger*

moreover have $\langle \text{language } \{\text{prenex } (\forall x. \varphi)\} = \text{language } \{\forall x. \varphi\} \rangle$

using *Forall prenex.simps(4) functions-form.simps(4) predicates-form.simps(4)*

unfolding *language-def functions-forms-def predicates-def* **by** *simp*

moreover have $\langle (\forall (I :: 'a \text{ intrp}) \beta. \text{dom } I \neq \{\} \longrightarrow I, \beta \models \text{prenex } (\forall x. \varphi) =$
 $I, \beta \models (\forall x. \varphi)) \rangle$

using *Forall holds.simps(4)* **by** *simp*

ultimately show *?case* **by** *blast*

qed

corollary *is-prenex-prenex* [*simp*]: $\langle \text{is-prenex } (\text{prenex } \varphi) \rangle$

and *FV-prenex* [*simp*]: $\langle FV (\text{prenex } \varphi) = FV \varphi \rangle$

and *language-prenex* [*simp*]: $\langle \text{language } \{\text{prenex } \varphi\} = \text{language } \{\varphi\} \rangle$

by (*auto simp: prenex-props*)

corollary *prenex-holds* [*simp*]: $\langle \text{dom } I \neq \{\} \implies (I, \beta \models (\text{prenex } \varphi)) \longleftrightarrow (I, \beta \models \varphi) \rangle$

by (*simp add: prenex-props*)

```

lemma prenex-satisfies [simp]:
  assumes dom M ≠ {}
  shows satisfies M {prenex φ} ↔ satisfies M {φ}
  using assms prenex-holds by (fastforce simp: satisfies-def)

```

end

```

theory Bumping
imports
  FOL-Semantics
  HOL-Library.Nat-Bijection
begin

```

```

abbreviation numpair where
  ⟨numpair m n ≡ prod-encode (m,n)⟩

```

```

abbreviation numfst where
  ⟨numfst k ≡ fst (prod-decode k)⟩

```

```

abbreviation numsnd where
  ⟨numsnd k ≡ snd (prod-decode k)⟩

```

```

definition bump-intrp :: 'm intrp ⇒ 'm intrp where
  ⟨bump-intrp M = Abs-intrp ((dom M), (λk zs. (intrp-fn M) (numsnd k) zs),
  (intrp-rel M))⟩

```

```

lemma bump-dom [simp]: ⟨dom (bump-intrp M) = dom M⟩
proof –
  have is-struct: ⟨struct (dom M)⟩
  by (simp add: intrp-is-struct)
  then show ?thesis unfolding bump-intrp-def using dom-Abs-is-fst by blast
qed

```

```

lemma bump-intrp-fn [simp]: ⟨intrp-fn (bump-intrp M) (numpair 0 f) ts = intrp-fn M f ts⟩
proof –
  have is-struct: ⟨struct (dom M)⟩
  by (smt (verit, best) intrp-is-struct struct-def)
  then show ?thesis unfolding bump-intrp-def by simp
qed

```

```

lemma bump-intrp-rel [simp]: ⟨intrp-rel (bump-intrp M) n = intrp-rel M n⟩
unfolding bump-intrp-def
by (smt (verit) intrp-is-struct intrp-rel-Abs-is-snd-snd struct-def)

```

fun *bump-nterm* :: *nterm* \Rightarrow *nterm* **where**
 $\langle \text{bump-nterm } (\text{Var } x) = \text{Var } x \rangle$
 $\mid \langle \text{bump-nterm } (\text{Fun } f \ ts) = \text{Fun } (\text{numpair } 0 \ f) \ (\text{map } \text{bump-nterm } \ ts) \rangle$

fun *bump-form* :: *form* \Rightarrow *form* **where**
 $\langle \text{bump-form } \perp = \perp \rangle$
 $\mid \langle \text{bump-form } (\text{Atom } p \ ts) = \text{Atom } p \ (\text{map } \text{bump-nterm } \ ts) \rangle$
 $\mid \langle \text{bump-form } (\varphi \longrightarrow \psi) = (\text{bump-form } \varphi) \longrightarrow (\text{bump-form } \psi) \rangle$
 $\mid \langle \text{bump-form } (\forall x. \varphi) = \forall x. (\text{bump-form } \varphi) \rangle$

lemma *bumpterm*: $\langle \llbracket t \rrbracket^{\mathcal{M}, \beta} = \llbracket \text{bump-nterm } t \rrbracket^{\text{bump-intrp } \mathcal{M}, \beta} \rangle$
proof (*induct t*)
case (*Var x*)
then show *?case*
by *simp*
next
case (*Fun f ts*)
then have $\langle \text{intrp-fn } \mathcal{M} \ f \ \llbracket t \rrbracket^{\mathcal{M}, \beta}. \ t \leftarrow \ ts \rangle =$
 $\text{intrp-fn } \mathcal{M} \ f \ \llbracket \text{bump-nterm } t \rrbracket^{\text{bump-intrp } \mathcal{M}, \beta}. \ t \leftarrow \ ts \rangle$
by (*metis (no-types, lifting) map-eq-conv*)
also have $\langle \dots =$
 $\text{intrp-fn } (\text{bump-intrp } \mathcal{M}) \ (\text{numpair } 0 \ f) \ \llbracket \text{bump-nterm } t \rrbracket^{\text{bump-intrp } \mathcal{M}, \beta}. \ t \leftarrow$
 $\ ts \rangle$
by (*simp add: bump-intrp-fn*)
also have $\langle \dots =$
 $\text{intrp-fn } (\text{bump-intrp } \mathcal{M}) \ (\text{numpair } 0 \ f) \ \llbracket t \rrbracket^{\text{bump-intrp } \mathcal{M}, \beta}. \ t \leftarrow (\text{map } \text{bump-nterm}$
 $\ ts) \rangle$
using *map-eq-conv* **by** *fastforce*
ultimately show *?case* **by** *auto*
qed

lemma *bump-intrp-rel-holds*: $\langle (\text{map } (\lambda t. \llbracket t \rrbracket^{\mathcal{M}, \beta}) \ ts \in \text{intrp-rel } \mathcal{M} \ n) =$
 $(\text{map } ((\lambda t. \llbracket t \rrbracket^{\text{bump-intrp } \mathcal{M}, \beta}) \circ \text{bump-nterm}) \ ts \in \text{intrp-rel } (\text{bump-intrp } \mathcal{M}) \ n) \rangle$
proof –
have $\langle (\lambda t. \llbracket t \rrbracket^{\mathcal{M}, \beta}) = (\lambda t. \llbracket t \rrbracket^{\text{bump-intrp } \mathcal{M}, \beta}) \circ \text{bump-nterm} \rangle$
using *bumpterm* **by** *fastforce*
then have $\langle \text{map } (\lambda t. \llbracket t \rrbracket^{\mathcal{M}, \beta}) \ ts = \text{map } ((\lambda t. \llbracket t \rrbracket^{\text{bump-intrp } \mathcal{M}, \beta}) \circ \text{bump-nterm})$
 $\ ts \rangle$
by *simp*
then show *?thesis*
by (*metis bump-intrp-rel*)
qed

lemma *bumpform*: $\langle \mathcal{M}, \beta \models \varphi = (\text{bump-intrp } \mathcal{M}), \beta \models (\text{bump-form } \varphi) \rangle$
proof (*induct φ arbitrary: β*)
case *Bot*
then show *?case*

```

      unfolding bump-intrp-def by auto
next
  case (Atom x1 x2)
  then show ?case
    using bump-intrp-rel-holds by auto
next
  case (Implies  $\varphi_1 \varphi_2$ )
  then show ?case
    unfolding bump-intrp-def by auto
next
  case (Forall x1  $\varphi$ )
  have  $\langle (\forall a \in \text{dom } \mathcal{M}. (\text{bump-intrp } \mathcal{M}), \beta(x1 := a) \models \text{bump-form } \varphi) =$ 
     $(\forall a \in \text{dom } \mathcal{M}. \mathcal{M}, \beta(x1 := a) \models \varphi) \rangle$ 
    using Forall by presburger
  then show ?case
    by simp
qed

lemma functions-form-bumpform:  $\langle (f, m) \in \text{functions-form } (\text{bump-form } \varphi) \implies$ 
   $\exists k. (f = \text{numpair } 0 k) \wedge (k, m) \in \text{functions-form } \varphi \rangle$ 
proof (induct  $\varphi$ )
  case (Atom p ts)
  then have  $\langle \exists t \in \text{set } ts. (f, m) \in \text{functions-term } (\text{bump-nterm } t) \rangle$  by simp
  then obtain t where t-in:  $\langle t \in \text{set } ts \rangle$  and fm-in-t:  $\langle (f, m) \in \text{functions-term}$ 
     $(\text{bump-nterm } t) \rangle$ 
    by blast
  have  $\langle \exists k. f = \text{numpair } 0 k \wedge (k, m) \in \text{functions-term } t \rangle$ 
    using fm-in-t
  proof (induction t)
    case (Var x)
    then show ?case by auto
  next
    case (Fun g us)
    have t-in-disj:  $\langle \text{functions-term } (\text{bump-nterm } (\text{Fun } g us)) =$ 
       $\{((\text{numpair } 0 g), \text{length } us)\} \cup (\bigcup u \in \text{set } us. \text{functions-term } (\text{bump-nterm}$ 
       $u)) \rangle$ 
      by simp
    then show ?case
      proof (cases  $(f, m) = ((\text{numpair } 0 g), \text{length } us)$ )
        case True
        then show ?thesis by auto
      next
        case False
        then have  $\langle (f, m) \in (\bigcup u \in \text{set } us. \text{functions-term } (\text{bump-nterm } u)) \rangle$ 
          using t-in-disj
          using Fun.premis by blast
        then show ?thesis
          using Fun(1) by fastforce
      qed
  qed
qed

```

qed
then have $\langle \exists k. f = \text{numpair } 0 \ k \wedge (\exists x \in \text{set } ts. (k, m) \in \text{functions-term } x) \rangle$
using *t-in* **by** *blast*
then show *?case* **using** *Atom* **by** *simp*
qed *auto*

lemma *bumpform-interpretation*: $\langle \text{is-interpretation (language } \{\varphi\}) \ \mathcal{M} \implies \text{is-interpretation (language } \{\text{bump-form } \varphi\}) \ (\text{bump-intrp } \mathcal{M}) \rangle$
unfolding *is-interpretation-def* *language-def*
by (*metis bump-dom bump-intrp-fn fst-conv functions-form-bumpform lang-singleton language-def*)

fun *unbump-nterm* :: *nterm* \Rightarrow *nterm* **where**
 $\langle \text{unbump-nterm (Var } x) = \text{Var } x \rangle$
 $\mid \langle \text{unbump-nterm (Fun } f \ ts) = \text{Fun (numsnd } f) \ (\text{map unbump-nterm } ts) \rangle$

fun *unbump-form* :: *form* \Rightarrow *form* **where**
 $\langle \text{unbump-form } \perp = \perp \rangle$
 $\mid \langle \text{unbump-form (Atom } p \ ts) = \text{Atom } p \ (\text{map unbump-nterm } ts) \rangle$
 $\mid \langle \text{unbump-form } (\varphi \longrightarrow \psi) = (\text{unbump-form } \varphi) \longrightarrow (\text{unbump-form } \psi) \rangle$
 $\mid \langle \text{unbump-form } (\forall x. \varphi) = \forall x. (\text{unbump-form } \varphi) \rangle$

lemma *unbump-term [simp]*: $\text{unbump-nterm (bump-nterm } t) = t$
by (*induct t*) (*simp add: list.map-ident-strong*) $+$

lemma *unbump-form [simp]*: $\langle \text{unbump-form (bump-form } \varphi) = \varphi \rangle$
by (*induct* φ) (*simp add: list.map-ident-strong*) $+$

definition *unbump-intrp* :: *'m intrp* \Rightarrow *'m intrp* **where**
 $\langle \text{unbump-intrp } \mathcal{M} = \text{Abs-intrp (dom } \mathcal{M}, (\lambda k \text{ zs. (intrp-fn } \mathcal{M}) \ (\text{numpair } 0 \ k) \ \text{zs}), (\text{intrp-rel } \mathcal{M})) \rangle$

lemma *unbump-term-intrp*: $\langle \llbracket \text{bump-nterm } t \rrbracket^{\mathcal{M}, \beta} = \llbracket t \rrbracket^{\text{unbump-intrp } \mathcal{M}, \beta} \rangle$

proof (*induct t*)

case (*Fun f ts*)

then show *?case*

unfolding *unbump-intrp-def*

by (*smt (verit, best) bump-nterm.simps(2) concat-map eval.simps(2) intrp-fn-Abs-is-fst-snd*

intrp-is-struct list.map-cong0 struct-def)

qed *simp*

lemma *unbump-holds*: $\langle (\mathcal{M}, \beta \models \text{bump-form } \varphi) = (\text{unbump-intrp } \mathcal{M}, \beta \models \varphi) \rangle$

proof (*induct* φ *arbitrary:* β)

```

case Bot
then show ?case
  by auto
next
case (Atom p ts)
then show ?case
  unfolding unbump-intrp-def using bump-intrp-def bumpform dom-Abs-is-fst
functions-form-bumpform
  holds-indep-intrp-if intrp-fn-Abs-is-fst-snd intrp-is-struct intrp-rel-Abs-is-snd-snd
struct-def
  by (smt (verit, ccfv-SIG) prod-encode-inverse snd-conv)
next
case (Implies  $\varphi 1 \varphi 2$ )
then show ?case
  by auto
next
case (Forall x  $\varphi$ )
then show ?case
  by (smt (verit, best) bump-form.simps(4) dom-Abs-is-fst holds.simps(4) in-
trp-is-struct
  struct-def unbump-intrp-def)
qed

```

abbreviation *numlist where*
⟨numlist ns ≡ list-encode ns⟩

fun *num-of-term :: nterm ⇒ nat where*
⟨num-of-term (Var x) = numpair 0 x
 | *⟨num-of-term (Fun f ts) = numpair 1 (numpair f (numlist (map num-of-term*
ts)))⟩

lemma *term-induct2:*

```

( $\bigwedge x y. P (Var x) (Var y)$ )
   $\implies (\bigwedge g us. P (Var x) (Fun g us))$ 
   $\implies (\bigwedge f ts y. P (Fun f ts) (Var y))$ 
   $\implies (\bigwedge f ts g us. (\bigwedge p q. p \in set ts \implies q \in set us \implies P p q) \implies P (Fun f ts)$ 
(Fun g us))
   $\implies P t1 t2$ 

```

proof (*induction t2 arbitrary: t1*)

```

case (Var y)
then show ?case by (metis is-FunE is-VarE)
next
case (Fun g us)
then have ⟨p ∈ set ts ⟹ q ∈ set us ⟹ P p q⟩ for ts p q
  by blast
then show ?case
  using Fun by (metis is-FunE is-VarE)
qed

```

```

lemma num-of-term-inj: ⟨num-of-term s = num-of-term t ⟷ s = t⟩
proof (induction s t rule: term-induct2)
  case (4 f ts g us)
  have ⟨(Fun f ts = Fun g us) ⟹ num-of-term (Fun f ts) = num-of-term (Fun g
us)⟩
  by auto
  moreover {
    assume ⟨num-of-term (Fun f ts) = num-of-term (Fun g us)⟩
    then have ⟨numpair f (numlist (map num-of-term ts)) = numpair g (numlist
(map num-of-term us))⟩
    by auto
    then have fun-eq: ⟨f = g⟩ and nl-eq: ⟨numlist (map num-of-term ts) = (numlist
(map num-of-term us))⟩
    by auto
    then have map num-of-term ts = map num-of-term us
    using list-encode-eq by blast
    then have args-eq: ⟨ts = us⟩
    using 4 by (metis list.inj-map-strong)
    have ⟨Fun f ts = Fun g us⟩
    using fun-eq args-eq by simp
  }
  ultimately show ?case by auto
qed auto

```

```

fun num-of-form :: form ⇒ nat where
  ⟨num-of-form ⊥ = numpair 0 0⟩
| ⟨num-of-form (Atom p ts) = numpair 1 (numpair p (numlist (map num-of-term
ts)))⟩
| ⟨num-of-form (φ ⟶ ψ) = numpair 2 (numpair (num-of-form φ) (num-of-form
ψ))⟩
| ⟨num-of-form (∀ x. φ) = numpair 3 (numpair x (num-of-form φ))⟩

