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

Sophie Tourret and Lawrence Paulson

March 17, 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* (¬)
no-notation *And* (**infix** ∧ *68*)
no-notation *Or* (**infix** ∨ *68*)

lemma *count-terms*:
OFCLASS((*'f*::*countable*, *'v*::*countable*) *term*,*countable-class*)
⟨*proof*⟩

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

⟨*proof*⟩

type-synonym *nterm* = ‹(*nat*, *nat*) *term*›

lemma *count-nterms*: *OFCLASS*(*nterm*,*countable-class*)
⟨*proof*⟩

instance *formula* :: (*countable*) *countable*
⟨*proof*⟩

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

abbreviation *functions-term* :: ‹*nterm* ⇒ (*nat* × *nat*) *set*› **where**
‹*functions-term t* ≡ *funas-term t*›

datatype *form* =
Bot (‹⊥›)
| *Atom* (*pred*:*nat*) (*args*:‹*nterm list*›)
| *Implies form form* (**infixl** ‹⟶› *85*)
| *Forall nat form* (‹∀ -. -› [*0*, *70*] *70*)

fun *functions-form* :: ‹*form* ⇒ (*nat* × *nat*) *set*› **where**
‹*functions-form* ⊥ = {}›
| ‹*functions-form* (*Atom p ts*) = (⋃ *t* ∈ *set ts*. *functions-term t*)›
| ‹*functions-form* (*φ* ⟶ *ψ*) = *functions-form φ* ∪ *functions-form ψ*›
| ‹*functions-form* (∀ *x*. *φ*) = *functions-form φ*›

fun *predicates-form* :: ‹*form* ⇒ (*nat* × *nat*) *set*› **where**
‹*predicates-form* ⊥ = {}›
| ‹*predicates-form* (*Atom p ts*) = {(*p*, *length ts*)}›
| ‹*predicates-form* (*φ* ⟶ *ψ*) = *predicates-form φ* ∪ *predicates-form ψ*›
| ‹*predicates-form* (∀ *x*. *φ*) = *predicates-form φ*›

definition *functions-forms* :: ‹*form set* ⇒ (*nat* × *nat*) *set*› **where**
‹*functions-forms fms* ≡ ⋃*f* ∈ *fms*. *functions-form f*›

definition *predicates* :: ‹*form set* ⇒ (*nat* × *nat*) *set*› **where**
‹*predicates fms* ≡ ⋃*f* ∈ *fms*. *predicates-form f*›

definition *language* :: ‹*form set* ⇒ ((*nat* × *nat*) *set* × (*nat* × *nat*) *set*)› **where**
‹*language fms* = (*functions-forms fms*, *predicates fms*)›

lemma *lang-singleton*: ‹*language* {*p*} = (*functions-form p*, *predicates-form p*)›
⟨*proof*⟩

abbreviation *Not* :: ‹*form* ⇒ *form*› (‹¬ -› [*90*] *90*) **where**
‹¬ *φ* ≡ *φ* ⟶ ⊥›

abbreviation *Top* :: ‹*form*› (‹⊤›) **where**
‹⊤ ≡ ¬ ⊥›

abbreviation *Or* :: ‹*form* ⇒ *form* ⇒ *form*›
(**infixl** ‹∨› *84*) **where**
‹$\varphi \vee \psi \equiv (\varphi \longrightarrow \psi) \longrightarrow \psi$›

abbreviation *And* :: ‹*form* ⇒ *form* ⇒ *form*›
(**infixl** ‹∧› *84*) **where**
‹$\varphi \wedge \psi \equiv \neg\ (\neg\ \varphi \vee \neg\ \psi)$›

abbreviation *Equiv* :: ‹*form* ⇒ *form* ⇒ *form*›
(**infix** ‹$\longleftrightarrow$› *70*) **where**
‹$\varphi \longleftrightarrow \psi \equiv (\varphi \longrightarrow \psi \wedge \psi \longrightarrow \varphi)$›

abbreviation *Exists* :: ‹*nat* ⇒ *form* ⇒ *form*›
(‹∃ -. -› [*0*, *70*] *70*) **where**
‹$\exists\ x.\ \varphi \equiv \neg\ (\forall\ x.\ \neg\ \varphi)$›

lemma *ex-all-distinct*: ‹$\forall\, x.\ \varphi \neq \exists\, y.\ \psi$›
⟨*proof*⟩

abbreviation *FVT* :: ‹*nterm* ⇒ *nat set*› **where**
‹*FVT* ≡ *vars-term*›

fun *FV* :: ‹*form* ⇒ *nat set*› **where**
‹*FV* ⊥ = {}›
| ‹*FV* (*Atom* - *ts*) = ($\bigcup$ *a* ∈ *set ts*. *FVT a*)›
| ‹*FV* ($\varphi \longrightarrow \psi$) = *FV* φ ∪ *FV* ψ›
| ‹*FV* (∀ *x*. φ) = *FV* φ − {*x*}›

lemma *FV-all-subs*: ‹*FV* φ ⊆ *FV* (∀ *x*. φ) ∪ {*x*}›
⟨*proof*⟩

lemma *FV-exists*: ‹*FV* (∃ *x*. φ) = *FV* φ − {*x*}›
⟨*proof*⟩

lemma *finite-FV*: ‹*finite* (*FV* φ)›
⟨*proof*⟩

fun *BV* :: ‹*form* ⇒ *nat set*› **where**
‹*BV* ⊥ = {}›
| ‹*BV* (*Atom* - *args'*) = {}›
| ‹*BV* ($\varphi \longrightarrow \psi$) = *BV* φ ∪ *BV* ψ›
| ‹*BV* (∀ *x*. φ) = *BV* φ ∪ {*x*}›

lemma *finite-BV*: ‹*finite* (*BV* φ)›
⟨*proof*⟩

definition *variant* :: ‹*nat set* ⇒ *nat*› **where**
‹*variant s* = *Max s* + *1*›

lemma *variant-finite*: ‹*finite s* ⟹ ¬ (*variant s* ∈ *s*)›
⟨*proof*⟩

lemma *variant-form*: ‹¬ *variant* (*FV* φ) ∈ *FV* φ›
⟨*proof*⟩

fun *formsubst* :: ‹*form* ⇒ (*nat*, *nat*) *subst* ⇒ *form*› (**infixl** ‹·$_{fm}$› *75*) **where**
‹⊥ ·$_{fm}$ - = ⊥›
| ‹(*Atom p ts*) ·$_{fm}$ σ = *Atom p* [*t* · σ. *t* ← *ts*]›
| ‹(φ ⟶ ψ) ·$_{fm}$ σ = (φ ·$_{fm}$ σ) ⟶ (ψ ·$_{fm}$ σ)›
| ‹(∀ *x*. φ) ·$_{fm}$ σ =
(*let* σ′ = σ(*x* := *Var x*);
z = *if* ∃ *y*. *y* ∈ *FV* (∀ *x*. φ) ∧ *x* ∈ *FVT* (σ′ *y*)
then variant (*FV* (φ ·$_{fm}$ σ′)) *else x in*
∀ *z*. (φ ·$_{fm}$ σ(*x* := *Var z*)))›

fun *formsubst2* :: ‹*form* ⇒ (*nat*, *nat*) *subst* ⇒ *form*› (**infixl** ‹·$_{fm}$*2*› *75*) **where**
‹⊥ ·$_{fm}$*2* - = ⊥›
| ‹(*Atom p ts*) ·$_{fm}$*2* σ = *Atom p* [*t* · σ. *t* ← *ts*]›
| ‹(φ ⟶ ψ) ·$_{fm}$*2* σ = (φ ·$_{fm}$*2* σ) ⟶ (ψ ·$_{fm}$*2* σ)›
| ‹(∀ *x*. φ) ·$_{fm}$*2* σ = (*let* σ′ = σ(*x* := *Var x*) *in*
(*if* ∃ *y*. *y* ∈ *FV* (∀ *x*. φ) ∧ *x* ∈ *FVT* (σ′ *y*)
then (*let z* = *variant* (*FV* (φ ·$_{fm}$*2* σ′)) *in*
∀ *z*. (φ ·$_{fm}$*2* σ(*x* := *Var z*)))
else ∀ *x*. (φ ·$_{fm}$*2* σ′)))›

lemma *formsubst-def-switch*: ‹φ ·$_{fm}$ σ = φ ·$_{fm}$*2* σ›
⟨*proof*⟩

lemma *termsubst-valuation*: ‹∀ *x*∈*FVT t*. σ *x* = σ′ *x* ⟹ *t* · σ = *t* · σ′›
⟨*proof*⟩

lemma *termsetsubst-valuation*: ‹∀ *y* ∈ *T*. ∀ *x*∈*FVT y*. σ *x* = σ′ *x* ⟹ *t* ∈ *T* ⟹ *t* · σ = *t* · σ′›
⟨*proof*⟩

lemma *formsubst-valuation*: ‹∀ *x*∈(*FV* φ). (*Var x*) · σ = (*Var x*) · σ′ ⟹ φ ·$_{fm}$ σ = φ ·$_{fm}$ σ′›
⟨*proof*⟩

lemma ‹{*x*. ∃ *y*. *y* ∈ (*s* ∪ *t*) ∧ *P x y*} = {*x*. ∃ *y*. *y* ∈ *s* ∧ *P x y*} ∪ {*x*. ∃ *y*. *y* ∈ *t* ∧ *P x y*}›
⟨*proof*⟩

lemma *formsubst-structure-bot*: ‹φ ·$_{fm}$ σ = ⊥ ⟷ φ = ⊥›

⟨*proof*⟩

lemma *formsubst-structure-pred*: ‹(∃ *p ts*. φ ·$_{fm}$ σ = *Atom p ts*) ⟷ (∃ *p ts*. φ = *Atom p ts*)›
⟨*proof*⟩

lemma *formsubst-structure-imp*: ‹(∃ φ*1* φ*2*. φ ·$_{fm}$ σ = φ*1* ⟶ φ*2*) ⟷ (∃ ψ*1* ψ*2*. φ = ψ*1* ⟶ ψ*2*)›
⟨*proof*⟩

lemma *formsubst-structure-all*: ‹(∃ *x* ψ. φ ·$_{fm}$ σ = (∀ *x*. ψ)) ⟷ (∃ *x* ψ. φ = (∀ *x*. ψ))›
⟨*proof*⟩

lemma *formsubst-structure-not*: ‹(∃ ψ. φ ·$_{fm}$ σ = *Not* ψ) ⟷ (∃ ψ. φ = *Not* ψ)›
⟨*proof*⟩

lemma *formsubst-structure-not-all-imp*:
‹(∃ *x* ψ. φ ·$_{fm}$ σ = (∀ *x*. ψ) ⟶ ⊥) ⟷ (∃ *x* ψ. φ = (∀ *x*. ψ) ⟶ ⊥)›
⟨*proof*⟩

lemma *formsubst-structure-all-not*:
‹(∃ *x* ψ. φ ·$_{fm}$ σ = (∀ *x*. ψ ⟶ ⊥)) ⟷ (∃ *x* ψ. φ = (∀ *x*. ψ ⟶ ⊥))›
⟨*proof*⟩

lemma *formsubst-structure-ex*: ‹(∃ *x* ψ. φ ·$_{fm}$ σ = (∃ *x*. ψ)) ⟷ (∃ *x* ψ. φ = (∃ *x*. ψ))›
⟨*proof*⟩

lemma *formsubst-structure*: ‹(φ ·$_{fm}$ σ = ⊥ ⟷ φ = ⊥) ∧
((∃ *p ts*. φ ·$_{fm}$ σ = *Atom p ts*) ⟷ (∃ *p ts*. φ = *Atom p ts*)) ∧
((∃ φ*1* φ*2*. φ ·$_{fm}$ σ = φ*1* ⟶ φ*2*) ⟷ (∃ ψ*1* ψ*2*. φ = ψ*1* ⟶ ψ*2*)) ∧
((∃ *x* ψ. φ ·$_{fm}$ σ = (∀ *x*. ψ)) ⟷ (∃ *x* ψ. φ = (∀ *x*. ψ)))›
⟨*proof*⟩

lemma *formsubst-fv*: ‹*FV* (φ ·$_{fm}$ σ) = {*x*. ∃ *y*. *y* ∈ (*FV* φ) ∧ *x* ∈ *FVT* ((*Var y*) · σ)}›
⟨*proof*⟩

lemma *subst-var* [*simp*]: ‹φ ·$_{fm}$ *Var* = φ›
⟨*proof*⟩

lemma *formsubst-rename*: ‹*FV* (φ ·$_{fm}$ (*subst x* (*Var y*))) − {*y*} = *FV* φ − {*x*} − {*y*}›
⟨*proof*⟩

lemma *termsubst-functions-term*:
‹*functions-term* (*t* · σ) = *functions-term t* ∪ {*x*. ∃ *y*. *y* ∈ *FVT t* ∧ *x* ∈ *functions-term* ((*Var y*) · σ)}›