```

```

lemma numlist-num-of-term: ⟨numlist (map num-of-term ts) = (numlist (map
num-of-term us)) ≡ ts = us⟩
  by (smt (verit) list.inj-map-strong list-encode-eq num-of-term-inj)

```

```

lemma num-of-form-inj: ⟨num-of-form φ = num-of-form ψ ⟷ φ = ψ⟩
proof
  show ⟨num-of-form φ = num-of-form ψ ⟹ φ = ψ⟩
  proof (induct φ arbitrary: ψ rule: num-of-form.induct)
    case 1
    then show ?case
    using num-of-form.elims num-of-form.simps(1) zero-neq-numeral zero-neq-one
by (metis prod.sel(1) prod-encode-inverse)
  next
    case (2 p ts)
    then show ?case

```

```

proof (cases  $\psi$ )
  case Bot
  then show ?thesis
    using 2 num-of-term-inj by fastforce
next
  case (Atom q us)
  then show ?thesis
    using 2 by (simp add: numlist-num-of-term)
next
  case (Implies  $\psi_1 \psi_2$ )
  then show ?thesis
    using 2 by simp
next
  case (Forall y  $\psi_1$ )
  then have  $\langle \exists k. \text{num-of-form } \psi = \text{numpair } 3 k \rangle$ 
    by auto
  moreover have  $\langle \exists k'. \text{num-of-form } (\text{Atom } p \text{ ts}) = \text{numpair } 1 k' \rangle$ 
    by auto
  ultimately show ?thesis using 2 by force
qed
next
  case ( $\exists \varphi_1 \varphi_2$ )
  then show ?case
    by (smt (verit, best) One-nat-def Pair-inject Suc-eq-numeral form.distinct(11)
      form.distinct(7) form.sel(3) form.sel(4) nat.simps(3) num-of-form.elims
numeral-3-eq-3
numerals(2) prod-encode-eq)
next
  case ( $\exists x \varphi_1$ )
  then show ?case
    by (smt (verit, ccfv-SIG) One-nat-def Zero-neq-Suc num-of-form.elims num-of-form.simps(4)
numeral-3-eq-3 numerals(2) old.nat.inject old.prod.inject prod-encode-eq)
qed
qed auto

consts term-of-num :: nat  $\Rightarrow$  nterm
specification (term-of-num)  $\langle \forall n. \text{term-of-num } n = (\text{THE } t. \text{num-of-term } t = n) \rangle$ 
  using num-of-term-inj by force

lemma term-of-num-of-term [simp]:  $\langle \text{term-of-num}(\text{num-of-term } t) = t \rangle$ 
  using num-of-term-inj HOL.nitpick-choice-spec by auto

consts form-of-num :: nat  $\Rightarrow$  form
specification (form-of-num)  $\langle \forall n. \text{form-of-num } n = (\text{THE } \varphi. \text{num-of-form } \varphi = n) \rangle$ 
  using num-of-form-inj by force

```

```

lemma form-of-num-of-form [simp]: ⟨form-of-num (num-of-form  $\varphi$ ) =  $\varphi$ ⟩
  using num-of-form-inj HOL.nitpick-choice-spec by auto

end

theory Skolem-Normal-Form
imports
  Prenex-Normal-Form
  Bumping
begin

lemma witness-imp-holds-exists:
  assumes ⟨is-interpretation (functions-term  $t$ , preds) ( $I :: (\text{nat}, \text{nat})$  term intrp)⟩
and
  ⟨is-valuation  $I$   $\beta$ ⟩ and
  ⟨ $I, \beta \models (\varphi \cdot_{fm} (\text{subst } x \ t))$ ⟩
shows ⟨ $I, \beta \models (\exists x. \varphi)$ ⟩
proof –
  have ⟨ $\lambda v. \llbracket \text{subst } x \ t \ v \rrbracket^{I, \beta} = \beta(x := \llbracket t \rrbracket^{I, \beta})$ ⟩
  proof –
    have  $\forall n. \llbracket \text{subst } x \ t \ n \rrbracket^{I, \beta} = (\beta(x := \llbracket t \rrbracket^{I, \beta})) \ n$ 
      by (simp add: subst-def)
    then show ?thesis
      by blast
  qed
moreover have ⟨ $\llbracket t \rrbracket^{I, \beta} \in \text{dom } I$ ⟩
  using assms(1)
proof (induct  $t$ )
  case (Var  $x$ )
  then show ?case
    using assms(2) by (auto simp: is-valuation-def)
next
  case (Fun  $f \ ts$ )
  then have ⟨ $u \in \text{set } ts \implies \llbracket u \rrbracket^{I, \beta} \in \text{dom } I$ ⟩ for  $u$ 
    by (smt (verit) UN-I Un-iff fst-conv funas-term.simps(2) is-interpretation-def
list.set-map)
  then show ?case
    using eval.simps(2) fst-conv imageE length-map list.set-map list-all-set assms(2)
    unfolding is-valuation-def
    by (smt (verit, best) Fun.premis Un-insert-left funas-term.simps(2) insert-iff
is-interpretation-def subsetI)
qed
ultimately have ⟨ $\exists a \in \text{dom } I. I, \beta(x := a) \models \varphi$ ⟩
  using assms(3) swap-subst-eval[of  $I$   $\beta$   $\varphi$   $\text{subst } x \ t$ ] by auto
then show ?thesis
  using holds-exists by blast
qed

```

definition *skolem1* :: *nat* \Rightarrow *nat* \Rightarrow *form* \Rightarrow *form* **where**
 $\langle \text{skolem1 } f \ x \ \varphi = \varphi \cdot_{fm} (\text{subst } x \ (\text{Fun } f \ (\text{map } \text{Var} \ (\text{sorted-list-of-set} \ (FV \ (\exists x. \varphi)))))) \rangle$

lemma *fvt-var-x-simp*:

$\langle FVT \ (\text{Var } x \cdot \text{subst } x \ (\text{Fun } f \ (\text{map } \text{Var} \ (\text{sorted-list-of-set} \ (FV \ \varphi - \{x\})))) = FV \ \varphi - \{x\} \rangle$

proof –

have *remove-list*:

$\langle \text{set} \ (\text{map } \text{Var} \ (\text{sorted-list-of-set} \ (FV \ \varphi - \{x\}))) = \text{Var} \ ' \ (FV \ \varphi - \{x\}) \rangle$

using *set-map set-sorted-list-of-set* **using** *finite-FV* **by** *auto*

have $\langle FVT \ (\text{Var } x \cdot \text{subst } x \ (\text{Fun } f \ (\text{map } \text{Var} \ (\text{sorted-list-of-set} \ (FV \ \varphi - \{x\})))) =$

$= FVT \ (\text{Fun } f \ (\text{map } \text{Var} \ (\text{sorted-list-of-set} \ (FV \ \varphi - \{x\}))) \rangle$

by *simp*

also have $\langle \dots = \bigcup \ (FVT \ ' \ \text{set} \ (\text{map } \text{Var} \ (\text{sorted-list-of-set} \ (FV \ \varphi - \{x\})))) \rangle$

using *term.set(4)* **by** *metis*

also have $\langle \dots = \bigcup \ (FVT \ ' \ \text{Var} \ ' \ (FV \ \varphi - \{x\})) \rangle$

using *remove-list* **by** *auto*

also have $\langle \dots = FV \ \varphi - \{x\} \rangle$

by *force*

finally show *?thesis* .

qed

lemma *holds-indep-intrp-if2*:

fixes *I I'* :: '*a intrp*

shows

$\langle \llbracket I, \beta \models \varphi; \text{dom } I = \text{dom } I'; \bigwedge p. \text{intrp-rel } I \ p = \text{intrp-rel } I' \ p; \rrbracket$

$\llbracket f \ zs. (f, \text{length } zs) \in \text{functions-form } \varphi \implies \text{intrp-fn } I \ f \ zs = \text{intrp-fn } I' \ f \ zs \rrbracket \implies$

$\langle I', \beta \models \varphi \rangle$

using *holds-indep-intrp-if* **by** *blast*

lemma *fun-upds-prop*: $\langle \text{length } xs = \text{length } zs \implies \forall z \in \text{set } zs. P \ z \implies \forall v. P \ (g \ v) \implies \forall v. P \ ((\text{foldr} \ (\lambda kv \ f. \text{fun-upd } f \ (\text{fst } kv) \ (\text{snd } kv)) \ (\text{zip } xs \ zs) \ g) \ v) \rangle$

proof (*induction zs arbitrary: xs g*)

case *Nil*

then show *?case*

by *simp*

next

case (*Cons a zs*)

obtain *x xsp* **where** *xs-is*: $\langle xs = x \ \# \ xsp \rangle$

using *Cons(2)* **by** (*metis length-Suc-conv*)

with *Cons* **show** *?case*

by *auto*

qed

lemma $\langle \{z. \exists y. y \in FV \varphi \wedge z \in \text{functions-term } (Var\ y \cdot \text{subst } x\ t)\} = \text{functions-term } t \vee$

$\{z. \exists y. y \in FV \varphi \wedge z \in \text{functions-term } (Var\ y \cdot \text{subst } x\ t)\} = \{\}\rangle$

proof –

have $\langle y \neq x \implies \text{functions-term } (Var\ y \cdot \text{subst } x\ t) = \{\}\rangle$ **for** y

by (*simp add: subst-def*)

moreover have $\langle y = x \implies \text{functions-term } (Var\ y \cdot \text{subst } x\ t) = \text{functions-term } t \rangle$ **for** y

by *simp*

ultimately show *?thesis*

by *blast*

qed

lemma *func-form-subst*: $\langle x \in FV \varphi \implies (f, \text{length } ts) \in \text{functions-form } (\varphi \cdot_{fm} \text{subst } x\ (Fun\ f\ ts)) \rangle$

proof (*induction* φ *rule: functions-form.induct*)

case 1

then show *?case* **by** *simp*

next

case (2 $p\ ts$)

then show *?case*

by (*metis (no-types, lifting) UnCI eval-term.simps(1) formsubst-functions-form*

funas-term.simps(2) mem-Collect-eq singletonI subst-simps(1))

next

case (3 $\varphi\ \psi$)

then show *?case*

by *auto*

next

case (4 $y\ \varphi$)

then show *?case*

by (*metis (no-types, lifting) UnI2 Un-commute eval-term.simps(1) formsubst-functions-form*

funas-term.simps(2) mem-Collect-eq singletonI subst-simps(1))

qed

lemma *holds-formsubst*:

$M, \beta \models (p \cdot_{fm} i) \iff M, (\lambda t. \llbracket t \rrbracket^{M, \beta}) \circ i \models p$

by (*simp add: holds-indep- β -if swap-subst-eval*)

lemma *holds-formsubst1*:

$M, \beta \models (p \cdot_{fm} Var(x:=t)) \iff M, \beta(x := \llbracket t \rrbracket^{M, \beta}) \models p$

by (*simp add: holds-indep- β -if swap-subst-eval*)

lemma *holds-formsubst2*:

$M, \beta \models (p \cdot_{fm} \text{subst } x \ t) \longleftrightarrow M, \beta(x := \llbracket t \rrbracket^{M, \beta}) \models p$
by (*simp add: holds-formsubst1 subst-def*)

lemma *size-nonzero* [*simp*]: *size fm > 0*
by (*induction fm*) *auto*

lemma

assumes *prenex-ex-phi*: $\langle \text{is-prenex } (\exists x. \varphi) \rangle$
and *notin-ff*: $\langle \neg (f, \text{card } (FV (\exists x. \varphi))) \in \text{functions-form } (\exists x. \varphi) \rangle$
shows *holds-skolem1a*: *is-prenex* (*skolem1 f x φ*) (**is** ?*A*)
and *holds-skolem1b*: $FV (\text{skolem1 } f \ x \ \varphi) = FV (\exists x. \varphi)$ (**is** ?*B*)
and *holds-skolem1c*:
 $\text{Prenex-Normal-Form.size } (\text{skolem1 } f \ x \ \varphi) < \text{Prenex-Normal-Form.size } (\exists x. \varphi)$ (**is** ?*C*)
and *holds-skolem1d*: $\text{predicates-form } (\text{skolem1 } f \ x \ \varphi) = \text{predicates-form } (\exists x. \varphi)$ (**is** ?*D*)
and *holds-skolem1e*: $\text{functions-form } (\exists x. \varphi) \subseteq \text{functions-form } (\text{skolem1 } f \ x \ \varphi)$ (**is** ?*E*)
and *holds-skolem1f*: $\text{functions-form } (\text{skolem1 } f \ x \ \varphi) \subseteq (f, \text{card } (FV (\exists x. \varphi))) \triangleright \text{functions-form } (\exists x. \varphi)$ (**is** ?*F*)

proof –

show ?*A*

by (*metis form.inject(2) form.inject(3) prenex-ex-phi prenex-formsubst prenex-imp*)

qfree-no-quantif skolem1-def)

show ?*B*

proof

show $\langle FV (\text{skolem1 } f \ x \ \varphi) \subseteq FV (\exists x. \varphi) \rangle$

proof

fix *z*

assume $\langle z \in FV (\text{skolem1 } f \ x \ \varphi) \rangle$

then obtain *y* **where** *y-in*: $\langle y \in FV \varphi \rangle$ **and**

z-in: $\langle z \in FVT (\text{Var } y \cdot \text{subst } x \ (\text{Fun } f \ (\text{map } \text{Var } (\text{sorted-list-of-set } (FV \varphi - \{x\})))))) \rangle$

unfolding *skolem1-def* **using** *formsubst-fv FV-exists* **by** *auto*

then have *neq-x*: $\langle y \neq x \implies z \in FV \varphi - \{x\} \rangle$

by (*simp add: subst-def*)

then show $\langle z \in FV (\exists x. \varphi) \rangle$

using *fvt-var-x-simp z-in* **by** *force*

qed

next

show $\langle FV (\exists x. \varphi) \subseteq FV (\text{skolem1 } f \ x \ \varphi) \rangle$

proof

fix *z*

assume *z-in*: $\langle z \in FV (\exists x. \varphi) \rangle$

then have $\langle FVT (\text{Var } z \cdot \text{subst } x \ (\text{Fun } f \ (\text{map } \text{Var } (\text{sorted-list-of-set } (FV (\exists x. \varphi)))))) = \{z\} \rangle$

by (*simp add: subst-def*)

```

    then show ⟨z ∈ FV (skolem1 f x φ)⟩
      unfolding skolem1-def using z-in formsubst-fv by auto
    qed
  qed
  show ?C
    by (simp add: size-indep-subst skolem1-def)
  show ?D
    by (simp add: formsubst-predicates skolem1-def)
  show ?E
    by (simp add: formsubst-functions-form skolem1-def)
  show ?F
  proof
    fix g
    assume g-in: ⟨g ∈ functions-form (skolem1 f x φ)⟩
    then have ⟨g ∈ functions-form φ ∪ {g. ∃ y. y ∈ FV φ ∧
      g ∈ functions-term (Var y · subst x (Fun f (map Var (sorted-list-of-set (FV
      ((∃ x. φ))))))⟩
      unfolding skolem1-def using formsubst-functions-form
      by blast
    moreover have ⟨{g. ∃ y ∈ FV φ.
      g ∈ functions-term (Var y · subst x (Fun f (map Var (sorted-list-of-set (FV
      ((∃ x. φ))))))}
      ⊆ (f, card (FV (∃ x. φ))) ▷ functions-form φ⟩
  proof
    fix h
    assume ⟨h ∈ {g. ∃ y ∈ FV φ. g ∈ functions-term (Var y · subst x (Fun f
      (map Var
        (sorted-list-of-set (FV ((∃ x. φ))))))}⟩
    then obtain y where y-in: ⟨y ∈ FV φ⟩ and h-in:
      ⟨h ∈ functions-term (Var y · subst x (Fun f (map Var (sorted-list-of-set (FV
      ((∃ x. φ))))))⟩
    by blast
    then have y-neq-x-case: ⟨y ≠ x ⟹ h ∈ functions-form φ⟩
      by (simp add: subst-def)
    have ⟨functions-term (Var x · subst x (Fun f (map Var (sorted-list-of-set (FV
      ((∃ x. φ)))))) =
      functions-term (Fun f (map Var (sorted-list-of-set (FV ((∃ x. φ))))))⟩
      by (simp add: subst-def)
    have ⟨functions-term (Fun f (map Var (sorted-list-of-set (FV ((∃ x. φ))))))
    =
      {(f, card (FV (∃ x. φ)))}⟩
      by auto
    then have y-eq-x-case: ⟨y = x ⟹ h = (f, card (FV (∃ x. φ)))⟩
      using y-in h-in by auto
    show ⟨h ∈ (f, card (FV (∃ x. φ))) ▷ functions-form φ⟩
      using y-neq-x-case y-eq-x-case by blast
  qed
  ultimately show ⟨g ∈ (f, card (FV (∃ x. φ))) ▷ functions-form (∃ x. φ)⟩
    by auto

```

qed
qed

definition *define-fn* $\equiv \lambda FN f n h. \lambda g zs. \text{if } g=f \wedge \text{length } zs = n \text{ then } h \text{ } zs \text{ else } FN \text{ } g \text{ } zs$

lemma *holds-skolem1g*:

assumes *prenex-ex-phi*: $\langle \text{is-prenex } (\exists x. \varphi) \rangle$
and *notin-ff*: $\langle \neg (f, \text{card } (FV (\exists x. \varphi))) \in \text{functions-form } (\exists x. \varphi) \rangle$
fixes *I* :: 'a intrp
assumes *interp-I*: *is-interpretation* (language $\{\varphi\}$) *I*
and *nempty-I*: $\text{dom } I \neq \{\}$
and *valid*: $\bigwedge \beta. \text{is-valuation } I \beta \implies I, \beta \models (\exists x. \varphi)$
obtains *M* **where** $\text{dom } M = \text{dom } I$
 $\text{intrp-rel } M = \text{intrp-rel } I$
 $\bigwedge g zs. g \neq f \vee \text{length } zs \neq \text{card } (FV (\exists x. \varphi)) \implies \text{intrp-fn } M g zs$
 $= \text{intrp-fn } I g zs$
is-interpretation (language $\{\text{skolem1 } f x \varphi\}$) *M*
 $\bigwedge \beta. \text{is-valuation } M \beta \implies M, \beta \models \text{skolem1 } f x \varphi$

proof –

have *ex-a-mod-phi*: $\exists a \in \text{dom } I. I, \beta(x := a) \models \varphi$
if $\forall v. \beta v \in \text{dom } I$ **for** β
using that *FOL-Semantics.holds-exists is-valuation-def valid by blast*
define *intrp-f* **where** — Using *fold* instead causes complications
 $\langle \text{intrp-f} \equiv \lambda zs. \text{foldr } (\lambda kv f. \text{fun-upd } f (fst kv) (snd kv))$
 $(\text{zip } (\text{sorted-list-of-set } (FV (\exists x. \varphi))) zs) (\lambda z. \text{SOME } c. c \in$
 $\text{dom } I) \rangle$
define *thex* **where** $\text{thex} \equiv \lambda zs. \text{SOME } a. a \in \text{dom } I \wedge (I, (\text{intrp-f } zs)(x:=a) \models$
 $\varphi)$
define *FN* **where** $FN \equiv \text{define-fn } (\text{intrp-fn } I) f (\text{card } (FV (\exists x. \varphi))) \text{thex}$

define *M* :: 'a intrp **where** $\langle M = \text{Abs-intrp } (\text{dom } I, FN, \text{intrp-rel } I) \rangle$

show *?thesis*

proof

show *dom-M-I-eq*: $\langle \text{dom } M = \text{dom } I \rangle$
unfolding *M-def* **by** *simp*
show *intrp-rel-eq*: $\langle \text{intrp-rel } M = \text{intrp-rel } I \rangle$
unfolding *M-def* **by** *simp*
show *intrp-fn-eq*: $\bigwedge g zs. g \neq f \vee \text{length } zs \neq \text{card } (FV (\exists x. \varphi)) \implies$
 $\text{intrp-fn } M g zs = \text{intrp-fn } I g zs$
unfolding *M-def FN-def define-fn-def*
by *fastforce*

have *in-dom-I*: $\langle \text{intrp-fn } M f zs \in \text{dom } I \rangle$

if *len-eq*: $\langle \text{length } zs = \text{card } (FV \varphi - \{x\}) \rangle$ **and** *zs-in*: $\langle \text{set } zs \subseteq \text{dom } M \rangle$
for *zs*

proof –

have *len-eq2*: $\langle \text{length } (\text{sorted-list-of-set } (FV (\exists x. \varphi))) = \text{length } zs \rangle$
using *len-eq* **by** *simp*
have *zs-in2*: $\langle \forall z \in \text{set } zs. z \in \text{dom } I \rangle$

```

    using dom-M-I-eq zs-in by force
  have fn-is-thex: ⟨(intrap-fn M) f zs = thex zs⟩
    using len-eq by (auto simp: M-def FN-def define-fn-def)
  have ⟨∀ v. (intrap-f zs) v ∈ dom I⟩
    using fun-upds-prop[OF len-eq2 zs-in2] nempty-I some-in-eq unfolding
intrap-f-def
    by (metis (mono-tags))
  then show ⟨intrap-fn M f zs ∈ dom I⟩
    using nempty-I ex-a-mod-phi interp-I unfolding is-interpretation-def
    by (metis (mono-tags, lifting) fn-is-thex someI-ex thex-def)
qed
show ⟨is-interpretation (language {skolem1 f x φ}) M⟩
  unfolding is-interpretation-def
  proof (intro strip)
    fix g l
    assume §: ⟨(g, length l) ∈ fst (language {skolem1 f x φ}) ∧ set l ⊆ dom M⟩
    then have ⟨(g, length l) ∈ fst (language {φ}) ∨ (g, length l) = (f, card (FV
φ - {x}))⟩
      using holds-skolem1f lang-singleton notin-ff prenex-ex-phi by auto
    with § show ⟨intrap-fn M g l ∈ dom M⟩
      by (metis FV-exists dom-M-I-eq in-dom-I interp-I intrap-fn-eq is-interpretation-def
prod.inject)
    qed
  show M,β ⊨ skolem1 f x φ if is-valuation M β for β
  proof -
    have M,β(x:=thex (map β (sorted-list-of-set(FV(∃ x. φ)))) ⊨ φ
    proof (rule holds-indep-intrap-if2)
      have I, (intrap-f (map β (sorted-list-of-set(FV(∃ x. φ)))))(x:=a) ⊨ φ ↔
I, β(x:=a) ⊨ φ
        for a
      proof (intro holds-indep-β-if strip)
        fix v
        assume v ∈ FV φ
        then have v=x ∨ v ∈ FV (∃ x. φ)
          using FV-exists by blast
        moreover have
          foldr (λkv f. f(fst kv := snd kv)) (zip vs (map β vs)) (λz. SOME c. c ∈
dom I) w = β w
          if w ∈ set vs set vs ⊆ FV (∃ x. φ) for vs w
          using that by (induction vs) auto
        ultimately
          show ((intrap-f (map β (sorted-list-of-set (FV (∃ x. φ)))))(x := a)) v =
(β(x := a)) v
          using finite-FV intrap-f-def by auto
        qed
      then show I,β (x := thex (map β (sorted-list-of-set (FV (∃ x. φ)))) ⊨ φ
        by (metis (mono-tags, lifting) dom-M-I-eq ex-a-mod-phi is-valuation-def
that thex-def
verit-sko-ex')

```

```

show  $\text{dom } I = \text{dom } M$ 
  using dom-M-I-eq by auto
show  $\bigwedge p. \text{intrp-rel } I \ p = \text{intrp-rel } M \ p$ 
  using intrp-rel-eq by auto
show  $\bigwedge f \text{ zs}. (f, \text{length } \text{zs}) \in \text{functions-form } \varphi \implies \text{intrp-fn } I \ f \ \text{zs} = \text{intrp-fn}$ 
M f zs
  using functions-form.simps notin-ff intrp-fn-eq
  by (metis sup-bot.right-neutral)
qed
moreover have  $\text{FN } f \ (\text{map } \beta \ (\text{sorted-list-of-set } (FV \ \varphi - \{x\}))) =$ 
   $\text{thex } (\text{map } \beta \ (\text{sorted-list-of-set } (FV \ \varphi - \{x\})))$ 
  by (simp add: FN-def define-fn-def)
ultimately show ?thesis
  by (simp add: holds-formsubst2 skolem1-def M-def o-def)
qed
qed
qed

lemma holds-skolem1h:
  assumes prenex-ex-phi:  $\langle \text{is-prenex } (\exists x. \varphi) \rangle$  and  $\langle \neg (f, \text{card } (FV \ (\exists x. \varphi))) \in$ 
functions-form } (\exists x. \varphi) \rangle
  assumes is-intrp: is-interpretation (language  $\{ \text{skolem1 } f \ x \ \varphi \}$ ) N
  and nempty-N:  $\text{dom } N \neq \{ \}$ 
  and is-val: is-valuation N  $\beta$ 
  and skol-holds:  $N, \beta \models \text{skolem1 } f \ x \ \varphi$ 
  shows  $N, \beta \models (\exists x. \varphi)$ 
proof -
  have  $\langle \exists a \in \text{dom } N. N, \beta(x := a) \models \varphi \rangle$ 
proof -
  have  $\langle N, (\lambda v. \llbracket \text{subst } x \ (\text{Fun } f \ (\text{map } \text{Var } (\text{sorted-list-of-set } (FV \ (\exists x. \varphi)))) \rrbracket$ 
 $v \rrbracket^{N, \beta}) \models \varphi \rangle$ 
  by (metis skol-holds skolem1-def swap-subst-eval)
  then have holds-eval-f:  $\langle N, \beta(x := \llbracket \text{Fun } f \ (\text{map } \text{Var } (\text{sorted-list-of-set } (FV$ 
 $(\exists x. \varphi))) \rrbracket^{N, \beta}) \models \varphi \rangle$ 
  by (smt (verit, best) eval.simps(1) fun-upd-other fun-upd-same holds-indep-beta-if
subst-def)
  show  $\langle \exists a \in \text{dom } N. N, \beta(x := a) \models \varphi \rangle$ 
proof (cases  $\langle x \in FV \ \varphi \rangle$ )
  case True
  have eval-to-intrp:  $\langle \llbracket \text{Fun } f \ (\text{map } \text{Var } (\text{sorted-list-of-set } (FV \ (\exists x. \varphi)))) \rrbracket^{N, \beta}$ 
  =
   $\text{intrp-fn } N \ f \ \llbracket [t]^{N, \beta}. t \leftarrow \text{map } \text{Var } (\text{sorted-list-of-set } (FV \ (\exists x. \varphi))) \rrbracket \rangle$ 
  by simp

  have  $\langle \llbracket [t]^{N, \beta}. t \leftarrow \text{map } \text{Var } (\text{sorted-list-of-set } (FV \ (\exists x. \varphi))) \rrbracket =$ 
   $\llbracket \beta \ t. t \leftarrow (\text{sorted-list-of-set } (FV \ (\exists x. \varphi))) \rrbracket \rangle$ 
  by auto
  then have all-in-dom:  $\langle \text{set } \llbracket [t]^{N, \beta}. t \leftarrow \text{map } \text{Var } (\text{sorted-list-of-set } (FV \ (\exists x.$ 
 $\varphi))) \rrbracket \subseteq \text{dom } N \rangle$ 

```

```

using is-val by (auto simp: is-valuation-def)
have  $\langle (f, \text{length } [\![t]\!]^{N,\beta}. t \leftarrow \text{map Var } (\text{sorted-list-of-set } (FV (\exists x. \varphi)))) \in$ 
   $\text{functions-form } (\varphi \cdot_{f_m} \text{subst } x (\text{Fun } f (\text{map Var } (\text{sorted-list-of-set } (FV (\exists x. \varphi)))))) \rangle$ 
using func-form-subst[OF True] is-intrp lang-singleton
unfolding skolem1-def is-interpretation-def by (metis length-map)
then have  $\langle [\![\text{Fun } f (\text{map Var } (\text{sorted-list-of-set } (FV (\exists x. \varphi)))]\!]^{N,\beta} \in \text{dom}$ 
   $N \rangle$ 
using is-intrp eval-to-intrp all-in-dom unfolding is-interpretation-def
skolem1-def
by (metis fst-conv lang-singleton)
then show ?thesis
using holds-eval-f by blast
next
case False
obtain a where a-in:  $\langle a \in \text{dom } N \rangle$ 
using nempty-N by blast
then have  $\langle N, \beta(x := a) \models \varphi \rangle$ 
using holds-eval-f False by (metis fun-upd-other holds-indep-beta-if)
then show ?thesis
using a-in by blast
qed
qed
then show ?thesis
by simp
qed

```

lemma *holds-skolem1*:

```

assumes  $\langle \text{is-prenex } (\exists x. \varphi) \rangle$  and  $\langle \neg (f, \text{card } (FV (\exists x. \varphi))) \in \text{functions-form}$ 
   $(\exists x. \varphi) \rangle$ 
shows  $\langle \text{is-prenex } (\text{skolem1 } f x \varphi) \wedge$ 
   $FV (\text{skolem1 } f x \varphi) = FV (\exists x. \varphi) \wedge$ 
   $\text{size } (\text{skolem1 } f x \varphi) < \text{size } (\exists x. \varphi) \wedge$ 
   $\text{predicates-form } (\text{skolem1 } f x \varphi) = \text{predicates-form } (\exists x. \varphi) \wedge$ 
   $\text{functions-form } (\exists x. \varphi) \subseteq \text{functions-form } (\text{skolem1 } f x \varphi) \wedge$ 
   $\text{functions-form } (\text{skolem1 } f x \varphi) \subseteq \text{insert } (f, \text{card } (FV (\exists x. \varphi))) (\text{functions-form}$ 
   $(\exists x. \varphi)) \wedge$ 
   $(\forall (I :: 'a \text{ intrp}). \text{is-interpretation } (\text{language } \{\varphi\}) I \wedge$ 
   $\neg (\text{dom } I = \{\}) \wedge$ 
   $(\forall \beta. \text{is-valuation } I \beta \longrightarrow (I, \beta \models (\exists x. \varphi))) \longrightarrow$ 
   $(\exists (M :: 'a \text{ intrp}). \text{dom } M = \text{dom } I \wedge$ 
   $\text{intrp-rel } M = \text{intrp-rel } I \wedge$ 
   $(\forall g \text{ zs}. \neg g=f \vee \neg(\text{length } \text{zs} = \text{card } (FV (\exists x. \varphi))) \longrightarrow \text{intrp-fn } M g \text{ zs} =$ 
   $\text{intrp-fn } I g \text{ zs}) \wedge$ 
   $\text{is-interpretation } (\text{language } \{\text{skolem1 } f x \varphi\}) M \wedge$ 
   $(\forall \beta. \text{is-valuation } M \beta \longrightarrow (M, \beta \models (\text{skolem1 } f x \varphi))) \rangle \wedge$ 
   $(\forall (N :: 'a \text{ intrp}). \text{is-interpretation } (\text{language } \{\text{skolem1 } f x \varphi\}) N \wedge$ 
   $\neg (\text{dom } N = \{\}) \longrightarrow$ 
   $(\forall \beta. \text{is-valuation } N \beta \wedge (N, \beta \models (\text{skolem1 } f x \varphi)) \longrightarrow (N, \beta \models (\exists x. \varphi)))$ 

```

\rangle
by (*smt* (*verit*, *ccfv-SIG*) *assms* *holds-skolem1a* *holds-skolem1b* *holds-skolem1c* *holds-skolem1d* *holds-skolem1e* *holds-skolem1f* *holds-skolem1g* *holds-skolem1h*)

lemma *skolems-ex*: $\langle \exists \text{skolems}. \forall \varphi. \text{skolems } \varphi = (\lambda k. \text{ppat } (\lambda x \psi. \forall x. (\text{skolems } \psi k)))$

$(\lambda x \psi. \text{skolems } (\text{skolem1 } (\text{numpair } J k) x \psi) (\text{Suc } k)) (\lambda \psi. \psi) \varphi \rangle$

proof (*intro* *size-rec* *strip*)

fix *skolems* :: *form* \Rightarrow *nat* \Rightarrow *form* **and** *g* φ

assume *IH*: $\langle \forall z. \text{size } z < \text{size } \varphi \longrightarrow \text{skolems } z = g z \rangle$

show $\langle \lambda k.$

$\text{ppat } (\lambda x \psi. \forall x. \text{skolems } \psi k) (\lambda x \psi. \text{skolems } (\text{skolem1 } (\text{numpair } J k) x \psi) (\text{Suc } k)) (\lambda \psi. \psi) \varphi =$