⟨*proof*⟩

lemma *formsubst-functions-form*:
‹*functions-form* $(\varphi \cdot_{fm} \sigma)$ = *functions-form* $\varphi \cup \{x.\ \exists y.\ y \in FV\ \varphi \wedge x \in$ *functions-term* $((Var\ y) \cdot \sigma)\}$›
⟨*proof*⟩

lemma *formsubst-predicates*: ‹*predicates-form* $(\varphi \cdot_{fm} \sigma)$ = *predicates-form* φ›
⟨*proof*⟩

lemma *formsubst-language-rename*: ‹*language* $\{\varphi \cdot_{fm}$ *subst* x $(Var\ y)\}$ = *language* $\{\varphi\}$›
⟨*proof*⟩

end

theory *FOL-Semantics*
imports *FOL-Syntax*
begin

locale *struct* =
fixes
M :: ‹'*m set*› **and**
FN :: ‹*nat* ⇒ '*m list* ⇒ '*m*› **and**
REL :: ‹*nat* ⇒ '*m list set*›
assumes
M-nonempty: ‹*M* ≠ {}›

typedef '*m intrp* =
‹{ (*M* :: '*m set*, *FN* :: *nat* ⇒ '*m list* ⇒ '*m*, *REL* :: *nat* ⇒ '*m list set*). *struct M*}›
⟨*proof*⟩

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

setup-lifting *type-definition-intrp*

lift-definition *dom* :: ‹'*m intrp* ⇒ '*m set*› **is** *fst* ⟨*proof*⟩
lift-definition *intrp-fn* :: ‹'*m intrp* ⇒ (*nat* ⇒ '*m list* ⇒ '*m*)› **is** ‹*fst* ∘ *snd*› ⟨*proof*⟩
lift-definition *intrp-rel* :: ‹'*m intrp* ⇒ (*nat* ⇒ '*m list set*)› **is** ‹*snd* ∘ *snd*› ⟨*proof*⟩

lemma *intrp-is-struct* [*iff*]: ‹*struct* (*dom* $\mathcal{M}$)›
⟨*proof*⟩

lemma *dom-Abs-is-fst* [*simp*]: ‹*struct M* ⟹ *dom* (*Abs-intrp* (*M*, *FN*, *REL*)) = *M*›
⟨*proof*⟩

lemma *intrp-fn-Abs-is-fst-snd* [*simp*]: ‹*struct M* ⟹ *intrp-fn* (*Abs-intrp* (*M*, *FN*,

REL)) = *FN*›
⟨*proof*⟩

lemma *intrp-rel-Abs-is-snd-snd* [*simp*]:
‹*struct M* ⟹ *intrp-rel* (*Abs-intrp* (*M*, *FN*, *REL*)) = *REL*›
⟨*proof*⟩

definition *is-valuation* :: ‹'*m intrp* ⇒ (*nat* ⇒ '*m*) ⇒ *bool*› **where**
‹*is-valuation* $\mathcal{M}$ β ⟷ (∀ *v*. β *v* ∈ *dom* $\mathcal{M}$)›

lemma *valuation-valmod*: ‹⟦*is-valuation* $\mathcal{M}$ β; *a* ∈ *dom* $\mathcal{M}$⟧ ⟹ *is-valuation* $\mathcal{M}$ (β(*x* := *a*))›
⟨*proof*⟩

fun *eval*
:: ‹*nterm* ⇒ '*m intrp* ⇒ (*nat* ⇒ '*m*) ⇒ '*m*›
(‹⟦-⟧⁻,⁻› [*50*, *0*, *0*] *70*) **where**
‹⟦*Var v*⟧⁻,β = β *v*›
| ‹⟦*Fun f ts*⟧$^{\mathcal{M},\beta}$ = *intrp-fn* $\mathcal{M}$ *f* [⟦*t*⟧$^{\mathcal{M},\beta}$. *t* ← *ts*]›

definition *list-all* :: ‹('*a* ⇒ *bool*) ⇒ '*a list* ⇒ *bool*› **where**
[*simp*]: ‹*list-all P ls* ⟷ (*fold* (λ*l b*. *b* ∧ *P l*) *ls True*)›

lemma *term-subst-eval*: ‹*intrp-fn M* = *Fun* ⟹ *t* · *v* = *eval t M v*›
⟨*proof*⟩

lemma *term-eval-triv*[*simp*]: ‹*intrp-fn M* = *Fun* ⟹ *eval t M Var* = *t*›
⟨*proof*⟩

lemma *fold-bool-prop*: ‹(*fold* (λ*l b*. *b* ∧ *P l*) *ls b*) = (*b* ∧ (∀ *l*∈*set ls*. *P l*))›
⟨*proof*⟩

lemma *list-all-set*: ‹*list-all P ls* = (∀ *l*∈*set ls*. *P l*)›
⟨*proof*⟩

hide-const *lang*

definition *is-interpretation* **where**
‹*is-interpretation lang* $\mathcal{M}$ ⟷
((∀ *f l*. (*f*, *length*(*l*)) ∈ *fst lang* ∧ *set l* ⊆ *dom* $\mathcal{M}$ ⟶ *intrp-fn* $\mathcal{M}$ *f l* ∈ *dom* $\mathcal{M}$))›

lemma *interpretation-sublanguage*: ‹*funs2* ⊆ *funs1* ⟹ *is-interpretation* (*funs1*, *pred1*) $\mathcal{M}$
⟹ *is-interpretation* (*funs2*, *preds2*) $\mathcal{M}$›
⟨*proof*⟩

lemma *interpretation-eval*:
assumes $\mathcal{M}$: ‹*is-interpretation* (*functions-term t*,*any*) $\mathcal{M}$› **and** *val*: ‹*is-valuation* $\mathcal{M}$ β›
shows ‹$[\![t]\!]^{\mathcal{M},\beta} \in$ *dom* $\mathcal{M}$›
⟨*proof*⟩

fun *holds*
:: ‹'*m intrp* ⇒ (*nat* ⇒ '*m*) ⇒ *form* ⇒ *bool*› (‹-,- ⊨ -› [30, 30, 80] 80) **where**
‹$\mathcal{M}$,β ⊨ ⊥ ⟷ *False*›
| ‹$\mathcal{M}$,β ⊨ *Atom p ts* ⟷ $[[t]]^{\mathcal{M},\beta}$. $t \leftarrow ts] \in$ *intrp-rel* $\mathcal{M}$ *p*›
| ‹$\mathcal{M}$,β ⊨ $\varphi \longrightarrow \psi$ ⟷ (($\mathcal{M}$,β ⊨ φ) ⟶ ($\mathcal{M}$,β ⊨ ψ))›
| ‹$\mathcal{M}$,β ⊨ ($\forall x.\ \varphi$) ⟷ ($\forall a \in$ *dom* $\mathcal{M}$. $\mathcal{M}$,$\beta(x := a) \models \varphi$)›

lemma *holds-exists*: ‹$\mathcal{M}$,β ⊨ ($\exists x.\ \varphi$) ⟷ ($\exists a \in$ *dom* $\mathcal{M}$. $\mathcal{M}$,$\beta(x := a) \models \varphi$)›
⟨*proof*⟩

lemma *holds-indep-β-if*:
‹$\forall\ v \in FV\ \varphi.\ \beta_1\ v = \beta_2\ v \Longrightarrow \mathcal{M},\beta_1 \models \varphi \longleftrightarrow \mathcal{M},\beta_2 \models \varphi$›
⟨*proof*⟩

lemma *holds-indep-intrp-if*:
fixes
φ :: *form* **and**
$\mathcal{M}$ $\mathcal{M}'$ :: ‹'*m intrp*›
assumes
dom-eq: ‹*dom* $\mathcal{M}$ = *dom* $\mathcal{M}'$› **and**
rel-eq: ‹$\forall$ *p*. *intrp-rel* $\mathcal{M}$ *p* = *intrp-rel* $\mathcal{M}'$ *p*› **and**
fn-eq: ‹$\forall$ *f ts*. (*f*, *length ts*) ∈ *functions-form* φ ⟶ *intrp-fn* $\mathcal{M}$ *f ts* = *intrp-fn* $\mathcal{M}'$ *f ts*›
shows
‹$\forall \beta$. $\mathcal{M}$,β ⊨ φ ⟷ $\mathcal{M}'$,β ⊨ φ›
⟨*proof*⟩

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

corollary *holds-cong*:
assumes
‹*dom* $\mathcal{M}$ = *dom* $\mathcal{M}'$›
‹$\bigwedge$*p*. *intrp-rel* $\mathcal{M}$ *p* = *intrp-rel* $\mathcal{M}'$ *p*›
‹$\bigwedge$*f ts*. (*f*, *length ts*) ∈ *functions-form* φ ⟹ *intrp-fn* $\mathcal{M}$ *f ts* = *intrp-fn* $\mathcal{M}'$ *f ts*›
shows ‹$\mathcal{M}$,β ⊨ φ ⟷ $\mathcal{M}'$,β ⊨ φ›
⟨*proof*⟩

abbreviation (*input*) ‹*termsubst* $\mathcal{M}$ β σ *v* ≡ $[\![\sigma\ v]\!]^{\mathcal{M},\beta}$›

lemma *subst-lemma-terms*: ‹$[\![t \cdot \sigma]\!]^{\mathcal{M},\beta} = [\![t]\!]^{\mathcal{M},termsubst\ \mathcal{M}\ \beta\ \sigma}$›
⟨*proof*⟩

lemma *eval-indep-β-if*:
assumes ‹$\forall\ v \in FVT\ t.\ \beta\ v = \beta'\ v$›
shows ‹$[\![t]\!]^{\mathcal{M},\beta} = [\![t]\!]^{\mathcal{M},\beta'}$›
⟨*proof*⟩

lemma *concat-map*: ‹$[f\ t.\ t \leftarrow [g\ t.\ t \leftarrow ts]] = [f\ (g\ t).\ t \leftarrow ts]$› ⟨*proof*⟩

lemma *swap-subst-eval*: ‹$\mathcal{M}$,$\beta \models (\varphi \cdot_{fm} \sigma) \longleftrightarrow \mathcal{M}$,$(\lambda v.\ termsubst\ \mathcal{M}\ \beta\ \sigma\ v) \models \varphi$›
⟨*proof*⟩