$(\lambda k. \text{ppat } (\lambda x \psi. \forall x. g \psi k) (\lambda x \psi. g (\text{skolem1 } (\text{numpair } J k) x \psi) (\text{Suc } k)) (\lambda \psi. \psi) \varphi)$

proof (*cases* $\exists x \varphi'. \varphi = \forall x. \varphi'$)

case *True*

then obtain $x \varphi'$ **where** *phi-is*: $\varphi = \forall x. \varphi'$

by *blast*

then have *smaller*: $\langle \text{size } (\varphi' \cdot_{fm} \sigma) < \text{size } \varphi \rangle$ **for** σ

using *size-indep-subst* **by** *simp*

have *ppat-to-skol*: $\langle \text{ppat } (\lambda x \psi. \forall x. (\text{skolems } \psi k))$

$(\lambda x \psi. \text{skolems } (\text{skolem1 } (\text{numpair } J k) x \psi) (\text{Suc } k)) (\lambda \psi. \psi) \varphi = (\forall x. \text{skolems } \varphi' k) \rangle$ **for** k

unfolding *ppat-def* **by** (*simp* *add*: *phi-is*)

have *skol-to-g*: $\langle (\forall x. \text{skolems } \varphi' k) = (\forall x. g \varphi' k) \rangle$ **for** k

using *IH* *smaller* **by** (*simp* *add*: *phi-is*)

have *g-to-ppat*: $\langle (\forall x. g \varphi' k) =$

$\text{ppat } (\lambda x \psi. \forall x. g \psi k) (\lambda x \psi. g (\text{skolem1 } (\text{numpair } J k) x \psi) (\text{Suc } k)) (\lambda \psi. \psi) \varphi \rangle$ **for** k

unfolding *ppat-def* **using** *phi-is* **by** *simp*

show *?thesis*

using *ppat-to-skol* *skol-to-g* *g-to-ppat* **by** *auto*

next

case *False*

assume *falseAll*: $\langle \neg (\exists x \varphi'. \varphi = \forall x. \varphi') \rangle$

then show *?thesis*

proof (*cases* $\exists x \varphi'. \varphi = \exists x. \varphi'$)

case *True*

then obtain $x \varphi'$ **where** *phi-is*: $\varphi = \exists x. \varphi'$

by *blast*

then have *smaller*: $\langle \text{size } (\varphi' \cdot_{fm} \sigma) < \text{size } \varphi \rangle$ **for** σ

using *size-indep-subst* **by** *simp*

have *ppat-to-skol*: $\langle \text{ppat } (\lambda x \psi. \forall x. (\text{skolems } \psi k))$

$(\lambda x \psi. \text{skolems } (\text{skolem1 } (\text{numpair } J k) x \psi) (\text{Suc } k)) (\lambda \psi. \psi) \varphi =$

$\text{skolems } (\text{skolem1 } (\text{numpair } J k) x \varphi') (\text{Suc } k) \rangle$ **for** k

unfolding *ppat-def* **using** *phi-is* **by** *simp*

```

have skol-to-g: ⟨skolems (skolem1 (numpair J k) x φ') (Suc k) =
  g (skolem1 (numpair J k) x φ') (Suc k)⟩ for k
using IH smaller phi-is by (simp add: skolem1-def)
have g-to-ppat: ⟨g (skolem1 (numpair J k) x φ') (Suc k) =
  ppat (λx ψ. ∀ x. g ψ k) (λx ψ. g (skolem1 (numpair J k) x ψ) (Suc k)) (λψ.
ψ) φ⟩ for k
unfolding ppat-def using phi-is by simp
show ?thesis
using ppat-to-skol skol-to-g g-to-ppat by simp
next
case False
then show ?thesis
using falseAll ppat-last unfolding ppat-def by auto
qed
qed
qed

```

```

consts skolems :: nat ⇒ form ⇒ nat ⇒ form
specification (skolems)
  skolems-eq: ⟨∧ J ψ k. skolems J ψ k
    = ppat (λx φ'. ∀ x. (skolems J φ' k)) (λx φ'. skolems J (skolem1
(numpair J k) x φ') (Suc k)) (λφ. φ) ψ⟩
using skolems-ex by meson

```

bounding the possible Skolem functions in a given formula

definition *skolems-bounded* ≡ λp J k. ∀ l m. (numpair J l, m) ∈ functions-form p
→ l < k

lemma *skolems-bounded-mono*: $\llbracket \text{skolems-bounded } p \ J \ k'; k' \leq k \rrbracket \implies \text{skolems-bounded } p \ J \ k$
by (meson dual-order.strict-trans1 skolems-bounded-def)

lemma *skolems-bounded-prenex*: $\text{skolems-bounded } \varphi \ K \ k \implies \text{skolems-bounded } (\text{prenex } \varphi) \ K \ k$
unfolding skolems-bounded-def
by (metis Pair-inject lang-singleton prenex-props)

Basic properties proved by induction on the number of Skolemisation steps. Harrison's gigantic conjunction broken up for more manageable proofs, at the cost of some repetition

first, the simplest properties

lemma *holds-skolems-induction-A*:
assumes size p = n **and** is-prenex p **and** skolems-bounded p J k
shows universal(skolems J p k) ∧
FV(skolems J p k) = FV p ∧
predicates-form (skolems J p k) = predicates-form p ∧
functions-form p ⊆ functions-form (skolems J p k) ∧
functions-form (skolems J p k) ⊆ {(numpair J l, m) | l m. k ≤ l} ∪
functions-form p

```

using assms
proof (induction n arbitrary: k p rule: less-induct)
  case (less n)
  show ?case
    using ⟨is-prenex p⟩
  proof cases
    case 1
    then show ?thesis
      by (metis (no-types, lifting) ppat-last-qfree skolems-eq universal.simps UnCI subsetI)
    next
      case (2 φ x)
      then have smaller: Prenex-Normal-Form.size φ < n and skbo: skolems-bounded φ J k
        using less.prem by (auto simp add: skolems-bounded-def)
        have skoeq: skolems J p k = (∀ x. skolems J φ k)
          by (metis 2(1) ppat-simpA skolems-eq)
        show ?thesis
          using less.IH [OF smaller refl ⟨is-prenex φ⟩, of k] skoeq
          by (simp add: 2 is-valuation-def lang-singleton skbo)
      next
        case (3 φ x)
        define φ' where φ' ≡ skolem1 (numpair J k) x φ
        have smaller: Prenex-Normal-Form.size φ' < n
          and pair-notin-ff: (numpair J k, card (FV (∃ x. φ))) ∉ functions-form (∃ x. φ)
          using 3 holds-skolem1c less.prem unfolding φ'-def skolems-bounded-def by blast+
          have pre: is-prenex φ'
          using 3(1) pair-notin-ff holds-skolem1a ⟨is-prenex p⟩ φ'-def by blast
          define φ'' where φ'' ≡ skolems J φ' (Suc k)
          have skos: skolems J (∃ x. φ) k = φ''
            by (metis φ'-def φ''-def ppat-simpB skolems-eq)
          have funsub: functions-form p ⊆ functions-form φ'
            functions-form φ' ⊆ insert (numpair J k, card (FV (∃ x. φ)))
            (functions-form p)
          using 3(1) pair-notin-ff φ'-def holds-skolem1e holds-skolem1f ⟨is-prenex p⟩
by presburger+
        have skbo: skolems-bounded φ' J (Suc k)
          using ⟨skolems-bounded p J k⟩ unfolding skolems-bounded-def less-Suc-eq
          using funsub(2) by fastforce
        have FV: FV φ' = FV φ - {x}
          using 3(1) pair-notin-ff holds-skolem1b ⟨is-prenex p⟩ φ'-def by auto
        have preq: predicates-form φ' = predicates-form φ
          using φ'-def formsubst-predicates skolem1-def by presburger
        show ?thesis
          using less.IH [OF smaller refl pre, of Suc k] skolems-eq [of J p] FV funsub 3
          by (force simp: preq skbo ppat-simpB simp flip: φ'-def)
qed

```

qed

the final conjunct of the HOL Light version

lemma *holds-skolems-induction-B*:

```
fixes N :: 'a intrp
assumes size p = n and is-prenex p and skolems-bounded p J k
      and is-interpretation (language {skolems J p k}) N dom N ≠ {}
      and is-valuation N β N,β ⊨ skolems J p k
shows N,β ⊨ p
using assms
proof (induction n arbitrary: N k p β rule: less-induct)
  case (less n)
  show ?case
    using ⟨is-prenex p⟩
  proof cases
    case 1
    with less show ?thesis
      by (metis (no-types, lifting) ppat-last-qfree skolems-eq)
  next
    case (2 φ x)
    then have smaller: Prenex-Normal-Form.size φ < n and skbo: skolems-bounded
      φ J k
      using less.prem1 by (auto simp add: skolems-bounded-def)
    have skolems J p k = (∀ x. skolems J φ k)
      by (metis 2(1) ppat-simpA skolems-eq)
    then show ?thesis
      using less.IH [OF smaller refl ⟨is-prenex φ⟩, of k] less.prem1
      by (simp add: lang-singleton skbo valuation-valmod 2)
  next
    case (3 φ x)
    define φ' where φ' ≡ skolem1 (numpair J k) x φ
    have smaller: Prenex-Normal-Form.size φ' < n
      and pair-notin-ff: (numpair J k, card (FV (∃ x. φ))) ∉ functions-form
      (∃ x. φ)
      using 3 holds-skolem1c less.prem1 unfolding φ'-def skolems-bounded-def by
      blast+
    have pre: is-prenex φ'
      using 3(1) pair-notin-ff holds-skolem1a ⟨is-prenex p⟩ φ'-def by blast
    define φ'' where φ'' ≡ skolems J φ' (Suc k)
    have skos: skolems J (∃ x. φ) k = φ''
      by (metis φ'-def φ''-def ppat-simpB skolems-eq)
    have functions-form φ' ⊆ insert (numpair J k, card (FV (∃ x. φ))) (functions-form
      p)
      using 3(1) pair-notin-ff φ'-def holds-skolem1f ⟨is-prenex p⟩ by presburger
    then have skbo: skolems-bounded φ' J (Suc k)
      using ⟨skolems-bounded p J k⟩ unfolding skolems-bounded-def less-Suc-eq by
      fastforce
    have pre: is-prenex (∃ x. φ)
      using 3 ⟨is-prenex p⟩ by blast
```

```

have functions-form (skolem1 (numpair J k) x  $\varphi$ )  $\subseteq$  functions-form (skolems
J  $\varphi'$  (Suc k))
  using  $\varphi'$ -def holds-skolems-induction-A pre skbo by blast
then show ?thesis
  using less.IH [OF smaller refl pre, of Suc k] less.premss skolems-eq [of J p]
  apply (simp add: skbo ppat-simpB 3)
  using holds-skolem1h [of x  $\varphi$  numpair J k] pair-notin-ff  $\langle$ is-prenex  $\varphi'$  $\rangle$ 
  by (metis prex  $\varphi'$ -def holds-exists interpretation-sublanguage lang-singleton)
qed
qed

```

the penultimate conjunct of the HOL Light version

lemma holds-skolems-induction-C:

```

fixes M :: 'a intrp
assumes size p = n and is-prenex p and skolems-bounded p J k
and is-interpretation (language {p}) M dom M  $\neq$  {} satisfies M {p}
shows  $\exists$  M'. dom M' = dom M  $\wedge$  intrp-rel M' = intrp-rel M  $\wedge$ 
  ( $\forall$  g zs. intrp-fn M' g zs  $\neq$  intrp-fn M g zs
 $\longrightarrow$  ( $\exists$  l. k  $\leq$  l  $\wedge$  g = numpair J l))  $\wedge$ 
  is-interpretation (language {skolems J p k}) M'  $\wedge$ 
  satisfies M' {skolems J p k}

using assms
proof (induction n arbitrary: M k p rule: less-induct)
case (less n)
show ?case
  using  $\langle$ is-prenex p $\rangle$ 
proof cases
case 1
with less show ?thesis
  by (metis (no-types, lifting) ppat-last-qfree skolems-eq)
next
case (2  $\varphi$  x)
then have smaller: Prenex-Normal-Form.size  $\varphi$  < n and skbo: skolems-bounded
 $\varphi$  J k
  using less.premss by (auto simp add: skolems-bounded-def)
have skoeq: skolems J p k = ( $\forall$  x. skolems J  $\varphi$  k)
  by (metis 2(1) ppat-simpA skolems-eq)
show ?thesis
  using less.IH [OF smaller refl  $\langle$ is-prenex  $\varphi$  $\rangle$ , of k M] skoeq less.premss
  apply (simp add: skbo 2 lang-singleton satisfies-def)
  by (metis fun-upd-triv is-valuation-def valuation-valmod)
next
case (3  $\varphi$  x)
define  $\varphi'$  where  $\varphi' \equiv$  skolem1 (numpair J k) x  $\varphi$ 
have smaller: Prenex-Normal-Form.size  $\varphi' < n$ 
and pair-notin-ff: (numpair J k, card (FV ( $\exists$  x.  $\varphi$ )))  $\notin$  functions-form
( $\exists$  x.  $\varphi$ )
  using 3 holds-skolem1c less.premss unfolding  $\varphi'$ -def skolems-bounded-def by
blast+

```

```

have pre: is-prenex  $\varphi'$ 
  using  $\exists(1)$  pair-notin-ff holds-skolem1a  $\langle$ is-prenex  $p\rangle$   $\varphi'$ -def by blast
define  $\varphi''$  where  $\varphi'' \equiv$  skolems  $J$   $\varphi'$  (Suc  $k$ )
have skos: skolems  $J$   $(\exists x. \varphi) k = \varphi''$ 
  by (metis  $\varphi'$ -def  $\varphi''$ -def ppat-simpB skolems-eq)
have functions-form  $\varphi' \subseteq$  insert (numpair  $J k$ , card (FV  $(\exists x. \varphi)$ )) (functions-form
 $p$ )
  using  $\exists(1)$  pair-notin-ff  $\varphi'$ -def holds-skolem1f  $\langle$ is-prenex  $p\rangle$  by presburger
then have skbo: skolems-bounded  $\varphi' J$  (Suc  $k$ )
  unfolding skolems-bounded-def less-Suc-eq
  by (meson insert-iff less.premB  $\exists$ ) old.prod.inject prod-encode-eq skolems-bounded-def
subsetD)
have prex: is-prenex  $(\exists x. \varphi)$ 
  using  $\exists(1)$   $\langle$ is-prenex  $p\rangle$  by blast
have **:  $\exists M'. \text{dom } M' = \text{dom } M \wedge \text{intrp-rel } M' = \text{intrp-rel } M$ 
 $\wedge (\forall g. (\exists zs. \text{intrp-fn } M' g zs \neq \text{intrp-fn } M g zs) \longrightarrow (\exists l \geq k. g = \text{numpair}$ 
 $J l))$ 
   $\wedge$  is-interpretation (language {skolems  $J \varphi'$  (Suc  $k$ )})  $M'$ 
   $\wedge$  satisfies  $M' \{ \text{skolems } J \varphi' (\text{Suc } k) \}$ 
if is-interpretation (language  $\{(\exists x. \varphi)\}$ )  $M$ 
  and  $\text{dom } M \neq \{\}$ 
  and  $M$ -extend:  $\bigwedge \beta. \text{is-valuation } M \beta \implies (\exists a \in \text{dom } M. M, \beta(x:=a) \models \varphi)$ 
for  $M :: 'a \text{ intrp}$ 
proof –
  have  $M$ : is-interpretation (language  $\{\varphi\}$ )  $M$ 
  using lang-singleton that(1) by auto
  with that show ?thesis
  using less.IH[OF smaller refl  $\langle$ is-prenex  $\varphi'\rangle$  skbo]
  using holds-skolem1g [OF prex pair-notin-ff  $M$ ] holds-exists
  by (smt (verit)  $\varphi'$ -def nat-le-linear not-less-eq-eq satisfies-def singleton-iff)
qed
show ?thesis
  using less.IH [OF smaller refl pre, of Suc  $k$ ] less.premB skolems-eq [of  $J p$ ] **
  by (simp add: skbo ppat-simpB  $\exists \varphi'$ -def satisfies-def)
qed
qed

```

corollary holds-skolems-prenex-A:

```

assumes is-prenex  $\varphi$  skolems-bounded  $\varphi K 0$ 
shows universal(skolems  $K \varphi 0$ )  $\wedge$  (FV (skolems  $K \varphi 0$ ) = FV  $\varphi$ )  $\wedge$ 
  predicates-form (skolems  $K \varphi 0$ ) = predicates-form  $\varphi$   $\wedge$ 
  functions-form  $\varphi \subseteq$  functions-form (skolems  $K \varphi 0$ )  $\wedge$ 
  functions-form (skolems  $K \varphi 0$ )  $\subseteq$   $\{( \text{numpair } K l, m) \mid l m. \text{True}\} \cup$ 
(functions-form  $\varphi$ )
using holds-skolems-induction-A [OF refl assms] by simp

```

corollary holds-skolems-prenex-B:

```

assumes is-prenex  $\varphi$  skolems-bounded  $\varphi K 0$ 

```

and is-interpretation (language $\{\text{skolems } K \ \varphi \ 0\}$) $M \text{ dom } M \neq \{\}$
and is-valuation $M \ \beta \ M, \beta \models \text{skolems } K \ \varphi \ 0$
shows $M, \beta \models \varphi$
using *holds-skolems-induction-B* [*OF refl assms*] **by** *simp*

corollary *holds-skolems-prenex-C*:

assumes *is-prenex* φ *skolems-bounded* $\varphi \ K \ 0$
and is-interpretation (language $\{\varphi\}$) $M \text{ dom } M \neq \{\}$ *satisfies* $M \ \{\varphi\}$
shows $\exists M'. \text{ dom } M' = \text{ dom } M \wedge \text{ intrp-rel } M' = \text{ intrp-rel } M \wedge$
 $(\forall g \text{ zs. } \text{ intrp-fn } M' \ g \ \text{zs} \neq \text{ intrp-fn } M \ g \ \text{zs} \longrightarrow (\exists l. g = \text{ numpair } K \ l))$
 \wedge
is-interpretation (language $\{\text{skolems } K \ \varphi \ 0\}$) $M' \wedge$
satisfies $M' \ \{\text{skolems } K \ \varphi \ 0\}$
using *holds-skolems-induction-C* [*OF refl assms*] **by** *simp*

definition *skopre where*

$\langle \text{skopre } k \ \varphi = \text{skolems } k \ (\text{prenex } \varphi) \ 0 \rangle$

corollary *skopre-model-A*:

assumes *skolems-bounded* $\varphi \ K \ 0$
shows *universal*(*skopre* $K \ \varphi$) $\wedge (FV (\text{skopre } K \ \varphi) = FV \ \varphi) \wedge$
predicates-form (*skopre* $K \ \varphi$) = *predicates-form* $\varphi \wedge$
functions-form $\varphi \subseteq \text{ functions-form } (\text{skopre } K \ \varphi) \wedge$
functions-form (*skopre* $K \ \varphi$) $\subseteq \{(\text{numpair } K \ l, m) \mid l \ m. \ \text{True}\} \cup (\text{ functions-form } \varphi)$
using *skolems-bounded-prenex holds-skolems-prenex-A*
by (*metis* (*no-types, lifting*) *Pair-inject assms lang-singleton prenex-props skopre-def*)

corollary *skopre-model-B*:

assumes *skolems-bounded* $\varphi \ K \ 0$
and is-interpretation (language $\{\text{skopre } K \ \varphi\}$) $M \text{ dom } M \neq \{\}$
and is-valuation $M \ \beta \ M, \beta \models \text{skopre } K \ \varphi$
shows $M, \beta \models \varphi$
using *skolems-bounded-prenex holds-skolems-prenex-B*
by (*metis assms prenex-props skopre-def*)

corollary *skopre-model-C*:

assumes *skolems-bounded* $\varphi \ K \ 0$
and is-interpretation (language $\{\varphi\}$) $M \text{ dom } M \neq \{\}$ *satisfies* $M \ \{\varphi\}$
shows $\exists M'. \text{ dom } M' = \text{ dom } M \wedge \text{ intrp-rel } M' = \text{ intrp-rel } M \wedge$
 $(\forall g \text{ zs. } \text{ intrp-fn } M' \ g \ \text{zs} \neq \text{ intrp-fn } M \ g \ \text{zs} \longrightarrow (\exists l. g = \text{ numpair } K \ l)) \wedge$
is-interpretation (language $\{\text{skopre } K \ \varphi\}$) $M' \wedge$
satisfies $M' \ \{\text{skopre } K \ \varphi\}$
using *holds-skolems-prenex-C skopre-def*
by (*metis assms prenex-props prenex-satisfies skolems-bounded-prenex*)

definition *skolemize where*

$\langle \text{skolemize } \varphi = \text{skopre } (\text{num-of-form } (\text{bump-form } \varphi) + 1) (\text{bump-form } \varphi) \rangle$

lemma *no-skolems-bump-nterm:*

shows $i > 0 \implies (\text{numpair } i \ l, \ m) \notin \text{functions-term } (\text{bump-nterm } t)$

proof (*induction t*)

case (*Var x*)

then show *?case*

by *auto*

next

case (*Fun ff ts*) **then show** *?case*

by *induction auto*

qed

lemma *no-skolems-bump-form: $i > 0 \implies \text{skolems-bounded } (\text{bump-form } \varphi) \ i \ 0$*

by (*induction φ*) (*auto simp: skolems-bounded-def no-skolems-bump-nterm*)

lemma *universal-skolemize [iff]: universal (skolemize φ)*

and *FV-skolemize [simp]: $FV (\text{skolemize } \varphi) = FV (\text{bump-form } \varphi)$*

and *predicates-form-skolemize [simp]: $\text{predicates-form } (\text{skolemize } \varphi) = \text{predicates-form } (\text{bump-form } \varphi)$*

by (*simp-all add: skolemize-def no-skolems-bump-form skopre-model-A*)

lemma *functions-bump-form: $\text{functions-form } (\text{bump-form } \varphi) \subseteq \text{functions-form } (\text{skolemize } \varphi)$*

by (*simp add: skolemize-def no-skolems-bump-form skopre-model-A*)

lemma *functions-skolemize:*

$\text{functions-form } (\text{skolemize } \varphi) \subseteq \{(\text{numpair } (\text{num-of-form } (\text{bump-form } \varphi) + 1) \ l, \ m) \mid k \ l \ m. \ \text{True}\} \cup \text{functions-form } (\text{bump-form } \varphi)$

unfolding *skolemize-def*

using *no-skolems-bump-form skopre-model-A* **by** *auto*

lemma *skolemize-imp-holds-bump-form:*

assumes *is-interpretation (language {skolemize φ }) $N \ \text{dom } N \neq \{\}$*

and *is-valuation $N \ \beta \ N, \beta \models \text{skolemize } \varphi$*

shows $N, \beta \models \text{bump-form } \varphi$

using *assms no-skolems-bump-form skolemize-def skopre-model-B* **by** *fastforce*

lemma *is-interpretation-skolemize:*

assumes *is-interpretation (language {bump-form φ }) $M \ \text{dom } M \neq \{\}$ satisfies $M \ \{\text{bump-form } \varphi\}$*

obtains M' **where** $\text{dom } M' = \text{dom } M$ *intrap-rel* $M' = \text{intrap-rel } M$

$\bigwedge g \ zs. \ \text{intrap-fn } M' \ g \ zs \neq \text{intrap-fn } M \ g \ zs \implies \exists l. \ g = \text{numpair } (\text{num-of-form } (\text{bump-form } \varphi) + 1) \ l$

is-interpretation (language {skolemize φ }) M' satisfies $M' \ \{\text{skolemize } \varphi\}$

by (metis add-gr-0 assms no-skolems-bump-form skolemize-def skopre-model-C zero-less-one)

lemma functions-form-skolemize:

assumes $\langle f, m \rangle \in \text{functions-form } (\text{skolemize } \varphi)$
obtains k **where** $\langle f = \text{numpair } 0 \ k \rangle \langle (k, m) \in \text{functions-form } \varphi \mid l$ **where** $\langle f = \text{numpair } (\text{num-of-form } (\text{bump-form } \varphi) + 1) \ l \rangle$
using functions-skolemize assms functions-form-bumpform **by** (fastforce dest: that)

definition skomod1 **where**

$\langle \text{skomod1 } \varphi \ M \equiv$
 if satisfies $M \ \{\varphi\}$
 then $(\text{SOME } M'. \text{dom } M' = \text{dom } (\text{bump-intrp } M) \wedge$
 $\text{intrp-rel } M' = \text{intrp-rel } (\text{bump-intrp } M) \wedge$
 $(\forall g \text{ zs. intrp-fn } M' \ g \ \text{zs} \neq \text{intrp-fn } (\text{bump-intrp } M) \ g \ \text{zs} \longrightarrow$
 $(\exists l. g = \text{numpair } (\text{num-of-form } (\text{bump-form } \varphi) + 1) \ l)) \wedge$
 $\text{is-interpretation } (\text{language } \{\text{skolemize } \varphi\}) \ M' \wedge \text{satisfies } M'$
 $\{\text{skolemize } \varphi\})$
 else $(\text{Abs-intrp } (\text{dom } M, (\lambda g \text{ zs. } (\text{SOME } a. a \in \text{dom } M)), \text{intrp-rel } M)) \rangle$

lemma skomod1-works:

assumes $M: \langle \text{is-interpretation } (\text{language } \{\varphi\}) \ M \rangle \langle \text{dom } M \neq \{\} \rangle$
shows $\langle \text{dom } (\text{skomod1 } \varphi \ M) = \text{dom } (\text{bump-intrp } M) \wedge$
 $\text{intrp-rel } (\text{skomod1 } \varphi \ M) = \text{intrp-rel } (\text{bump-intrp } M) \wedge$
 $\text{is-interpretation } (\text{language } \{\text{skolemize } \varphi\}) \ (\text{skomod1 } \varphi \ M) \wedge$
 $(\text{satisfies } M \ \{\varphi\} \longrightarrow$
 $(\forall g \text{ zs. intrp-fn } (\text{skomod1 } \varphi \ M) \ g \ \text{zs} \neq \text{intrp-fn } (\text{bump-intrp } M) \ g \ \text{zs} \longrightarrow$
 $(\exists l. g = \text{numpair } (\text{num-of-form } (\text{bump-form } \varphi) + 1) \ l)) \wedge$
 $\text{satisfies } (\text{skomod1 } \varphi \ M) \ \{\text{skolemize } \varphi\} \rangle$

proof (cases $\langle \text{satisfies } M \ \{\varphi\} \rangle$)

case True

obtain M' **where**

$\text{dom } M' = \text{dom } (\text{bump-intrp } M) \ \text{intrp-rel } M' = \text{intrp-rel } (\text{bump-intrp } M)$
 $\wedge g \text{ zs. intrp-fn } M' \ g \ \text{zs} \neq \text{intrp-fn } (\text{bump-intrp } M) \ g \ \text{zs} \implies \exists l. g = \text{numpair}$
 $(\text{num-of-form } (\text{bump-form } \varphi) + 1) \ l$
 $\text{is-interpretation } (\text{language } \{\text{skolemize } \varphi\}) \ M' \ \text{satisfies } M' \ \{\text{skolemize } \varphi\}$

proof (rule is-interpretation-skolemize)

show $\text{is-interpretation } (\text{language } \{\text{bump-form } \varphi\}) \ (\text{bump-intrp } M)$

by (simp add: assms(1) bumpform-interpretation)

next

show $\text{dom } (\text{bump-intrp } M) \neq \{\}$

by (simp add: assms(2))

next

show $\text{satisfies } (\text{bump-intrp } M) \ \{\text{bump-form } \varphi\}$

```

    by (metis True bump-dom bumpform is-valuation-def satisfies-def singleton-iff)
  qed metis
  then show ?thesis
    apply (simp only: skomod1-def True)
    by (smt (verit, del-insts) someI)
next
  case False
  then show ?thesis
    by (simp add: skomod1-def assms bump-intrp-def intrp-is-struct is-interpretation-def
some-in-eq)
qed

```

definition $skomod-FN \equiv \lambda M g$ *zs. if numfst $g = 0$ then intrp-fn M (numsnd g) zs else intrp-fn (skomod1 (unbump-form (form-of-num (numfst $g - 1$))) M) g zs*

definition *skomod where*
 $\langle skomod\ M \equiv Abs-intrp\ (dom\ M,\ skomod-FN\ M,\ intrp-rel\ M) \rangle$

lemma *skomod-interpretation:*

```

  assumes  $\langle is-interpretation\ (language\ \{\varphi\})\ M \rangle$   $\langle dom\ M \neq \{\} \rangle$ 
  shows  $\langle is-interpretation\ (language\ \{skolemize\ \varphi\})\ (skomod\ M) \rangle$ 
  proof -
    have  $stM: struct\ (dom\ M)$ 
      by (simp add: intrp-is-struct)
    have  $indom: intrp-fn\ M\ f\ l \in dom\ M$ 
      if  $(f, length\ l) \in functions-form\ \varphi$  and  $set\ l \subseteq dom\ M$  for  $f\ l$ 
      using  $assms$  that by (auto simp: is-interpretation-def lang-singleton)
    show ?thesis
    proof -
      have  $intrp-fn\ (skomod\ M)\ f\ l \in dom\ (skomod\ M)$ 
        if  $fl: (f, length\ l) \in functions-form\ (skolemize\ \varphi)$  and  $set\ l \subseteq dom\ (skomod\ M)$ 
        for  $f\ l$ 
      proof -
        consider  $(0)\ k$  where  $\langle f = numpair\ 0\ k \rangle$   $\langle (k, length\ l) \in functions-form\ \varphi \rangle$ 
          |  $(1)\ l$  where  $\langle f = numpair\ (num-of-form\ (bump-form\ \varphi) + 1)\ l \rangle$ 
          using  $functions-form-skolemize\ [OF\ fl]$  by metis
        then show ?thesis
      proof cases
        case 0
          with that show ?thesis
            by (simp add: stM indom skomod-def skomod-FN-def)
      next
        case  $(1\ l')$ 
          then show ?thesis
            using that skomod1-works [OF assms]
            by (force simp add: stM indom skomod-def skomod-FN-def lang-singleton
is-interpretation-def)
      end
    end
  end

```

```

    qed
  qed
  then show ?thesis
    by (auto simp: lang-singleton is-interpretation-def)
  qed
qed

```

proposition *skomod-works*:

```

  assumes ‹is-interpretation (language {φ}) M› ‹dom M ≠ {}›
  shows ‹satisfies M {φ} ‹longleftrightarrow› satisfies (skomod M) {skolemize φ}›
proof
  assume φ: satisfies M {φ}
  have Abs-intrp (dom M, skomod-FN M, intrp-rel M), β ⊨ skolemize φ
    if is-valuation (Abs-intrp (dom M, skomod-FN M, intrp-rel M)) β
    for β :: nat ⇒ 'a
  proof –
    have is-valuation (skomod1 φ M) β
      by (metis assms bump-dom dom-Abs-is-fst is-valuation-def skomod1-works
        struct.intro that)
    then have skomod1 φ M, β ⊨ skolemize φ
      by (meson φ assms insertCI satisfies-def skomod1-works)
    then show ?thesis
      proof (rule holds-indep-intrp-if2)
        fix f and zs :: 'a list
        assume f: (f, length zs) ∈ functions-form (skolemize φ)
        show intrp-fn (skomod1 φ M) f zs = intrp-fn (Abs-intrp (dom M, skomod-FN
          M, intrp-rel M)) f zs
          using functions-form-skolemize [OF f]
        proof cases
          case (1 k)
            with φ skomod1-works [OF assms] show ?thesis
              apply (simp add: skomod-FN-def bump-intrp-def)
              by (metis Zero-neq-Suc prod.inject prod.encode-inverse sndI)
            qed (simp add: skomod-FN-def)
          qed (simp-all add: assms skomod1-works)
        qed
      then show satisfies (skomod M) {skolemize φ}
        by (simp add: satisfies-def skomod-def skomod-FN-def)
    next
      assume φ: satisfies (skomod M) {skolemize φ}
      have bump-intrp M, β ⊨ bump-form φ if is-valuation (bump-intrp M) β for β ::
        nat ⇒ 'a
      proof –
        have skomod M, β ⊨ skolemize φ
          using φ that by (simp add: satisfies-def is-valuation-def skomod-def)
        with assms that skomod-interpretation [OF assms] skolemize-imp-holds-bump-form
        have skomod M, β ⊨ bump-form φ