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

lemma *satisfies-iff-satisfies-sing*: ‹*satisfies M S* $\longleftrightarrow (\forall \varphi \in S.$ *satisfies M* $\{\varphi\})$›
⟨*proof*⟩

end

theory *Ground-FOL-Compactness*
imports
FOL-Semantics
begin

fun *qfree* :: ‹*form* ⇒ *bool*› **where**
‹*qfree* $\bot$ = *True*›
| ‹*qfree* (*Atom p ts*) = *True*›
| ‹*qfree* $(\varphi \longrightarrow \psi)$ = (*qfree* $\varphi \wedge$ *qfree* $\psi)$›
| ‹*qfree* $(\forall\ x.\ \varphi)$ = *False*›

lemma *qfree-iff-BV-empty*: *qfree* $\varphi \longleftrightarrow BV\ \varphi = \{\}$
⟨*proof*⟩

lemma *qfree-no-quantif*: ‹*qfree r* $\Longrightarrow \neg(\exists x\ p.\ r = \forall x.\ p) \wedge \neg(\exists x\ p.\ r = \exists x.\ p)$›
⟨*proof*⟩

lemma *qfree-formsubst*: ‹*qfree* $\varphi \equiv$ *qfree* $(\varphi \cdot_{fm} \sigma)$›
⟨*proof*⟩

fun *form-to-formula* :: *form* ⇒ (*nat*×*nterm list*) *formula* **where**

‹*form-to-formula* $\bot$ = $\bot$›
| ‹*form-to-formula* (*Atom p ts*) = *formula.Atom* (*p*,*ts*)›
| ‹*form-to-formula* ($\varphi \longrightarrow \psi$) = *Imp* (*form-to-formula* φ) (*form-to-formula* ψ)›
| ‹*form-to-formula* ($\forall$ *x*. φ) = $\bot$›

fun *pholds* :: ‹(*form* ⇒ *bool*) ⇒ *form* ⇒ *bool*› (‹- $\models_p$ -› [*30*, *80*] *80*) **where**
‹*I* $\models_p$ $\bot$ $\longleftrightarrow$ *False*›
| ‹*I* $\models_p$ *Atom p ts* $\longleftrightarrow$ *I* (*Atom p ts*)›
| ‹*I* $\models_p$ $\varphi \longrightarrow \psi$ $\longleftrightarrow$ ((*I* $\models_p$ φ) $\longrightarrow$ (*I* $\models_p$ ψ))›
| ‹*I* $\models_p$ ($\forall$ *x*. φ) $\longleftrightarrow$ *I* ($\forall$ *x*. φ)›

definition *psatisfiable* :: *form set* ⇒ *bool* **where**
‹*psatisfiable S* ≡ ∃ *I*. ∀ φ∈*S*. *I* $\models_p$ φ›

abbreviation *psatisfies* **where** ‹*psatisfies I* Φ ≡ ∀ φ∈Φ. *pholds I* φ›

definition *val-to-prop-val* :: (*form* ⇒ *bool*) ⇒ ((*nat* × *nterm list*) ⇒ *bool*) **where**
‹*val-to-prop-val I* = (λ*x*. *I* (*Atom* (*fst x*) (*snd x*)))›

lemma *pholds-Not*: ‹*I* $\models_p$ *Not* φ $\longleftrightarrow$ ¬ (*I* $\models_p$ φ)›
⟨*proof*⟩

lemma *pentails-equiv*: ‹*qfree* φ ⟹ (*I* $\models_p$ φ ≡ (*val-to-prop-val I*) ⊨ (*form-to-formula* φ))›
⟨*proof*⟩

lemma *pentails-equiv-set*:
assumes *all-qfree*: ‹∀ φ∈*S*. *qfree* φ›
shows ‹*psatisfiable S* ≡ *sat* (*form-to-formula* ‘ *S*)›
⟨*proof*⟩

definition *finsat* :: *form set* ⇒ *bool* **where**
‹*finsat S* ≡ ∀ *T*⊆*S*. *finite T* ⟶ *psatisfiable T*›

lemma *finsat-fin-sat-eq*:
assumes *all-qfree*: ‹∀ φ∈*S*. *qfree* φ›
shows ‹*finsat S* $\longleftrightarrow$ *fin-sat* (*form-to-formula* ‘ *S*)›
⟨*proof*⟩

lemma *psatisfiable-mono*: ‹*psatisfiable S* ⟹ *T* ⊆ *S* ⟹ *psatisfiable T*›
⟨*proof*⟩

lemma *finsat-mono*: ‹*finsat S* ⟹ *T* ⊆ *S* ⟹ *finsat T*›
⟨*proof*⟩

lemma *finsat-satisfiable*: ‹*psatisfiable S* ⟹ *finsat S*›
⟨*proof*⟩

lemma *prop-compactness*: ‹(∀ φ ∈ *S*. *qfree* φ) ⟹ *finsat S* = *psatisfiable S*›

⟨*proof*⟩

as above, more in the style of HOL Light

lemma *compact-prop*:
assumes ‹$\bigwedge B.$ ⟦*finite* B; $B \subseteq A$⟧ $\Longrightarrow \exists I.$ *psatisfies* I B› **and** ‹$\bigwedge \varphi.$ $\varphi \in A \longrightarrow$ *qfree* φ›
shows ‹$\exists I.$ *psatisfies* I A›
⟨*proof*⟩

Three results required for the FOL uniformity theorem

lemma *compact-prop-alt*:
assumes ‹$\bigwedge I.$ $\exists \varphi \in A.$ $I \models_p \varphi$› ‹$\bigwedge \varphi.$ $\varphi \in A \longrightarrow$ *qfree* φ›
obtains B **where** ‹*finite* B› ‹$B \subseteq A$› ‹$\bigwedge I.$ $\exists \varphi \in B.$ $I \models_p \varphi$›
⟨*proof*⟩

lemma *finite-disj-lemma*:
assumes ‹*finite* A›
shows ‹$\exists \Phi.$ *set* $\Phi \subseteq A \wedge (\forall I.$ $I \models_p$ *foldr* $(\vee)$ Φ $\bot \longleftrightarrow (\exists \varphi \in A.$ $I \models_p \varphi))$›
⟨*proof*⟩

lemma *compact-disj*:
assumes ‹$\bigwedge I.$ $\exists \varphi \in A.$ $I \models_p \varphi$› ‹$\bigwedge \varphi.$ $\varphi \in A \longrightarrow$ *qfree* φ›
obtains Φ **where** ‹*set* $\Phi \subseteq A$› ‹$\bigwedge I.$ $I \models_p$ *foldr* $(\vee)$ Φ $\bot$›
⟨*proof*⟩

end

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

inductive *is-prenex* :: *form* ⇒ *bool* **where**
‹*qfree* $\varphi \Longrightarrow$ *is-prenex* φ›
| ‹*is-prenex* $\varphi \Longrightarrow$ *is-prenex* $(\forall x.\ \varphi)$›
| ‹*is-prenex* $\varphi \Longrightarrow$ *is-prenex* $(\exists x.\ \varphi)$›

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*: ‹*is-prenex* $\varphi \Longrightarrow$ *is-prenex* $(\varphi \cdot_{fm} \sigma)$›
⟨*proof*⟩

lemma *prenex-formsubst2*: ‹*is-prenex* $(\varphi \cdot_{fm} \sigma) \Longrightarrow$ *is-prenex* φ›

⟨*proof*⟩

lemma *prenex-formsubst*: ‹*is-prenex* ($\varphi \cdot_{fm} \sigma$) ≡ *is-prenex* φ›
⟨*proof*⟩

lemma *prenex-imp*: ‹*is-prenex* ($\varphi \longrightarrow \psi$) ⟹
qfree ($\varphi \longrightarrow \psi$) ∨ ($\psi = \bot \wedge (\exists\, x\ \varphi'.$ *is-prenex* $\varphi' \wedge \varphi = (\forall\, x.\ \varphi' \longrightarrow \bot))$)›
⟨*proof*⟩

inductive *universal* :: *form* ⇒ *bool* **where**
‹*qfree* φ ⟹ *universal* φ›
| ‹*universal* φ ⟹ *universal* ($\forall\, x.\ \varphi$)›

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

fun *size* :: *form* ⇒ *nat* **where**
‹*size* $\bot$ = *1*›
| ‹*size* (*Atom p ts*) = *1*›
| ‹*size* ($\varphi \longrightarrow \psi$) = *size* φ + *size* ψ›
| ‹*size* ($\forall\ x.\ \varphi$) = *1* + *size* φ›

lemma *wf-size*: ‹*wfP* ($\lambda\varphi\ \psi.$ *size* φ < *size* ψ)›
⟨*proof*⟩

lemma *size-indep-subst*: ‹*size* ($\varphi \cdot_{fm} \sigma$) = *size* φ›
⟨*proof*⟩

lemma *prenex-distinct*: ‹($\forall\, x.\ \varphi$) ≠ ($\exists\, y.\ \psi$)›
⟨*proof*⟩

lemma *uniq-all-x*: *Uniq* ($\lambda x.\ \exists\, p.\ r = \forall\, x.\ p$)
⟨*proof*⟩

lemma *uniq-all-p*: ‹*Uniq* (($\lambda p.\ r = \forall\,$ (*THE* $x.\ \exists\, p.\ r = \forall\, x.\ p$). p))›
⟨*proof*⟩

lemma *uniq-ex-x*: *Uniq* ($\lambda x.\ \exists\, p.\ r = \exists\, x.\ p$)
⟨*proof*⟩

lemma *uniq-ex-p*: ‹*Uniq* (($\lambda p.\ r = \exists\,$ (*THE* $x.\ \exists\, p.\ r = \exists\, x.\ p$). p))›
⟨*proof*⟩

definition *ppat* :: (*nat* ⇒ *form* ⇒ *form*) ⇒ (*nat* ⇒ *form* ⇒ *form*) ⇒ (*form* ⇒ *form*) ⇒ *form* ⇒ *form* **where**

‹*ppat A B C r* = (*if* (∃ *x p*. *r* = ∀ *x*. *p*) *then*
A (*THE x*. ∃ *p*. *r* = ∀ *x*. *p*) (*THE p*. *r* = ∀ (*THE x*. ∃ *p*. *r* = ∀ *x*. *p*). *p*)
else (*if* ∃ *x p*. *r* = ∃ *x*. *p then*
B (*THE x*. ∃ *p*. *r* = ∃ *x*. *p*) (*THE p*. *r* = ∃ (*THE x*. ∃ *p*. *r* = ∃ *x*. *p*). *p*)
else C r))›

lemma *ppat-simpA*: ‹∀ *x p*. *ppat A B C* (∀ *x*. *p*) = *A x p*›
⟨*proof*⟩

lemma *ppat-simpB*: ‹∀ *x p*. *ppat A B C* (∃ *x*. *p*) = *B x p*›
⟨*proof*⟩

lemma *ppat-last*: ‹(∀ *r*. ¬(∃ *x p*. *r* = ∀ *x*. *p*) ∧ ¬(∃ *x p*. *r* = ∃ *x*. *p*)) ⟹ *ppat A B C r* = *C r*›
⟨*proof*⟩

lemma *ppat-last-qfree*: ‹*qfree r* ⟹ *ppat A B C r* = *C r*›
⟨*proof*⟩

lemma *ppat-to-ex-qfree*:
‹(∃ *f*. (∀ *x p q*. *f p* (∀ *x*. *q*) = ((*A* :: *form* ⇒ *nat* ⇒ *form* ⇒ *form*) *p*) *x q*) ∧
(∀ *x p q*. *f p* (∃ *x*. *q*) = (*B p*) *x q*) ∧
(∀ *p q*. *qfree q* ⟶ *f p q* = (*C p*) *q*))›
⟨*proof*⟩

lemma *size-rec*:
‹∀ *f g x*. (∀ (*z*::*form*). *size z* < *size x* ⟶ (*f z* = *g z*)) ⟶ (*H f x* = *H g x*) ⟹ (∃ *f*. ∀ *x*. *f x* = *H f x*)›
⟨*proof*⟩

abbreviation *prenex-right-forall* :: (*form* ⇒ *form* ⇒ *form*) ⇒ *form* ⇒ *nat* ⇒ *form* ⇒ *form* **where**
‹*prenex-right-forall* ≡
(λ*p φ x ψ*. (*let y* = *variant*(*FV φ* ∪ *FV* (∀ *x*. *ψ*)) *in* (∀ *y*. *p φ* (*ψ* ·$_{fm}$ (*subst x* (*Var y*))))))›

abbreviation *prenex-right-exists* :: (*form* ⇒ *form* ⇒ *form*) ⇒ *form* ⇒ *nat* ⇒ *form* ⇒ *form* **where**
‹*prenex-right-exists* ≡
(λ*p φ x ψ*. (*let y* = *variant*(*FV φ* ∪ *FV* (∃ *x*. *ψ*)) *in* (∃ *y*. *p φ* (*ψ* ·$_{fm}$ (*subst x* (*Var y*))))))›

lemma *prenex-right-ex*:
‹∃ *prenex-right*. (∀ *φ x ψ*. *prenex-right φ* (∀ *x*. *ψ*) = *prenex-right-forall prenex-right φ x ψ*)
∧ (∀ *φ x ψ*. *prenex-right φ* (∃ *x*. *ψ*) = *prenex-right-exists prenex-right φ x ψ*)