```

```

    by (simp add: is-valuation-def skolemize-imp-holds-bump-form skomod-def)
  then show ?thesis
proof (rule holds-indep-intrp-if2)
  fix f and zs :: 'a list
  assume f: (f, length zs) ∈ functions-form (bump-form φ)
  show intrp-fn (skomod M) f zs = intrp-fn (bump-intrp M) f zs
    using functions-form-bumpform [OF f]
    by (auto simp: skomod-FN-def skomod-def)
qed (simp-all add: skomod-def)
qed
then have satisfies (bump-intrp M) {bump-form φ}
  by (auto simp: satisfies-def)
then show satisfies M {φ}
  by (metis bump-dom bumpform is-valuation-def satisfies-def singleton-iff)
qed

```

proposition *skolemize-satisfiable*:

$$\langle \exists M::'a \text{ intrp. } \text{dom } M \neq \{\} \wedge \text{is-interpretation (language } S) M \wedge \text{satisfies } M S \rangle \longleftrightarrow$$

$$\langle \exists M::'a \text{ intrp. } \text{dom } M \neq \{\} \wedge \text{is-interpretation (language (skolemize ' } S)) M \wedge \text{satisfies } M (\text{skolemize ' } S) \rangle \quad (\text{is } ?lhs = ?rhs)$$

proof

```

assume ?lhs
then obtain M::'a intrp where dom M ≠ {}
  and int: is-interpretation (language S) M and sat: satisfies M S
  by auto
show ?rhs
proof (intro exI conjI)
  show dom (skomod M) ≠ {}
    using ⟨dom M ≠ {}⟩ by (simp add: dom-def skomod-def struct-def)
  have intrp-fn (skomod M) f l ∈ dom (skomod M)
    if l: set l ⊆ dom (skomod M)
      and φ ∈ S
      and f: (f, length l) ∈ functions-form (skolemize φ)
    for f l φ
  proof -
    have is-interpretation (language {φ}) M
      using ⟨φ ∈ S⟩ unfolding lang-singleton
    by (metis Sup-upper functions-forms-def image-iff int interpretation-sublanguage
language-def)
    then have is-interpretation (language {skolemize φ}) (skomod M)
      by (intro skomod-interpretation ⟨dom M ≠ {}⟩)
    then show ?thesis
      by (simp add: f is-interpretation-def l lang-singleton)
  qed
then show is-interpretation (language (skolemize ' S)) (skomod M)
  by (auto simp add: is-interpretation-def language-def functions-forms-def)

```

```

next
  have skomod  $M, \beta \models \text{skolemize } \varphi$ 
  if is-valuation (skomod  $M$ )  $\beta$  and  $\varphi \in S$  for  $\beta \varphi$ 
  proof –
    have is-interpretation (language  $\{\varphi\}$ )  $M$ 
    using  $\langle \varphi \in S \rangle$  unfolding lang-singleton
    by (metis Sup-upper functions-forms-def image-iff int interpretation-sublanguage
language-def)
    then show ?thesis
    using that sat  $\langle \text{dom } M \neq \{\} \rangle$ 
    by (metis satisfies-def singleton-iff skomod-works)
  qed
  then show satisfies (skomod  $M$ ) (skolemize ‘  $S$ ’)
  by (auto simp add: satisfies-def image-iff)
  qed
next
assume ?rhs
then obtain  $M :: 'a \text{ intrp}$  where  $\text{dom } M \neq \{\}$ 
  and int: is-interpretation (language (skolemize ‘  $S$ ’))  $M$ 
  and sat: satisfies  $M$  (skolemize ‘  $S$ ’)
  by auto
show ?lhs
proof (intro exI conjI)
  show  $\text{dom } (\text{unbump-intrp } M) \neq \{\}$ 
  using  $\langle \text{dom } M \neq \{\} \rangle$  struct-def by blast
next
  have functions-forms (bump-form ‘  $S$ ’)  $\subseteq$  functions-forms (skolemize ‘  $S$ ’)
  using functions-bump-form functions-forms-def by auto
  then have *: is-interpretation (language (bump-form ‘  $S$ ’))  $M$ 
  by (metis int interpretation-sublanguage language-def)
  have is-interpretation (language  $\{\varphi\}$ ) (unbump-intrp  $M$ )
  if is-interpretation (language  $\{\text{bump-form } \varphi\}$ )  $M$  for  $\varphi$ 
  using that
  proof (induction  $\varphi$ )
  case (Atom  $p$   $ts$ )
  have **:  $(f, k) \in \text{Union } (\text{set } (\text{map functions-term } l))$ 
     $\implies (\text{numpair } 0 f, k) \in \text{Union } (\text{set } (\text{map functions-term } (\text{map bump-nterm } l)))$ 
  for  $l$ 
  proof (induction  $l$ )
  case Nil
  then show ?case by simp
  next
  case (Cons  $a$   $l$ )
  have  $(f, k) \in \text{functions-term } t \implies (\text{numpair } 0 f, k) \in \text{functions-term } (\text{bump-nterm } t)$ 
  for  $t$ 
  by (induction  $t$ ) auto
  with Cons show ?case
  by auto
  qed
  qed
  qed

```

```

    show ?case
      using Atom ** by (auto simp: is-interpretation-def lang-singleton un-
bump-intrp-def)
    qed (auto simp: is-interpretation-def lang-singleton)
    with * show is-interpretation (language S) (unbump-intrp M)
      unfolding is-interpretation-def lang-singleton
      by (simp add: language-def functions-forms-def) blast
  next
    have unbump-intrp M,β ⊨ φ
      if is-valuation (unbump-intrp M) β and φ ∈ S for β φ
    proof -
      have is-interpretation (language {skolemize φ}) M
        using ⟨φ ∈ S⟩ unfolding lang-singleton
      by (metis Sup-upper functions-forms-def image-eqI int interpretation-sublanguage
language-def)
      then show ?thesis
        using that ⟨dom M ≠ {}⟩ sat unfolding satisfies-def
        by (metis dom-Abs-is-fst image-eqI intrp-is-struct is-valuation-def
skolemize-imp-holds-bump-form unbump-holds unbump-intrp-def)
    qed
    then show satisfies (unbump-intrp M) S
      by (auto simp add: satisfies-def image-iff)
  qed
qed

```

```

fun specialize :: form ⇒ form where
  ⟨specialize ⊥ = ⊥⟩
| ⟨specialize (Atom p ts) = Atom p ts⟩
| ⟨specialize (φ ⟶ ψ) = φ ⟶ ψ⟩
| ⟨specialize (∀ x. φ) = specialize φ⟩

```

lemma *specialize-satisfies*:

```

  fixes M :: 'a intrp
  assumes ⟨dom M ≠ {}⟩
  shows ⟨satisfies M (specialize ' S) ⟷ satisfies M S⟩
proof -
  have satisfies M {specialize φ} ⟷ satisfies M {φ} for φ
  proof (induction φ)
    case (Forall x1 φ)
    show ?case
    proof
      show satisfies M {specialize (∀ x1. φ)} ⟹ satisfies M {∀ x1. φ}
        using Forall by (auto simp: satisfies-def valuation-valmod)
      show satisfies M {specialize (∀ x1. φ)} if §: satisfies M {∀ x1. φ}
    proof -
      have M,β ⊨ specialize φ if is-valuation M β for β
        using that § Forall unfolding is-valuation-def satisfies-def singleton-iff

```

```

      by (metis fun-upd-triv holds.simps(4))
    then show ?thesis
      by (auto simp: satisfies-def)
  qed
qed
qed auto
then show ?thesis
  by (auto simp add: satisfies-def)
qed

```

lemma *specialize-qfree*: $\langle \text{universal } \varphi \implies \text{qfree } (\text{specialize } \varphi) \rangle$
 by (induction rule: universal.induct) (auto elim: qfree.elims)

lemma *functions-form-specialize* [simp]: $\text{functions-form}(\text{specialize } \varphi) = \text{functions-form } \varphi$
 by (induction φ) auto

lemma *predicates-form-form-specialize* [simp]: $\text{predicates-form}(\text{specialize } \varphi) = \text{predicates-form } \varphi$
 by (induction φ) auto

lemma *specialize-language*: $\langle \text{language } (\text{specialize } 'S) = \text{language } S \rangle$
 by (simp add: language-def functions-forms-def predicates-def)

definition *skolem* :: $\text{form} \Rightarrow \text{form}$ **where**
 $\langle \text{skolem } \varphi = \text{specialize}(\text{skolemize } \varphi) \rangle$

lemma *skolem-qfree*: $\langle \text{qfree } (\text{skolem } \varphi) \rangle$
 by (simp add: skolem-def specialize-qfree)

theorem *skolem-satisfiable*:
 $\langle (\exists M :: 'a \text{ intrp. } \text{dom } M \neq \{\} \wedge \text{is-interpretation } (\text{language } S) M \wedge \text{satisfies } M S) \longleftrightarrow (\exists M :: 'a \text{ intrp. } \text{dom } M \neq \{\} \wedge \text{is-interpretation } (\text{language } (\text{skolem } 'S)) M \wedge \text{satisfies } M (\text{skolem } 'S)) \rangle$

proof –
 have $\text{is-interpretation } (\text{language } (\text{skolemize } 'S)) M \longleftrightarrow \text{is-interpretation } (\text{language } (\text{skolem } 'S)) M$
 for $M :: 'a \text{ intrp}$
 by (simp add: functions-forms-def is-interpretation-def language-def skolem-def)
moreover
 have $\text{satisfies } M (\text{skolemize } 'S) \longleftrightarrow \text{satisfies } M (\text{skolem } 'S)$
 if $\text{dom } M \neq \{\}$ for $M :: 'a \text{ intrp}$
 by (smt (verit, del-insts) image-iff satisfies-def skolem-def specialize-satisfies that)
 ultimately show ?thesis

by (*metis skolemize-satisfiable*)
qed

end

theory *Canonical-Models*
 imports *Skolem-Normal-Form*
begin

inductive-set *terms-set* :: $\langle (\text{nat} \times \text{nat}) \text{ set} \Rightarrow \text{nterm set} \rangle$ **for** *fns* :: $\langle (\text{nat} \times \text{nat}) \text{ set} \rangle$ **where**
 vars: $\langle (\text{Var } v) \in \text{terms-set fns} \rangle$
 | *fn*: $\langle (\text{Fun } f \text{ ts}) \in \text{terms-set fns} \rangle$
 if $\langle (f, \text{length ts}) \in \text{fns} \rangle \langle \bigwedge t. t \in \text{set ts} \implies t \in \text{terms-set fns} \rangle$

lemma *struct-terms-set [iff]*: $\langle \text{struct } (\text{terms-set } A) \rangle$
 by (*metis empty-iff struct.intro terms-set.vars*)

lemma *stupid-canondom*: $\langle t \in \text{terms-set } (\text{fst } \mathcal{L}) \implies \text{functions-term } t \subseteq (\text{fst } \mathcal{L}) \rangle$
 by (*induction t rule: terms-set.induct*) *auto*

lemma *finite-subset-instance*: $\langle \text{finite } t' \implies t' \subseteq \{\varphi \cdot_{fm} \sigma \mid \sigma \varphi. P \sigma \wedge \varphi \in s\} \implies (\exists t. \text{finite } t \wedge t \subseteq s \wedge t' \subseteq \{\varphi \cdot_{fm} \sigma \mid \sigma \varphi. P \sigma \wedge \varphi \in t\}) \rangle$

proof (*induction t' rule: finite.induct*)

case *emptyI*

then show *?case* **by** *blast*

next

case (*insertI A a*)

obtain φa **where** *phi-in*: $\langle \varphi a \in s \rangle$ **and** *phi-ex-subs*: $\langle \exists \sigma. P \sigma \wedge a = \varphi a \cdot_{fm} \sigma \rangle$

using *insertI(3)* **by** *auto*

obtain Φ **where** *Phi-subs*: $\langle \Phi \subseteq s \rangle$ **and** $\langle \text{finite } \Phi \rangle$ **and** *Phi-set*: $\langle A \subseteq \{\varphi \cdot_{fm} \sigma \mid \sigma \varphi. P \sigma \wedge \varphi \in \Phi\} \rangle$

using *insertI(3) insertI(2)* **by** *auto*

then have $\langle \text{finite } (\varphi a \triangleright \Phi) \rangle$

by *auto*

moreover have $\langle (\varphi a \triangleright \Phi) \subseteq s \rangle$

using *phi-in Phi-subs* **by** *auto*

moreover have $\langle a \triangleright A \subseteq \{\varphi \cdot_{fm} \sigma \mid \sigma \varphi. P \sigma \wedge \varphi \in (\varphi a \triangleright \Phi)\} \rangle$

using *phi-ex-subs Phi-set* **by** *blast*

ultimately show *?case* **by** *blast*

qed

lemma *finite-subset-skolem*: $\langle \text{finite } u \implies u \subseteq \{\text{skolem } \varphi \mid \varphi. \varphi \in s\} \implies (\exists t. \text{finite } t \wedge t \subseteq s \wedge u = \{\text{skolem } \varphi \mid \varphi. \varphi \in t\}) \rangle$

```

proof (induction u rule: finite.induct)
  case emptyI
  then show ?case by auto
next
  case (insertI A a)
  obtain  $\varphi a$  where phi-in:  $\langle \varphi a \in s \rangle$  and phi-ex-subs:  $\langle a = \text{skolem } \varphi a \rangle$ 
    using insertI(3) by auto
  obtain  $\Phi$  where Phi-subs:  $\langle \Phi \subseteq s \rangle$  and  $\langle \text{finite } \Phi \rangle$  and Phi-set:  $\langle A = \{ \text{skolem } \varphi \mid \varphi. \varphi \in \Phi \} \rangle$ 
    using insertI(3) insertI(2) by auto
  then have  $\langle \text{finite } (\varphi a \triangleright \Phi) \rangle$ 
    by auto
  moreover have  $\langle (\varphi a \triangleright \Phi) \subseteq s \rangle$ 
    using phi-in Phi-subs by auto
  moreover have  $\langle a \triangleright A = \{ \text{skolem } \varphi \mid \varphi. \varphi \in (\varphi a \triangleright \Phi) \} \rangle$ 
    using phi-ex-subs Phi-set by blast
  ultimately show ?case
    by blast
qed

```

```

lemma valuation-exists:  $\langle \neg (\text{dom } M = \{\}) \implies \exists \beta. \text{is-valuation } M \beta \rangle$ 
  unfolding dom-def is-valuation-def by fast

```

```

lemma holds-itlist-exists:
   $\langle (M, \beta \models (\text{foldr } (\lambda x p. \exists x. p) xs \varphi)) \longleftrightarrow$ 
     $(\exists as. \text{length } as = \text{length } xs \wedge \text{set } as \subseteq \text{dom } M \wedge$ 
       $(M, (\text{foldr } (\lambda u \beta. \beta(\text{fst } u := \text{snd } u)) (\text{rev } (\text{zip } xs as)) \beta) \models \varphi) \rangle$ 
proof (induction xs arbitrary:  $\beta \varphi$ )
  case Nil
  then show ?case by simp
next
  case (Cons x xs)
  show ?case
    by (force simp add: Cons length-Suc-conv simp flip: fun-upd-def)
qed

```

```

definition canonical ::  $\langle (\text{nat} \times \text{nat}) \text{ set} \times (\text{nat} \times \text{nat}) \text{ set} \Rightarrow \text{nterm intrp} \Rightarrow \text{bool} \rangle$ 
  where
   $\langle \text{canonical } \mathcal{L} \mathcal{M} \equiv (\text{dom } \mathcal{M} = \text{terms-set } (\text{fst } \mathcal{L})) \wedge (\forall f. \text{intrp-fn } \mathcal{M} f = \text{Fun } f) \rangle$ 

```

```

definition pintrp-of-intrp ::  $\langle 'm \text{ intrp} \Rightarrow (\text{nat} \Rightarrow 'm) \Rightarrow (\text{form} \Rightarrow \text{bool}) \rangle$  where
   $\langle \text{pintrp-of-intrp } \mathcal{M} \beta = (\lambda \varphi. \mathcal{M}, \beta \models \varphi) \rangle$ 

```

```

definition

```

canon-of-prop :: $\langle ((nat \times nat) set \times (nat \times nat) set) \Rightarrow (form \Rightarrow bool) \Rightarrow nterm intrp \rangle$ **where**
 $\langle canon-of-prop \mathcal{L} I \equiv Abs-intrp (terms-set (fst \mathcal{L}), Fun, (\lambda p. \{ts. I (Atom p ts)\})) \rangle$

lemma *dom-canon-of-prop* [*simp*]: $\langle dom (canon-of-prop \mathcal{L} I) = terms-set (fst \mathcal{L}) \rangle$
by (*simp add: canon-of-prop-def*)

lemma *intrp-fn-canon-of-prop* [*simp*]: $\langle intrp-fn (canon-of-prop \mathcal{L} I) = Fun \rangle$
by (*simp add: canon-of-prop-def*)

lemma *intrp-rel-canon-of-prop* [*simp*]: $\langle intrp-rel (canon-of-prop \mathcal{L} I) = (\lambda p. \{ts. I (Atom p ts)\}) \rangle$
by (*simp add: canon-of-prop-def*)

lemma *pholds-pintrp-of-intrp*:
 $\langle qfree \varphi \implies (pintrp-of-intrp \mathcal{M} \beta) \models_p \varphi \longleftrightarrow \mathcal{M}, \beta \models \varphi \rangle$
unfolding *pintrp-of-intrp-def* **by** (*induction \varphi simp+*)

lemma *intrp-of-canon-of-prop* [*simp*]:
 $\langle pintrp-of-intrp (canon-of-prop \mathcal{L} I) Var (Atom p ts) = I (Atom p ts) \rangle$
proof –
have $\S: \langle terms-set (fst \mathcal{L}) \neq \{\} \rangle$
using *terms-set.vars* **by** *auto*
have $\langle \forall t \in set ts. \llbracket t \rrbracket Abs-intrp (terms-set (fst \mathcal{L}), Fun, \lambda p. \{ts. I (form.Atom p ts)\}), Var = t \rangle$
proof (*induction ts*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons t ts*)
have $\langle \llbracket t \rrbracket Abs-intrp (terms-set (fst \mathcal{L}), Fun, \lambda p. \{ts. I (form.Atom p ts)\}), Var = t \rangle$
proof (*induction t*)
case (*Var x*)
then show *?case*
by *simp*
next
case *Fun*
then show *?case*
by (*simp add: \S struct-def map-idI*)
qed
with *Cons* **show** *?case*
by *simp*
qed
then show *?thesis*
by (*simp add: \S struct-def canon-of-prop-def pintrp-of-intrp-def holds-def map-idI*)
qed

lemma *holds-canon-of-prop*:
assumes $\langle \text{qfree } \varphi \rangle$ **shows** $\langle (\text{canon-of-prop } \mathcal{L} I), \text{Var} \models \varphi \longleftrightarrow I \models_p \varphi \rangle$
proof –
have $\langle \text{pintrp-of-intrp } (\text{canon-of-prop } \mathcal{L} I) \text{ Var} \models_p \varphi \longleftrightarrow I \models_p \varphi \rangle$
using *assms*
by (*induction* φ) *auto*
with *assms* **show** *?thesis*
using *pholds-pintrp-of-intrp* **by** *blast*
qed

lemma *holds-canon-of-prop-general*:
assumes $\langle \text{qfree } \varphi \rangle$ **shows** $\langle (\text{canon-of-prop } \mathcal{L} I), \beta \models \varphi \longleftrightarrow I \models_p (\varphi \cdot_{fm} \beta) \rangle$
proof –
have $\langle \text{pintrp-of-intrp } (\text{canon-of-prop } \mathcal{L} I) \beta \models_p \varphi \longleftrightarrow I \models_p (\varphi \cdot_{fm} \beta) \rangle$
using *assms*
proof (*induction* φ)
case *Atom*
have $\langle \llbracket t \rrbracket \text{Abs-intrp } (\text{terms-set } (\text{fst } \mathcal{L}), \text{Fun}, \lambda p. \{ts. I (\text{form.Atom } p \ ts)\}), \beta = t \cdot \beta \rangle$
for *t*
using *term-subst-eval* **by** (*metis empty-iff intrp-fn-Abs-is-fst-snd struct-def terms-set.simps*)
moreover **have** $\langle \text{struct } (\text{terms-set } (\text{fst } \mathcal{L})) \rangle$
by (*metis empty-iff struct.intro terms-set.vars*)
ultimately **show** *?case*
by (*simp add: canon-of-prop-def pintrp-of-intrp-def Atom*)
qed *auto*
with *assms* **show** *?thesis*
by (*simp add: pholds-pintrp-of-intrp*)
qed

lemma *canonical-canon-of-prop*: $\langle \text{canonical } \mathcal{L} (\text{canon-of-prop } \mathcal{L} I) \rangle$
unfolding *canonical-def canon-of-prop-def*
by (*metis dom-Abs-is-fst emptyE intrp-fn-Abs-is-fst-snd struct-def terms-set.vars*)

lemma *interpretation-canon-of-prop*: $\langle \text{is-interpretation } \mathcal{L} (\text{canon-of-prop } \mathcal{L} I) \rangle$
unfolding *is-interpretation-def canon-of-prop-def*
by (*metis (no-types, lifting) canonical-canon-of-prop canonical-def dom-Abs-is-fst intrp-fn-Abs-is-fst-snd intrp-is-struct subset-code(1) terms-set.fn*)

lemma *prop-valid-imp-fol-valid*: $\langle \text{qfree } \varphi \wedge (\forall I. I \models_p \varphi) \implies (\forall \mathcal{M} \beta. \mathcal{M}, \beta \models \varphi) \rangle$
using *pholds-pintrp-of-intrp* **by** *fast*

lemma *fol-valid-imp-prop-valid*: $\langle \text{qfree } \varphi \wedge (\forall \mathcal{M} \beta. \text{canonical (language } \{\varphi\}) \mathcal{M} \rightarrow \mathcal{M}, \beta \models \varphi) \Rightarrow \forall I. I \models_p \varphi \rangle$
using *canonical-canon-of-prop holds-canon-of-prop* **by** *blast*

lemma *satisfies-psatisfies*: $\langle \llbracket \varphi \in \Phi; \Phi \subseteq \{\varphi. \text{qfree } \varphi\}; \text{satisfies } \mathcal{M} \Phi; \text{is-valuation } \mathcal{M} \beta \rrbracket \Rightarrow \text{psatisfies (pintrp-of-intrp } \mathcal{M} \beta) \Phi \rangle$
using *pholds-pintrp-of-intrp satisfies-def* **by** *blast*

lemma *psatisfies-instances*:
assumes *qf*: $\langle \Phi \subseteq \{\varphi. \text{qfree } \varphi\} \rangle$
and *ps*: $\langle \text{psatisfies } I \{\varphi \cdot_{fm} \beta \mid \varphi \beta. (\forall x. \beta x \in \text{terms-set (fst } \mathcal{L})) \wedge \varphi \in \Phi\} \rangle$
shows $\langle \text{satisfies (canon-of-prop } \mathcal{L} I) \Phi \rangle$
unfolding *satisfies-def is-valuation-def*
by (*smt (verit, best) dom-canon-of-prop holds-canon-of-prop-general mem-Collect-eq ps qf subset-iff*)

lemma *satisfies-instances*:
assumes $\langle \text{is-interpretation (language } \Xi) \mathcal{M} \rangle$
shows $\langle \text{satisfies } \mathcal{M} \{\varphi \cdot_{fm} \sigma \mid \varphi \sigma. \varphi \in \Phi \wedge (\forall x. \sigma x \in \text{terms-set (fst (language } \Xi)))\} \leftrightarrow \text{satisfies } \mathcal{M} \Phi \rangle$
unfolding *satisfies-def mem-Collect-eq*
proof (*intro iffI strip*)
fix $\beta \varphi$
assume \mathcal{M} : $\langle \forall \beta \varphi. \text{is-valuation } \mathcal{M} \beta \rightarrow \varphi \in \Phi \rightarrow \mathcal{M}, \beta \models \varphi \rangle \langle \text{is-valuation } \mathcal{M} \beta \rangle$
and $\langle \exists \varphi' \sigma. \varphi = \varphi' \cdot_{fm} \sigma \wedge \varphi' \in \Phi \wedge (\forall x. \sigma x \in \text{terms-set (fst (language } \Xi))) \rangle$
then obtain $\varphi' \sigma$ **where** σ : $\langle \varphi = \varphi' \cdot_{fm} \sigma \rangle \langle \varphi' \in \Phi \rangle \langle \forall x. \sigma x \in \text{terms-set (functions-forms } \Xi) \rangle$
by (*auto simp add: language-def*)
with \mathcal{M} **assms** **have** $\langle \mathcal{M}, (\lambda v. \llbracket \sigma v \rrbracket^{\mathcal{M}, \beta}) \models \varphi' \rangle$
by (*metis (no-types, lifting) eq-fst-iff interpretation-eval interpretation-sublanguage is-valuation-def language-def stupid-canondom*)
with **assms** **show** $\langle \mathcal{M}, \beta \models \varphi \rangle$
by (*simp add: sigma swap-subst-eval*)
qed (*metis subst-var terms-set.vars*)

lemma *compact-canon-qfree*:

assumes qf : $\langle \Phi \subseteq \{\varphi. qfree \varphi\} \rangle$
and int : $\langle \bigwedge \Psi. \llbracket finite \Psi; \Psi \subseteq \Phi \rrbracket \implies \exists \mathcal{M}::'a \text{ intrp. } is\text{-interpretation (language } \Xi) \mathcal{M} \wedge dom \mathcal{M} \neq \{\} \wedge \text{satisfies } \mathcal{M} \Psi \rangle$
obtains \mathcal{C} **where** $\langle is\text{-interpretation (language } \Xi) \mathcal{C} \rangle \langle canonical (language \Xi) \mathcal{C} \rangle \langle satisfies \mathcal{C} \Phi \rangle$
proof –
define Γ **where** $\langle \Gamma \equiv \lambda X. \{\varphi \cdot_{fm} \beta \mid \beta \varphi. (\forall x. \beta x \in terms\text{-set (fst (language } \Xi))) \wedge \varphi \in X\} \rangle$
have $\langle psatisfiable (\Gamma \Phi) \rangle$
unfolding $psatisfiable\text{-def}$
proof ($rule \text{ compact-prop}$)
fix Θ
assume $\langle finite \Theta \rangle \langle \Theta \subseteq \Gamma \Phi \rangle$
then have $\langle \exists \Psi. finite \Psi \wedge \Psi \subseteq \Phi \wedge \Theta \subseteq \Gamma \Psi \rangle$
using $finite\text{-subset-instance } \Gamma\text{-def}$ **by** $force$
then obtain Ψ **where** Ψ : $\langle finite \Psi \rangle \langle \Psi \subseteq \Phi \rangle \langle \Theta \subseteq \Gamma \Psi \rangle$
by $auto$
have $\langle psatisfiable \Theta \rangle$
proof ($rule \text{ psatisfiable-mono}$)
obtain $\mathcal{M}::'a \text{ intrp}$ **where** \mathcal{M} : $\langle is\text{-interpretation (language } \Xi) \mathcal{M} \rangle \langle dom \mathcal{M} \neq \{\} \rangle$
 $\langle satisfies \mathcal{M} \Psi \rangle$
using $int \Psi$ **by** $meson$
then obtain β **where** β : $\langle is\text{-valuation } \mathcal{M} \beta \rangle$
by ($meson \text{ valuation-exists}$)
moreover have $\langle \Gamma \Psi \subseteq \{\varphi. qfree \varphi\} \rangle$
using $\Gamma\text{-def } \Psi$ $qf \text{ qfree-formsubst}$ **by** $auto$
moreover have $\langle satisfies \mathcal{M} (\Gamma \Psi) \rangle$
using \mathcal{M} **unfolding** $\Gamma\text{-def}$
by ($smt (verit, del\text{-insts}) \text{ mem-Collect-eq satisfies-def satisfies-instances}$)
ultimately show $\langle psatisfiable (\Gamma \Psi) \rangle$
by ($meson \text{ psatisfiable-def satisfies-psatisfies}$)
qed ($use \Psi$ **in** $auto$)
then show $\langle \exists I. psatisfies I \Theta \rangle$
using $psatisfiable\text{-def}$ **by** $blast$
qed ($use \text{ qf qfree-formsubst } \Gamma\text{-def}$ **in** $force$)
with qf **that show** $thesis$
unfolding $\Gamma\text{-def}$ $psatisfiable\text{-def}$
by ($smt (verit, ccfv\text{-threshold}) \text{ canonical-canon-of-prop interpretation-canon-of-prop mem-Collect-eq psatisfies-instances}$)
qed

lemma $interpretation\text{-restrictlanguage}$:
 $\langle \Psi \subseteq \Phi \implies is\text{-interpretation (language } \Phi) \mathcal{M} \implies is\text{-interpretation (language } \Psi) \mathcal{M} \rangle$
unfolding $is\text{-interpretation-def language-def functions-forms-def predicates-def}$

by (metis Union-mono fstI image-mono in-mono)

lemma *interpretation-extendlanguage*:

fixes \mathcal{M} :: $\langle 'a \text{ intrp} \rangle$

assumes *int*: $\langle \text{is-interpretation (language } \Psi) \mathcal{M} \rangle$ and $\langle \text{dom } \mathcal{M} \neq \{\} \rangle$
and $\langle \text{satisfies } \mathcal{M} \Psi \rangle$

obtains \mathcal{N} where $\langle \text{dom } \mathcal{N} = \text{dom } \mathcal{M} \rangle \langle \text{intrp-rel } \mathcal{N} = \text{intrp-rel } \mathcal{M} \rangle$
 $\langle \text{is-interpretation (language } \Phi) \mathcal{N} \rangle \langle \text{satisfies } \mathcal{N} \Psi \rangle$

proof

define m where $\langle m \equiv (\text{SOME } a. a \in \text{dom } \mathcal{M}) \rangle$

have m : $\langle m \in \text{dom } \mathcal{M} \rangle$

by (simp add: $\langle \text{dom } \mathcal{M} \neq \{\} \rangle$ *m-def some-in-eq*)

define \mathcal{N} where $\langle \mathcal{N} \equiv \text{Abs-intrp}$

$(\text{dom } \mathcal{M},$

$\lambda g \text{ zs. if } (g, \text{length } \text{zs}) \in \text{functions-forms } \Psi \text{ then intrp-fn } \mathcal{M} \ g \ \text{zs}$

else m),

$\text{intrp-rel } \mathcal{M}) \rangle$

show *eq*: $\langle \text{dom } \mathcal{N} = \text{dom } \mathcal{M} \rangle \langle \text{intrp-rel } \mathcal{N} = \text{intrp-rel } \mathcal{M} \rangle$

by (simp-all add: *N-def*)

show $\langle \text{is-interpretation (language } \Phi) \mathcal{N} \rangle$

proof –

have \S : $\langle \text{fst (language } \Psi) = \text{functions-forms } \Psi \rangle$

by (simp add: *language-def*)

obtain β where $\langle \text{is-valuation } \mathcal{M} (\beta \ \mathcal{M}) \rangle$

by (meson $\langle \text{dom } \mathcal{M} \neq \{\} \rangle$ *valuation-exists*)

then have $\langle \forall n. \beta \ \mathcal{M} \ n \in \text{dom } \mathcal{M} \rangle$

using *is-valuation-def* by blast

with *eq m int* show *?thesis*

unfolding *N-def is-interpretation-def*

by (smt (verit, ccfv-SIG) \S *intrp-fn-Abs-is-fst-snd intrp-is-struct*)

qed

show $\langle \text{satisfies } \mathcal{N} \Psi \rangle$

unfolding *satisfies-def*

proof (*intro strip*)

fix $\beta \ \varphi$

assume β : $\langle \text{is-valuation } \mathcal{N} \ \beta \rangle$ and $\langle \varphi \in \Psi \rangle$

then have $\langle \text{is-valuation } \mathcal{M} \ \beta \rangle$

by (simp add: *eq is-valuation-def*)

then have $\langle \mathcal{M}, \beta \models \varphi \rangle$

using $\langle \varphi \in \Psi \rangle \langle \text{satisfies } \mathcal{M} \Psi \rangle$ by (simp add: *satisfies-def*)

moreover

have $\langle (\mathcal{N}, \beta \models \varphi) \longleftrightarrow (\mathcal{M}, \beta \models \varphi) \rangle$

proof (*intro holds-cong*)

fix $f :: \text{nat}$ and $ts :: \langle 'a \text{ list} \rangle$

assume $\langle (f, \text{length } ts) \in \text{functions-form } \varphi \rangle$

then show $\langle \text{intrp-fn } \mathcal{N} \ f \ ts = \text{intrp-fn } \mathcal{M} \ f \ ts \rangle$

using *N-def* $\langle \varphi \in \Psi \rangle$ *functions-forms-def* by auto

qed (*auto simp: eq*)

ultimately show $\langle \mathcal{N}, \beta \models \varphi \rangle$ by *auto*
qed
qed

theorem *compact-ls*:

assumes $\langle \bigwedge \Psi. [\text{finite } \Psi; \Psi \subseteq \Phi] \implies \exists \mathcal{M}::'a \text{ intrp. is-interpretation (language } \Phi) \mathcal{M} \wedge \text{dom } \mathcal{M} \neq \{\} \wedge \text{satisfies } \mathcal{M} \Psi \rangle$

obtains $\mathcal{C}::\langle \text{nterm intrp} \rangle$ **where** $\langle \text{is-interpretation (language } \Phi) \mathcal{C} \rangle \langle \text{dom } \mathcal{C} \neq \{\} \rangle \langle \text{satisfies } \mathcal{C} \Phi \rangle$

proof –

have $\langle \exists \mathcal{M}::'a \text{ intrp. is-interpretation (language (skolem } \Phi)) \mathcal{M} \wedge \text{dom } \mathcal{M} \neq \{\} \wedge \text{satisfies } \mathcal{M} \Psi \rangle$

if $\Psi: \langle \text{finite } \Psi \rangle \langle \Psi \subseteq \text{skolem } \Phi \rangle$ **for** Ψ

by (*smt (verit, ccfv-threshold) assms finite-subset-image interpretation-extendlanguage*

interpretation-restrictlanguage skolem-satisfiable that)

with *compact-canon-qfree skolem-qfree*

obtain \mathcal{C} **where** $\mathcal{C}: \langle \text{is-interpretation (language (skolem } \Phi)) \mathcal{C} \rangle$