$\wedge$ $(\forall\, \varphi\ \psi.\ qfree\ \psi \longrightarrow prenex\text{-}right\ \varphi\ \psi = (\varphi \longrightarrow \psi))$›
⟨*proof*⟩

consts *prenex-right* :: *form* ⇒ *form* ⇒ *form*
specification (*prenex-right*) ‹
$(\forall\, \varphi\ x\ \psi.\ prenex\text{-}right\ \varphi\ (\forall\, x.\ \psi) = prenex\text{-}right\text{-}forall\ prenex\text{-}right\ \varphi\ x\ \psi) \wedge$
$(\forall\, \varphi\ x\ \psi.\ prenex\text{-}right\ \varphi\ (\exists\, x.\ \psi) = prenex\text{-}right\text{-}exists\ prenex\text{-}right\ \varphi\ x\ \psi) \wedge$
$(\forall\, \varphi\ \psi.\ qfree\ \psi \longrightarrow prenex\text{-}right\ \varphi\ \psi = (\varphi \longrightarrow \psi))$›
⟨*proof*⟩

lemma *prenex-right-qfree-case*: ‹*qfree* $\psi \Longrightarrow$ *prenex-right* $\varphi\ \psi = (\varphi \longrightarrow \psi)$›
⟨*proof*⟩

lemma *prenex-right-all-case*: ‹*prenex-right* $\varphi\ (\forall\, x.\ \psi) =$ *prenex-right-forall prenex-right* $\varphi\ x\ \psi$›
⟨*proof*⟩

lemma *prenex-right-exist-case*: ‹*prenex-right* $\varphi\ (\exists\, x.\ \psi) =$ *prenex-right-exists prenex-right* $\varphi\ x\ \psi$›
⟨*proof*⟩

lemma *prenex-right-exists-shape-case*:
‹$\exists\, x2\ \sigma.$ *prenex-right* $\varphi\ (\exists\, x.\ \psi) = \exists\, x2.$ *prenex-right* $\varphi\ (\psi \cdot_{fm} \sigma)$›
⟨*proof*⟩

abbreviation *prenex-left-forall* :: (*form* ⇒ *form* ⇒ *form*) ⇒ *form* ⇒ *nat* ⇒ *form* ⇒ *form* **where**
‹*prenex-left-forall* ≡
$(\lambda p\ \varphi\ x\ \psi.\ (let\ y = variant(FV\ (\forall\, x.\ \varphi) \cup FV\ \psi)\ in\ (\exists\, y.\ p\ (\varphi \cdot_{fm} (subst\ x\ (Var\ y)))\ \psi)))$›

abbreviation *prenex-left-exists* :: (*form* ⇒ *form* ⇒ *form*) ⇒ *form* ⇒ *nat* ⇒ *form* ⇒ *form* **where**
‹*prenex-left-exists* ≡
$(\lambda p\ \varphi\ x\ \psi.\ (let\ y = variant(FV\ (\exists\, x.\ \varphi) \cup FV\ \psi)\ in\ (\forall\, y.\ p\ (\varphi \cdot_{fm} (subst\ x\ (Var\ y)))\ \psi)))$›

lemma *prenex-left-ex*:
‹$\exists$ *prenex-left*. $(\forall\, \varphi\ x\ \psi.\ prenex\text{-}left\ (\forall\, x.\ \varphi)\ \psi = prenex\text{-}left\text{-}forall\ prenex\text{-}left\ \varphi\ x\ \psi)$
$\wedge\ (\forall\, \varphi\ x\ \psi.\ prenex\text{-}left\ (\exists\, x.\ \varphi)\ \psi = prenex\text{-}left\text{-}exists\ prenex\text{-}left\ \varphi\ x\ \psi)$
$\wedge\ (\forall\, \varphi\ \psi.\ qfree\ \varphi \longrightarrow prenex\text{-}left\ \varphi\ \psi = prenex\text{-}right\ \varphi\ \psi)$›
⟨*proof*⟩

definition *prenex-left* **where** ‹*prenex-left* = (*SOME prenex-left*.
$(\forall\, \varphi\ x\ \psi.\ prenex\text{-}left\ (\forall\, x.\ \varphi)\ \psi = prenex\text{-}left\text{-}forall\ prenex\text{-}left\ \varphi\ x\ \psi) \wedge$
$(\forall\, \varphi\ x\ \psi.\ prenex\text{-}left\ (\exists\, x.\ \varphi)\ \psi = prenex\text{-}left\text{-}exists\ prenex\text{-}left\ \varphi\ x\ \psi) \wedge$

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

lemma *prenex-left-forall-case*: ‹*prenex-left* $(\forall\, x.\ \varphi)\ \psi$ = *prenex-left-forall prenex-left* $\varphi\ x\ \psi$›
⟨*proof*⟩

lemma *prenex-left-qfree-case*: ‹*qfree* $\varphi \Longrightarrow$ *prenex-left* $\varphi\ \psi$ = *prenex-right* $\varphi\ \psi$›
⟨*proof*⟩

lemma *prenex-left-exists-case*: ‹*prenex-left* $(\exists\, x.\ \varphi)\ \psi$ = *prenex-left-exists prenex-left* $\varphi\ x\ \psi$›
⟨*proof*⟩

lemma *prenex-left-exists-shape-case*:
‹$\exists\, x2\ \sigma.$ *prenex-left* $(\exists\, x.\ \varphi)\ \psi = \forall\, x2.$ *prenex-left* $(\varphi \cdot_{fm} \sigma)\ \psi$›
⟨*proof*⟩

fun *prenex* **where**
‹*prenex* $\bot = \bot$›
| ‹*prenex* (*Atom p ts*) = *Atom p ts*›
| ‹*prenex* $(\varphi \longrightarrow \psi)$ = *prenex-left* (*prenex* φ) (*prenex* ψ)›
| ‹*prenex* $(\forall\, x.\ \varphi) = \forall\, x.$ (*prenex* φ)›

lemma *holds-indep-forall*:
assumes *y-notin*: ‹$y \notin FV\ (\forall\, x.\ \varphi)$›
shows ‹$(I,\beta \models (\forall\, x.\ \varphi) \longleftrightarrow I,\beta \models (\forall\, y.\ \varphi \cdot_{fm}$ (*subst x* (*Var y*))))›
⟨*proof*⟩

lemma *forall-imp-commute*:
assumes *y-notin*: ‹$y \notin FV\ \varphi$›
shows ‹$((I :: 'a\ intrp), \beta \models (\varphi \longrightarrow (\forall\, y.\ \psi)) \longleftrightarrow I, \beta \models (\forall\, y.\ \varphi \longrightarrow \psi))$›
⟨*proof*⟩

lemma *forall-imp-exists*:
assumes *y-notin*: ‹$y \notin FV\ \psi$›
shows ‹$((I :: 'a\ intrp), \beta \models ((\forall\, y.\varphi) \longrightarrow \psi) \longleftrightarrow I, \beta \models (\exists\, y.\ (\varphi \longrightarrow \psi)))$›
⟨*proof*⟩

lemma *exists-imp-forall*:
assumes *y-notin*: ‹$y \notin FV\ \psi$›
shows ‹$(I, \beta \models ((\exists\, y.\varphi) \longrightarrow \psi) \longleftrightarrow I, \beta \models (\forall\, y.\ (\varphi \longrightarrow \psi)))$›
⟨*proof*⟩

lemma *exists-imp-commute*:
assumes *y-notin*: ‹$y \notin FV\ \varphi$›
shows ‹$((I :: 'a\ intrp), \beta \models (\varphi \longrightarrow (\exists\, y.\ \psi)) \longleftrightarrow I, \beta \models (\exists\, y.\ \varphi \longrightarrow \psi))$›
⟨*proof*⟩

lemma *holds-indep-exists*:
‹$y \notin FV\ (\exists\, x.\ \varphi) \Longrightarrow (I,\beta \models (\exists\, x.\ \varphi) \longleftrightarrow I,\beta \models (\exists\, y.\ \varphi \cdot_{fm} (subst\ x\ (Var\ y))))$›
⟨*proof*⟩

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

lemma *prenex-right-exists-is*:
assumes ‹$dom\ I \neq \{\}$›
shows ‹$((I,\ \beta \models \varphi \longrightarrow (\exists\, x.\ \psi)) \longleftrightarrow$
$(I,\ \beta \models (\exists\, (variant\ (FV\ \varphi \cup FV\ (\exists\, x.\ \psi))).$
$(\varphi \longrightarrow (\psi \cdot_{fm} (subst\ x\ (Var\ (variant\ (FV\ \varphi \cup FV\ (\exists\, x.\ \psi))))))))))$›
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *prenex-left-forall-is*:
assumes ‹$dom\ I \neq \{\}$›
shows ‹$(I,\ \beta \models ((\forall\, x.\ \varphi) \longrightarrow \psi)) \equiv (I,\ \beta \models (\exists\, (variant\ (FV\ (\forall\, x.\ \varphi) \cup FV\ \psi)).$
$((\varphi \cdot_{fm} (subst\ x\ (Var\ (variant\ (FV\ (\forall\, x.\ \varphi) \cup FV\ \psi))))) \longrightarrow \psi)))$›
⟨*proof*⟩

lemma *prenex-left-exists-is*:
assumes ‹$dom\ I \neq \{\}$›
shows ‹$(I,\ \beta \models ((\exists\, x.\ \varphi) \longrightarrow \psi)) \equiv (I,\ \beta \models (\forall\, (variant\ (FV\ (\exists\, x.\ \varphi) \cup FV\ \psi)).$
$((\varphi \cdot_{fm} (subst\ x\ (Var\ (variant\ (FV\ (\exists\, x.\ \varphi) \cup FV\ \psi))))) \longrightarrow \psi)))$›
⟨*proof*⟩

lemma *prenex-right-forall-FV*: ‹$FV\ (\varphi \longrightarrow (\forall\, x.\ \psi)) =$
$FV\ (\forall\, (variant\ (FV\ \varphi \cup FV\ (\forall\, x.\ \psi))).\ (\varphi \longrightarrow (\psi \cdot_{fm} (subst\ x\ (Var\ (variant\ (FV\ \varphi \cup FV\ (\forall\, x.\ \psi))))))))$›
⟨*proof*⟩

lemma *prenex-right-exists-FV*: ‹$FV\ (\varphi \longrightarrow (\exists\, x.\ \psi)) =$
$FV\ (\forall\, (variant\ (FV\ \varphi \cup FV\ (\exists\, x.\ \psi))).\ (\varphi \longrightarrow (\psi \cdot_{fm} (subst\ x\ (Var\ (variant\ (FV\ \varphi \cup FV\ (\exists\, x.\ \psi))))))))$›
⟨*proof*⟩

lemma *prenex-left-forall-FV*: ‹$FV\ ((\forall\, x.\ \varphi) \longrightarrow \psi) =$

$FV\ (\exists\,(variant\ (FV\ (\forall\, x.\ \varphi)\ \cup\ FV\ \psi)).\ ((\varphi\ \cdot_{fm}\ (subst\ x\ (Var\ (variant\ (FV\ (\forall\, x.\ \varphi)\ \cup\ FV\ \psi))))) \longrightarrow \psi))$›
⟨*proof*⟩

lemma *prenex-left-exists-FV*: ‹$FV\ ((\exists\, x.\ \varphi) \longrightarrow \psi) =$
$FV\ (\forall\,(variant\ (FV\ (\exists\, x.\ \varphi)\ \cup\ FV\ \psi)).\ ((\varphi\ \cdot_{fm}\ (subst\ x\ (Var\ (variant\ (FV\ (\exists\, x.\ \varphi)\ \cup\ FV\ \psi))))) \longrightarrow \psi))$›
⟨*proof*⟩

lemma *prenex-right-forall-language*: ‹$language\ \{\varphi \longrightarrow (\forall\, x.\ \psi)\} =$
$language\ \{\forall\,(variant\ (FV\ \varphi\ \cup\ FV\ (\forall\, x.\ \psi))).\ (\varphi \longrightarrow (\psi\ \cdot_{fm}\ (subst\ x\ (Var\ (variant\ (FV\ \varphi \cup FV\ (\forall\, x.\ \psi))))))\}$›
⟨*proof*⟩

lemma *prenex-right-exists-language*: ‹$language\ \{\varphi \longrightarrow (\exists\, x.\ \psi)\} =$
$language\ \{\exists\,(variant\ (FV\ \varphi\ \cup\ FV\ (\exists\, x.\ \psi))).\ (\varphi \longrightarrow (\psi\ \cdot_{fm}\ (subst\ x\ (Var\ (variant\ (FV\ \varphi \cup FV\ (\exists\, x.\ \psi))))))\}$›
⟨*proof*⟩

lemma *prenex-left-forall-language*: ‹$language\ \{(\forall\, x.\ \varphi) \longrightarrow \psi\} =$
$language\ \{\exists\,(variant\ (FV\ (\forall\, x.\ \varphi)\ \cup\ FV\ \psi)).\ ((\varphi\ \cdot_{fm}\ (subst\ x\ (Var\ (variant\ (FV\ (\forall\, x.\ \varphi)\ \cup\ FV\ \psi))))) \longrightarrow \psi)\}$›
⟨*proof*⟩