$\langle \text{canonical (language (skolem } \Phi)) \mathcal{C} \rangle$

$\langle \text{satisfies } \mathcal{C} (\text{skolem } \Phi) \rangle$

by *blast*

have $\langle \text{dom } \mathcal{C} \neq \{\} \rangle$

using *struct-def* **by** *blast*

with *skolem-satisfiable[of } \Phi] \mathcal{C} **that show** *thesis**

by *metis*

qed

lemma *canon*:

assumes $\langle \text{is-interpretation (language } \Phi) \mathcal{M} \rangle \langle \text{dom } \mathcal{M} \neq \{\} \rangle \langle \text{satisfies } \mathcal{M} \Phi \rangle$

obtains $\mathcal{C}::\langle \text{nterm intrp} \rangle$ **where** $\langle \text{is-interpretation (language } \Phi) \mathcal{C} \rangle \langle \text{dom } \mathcal{C} \neq \{\} \rangle \langle \text{satisfies } \mathcal{C} \Phi \rangle$

using *compact-ls assms unfolding satisfies-def* **by** *blast*

definition *lowmod* :: $\langle \text{nterm intrp} \Rightarrow \text{nat intrp} \rangle$ **where**

$\langle \text{lowmod } \mathcal{M} \equiv \text{Abs-intrp (num-of-term } \Phi (\text{dom } \mathcal{M}),$

$(\lambda g \text{ ns. num-of-term (intrp-fn } \mathcal{M} g (\text{map term-of-num ns}))),$

$(\lambda p. \{ \text{ns. (map term-of-num ns) } \in \text{intrp-rel } \mathcal{M} p \}) \rangle$

lemma *dom-lowmod [simp]*: $\langle \text{dom (lowmod } \mathcal{M}) = \text{num-of-term } \Phi (\text{dom } \mathcal{M}) \rangle$

by (*metis (no-types, lifting) dom-Abs-is-fst image-is-empty intrp-is-struct lowmod-def struct-def*)

lemma *intrp-fn-lowmod [simp]*: $\langle \text{intrp-fn (lowmod } \mathcal{M}) f \text{ ns} = \text{num-of-term (intrp-fn } \mathcal{M} f (\text{map term-of-num ns})) \rangle$

by (*metis dom-lowmod intrp-fn-Abs-is-fst-snd intrp-is-struct lowmod-def*)

lemma *intrp-rel-lowmod* [*simp*]: $\langle \text{intrp-rel } (\text{lowmod } \mathcal{M}) \text{ } p = \{ns. (\text{map term-of-num } ns) \in \text{intrp-rel } \mathcal{M} \text{ } p\} \rangle$
by (*metis* (*no-types*, *lifting*) *dom-lowmod intrp-is-struct intrp-rel-Abs-is-snd-snd lowmod-def*)

lemma *is-valuation-lowmod*: $\langle \text{is-valuation } (\text{lowmod } \mathcal{C}) (\text{num-of-term } \circ \beta) \longleftrightarrow \text{is-valuation } \mathcal{C} \beta \rangle$
by (*simp* *add: is-valuation-def image-iff num-of-term-inj*)

lemma *lowmod-dom-empty*: $\langle \text{dom } (\text{lowmod } \mathcal{M}) = \{\} \longleftrightarrow \text{dom } \mathcal{M} = \{\} \rangle$
by *simp*

lemma *lowmod-termval*:
assumes $\langle \text{is-valuation } (\text{lowmod } \mathcal{M}) \beta \rangle$
shows $\langle \text{eval } t (\text{lowmod } \mathcal{M}) \beta = \text{num-of-term } (\text{eval } t \mathcal{M} (\text{term-of-num } \circ \beta)) \rangle$
proof (*induction* *t*)
case (*Var* *x*)
then show *?case*
using *assms unfolding is-valuation-def*
by (*metis* (*no-types*, *lifting*) *comp-apply dom-lowmod eval.simps(1) image-iff term-of-num-of-term*)
next
case (*Fun* *f* *args*)
then show *?case*
using *assms unfolding is-valuation-def comp-apply intrp-fn-lowmod eval.simps*
by (*smt* (*verit*) *concat-map map-eq-conv term-of-num-of-term*)
qed

lemma *lowmod-holds*:
assumes $\langle \text{is-valuation } (\text{lowmod } \mathcal{M}) \beta \rangle$
shows $\langle (\text{lowmod } \mathcal{M}), \beta \models \varphi \longleftrightarrow \mathcal{M}, (\text{term-of-num } \circ \beta) \models \varphi \rangle$
using *assms*
proof (*induction* φ *arbitrary:* β)
case (*Atom* *x1* *x2*)
then show *?case*
using *lowmod-termval [OF Atom] by (simp add: comp-def)*
next
case (*Forall* *x1* φ)
then have $\langle \bigwedge a. a \in \text{dom } \mathcal{M} \implies \text{is-valuation } (\text{lowmod } \mathcal{M}) (\beta(x1 := \text{num-of-term } a)) \rangle$
by (*simp* *add: valuation-valmod*)
with *Forall* **show** *?case*
apply *simp*
by (*smt* (*verit*, *best*) *fun-upd-apply holds-indep- β -if term-of-num-of-term*)
qed *auto*

lemma *lowmod-intrp*: $\langle \text{is-interpretation } \mathcal{L} (\text{lowmod } \mathcal{M}) = \text{is-interpretation } \mathcal{L} \mathcal{M} \rangle$
proof
have *inj*: $\langle \text{inj num-of-term} \rangle$
by (*meson injI num-of-term-inj*)
show $\langle \text{is-interpretation } \mathcal{L} (\text{lowmod } \mathcal{M}) \implies \text{is-interpretation } \mathcal{L} \mathcal{M} \rangle$
unfolding *is-interpretation-def dom-lowmod intrp-fn-lowmod inj-image-mem-iff*
 [*OF inj*]
by (*metis (no-types, lifting) concat-map image-mono image-set length-map map-idI term-of-num-of-term*)
show $\langle \text{is-interpretation } \mathcal{L} \mathcal{M} \implies \text{is-interpretation } \mathcal{L} (\text{lowmod } \mathcal{M}) \rangle$
unfolding *is-interpretation-def dom-lowmod intrp-fn-lowmod subset-iff*
by (*smt (verit, best) image-iff length-map list.set-map term-of-num-of-term*)
qed

lemma *loewenheim-skolem*:
assumes $\langle \text{is-interpretation (language } \Phi) \mathcal{M} \rangle \langle \text{dom } \mathcal{M} \neq \{\} \rangle$
assumes $\langle \bigwedge \varphi. \varphi \in \Phi \implies \text{qfree } \varphi \rangle \langle \text{satisfies } \mathcal{M} \Phi \rangle$
obtains $\mathcal{N} :: \langle \text{nat intrp} \rangle$ **where** $\langle \text{is-interpretation (language } \Phi) \mathcal{N} \rangle \langle \text{dom } \mathcal{N} \neq \{\} \rangle \langle \text{satisfies } \mathcal{N} \Phi \rangle$
proof –
obtain $\mathcal{C} :: \langle \text{nterm intrp} \rangle$ **where** \mathcal{C} : $\langle \text{is-interpretation (language } \Phi) \mathcal{C} \rangle \langle \text{dom } \mathcal{C} \neq \{\} \rangle \langle \text{satisfies } \mathcal{C} \Phi \rangle$
using *assms canon by blast*
show *?thesis*
proof
show $\langle \text{is-interpretation (language } \Phi) (\text{lowmod } \mathcal{C}) \rangle$
by (*simp add: C lowmod-intrp*)
show $\langle \text{dom (lowmod } \mathcal{C}) \neq \{\} \rangle$
by (*simp add: C*)
show $\langle \text{satisfies (lowmod } \mathcal{C}) \Phi \rangle$
using \mathcal{C} **unfolding** *satisfies-def*
by (*smt (verit, ccfv-SIG) comp-apply dom-lowmod image-iff is-valuation-def lowmod-holds term-of-num-of-term*)
qed
qed

theorem *uniformity*:
assumes $\langle \text{qfree } \varphi \rangle$
 $\langle \bigwedge \mathcal{C} :: \langle \text{nterm intrp} \rangle. \bigwedge \beta. [\text{dom } \mathcal{C} \neq \{\}; \text{is-valuation } \mathcal{C} \beta] \implies \mathcal{C}, \beta \models \text{foldr Exists } xs \varphi \rangle$
obtains σs **where** $\langle \bigwedge \sigma x. \sigma \in \text{set } \sigma s \implies \sigma x \in \text{terms-set (fst (language } \{\varphi\}) \rangle$
 $\langle \bigwedge I. I \models_p (\text{foldr } (\lambda \varphi \psi. \varphi \vee \psi) (\text{map } (\lambda \sigma. \varphi \cdot_{fm} \sigma) \sigma s) \perp) \rangle$
proof –
define A **where** $\langle A \equiv \text{formsubst } \varphi \text{ ' } \{\sigma. \forall x. \sigma x \in \text{terms-set (fst (language } \{\varphi\}) \rangle} \rangle$

```

have ⟨ $\exists \varphi' \in A. I \models_p \varphi'$ ⟩ for  $I$ 
proof –
  have *: False if ⟨satisfies (canon-of-prop (language { $\varphi$ })  $I$ )  $\{\neg \varphi\}$ ⟩
  proof –
    obtain  $\beta$  where  $\beta$ : ⟨is-valuation (canon-of-prop (language { $\varphi$ })  $I$ )  $\beta$ ⟩
    by (metis struct-def valuation-exists intrp-is-struct)
    then have ⟨canon-of-prop (language { $\varphi$ })  $I$ ,  $\beta \models$  foldr Exists  $xs \ \varphi$ ⟩
    using assms struct-def by fastforce
    then obtain  $as$  where  $len$ : ⟨length  $as =$  length  $xs$ ⟩
      and  $sub$ : ⟨set  $as \subseteq$  terms-set (fst (language { $\varphi$ }))⟩
      and  $sat0$ : ⟨canon-of-prop (language { $\varphi$ })  $I$ ,
        foldr ( $\lambda u \ \beta. \beta(\text{fst } u := \text{snd } u)$ ) (rev (zip  $xs \ as$ ))  $\beta \models \varphi$ ⟩
    by (force simp add: holds-itlist-exists)
    define  $F$  where  $F \equiv \lambda as::nterm \ list. \lambda xs::nat \ list. \text{foldr } (\lambda u \ \beta. \beta(\text{fst } u := \text{snd } u)) (\text{rev } (\text{zip } xs \ as))$ 
    have  $F\text{-Cons}$ : ⟨ $F (a\#as) (x\#xs) \beta = F as \ xs ((\beta(x := a)))$ ⟩ for  $a \ as \ x \ xs \ \beta$ 
    by (simp add: F-def)
    have  $sat$ : ⟨canon-of-prop (language { $\varphi$ })  $I$ ,  $F as \ xs \ \beta \models \varphi$ ⟩
    using  $sat0$  by (simp add: F-def)
    have ⟨ $\llbracket$ length  $as =$  length  $xs$ ;
      set  $as \subseteq$  dom (canon-of-prop (language { $\varphi$ })  $I$ );
      is-valuation (canon-of-prop (language { $\varphi$ })  $I$ )  $\beta \rrbracket$ 
       $\implies$  is-valuation (canon-of-prop (language { $\varphi$ })  $I$ ) ( $F as \ xs \ \beta$ )⟩
    for  $xs \ as$ 
  proof (induction  $xs$  arbitrary:  $as \ \beta$ )
    case Nil
    then show ?case
    by (simp add: F-def is-valuation-def)
  next
    case (Cons  $x \ xs \ as \ \beta$ )
    then obtain  $a \ as'$  where  $aas'$ : ⟨ $as = a\#as'$ ⟩ ⟨length  $as' =$  length  $xs$ ⟩
    by (metis length-Suc-conv)
    with Cons show ?case
    by (simp add: is-valuation-def aas' F-Cons)
  qed
  then have ⟨is-valuation (canon-of-prop (language { $\varphi$ })  $I$ ) ( $F as \ xs \ \beta$ )⟩
  by (metis (no-types, lifting)  $\beta$  dom-canon-of-prop len sub)
  then show ?thesis
  using  $sat$  satisfies-def that by (force simp: F-def)
  qed
  then show ?thesis
  using psatisfies-instances [of concl: ⟨language { $\varphi$ ⟩  $I$  ⟨ $\{\neg \varphi\}$ ⟩] ⟨qfree  $\varphi$ ⟩
  by (force simp: A-def)
  qed
  then obtain  $\Phi$  where  $\Phi$ : ⟨set  $\Phi \subseteq A$ ⟩ ⟨ $\bigwedge I. I \models_p \text{foldr } (\vee) \ \Phi \ \perp$ ⟩
  by (smt (verit, ccfv-threshold) A-def assms(1) compact-disj image-iff qfree-formsubst)
  show thesis
  proof
    define  $sf$  where  $sf \equiv \lambda q. @\sigma. (\forall i. \sigma \ i \in \text{terms-set } (\text{fst } (\text{language } \{\varphi\}))) \wedge q$ 

```

```

=  $\varphi \cdot_{fm} \sigma$ 
  have sf-works:  $\langle (\forall i. sf\ a\ i \in terms\text{-}set\ (fst\ (language\ \{\varphi\}))) \wedge a = \varphi \cdot_{fm} (sf\ a) \rangle$ 
    if  $\langle a \in A \rangle$  for  $a$ 
      using that unfolding A-def sf-def image-iff Bex-def mem-Collect-eq
      by (rule someI-ex)
      show  $\langle \sigma\ i \in terms\text{-}set\ (fst\ (language\ \{\varphi\})) \rangle$ 
        if  $\langle \sigma \in set\ (map\ sf\ \Phi) \rangle$  for  $\sigma\ i$ 
          proof –
            have *:  $\langle set\ \Theta \subseteq A \implies \sigma \in set\ (map\ sf\ \Theta) \implies \sigma\ i \in terms\text{-}set\ (fst\ (language\ \{\varphi\})) \rangle$  for  $\Theta$ 
              by (induction  $\Theta$ ) (auto simp: sf-works)
              then show ?thesis
                using  $\Phi$  that by fastforce
            qed
          show  $\langle I \models_p\ foldr\ (\vee)\ (map\ ((\cdot)_{fm})\ \varphi)\ (map\ sf\ \Phi) \perp \rangle$  for  $I$ 
            using  $\Phi(2)$  [of I]  $\Phi(1)$ 
            by (induction  $\Phi$ ) (use sf-works in force)+
          qed
        qed
      qed
    end

```

References

- [1] J. Harrison. Formalizing basic first order model theory. In *TPHOLs*, volume 1479 of *Lecture Notes in Computer Science*, pages 153–170. Springer, 1998.