lemma *prenex-left-exists-language*: ‹$language\ \{(\exists\, x.\ \varphi) \longrightarrow \psi\} =$
$language\ \{\forall\,(variant\ (FV\ (\exists\, x.\ \varphi)\ \cup\ FV\ \psi)).\ ((\varphi\ \cdot_{fm}\ (subst\ x\ (Var\ (variant\ (FV\ (\exists\, x.\ \varphi)\ \cup\ FV\ \psi))))) \longrightarrow \psi)\}$›
⟨*proof*⟩

lemma *prenex-props-forall*: ‹$P \wedge FV\ \varphi = FV\ \psi \wedge language\ \{\varphi\} = language\ \{\psi\}$
$\wedge$
$(\forall\,(I :: 'a\ intrp)\ \beta.\ dom\ I \neq \{\} \longrightarrow (I,\beta \models \varphi \longleftrightarrow I,\beta \models \psi)) \Longrightarrow$
$P \wedge FV\ (\forall\, x.\ \varphi) = FV\ (\forall\, x.\ \psi) \wedge language\ \{(\forall\, x.\ \varphi)\} = language\ \{(\forall\, x.\ \psi)\} \wedge$
$(\forall\,(I :: 'a\ intrp)\ \beta.\ dom\ I \neq \{\} \longrightarrow (I,\beta \models (\forall\, x.\ \varphi) \longleftrightarrow I,\beta \models (\forall\, x.\ \psi)))$
›
⟨*proof*⟩

lemma *prenex-props-exists*: ‹$P \wedge FV\ \varphi = FV\ \psi \wedge language\ \{\varphi\} = language\ \{\psi\}$
$\wedge$
$(\forall\,(I :: 'a\ intrp)\ \beta.\ dom\ I \neq \{\} \longrightarrow (I,\beta \models \varphi \longleftrightarrow I,\beta \models \psi)) \Longrightarrow$
$P \wedge FV\ (\exists\, x.\ \varphi) = FV\ (\exists\, x.\ \psi) \wedge language\ \{(\exists\, x.\ \varphi)\} = language\ \{(\exists\, x.\ \psi)\} \wedge$
$(\forall\,(I :: 'a\ intrp)\ \beta.\ dom\ I \neq \{\} \longrightarrow (I,\beta \models (\exists\, x.\ \varphi) \longleftrightarrow I,\beta \models (\exists\, x.\ \psi)))$
›
⟨*proof*⟩

lemma *prenex-right-props-imp0*:
assumes ‹*qfree* φ›

shows ‹*is-prenex* $\psi \Longrightarrow$ *is-prenex* (*prenex-right* $\varphi\ \psi$)›
⟨*proof*⟩

lemma *prenex-right-props-imp*:
assumes ‹*qfree* φ›
shows ‹*is-prenex* $\psi \Longrightarrow$
is-prenex (*prenex-right* $\varphi\ \psi$) $\wedge$
FV (*prenex-right* $\varphi\ \psi$) = FV ($\varphi \longrightarrow \psi$) $\wedge$
language {*prenex-right* $\varphi\ \psi$} = *language* {($\varphi \longrightarrow \psi$)} $\wedge$
($\forall$ (I :: *'a intrp*) β. *dom* $I \neq \{\} \longrightarrow$ ((I,β $\models$ (*prenex-right* $\varphi\ \psi$)) $\longleftrightarrow$ (I,β $\models$ ($\varphi \longrightarrow \psi$)))))›
(**is** ‹*is-prenex* $\psi \Longrightarrow$ *?P* ψ›)
⟨*proof*⟩

lemma *prenex-right-props*:
‹*qfree* $\varphi \wedge$ *is-prenex* $\psi \Longrightarrow$
is-prenex (*prenex-right* $\varphi\ \psi$) $\wedge$
FV (*prenex-right* $\varphi\ \psi$) = FV ($\varphi \longrightarrow \psi$) $\wedge$
language {*prenex-right* $\varphi\ \psi$} = *language* {($\varphi \longrightarrow \psi$)} $\wedge$
($\forall$ (I :: *'a intrp*) β. *dom* $I \neq \{\} \longrightarrow$ ((I,β $\models$ (*prenex-right* $\varphi\ \psi$)) $\longleftrightarrow$ (I,β $\models$ ($\varphi \longrightarrow \psi$)))))›
⟨*proof*⟩

lemma *prenex-left-props-imp0*:
assumes ‹*is-prenex* ψ›
shows ‹*is-prenex* $\varphi \Longrightarrow$ *is-prenex* (*prenex-left* $\varphi\ \psi$)›
⟨*proof*⟩

lemma *prenex-left-props-imp*:
assumes ‹*is-prenex* ψ›
shows ‹*is-prenex* $\varphi \Longrightarrow$
is-prenex (*prenex-left* $\varphi\ \psi$) $\wedge$
FV (*prenex-left* $\varphi\ \psi$) = FV ($\varphi \longrightarrow \psi$) $\wedge$
(*language* {(*prenex-left* $\varphi\ \psi$)} = *language* {($\varphi \longrightarrow \psi$)}) $\wedge$
($\forall$ (I :: *'a intrp*) β. *dom* $I \neq \{\} \longrightarrow$ (I,β $\models$ *prenex-left* $\varphi\ \psi \longleftrightarrow$ I,β $\models$ $\varphi \longrightarrow \psi$))›
(**is** ‹*is-prenex* $\varphi \Longrightarrow$ *?P* φ›)
⟨*proof*⟩

lemma *prenex-left-props*:
‹*is-prenex* $\varphi \wedge$ *is-prenex* $\psi \Longrightarrow$
is-prenex (*prenex-left* $\varphi\ \psi$) $\wedge$
FV (*prenex-left* $\varphi\ \psi$) = FV ($\varphi \longrightarrow \psi$) $\wedge$
(*language* {(*prenex-left* $\varphi\ \psi$)} = *language* {($\varphi \longrightarrow \psi$)}) $\wedge$
($\forall$ (I :: *'a intrp*) β. *dom* $I \neq \{\} \longrightarrow$ (I,β $\models$ *prenex-left* $\varphi\ \psi \longleftrightarrow$ I,β $\models$ $\varphi \longrightarrow \psi$))›
⟨*proof*⟩

theorem *prenex-props*: ‹*is-prenex* (*prenex* φ) $\wedge$ (*FV* (*prenex* φ) = *FV* φ) $\wedge$
(*language* {*prenex* φ} = *language* {φ}) $\wedge$
($\forall$(*I* :: 'a *intrp*) β. *dom I* $\neq$ {} $\longrightarrow$ (*I*, β $\models$ (*prenex* φ)) $\longleftrightarrow$ (*I*, β $\models$ φ))›
⟨*proof*⟩

corollary *is-prenex-prenex* [*simp*]: ‹*is-prenex* (*prenex* φ)›
and *FV-prenex* [*simp*]: ‹*FV* (*prenex* φ) = *FV* φ›
and *language-prenex* [*simp*]: ‹*language* {*prenex* φ} = *language* {φ}›
⟨*proof*⟩

corollary *prenex-holds* [*simp*]: ‹*dom I* $\neq$ {} $\Longrightarrow$ (*I*,β $\models$ (*prenex* φ)) $\longleftrightarrow$ (*I*,β $\models$ φ)›
⟨*proof*⟩

lemma *prenex-satisfies* [*simp*]:
assumes *dom M* $\neq$ {}
shows *satisfies M* {*prenex* φ} $\longleftrightarrow$ *satisfies M* {φ}
⟨*proof*⟩

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* $\mathcal{M}$ = *Abs-intrp* ((*dom* $\mathcal{M}$), (λ*k zs*. (*intrp-fn* $\mathcal{M}$) (*numsnd k*) *zs*), (*intrp-rel* $\mathcal{M}$))›

lemma *bump-dom* [*simp*]: ‹*dom* (*bump-intrp* $\mathcal{M}$) = *dom* $\mathcal{M}$›
⟨*proof*⟩

lemma *bump-intrp-fn* [*simp*]: ‹*intrp-fn* (*bump-intrp* $\mathcal{M}$) (*numpair 0 f*) *ts* = *intrp-fn* $\mathcal{M}$ *f ts*›
⟨*proof*⟩

lemma *bump-intrp-rel* [*simp*]: ‹*intrp-rel* (*bump-intrp* $\mathcal{M}$) n = *intrp-rel* $\mathcal{M}$ n›
⟨*proof*⟩

fun *bump-nterm* :: *nterm* ⇒ *nterm* **where**
‹*bump-nterm* (*Var* x) = *Var* x›
| ‹*bump-nterm* (*Fun* f *ts*) = *Fun* (*numpair* 0 f) (*map* *bump-nterm* *ts*)›

fun *bump-form* :: *form* ⇒ *form* **where**
‹*bump-form* ⊥ = ⊥›
| ‹*bump-form* (*Atom* p *ts*) = *Atom* p (*map* *bump-nterm* *ts*)›
| ‹*bump-form* ($\varphi \longrightarrow \psi$) = (*bump-form* φ) ⟶ (*bump-form* ψ)›
| ‹*bump-form* ($\forall$ x. φ) = $\forall$ x. (*bump-form* φ)›

lemma *bumpterm*: ‹$[\![t]\!]^{\mathcal{M},\beta}$ = $[\![$*bump-nterm* $t]\!]^{$*bump-intrp* $\mathcal{M},\beta}$›
⟨*proof*⟩

lemma *bump-intrp-rel-holds*: ‹(*map* (λt. $[\![t]\!]^{\mathcal{M},\beta}$) *ts* ∈ *intrp-rel* $\mathcal{M}$ n) =
(*map* ((λt. $[\![t]\!]^{$*bump-intrp* $\mathcal{M},\beta}$) ∘ *bump-nterm*) *ts* ∈ *intrp-rel* (*bump-intrp* $\mathcal{M}$) n)›
⟨*proof*⟩

lemma *bumpform*: ‹$\mathcal{M}$, β ⊨ φ = (*bump-intrp* $\mathcal{M}$), β ⊨ (*bump-form* φ)›
⟨*proof*⟩

lemma *functions-form-bumpform*: ‹(f, m) ∈ *functions-form* (*bump-form* φ) ⟹
∃ k. (f = *numpair* 0 k) ∧ (k, m) ∈ *functions-form* φ›
⟨*proof*⟩

lemma *bumpform-interpretation*: ‹*is-interpretation* (*language* {φ}) $\mathcal{M}$ ⟹
is-interpretation (*language* {(*bump-form* φ)}) (*bump-intrp* $\mathcal{M}$)›
⟨*proof*⟩

fun *unbump-nterm* :: *nterm* ⇒ *nterm* **where**
‹*unbump-nterm* (*Var* x) = *Var* x›
| ‹*unbump-nterm* (*Fun* f *ts*) = *Fun* (*numsnd* f) (*map* *unbump-nterm* *ts*)›

fun *unbump-form* :: *form* ⇒ *form* **where**
‹*unbump-form* ⊥ = ⊥›
| ‹*unbump-form* (*Atom* p *ts*) = *Atom* p (*map* *unbump-nterm* *ts*)›
| ‹*unbump-form* ($\varphi \longrightarrow \psi$) = (*unbump-form* φ) ⟶ (*unbump-form* ψ)›
| ‹*unbump-form* ($\forall$ x. φ) = $\forall$ x. (*unbump-form* φ)›

lemma *unbumpterm* [*simp*]: *unbump-nterm* (*bump-nterm* t) = t
⟨*proof*⟩

lemma *unbumpform* [*simp*]: ‹*unbump-form* (*bump-form* φ) = φ›
⟨*proof*⟩

definition *unbump-intrp* :: *'m intrp* ⇒ *'m intrp* **where**
‹*unbump-intrp* $\mathcal{M}$ = *Abs-intrp* (*dom* $\mathcal{M}$, (λ*k zs*. (*intrp-fn* $\mathcal{M}$) (*numpair 0 k*) *zs*), (*intrp-rel* $\mathcal{M}$))›

lemma *unbump-term-intrp*: ‹$[\![$*bump-nterm t*$]\!]^{\mathcal{M},\beta}$ = $[\![t]\!]^{$*unbump-intrp* $\mathcal{M},\beta}$›
⟨*proof*⟩

lemma *unbump-holds*: ‹($\mathcal{M}$,β ⊨ *bump-form* φ) = (*unbump-intrp* $\mathcal{M}$,β ⊨ φ)›
⟨*proof*⟩

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*:
(⋀*x y*. *P* (*Var x*) (*Var y*))
⟹ (⋀*x g us*. *P* (*Var x*) (*Fun g us*))
⟹ (⋀*f ts y*. *P* (*Fun f ts*) (*Var y*))
⟹ (⋀*f ts g us*. (⋀*p q*. *p* ∈ *set ts* ⟹ *q* ∈ *set us* ⟹ *P p q*) ⟹ *P* (*Fun f ts*) (*Fun g us*))
⟹ *P t1 t2*
⟨*proof*⟩

lemma *num-of-term-inj*: ‹*num-of-term s* = *num-of-term t* ⟷ *s* = *t*›
⟨*proof*⟩

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*›

⟨*proof*⟩

lemma *num-of-form-inj*: ‹*num-of-form* φ = *num-of-form* $\psi \longleftrightarrow \varphi = \psi$›
⟨*proof*⟩

consts *term-of-num* :: *nat* ⇒ *nterm*
specification (*term-of-num*) ‹∀ *n*. *term-of-num n* = (*THE t*. *num-of-term t* = *n*)›
⟨*proof*⟩

lemma *term-of-num-of-term* [*simp*]: ‹*term-of-num*(*num-of-term t*) = *t*›
⟨*proof*⟩

consts *form-of-num* :: *nat* ⇒ *form*
specification (*form-of-num*) ‹∀ *n*. *form-of-num n* = (*THE* φ. *num-of-form* φ = *n*)›
⟨*proof*⟩

lemma *form-of-num-of-form* [*simp*]: ‹*form-of-num* (*num-of-form* φ) = φ›
⟨*proof*⟩

end

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

lemma *witness-imp-holds-exists*:
assumes ‹*is-interpretation* (*functions-term t*, *preds*) (*I* :: (*nat*, *nat*) *term intrp*)›
and
‹*is-valuation I* β› **and**
‹*I*, $\beta \models$ (φ ·$_{fm}$ (*subst x t*))›
shows ‹*I*, $\beta \models$ (∃ *x*. φ)›
⟨*proof*⟩

definition *skolem1* :: *nat* ⇒ *nat* ⇒ *form* ⇒ *form* **where**
‹*skolem1 f x* φ = φ ·$_{fm}$ (*subst x* (*Fun f* (*map Var* (*sorted-list-of-set* (*FV* (∃ *x*. φ)))))))›

lemma *fvt-var-x-simp*:
‹*FVT* (*Var x* · *subst x* (*Fun f* (*map Var* (*sorted-list-of-set* (*FV* φ − {*x*}))))) = *FV* φ − {*x*}›
⟨*proof*⟩

lemma *holds-indep-intrp-if2*:
fixes *I I'* :: *'a intrp*
shows
‹⟦*I*,*β* ⊨ *φ*; *dom I* = *dom I'*; ⋀*p*. *intrp-rel I p* = *intrp-rel I' p*;
⋀*f zs*. (*f*, *length zs*) ∈ *functions-form φ* ⟹ *intrp-fn I f zs* = *intrp-fn I' f zs*⟧ ⟹
I',*β* ⊨ *φ*›
⟨*proof*⟩

lemma *fun-upds-prop*: ‹*length xs* = *length zs* ⟹ ∀ *z*∈*set zs*. *P z* ⟹ ∀ *v*. *P* (*g v*)
⟹ ∀ *v*. *P* ((*foldr* (λ*kv f*. *fun-upd f* (*fst kv*) (*snd kv*)) (*zip xs zs*) *g*) *v*)›
⟨*proof*⟩

lemma ‹{*z*. ∃ *y*. *y* ∈ *FV φ* ∧ *z* ∈ *functions-term* (*Var y* · *subst x t*)} = *functions-term t* ∨
{*z*. ∃ *y*. *y* ∈ *FV φ* ∧ *z* ∈ *functions-term* (*Var y* · *subst x t*)} = {}›
⟨*proof*⟩

lemma *func-form-subst*: ‹*x* ∈ *FV φ* ⟹ (*f*, *length ts*) ∈ *functions-form* (*φ* ·*fm* *subst x* (*Fun f ts*))›
⟨*proof*⟩

lemma *holds-formsubst*:
M,*β* ⊨ (*p* ·*fm* *i*) ⟷ *M*,(λ*t*. ⟦*t*⟧^*M*,*β*) ∘ *i* ⊨ *p*
⟨*proof*⟩

lemma *holds-formsubst1*:
M,*β* ⊨ (*p* ·*fm* *Var*(*x*:=*t*)) ⟷ *M*,*β*(*x* := ⟦*t*⟧^*M*,*β*) ⊨ *p*
⟨*proof*⟩

lemma *holds-formsubst2*:
M,*β* ⊨ (*p* ·*fm* *subst x t*) ⟷ *M*,*β*(*x* := ⟦*t*⟧^*M*,*β*) ⊨ *p*
⟨*proof*⟩

lemma *size-nonzero* [*simp*]: *size fm* > *0*
⟨*proof*⟩

lemma
assumes *prenex-ex-phi*: ‹*is-prenex* (∃ *x*. *φ*)›
and *notin-ff*: ‹¬ (*f*, *card* (*FV* (∃ *x*. *φ*))) ∈ *functions-form* (∃ *x*. *φ*)›
shows *holds-skolem1a*: *is-prenex* (*skolem1 f x φ*) (**is** *?A*)
and *holds-skolem1b*: *FV* (*skolem1 f x φ*) = *FV* (∃ *x*. *φ*) (**is** *?B*)
and *holds-skolem1c*:
Prenex-Normal-Form.size (*skolem1 f x φ*) < *Prenex-Normal-Form.size* (∃ *x*.

φ) (**is** *?C*)
and *holds-skolem1d*: *predicates-form* (*skolem1 f x* φ) = *predicates-form* ($\exists$ *x.* φ) (**is** *?D*)
and *holds-skolem1e*: *functions-form* ($\exists$ *x.* φ) $\subseteq$ *functions-form* (*skolem1 f x* φ) (**is** *?E*)
and *holds-skolem1f*: *functions-form* (*skolem1 f x* φ)
$\subseteq$ (*f*, *card* (*FV* ($\exists$ *x.* φ))) $\triangleright$ *functions-form* ($\exists$ *x.* φ) (**is** *?F*)
⟨*proof*⟩

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

lemma *holds-skolem1g*:
assumes *prenex-ex-phi*: ‹*is-prenex* ($\exists$ *x.* φ)›
and *notin-ff*: ‹$\neg$ (*f*, *card* (*FV* ($\exists$ *x.* φ))) $\in$ *functions-form* ($\exists$ *x.* φ)›
fixes *I* :: *'a intrp*
assumes *interp-I*: *is-interpretation* (*language* {φ}) *I*
and *nempty-I*: *dom I* $\neq$ {}
and *valid*: $\bigwedge\beta$. *is-valuation I* β $\Longrightarrow$ *I*,β $\models$ ($\exists$ *x.* φ)
obtains *M* **where** *dom M* = *dom I*
intrp-rel M = *intrp-rel I*
$\bigwedge$*g zs. g* $\neq$ *f* $\vee$ *length zs* $\neq$ *card* (*FV* ($\exists$ *x.* φ)) $\Longrightarrow$ *intrp-fn M g zs* = *intrp-fn I g zs*
is-interpretation (*language* {*skolem1 f x* φ}) *M*
$\bigwedge\beta$. *is-valuation M* β $\Longrightarrow$ *M*,β $\models$ *skolem1 f x* φ
⟨*proof*⟩

lemma *holds-skolem1h*:
assumes *prenex-ex-phi*: ‹*is-prenex* ($\exists$ *x.* φ)› **and** ‹$\neg$ (*f*, *card* (*FV* ($\exists$ *x.* φ))) $\in$ *functions-form* ($\exists$ *x.* φ)›
assumes *is-intrp*: *is-interpretation* (*language* {*skolem1 f x* φ}) *N*
and *nempty-N*: *dom N* $\neq$ {}
and *is-val*: *is-valuation N* β
and *skol-holds*: *N*,β $\models$ *skolem1 f x* φ
shows *N*,β $\models$ ($\exists$ *x.* φ)
⟨*proof*⟩

lemma *holds-skolem1*:
assumes ‹*is-prenex* ($\exists$ *x.* φ)› **and** ‹$\neg$ (*f*, *card* (*FV* ($\exists$ *x.* φ))) $\in$ *functions-form* ($\exists$ *x.* φ)›
shows ‹*is-prenex* (*skolem1 f x* φ) $\wedge$
FV (*skolem1 f x* φ) = *FV* ($\exists$ *x.* φ) $\wedge$
size (*skolem1 f x* φ) < *size* ($\exists$ *x.* φ) $\wedge$
predicates-form (*skolem1 f x* φ) = *predicates-form* ($\exists$ *x.* φ) $\wedge$
functions-form ($\exists$ *x.* φ) $\subseteq$ *functions-form* (*skolem1 f x* φ) $\wedge$
functions-form (*skolem1 f x* φ) $\subseteq$ *insert* (*f*, *card* (*FV* ($\exists$ *x.* φ))) (*functions-form* ($\exists$ *x.* φ)) $\wedge$
($\forall$ (*I* :: *'a intrp*). *is-interpretation* (*language* {φ}) *I* $\wedge$
$\neg$ (*dom I* = {}) $\wedge$

(∀ β. *is-valuation I β* ⟶ (*I*, β ⊨ (∃ *x*. φ))) ⟶
(∃ (*M* :: 'a *intrp*). *dom M* = *dom I* ∧
intrp-rel M = *intrp-rel I* ∧
(∀ *g zs*. ¬*g*=*f* ∨ ¬(*length zs* = *card* (*FV* (∃ *x*. φ))) ⟶ *intrp-fn M g zs* = *intrp-fn I g zs*) ∧
is-interpretation (*language* {*skolem1 f x* φ}) *M* ∧
(∀ β. *is-valuation M* β ⟶ (*M*, β ⊨ (*skolem1 f x* φ))))) ∧
(∀ (*N* :: 'a *intrp*). *is-interpretation* (*language* {*skolem1 f x* φ}) *N* ∧
¬ (*dom N* = {}) ⟶
(∀ β. *is-valuation N* β ∧ (*N*, β ⊨ (*skolem1 f x* φ)) ⟶ (*N*, β ⊨ (∃ *x*. φ))))
›
⟨*proof*⟩

lemma *skolems-ex*: ‹∃ *skolems*. ∀ φ. *skolems* φ = (λ*k*. *ppat* (λ*x* ψ. ∀ *x*. (*skolems* ψ *k*))
(λ*x* ψ. *skolems* (*skolem1* (*numpair J k*) *x* ψ) (*Suc k*)) (λψ. ψ) φ)›
⟨*proof*⟩

consts *skolems* :: *nat* ⇒ *form* ⇒ *nat* ⇒ *form*
specification (*skolems*)
skolems-eq: ‹⋀*J* ψ *k*. *skolems J* ψ *k*
= *ppat* (λ*x* φ'. ∀ *x*. (*skolems J* φ' *k*)) (λ*x* φ'. *skolems J* (*skolem1* (*numpair J k*) *x* φ') (*Suc k*)) (λφ. φ) ψ›
⟨*proof*⟩

bounding the possible Skolem functions in a given formula

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

lemma *skolems-bounded-mono*: ⟦*skolems-bounded p J k'*; *k'*≤*k*⟧ ⟹ *skolems-bounded p J k*
⟨*proof*⟩

lemma *skolems-bounded-prenex*: *skolems-bounded* φ *K k* ⟹ *skolems-bounded* (*prenex* φ) *K k*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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* ≠ {} *satisfies M* {*p*}
shows ∃ *M'*. *dom M'* = *dom M* ∧ *intrp-rel M'* = *intrp-rel M* ∧
(∀ *g zs*. *intrp-fn M' g zs* ≠ *intrp-fn M g zs*
⟶ (∃ *l*. *k* ≤ *l* ∧ *g* = *numpair J l*)) ∧
is-interpretation (*language* {*skolems J p k*}) *M'* ∧
satisfies M' {*skolems J p k*}
⟨*proof*⟩

corollary *holds-skolems-prenex-A*:
assumes *is-prenex φ skolems-bounded φ K 0*
shows *universal*(*skolems K φ 0*) ∧ (*FV* (*skolems K φ 0*) = *FV φ*) ∧
predicates-form (*skolems K φ 0*) = *predicates-form φ* ∧
functions-form φ ⊆ *functions-form* (*skolems K φ 0*) ∧
functions-form (*skolems K φ 0*) ⊆ {(*numpair K l*,*m*) | *l m*. *True*} ∪ (*functions-form φ*)
⟨*proof*⟩

corollary *holds-skolems-prenex-B*:
assumes *is-prenex φ skolems-bounded φ K 0*
and *is-interpretation* (*language* {*skolems K φ 0*}) *M dom M* ≠ {}
and *is-valuation M β M*,*β* ⊨ *skolems K φ 0*
shows *M*,*β* ⊨ *φ*
⟨*proof*⟩

corollary *holds-skolems-prenex-C*:
assumes *is-prenex φ skolems-bounded φ K 0*
and *is-interpretation* (*language* {*φ*}) *M dom M* ≠ {} *satisfies M* {*φ*}
shows ∃ *M'*. *dom M'* = *dom M* ∧ *intrp-rel M'* = *intrp-rel M* ∧
(∀ *g zs*. *intrp-fn M' g zs* ≠ *intrp-fn M g zs* ⟶ (∃ *l*. *g* = *numpair K l*)) ∧
is-interpretation (*language* {*skolems K φ 0*}) *M'* ∧

satisfies M' {*skolems* K φ 0}
⟨*proof*⟩

definition *skopre* **where**
‹*skopre* k φ = *skolems* k (*prenex* φ) 0›

corollary *skopre-model-A*:
assumes *skolems-bounded* φ K 0
shows *universal*(*skopre* K φ) ∧ (*FV* (*skopre* K φ) = *FV* φ) ∧
predicates-form (*skopre* K φ) = *predicates-form* φ ∧
functions-form φ ⊆ *functions-form* (*skopre* K φ) ∧
functions-form (*skopre* K φ) ⊆ {(*numpair* K *l*,*m*) | *l m*. *True*} ∪ (*functions-form* φ)
⟨*proof*⟩

corollary *skopre-model-B*:
assumes *skolems-bounded* φ K 0
and *is-interpretation* (*language* {*skopre* K φ}) M *dom* M ≠ {}
and *is-valuation* M β M,β ⊨ *skopre* K φ
shows M,β ⊨ φ
⟨*proof*⟩

corollary *skopre-model-C*:
assumes *skolems-bounded* φ K 0
and *is-interpretation* (*language* {φ}) M *dom* M ≠ {} *satisfies* M {φ}
shows ∃ M'. *dom* M' = *dom* M ∧ *intrp-rel* M' = *intrp-rel* M ∧
(∀ *g zs*. *intrp-fn* M' *g zs* ≠ *intrp-fn* M *g zs* ⟶ (∃ *l*. *g* = *numpair* K *l*)) ∧
is-interpretation (*language* {*skopre* K φ}) M' ∧
satisfies M' {*skopre* K φ}
⟨*proof*⟩

definition *skolemize* **where**
‹*skolemize* φ = *skopre* (*num-of-form* (*bump-form* φ) + 1) (*bump-form* φ)›

lemma *no-skolems-bump-nterm*:
shows *i*>0 ⟹ (*numpair i l*, *m*) ∉ *functions-term* (*bump-nterm t*)
⟨*proof*⟩

lemma *no-skolems-bump-form*: *i*>0 ⟹ *skolems-bounded* (*bump-form* φ) *i* 0
⟨*proof*⟩

lemma *universal-skolemize* [*iff*]: *universal* (*skolemize* φ)
and *FV-skolemize* [*simp*]: *FV* (*skolemize* φ) = *FV* (*bump-form* φ)
and *predicates-form-skolemize* [*simp*]: *predicates-form* (*skolemize* φ) = *predi-*

cates-form (*bump-form* φ)
⟨*proof*⟩

lemma *functions-bump-form*: *functions-form* (*bump-form* φ) $\subseteq$ *functions-form* (*skolemize* φ)
⟨*proof*⟩

lemma *functions-skolemize*:
functions-form (*skolemize* φ) $\subseteq$ {(*numpair* (*num-of-form* (*bump-form* φ)+*1*) *l*, *m*) |*k l m*. *True*} $\cup$ *functions-form* (*bump-form* φ)
⟨*proof*⟩

lemma *skolemize-imp-holds-bump-form*:
assumes *is-interpretation* (*language* {*skolemize* φ}) *N dom N* $\neq$ {}
and *is-valuation N* β *N*,β $\models$ *skolemize* φ
shows *N*,β $\models$ *bump-form* φ
⟨*proof*⟩

lemma *is-interpretation-skolemize*:
assumes *is-interpretation* (*language* {*bump-form* φ}) *M dom M* $\neq$ {} *satisfies M* {*bump-form* φ}
obtains *M'* **where** *dom M' = dom M intrp-rel M' = intrp-rel M*
$\bigwedge$*g zs. intrp-fn M' g zs* $\neq$ *intrp-fn M g zs* $\Longrightarrow$ $\exists$ *l. g = numpair* (*num-of-form* (*bump-form* φ) + *1*) *l*
is-interpretation (*language* {*skolemize* φ}) *M' satisfies M'* {*skolemize* φ}
⟨*proof*⟩

lemma *functions-form-skolemize*:
assumes ‹(*f*,*m*) $\in$ *functions-form* (*skolemize* φ)›
obtains *k* **where** ‹*f = numpair 0 k*› ‹(*k*,*m*) $\in$ *functions-form* φ› | *l* **where** ‹*f = numpair* (*num-of-form* (*bump-form* φ) + *1*) *l*›
⟨*proof*⟩

definition *skomod1* **where**
‹*skomod1* φ *M* $\equiv$
if satisfies M {φ}
then (*SOME M'*. *dom M' = dom* (*bump-intrp M*) $\wedge$
intrp-rel M' = intrp-rel (*bump-intrp M*) $\wedge$
($\forall$ *g zs. intrp-fn M' g zs* $\neq$ *intrp-fn* (*bump-intrp M*) *g zs* $\longrightarrow$
($\exists$ *l. g = numpair* (*num-of-form* (*bump-form* φ) + *1*) *l*)) $\wedge$
is-interpretation (*language* {*skolemize* φ}) *M'* $\wedge$ *satisfies M'* {*skolemize* φ})
else (*Abs-intrp* (*dom M*, (λ*g zs.* (*SOME a. a* $\in$ *dom M*)), *intrp-rel M*))›

lemma *skomod1-works*:

assumes M: ‹*is-interpretation* (*language* {φ}) M› ‹*dom* $M \neq \{\}$›
shows ‹*dom* (*skomod1* φ M) = *dom* (*bump-intrp* M) ∧
intrp-rel (*skomod1* φ M) = *intrp-rel* (*bump-intrp* M) ∧
is-interpretation (*language* {*skolemize* φ}) (*skomod1* φ M) ∧
(*satisfies* M {φ} ⟶
(∀ *g zs*. *intrp-fn* (*skomod1* φ M) *g zs* ≠ *intrp-fn* (*bump-intrp* M) *g zs* ⟶
(∃ *l*. *g* = *numpair* (*num-of-form* (*bump-form* φ) + *1*) *l*)) ∧
satisfies (*skomod1* φ M) {*skolemize* φ})›
⟨*proof*⟩

definition *skomod-FN* ≡ λ*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**
‹*skomod M* ≡ *Abs-intrp* (*dom M*, *skomod-FN M*, *intrp-rel M*)›

lemma *skomod-interpretation*:
assumes ‹*is-interpretation* (*language* {φ}) M› ‹*dom* $M \neq \{\}$›
shows ‹*is-interpretation* (*language* {*skolemize* φ}) (*skomod M*)›
⟨*proof*⟩

proposition *skomod-works*:
assumes ‹*is-interpretation* (*language* {φ}) M› ‹*dom* $M \neq \{\}$›
shows ‹*satisfies* M {φ} ⟷ *satisfies* (*skomod M*) {*skolemize* φ}›
⟨*proof*⟩

proposition *skolemize-satisfiable*:
‹(∃ *M*::'*a intrp*. *dom M* ≠ {} ∧ *is-interpretation* (*language S*) *M* ∧
satisfies M S) ⟷
(∃ *M*::'*a intrp*. *dom M* ≠ {} ∧ *is-interpretation* (*language* (*skolemize* ‘ *S*)) *M*
∧ *satisfies M* (*skolemize* ‘ *S*))› (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

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*⟩

lemma *specialize-qfree*: ‹*universal* φ ⟹ *qfree* (*specialize* φ)›
⟨*proof*⟩

lemma *functions-form-specialize* [*simp*]: *functions-form*(*specialize* φ) = *functions-form* φ
⟨*proof*⟩

lemma *predicates-form-form-specialize* [*simp*]: *predicates-form*(*specialize* φ) = *predicates-form* φ
⟨*proof*⟩

lemma *specialize-language*: ‹*language* (*specialize* ‘ *S*) = *language S*›
⟨*proof*⟩

definition *skolem* :: *form* ⇒ *form* **where**
‹*skolem* φ = *specialize*(*skolemize* φ)›

lemma *skolem-qfree*: ‹*qfree* (*skolem* φ)›
⟨*proof*⟩

theorem *skolem-satisfiable*:
‹(∃ *M*::'*a intrp*. *dom M* ≠ {} ∧ *is-interpretation* (*language S*) *M* ∧ *satisfies M S*)
⟷ (∃ *M*::'*a intrp*. *dom M* ≠ {} ∧ *is-interpretation* (*language* (*skolem* ‘ *S*)) *M* ∧
satisfies M (*skolem* ‘ *S*))›
⟨*proof*⟩

end

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

inductive-set *terms-set* :: ‹(*nat* × *nat*) *set* ⇒ *nterm set*› **for** *fns* :: ‹(*nat* × *nat*) *set*› **where**
vars: ‹(*Var v*) ∈ *terms-set fns*›
| *fn*: ‹(*Fun f ts*) ∈ *terms-set fns*›
if ‹(*f*, *length ts*) ∈ *fns*› ‹⋀*t*. *t* ∈ *set ts* ⟹ *t* ∈ *terms-set fns*›

lemma *struct-terms-set* [*iff*]: ‹*struct* (*terms-set A*)›

⟨*proof*⟩

lemma *stupid-canondom*: ‹$t \in$ *terms-set* (*fst* $\mathcal{L}$) $\Longrightarrow$ *functions-term* $t \subseteq$ (*fst* $\mathcal{L}$)›
⟨*proof*⟩

lemma *finite-subset-instance*: ‹*finite* $t' \Longrightarrow t' \subseteq \{\varphi \cdot_{fm} \sigma \,|\sigma\ \varphi.\ P\ \sigma \wedge \varphi \in s\} \Longrightarrow$
($\exists\, t.$ *finite* $t \wedge t \subseteq s \wedge t' \subseteq \{\varphi \cdot_{fm} \sigma \,|\sigma\ \varphi.\ P\ \sigma \wedge \varphi \in t\}$)›
⟨*proof*⟩

lemma *finite-subset-skolem*: ‹*finite* $u \Longrightarrow u \subseteq \{$*skolem* $\varphi \,|\varphi.\ \varphi \in s\} \Longrightarrow$
($\exists\, t.$ *finite* $t \wedge t \subseteq s \wedge u = \{$*skolem* $\varphi \,|\varphi.\ \varphi \in t\}$)›
⟨*proof*⟩

lemma *valuation-exists*: ‹$\neg$ (*dom* $M = \{\}) \Longrightarrow \exists\, \beta.$ *is-valuation* $M\ \beta$›
⟨*proof*⟩

lemma *holds-itlist-exists*:
‹(M,$\beta \models$ (*foldr* ($\lambda x\ p.\ \exists\, x.\ p$) *xs* φ)) $\longleftrightarrow$
($\exists\, as.$ *length as* = *length xs* $\wedge$ *set as* $\subseteq$ *dom* $M \wedge$
(M,(*foldr* ($\lambda u\ \beta.\ \beta$(*fst* u := *snd* u)) (*rev* (*zip xs as*)) β) $\models \varphi$))›
⟨*proof*⟩

definition *canonical* :: ‹(*nat* × *nat*) *set* × (*nat* × *nat*) *set* ⇒ *nterm intrp* ⇒ *bool*›
where
‹*canonical* $\mathcal{L}$ $\mathcal{M}$ ≡ (*dom* $\mathcal{M}$ = *terms-set* (*fst* $\mathcal{L}$) ∧ (∀f. *intrp-fn* $\mathcal{M}$ f = *Fun* f))›

definition *pintrp-of-intrp* :: ‹'*m intrp* ⇒ (*nat* ⇒ '*m*) ⇒ (*form* ⇒ *bool*)› **where**
‹*pintrp-of-intrp* $\mathcal{M}$ β = (λφ. $\mathcal{M}$,$\beta \models \varphi$)›

definition
canon-of-prop :: ‹((*nat* × *nat*) *set* × (*nat* × *nat*) *set*) ⇒ (*form* ⇒ *bool*) ⇒ *nterm intrp*› **where**
‹*canon-of-prop* $\mathcal{L}$ I ≡ *Abs-intrp* (*terms-set* (*fst* $\mathcal{L}$), *Fun*, (λp. {*ts*. I (*Atom* p *ts*)}))›

lemma *dom-canon-of-prop* [*simp*]: ‹*dom* (*canon-of-prop* $\mathcal{L}$ I) = *terms-set* (*fst* $\mathcal{L}$)›
⟨*proof*⟩

lemma *intrp-fn-canon-of-prop* [*simp*]: ‹*intrp-fn* (*canon-of-prop* $\mathcal{L}$ I) = *Fun*›
⟨*proof*⟩

lemma *intrp-rel-canon-of-prop* [*simp*]: ‹*intrp-rel* (*canon-of-prop* $\mathcal{L}$ I) = (λp. {*ts*. I (*Atom* p *ts*)})›
⟨*proof*⟩

lemma *pholds-pintrp-of-intrp*:
‹*qfree* $\varphi \Longrightarrow$ (*pintrp-of-intrp* $\mathcal{M}$ β) $\models_p$ $\varphi \longleftrightarrow \mathcal{M}$,$\beta \models \varphi$›
⟨*proof*⟩

lemma *intrp-of-canon-of-prop* [*simp*]:
‹*pintrp-of-intrp* (*canon-of-prop* $\mathcal{L}$ I) *Var* (*Atom* p *ts*) = I (*Atom* p *ts*)›
⟨*proof*⟩

lemma *holds-canon-of-prop*:
assumes ‹*qfree* φ› **shows** ‹(*canon-of-prop* $\mathcal{L}$ I),*Var* $\models \varphi \longleftrightarrow I \models_p \varphi$›
⟨*proof*⟩

lemma *holds-canon-of-prop-general*:
assumes ‹*qfree* φ› **shows** ‹(*canon-of-prop* $\mathcal{L}$ I),$\beta \models \varphi \longleftrightarrow I \models_p (\varphi \cdot_{fm} \beta)$›
⟨*proof*⟩

lemma *canonical-canon-of-prop*: ‹*canonical* $\mathcal{L}$ (*canon-of-prop* $\mathcal{L}$ I)›
⟨*proof*⟩

lemma *interpretation-canon-of-prop*: ‹*is-interpretation* $\mathcal{L}$ (*canon-of-prop* $\mathcal{L}$ I)›
⟨*proof*⟩

lemma *prop-valid-imp-fol-valid*: ‹*qfree* $\varphi \wedge (\forall I.\ I \models_p \varphi) \Longrightarrow (\forall \mathcal{M}\ \beta.\ \mathcal{M}$,$\beta \models \varphi)$›
⟨*proof*⟩

lemma *fol-valid-imp-prop-valid*: ‹*qfree* $\varphi \wedge (\forall \mathcal{M}\ \beta.$ *canonical* (*language* {φ}) $\mathcal{M}$ $\longrightarrow \mathcal{M}$,$\beta \models \varphi$)
$\Longrightarrow \forall I.\ I \models_p \varphi$›
⟨*proof*⟩

lemma *satisfies-psatisfies*: ‹⟦$\varphi \in \Phi$; $\Phi \subseteq$ {φ. *qfree* φ}; *satisfies* $\mathcal{M}$ Φ; *is-valuation* $\mathcal{M}$ β⟧ $\Longrightarrow$
psatisfies (*pintrp-of-intrp* $\mathcal{M}$ β) Φ›
⟨*proof*⟩

lemma *psatisfies-instances*:
assumes *qf*: ‹$\Phi \subseteq \{\varphi.\ qfree\ \varphi\}$›
and *ps*: ‹*psatisfies* I $\{\varphi \cdot_{fm} \beta\ |\varphi\ \beta.\ (\forall x.\ \beta\ x \in terms\text{-}set\ (fst\ \mathcal{L})) \wedge \varphi \in \Phi\}$›
shows ‹*satisfies* (*canon-of-prop* $\mathcal{L}$ I) Φ›
⟨*proof*⟩

lemma *satisfies-instances*:
assumes ‹*is-interpretation* (*language* Ξ) $\mathcal{M}$›
shows ‹*satisfies* $\mathcal{M}$ $\{\varphi \cdot_{fm} \sigma\ |\varphi\ \sigma.\ \varphi \in \Phi \wedge (\forall x.\ \sigma\ x \in terms\text{-}set\ (fst\ (language\ \Xi)))\} \longleftrightarrow$
satisfies $\mathcal{M}$ Φ›
⟨*proof*⟩

lemma *compact-canon-qfree*:
assumes *qf*: ‹$\Phi \subseteq \{\varphi.\ qfree\ \varphi\}$›
and *int*: ‹$\bigwedge\Psi.\ [\![finite\ \Psi;\ \Psi \subseteq \Phi]\!] \Longrightarrow \exists \mathcal{M}::'a\ intrp.\ is\text{-}interpretation\ (language\ \Xi)\ \mathcal{M} \wedge dom\ \mathcal{M} \neq \{\} \wedge satisfies\ \mathcal{M}\ \Psi$›
obtains $\mathcal{C}$ **where** ‹*is-interpretation* (*language* Ξ) $\mathcal{C}$› ‹*canonical* (*language* Ξ) $\mathcal{C}$› ‹*satisfies* $\mathcal{C}$ Φ›
⟨*proof*⟩

lemma *interpretation-restrictlanguage*:
‹$\Psi \subseteq \Phi \Longrightarrow$ *is-interpretation* (*language* Φ) $\mathcal{M} \Longrightarrow$ *is-interpretation* (*language* Ψ) $\mathcal{M}$›
⟨*proof*⟩

lemma *interpretation-extendlanguage*:
fixes $\mathcal{M}$:: ‹$'a$ *intrp*›
assumes *int*: ‹*is-interpretation* (*language* Ψ) $\mathcal{M}$› **and** ‹*dom* $\mathcal{M} \neq \{\}$›
and ‹*satisfies* $\mathcal{M}$ Ψ›
obtains $\mathcal{N}$ **where** ‹*dom* $\mathcal{N}$ = *dom* $\mathcal{M}$› ‹*intrp-rel* $\mathcal{N}$ = *intrp-rel* $\mathcal{M}$›
‹*is-interpretation* (*language* Φ) $\mathcal{N}$› ‹*satisfies* $\mathcal{N}$ Ψ›
⟨*proof*⟩

theorem *compact-ls*:
assumes ‹$\bigwedge\Psi.\ [\![finite\ \Psi;\ \Psi \subseteq \Phi]\!] \Longrightarrow \exists \mathcal{M}::'a\ intrp.\ is\text{-}interpretation\ (language\ \Phi)\ \mathcal{M} \wedge dom\ \mathcal{M} \neq \{\} \wedge satisfies\ \mathcal{M}\ \Psi$›
obtains $\mathcal{C}$::‹*nterm intrp*› **where** ‹*is-interpretation* (*language* Φ) $\mathcal{C}$› ‹*dom* $\mathcal{C} \neq \{\}$› ‹*satisfies* $\mathcal{C}$ Φ›
⟨*proof*⟩

lemma *canon*:
assumes ‹*is-interpretation* (*language* Φ) $\mathcal{M}$› ‹*dom* $\mathcal{M} \neq \{\}$› ‹*satisfies* $\mathcal{M}$ Φ›
obtains $\mathcal{C}$::‹*nterm intrp*› **where** ‹*is-interpretation* (*language* Φ) $\mathcal{C}$› ‹*dom* $\mathcal{C} \neq \{\}$› ‹*satisfies* $\mathcal{C}$ Φ›
⟨*proof*⟩

definition *lowmod* :: ‹*nterm intrp* ⇒ *nat intrp*› **where**
‹*lowmod* $\mathcal{M}$ ≡ *Abs-intrp* (*num-of-term* ' (*dom* $\mathcal{M}$),
(λ*g ns*. *num-of-term* (*intrp-fn* $\mathcal{M}$ *g* (*map term-of-num ns*))),
(λ*p*. {*ns*. (*map term-of-num ns*) ∈ *intrp-rel* $\mathcal{M}$ *p*}))›

lemma *dom-lowmod* [*simp*]: ‹*dom* (*lowmod* $\mathcal{M}$) = *num-of-term* ' (*dom* $\mathcal{M}$)›
⟨*proof*⟩

lemma *intrp-fn-lowmod* [*simp*]: ‹*intrp-fn* (*lowmod* $\mathcal{M}$) *f ns* = *num-of-term* (*intrp-fn* $\mathcal{M}$ *f* (*map term-of-num ns*))›
⟨*proof*⟩

lemma *intrp-rel-lowmod* [*simp*]: ‹*intrp-rel* (*lowmod* $\mathcal{M}$) *p* = {*ns*. (*map term-of-num ns*) ∈ *intrp-rel* $\mathcal{M}$ *p*}›
⟨*proof*⟩

lemma *is-valuation-lowmod*: ‹*is-valuation* (*lowmod* $\mathcal{C}$) (*num-of-term* ∘ β) ⟷ *is-valuation* $\mathcal{C}$ β›
⟨*proof*⟩

lemma *lowmod-dom-empty*: ‹*dom* (*lowmod* $\mathcal{M}$) = {} ⟷ *dom* $\mathcal{M}$ = {}›
⟨*proof*⟩

lemma *lowmod-termval*:
assumes ‹*is-valuation* (*lowmod* $\mathcal{M}$) β›
shows ‹*eval t* (*lowmod* $\mathcal{M}$) β = *num-of-term* (*eval t* $\mathcal{M}$ (*term-of-num* ∘ β))›
⟨*proof*⟩

lemma *lowmod-holds*:
assumes ‹*is-valuation* (*lowmod* $\mathcal{M}$) β›
shows ‹(*lowmod* $\mathcal{M}$),β ⊨ φ ⟷ $\mathcal{M}$,(*term-of-num* ∘ β) ⊨ φ›
⟨*proof*⟩

lemma *lowmod-intrp*: ‹*is-interpretation* $\mathcal{L}$ (*lowmod* $\mathcal{M}$) = *is-interpretation* $\mathcal{L}$ $\mathcal{M}$›
⟨*proof*⟩

lemma *loewenheim-skolem*:
assumes ‹*is-interpretation* (*language* Φ) $\mathcal{M}$› ‹*dom* $\mathcal{M} \neq \{\}$›
assumes ‹$\bigwedge\varphi$. $\varphi \in \Phi \Longrightarrow$ *qfree* φ› ‹*satisfies* $\mathcal{M}$ Φ›
obtains $\mathcal{N}$ :: ‹*nat intrp*› **where** ‹*is-interpretation* (*language* Φ) $\mathcal{N}$› ‹*dom* $\mathcal{N} \neq \{\}$› ‹*satisfies* $\mathcal{N}$ Φ›
⟨*proof*⟩

theorem *uniformity*:
assumes ‹*qfree* φ›
‹$\bigwedge\mathcal{C}$::*nterm intrp*. $\bigwedge\beta$. ⟦*dom* $\mathcal{C} \neq \{\}$; *is-valuation* $\mathcal{C}$ β⟧ $\Longrightarrow$ $\mathcal{C}$,β $\models$ *foldr Exists xs* φ›
obtains σs **where** ‹$\bigwedge\sigma$ x. $\sigma \in$ *set* $\sigma s \Longrightarrow \sigma\ x \in$ *terms-set* (*fst* (*language* $\{\varphi\}$))›
‹$\bigwedge I$. $I \models_p$ (*foldr* ($\lambda\varphi\ \psi$. $\varphi \vee \psi$) (*map* ($\lambda\sigma$. $\varphi \cdot_{fm} \sigma$) σs) $\bot$)›
⟨*proof*⟩

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.