A deep embedding of HOL in HOL: soundness, completeness, consistency

Christoph Benzmüller Daniel Kirchner

August 7, 2026

Abstract

This entry presents a minimal, machine-checked deep embedding of classical higher-order logic (HOL) in Isabelle/HOL, following Benzmüller, Brown and Kohlhase (BKK) [4]. Throughout, HOL means Church's simple theory of types [5]: it is the embedded *object* logic, while Isabelle/HOL serves as the ambient meta-logic. The language of BKK §2 is formalised in a *locally nameless* representation, which makes BKK's convention of identifying α-equivalent terms literally true and eliminates capture-avoiding substitution and renaming machinery altogether. The semantics of BKK §3 is rendered abstractly: applicative structures, Σ-evaluations $(D, @, E)$ subject to BKK's four conditions on the evaluation function, Σ-models $(D, @, E, \upsilon)$, and the class $\mathcal{M}_{\beta\mathfrak{fb}}$ of Σ-Henkin models (BKK's property q is automatic here, since primitive equality is part of the signature). Standard models arise as the full special case, and the familiar recursive denotation over frames enters only as the canonical way of constructing concrete models.

On this basis we prove the natural-deduction calculus NK of BKK §7 sound (Theorem 7.3, including BKK's evaluation-variant argument) and complete in the sense of Henkin, via the abstract-consistency and model-existence method of BKK §6, with the term model realised as a term evaluation. Completeness is established in a form stronger than BKK's Corollary 7.7: it holds at every infinite value carrier, for every signature with infinitely many parameters, and for open formulas. Consistency follows by exhibiting a concrete Σ-standard model over finite domains. The signature includes BKK's optional primitive equality (Remark 7.9) and—going beyond BKK, following Andrews [1]—typed description operators. As an illustration, Cantor's theorem is derived *inside* NK, in surjective and injective form and at every type, with the Cantor sentences stated as in the Stanford Encyclopedia entry on Church's type theory [3].

What is new. To the best of our knowledge, this entry contains the first machine-checked *completeness* proof for a deep embedding of HOL in HOL. The most closely related AFP entry, *Metatheory of* $\mathcal{Q}_0$ by Díaz [6], formalises

Andrews' system $\mathcal{Q}_0$ ([2], Chapter 5) with named variables up to and including soundness and consistency; completeness is not covered there. Harrison's classic self-verification of HOL Light [8] likewise concerns soundness and consistency of HOL in HOL, not completeness. The nearest completeness mechanisations concern weaker logics: Koch and Kirst [9] prove completeness of second-order logic with respect to Henkin semantics in Coq, by translation to mono-sorted first-order logic — second-order logic is a proper fragment of Church's type theory, and the translation route differs from a direct model-existence proof for a higher-order calculus — and From [7] provides a synthetic-completeness framework in Isabelle/HOL, instantiated to propositional, first-order, modal and hybrid logic. Second, the entry demonstrates that BKK's abstract consistency / model-existence technique (BKK §6)—the basis of their uniform completeness proofs for the whole landscape of model classes—can indeed be formalised. Third, the formalisation *extends* BKK by typed description operators, following Andrews [1]: the rule $NK(\iota)$ and the corresponding model condition are carried through soundness, model existence and completeness. Finally, completeness is established in a form stronger than BKK's Corollary 7.7: for open formulas, at every infinite value carrier, and for every signature with infinitely many parameters, with no cardinality link between signature and carrier.

This contribution is part of the LogiKEy project (https://logikey.org).

Contents

1 Syntax: a deep embedding of HOL in HOL

BKK's language of classical higher-order logic — HOL, by which we mean Church's simple theory of types throughout (BKK Sections 2.1–2.2) — in a *locally nameless* representation: bound variables are de Bruijn indices, free variables and parameters carry their type. BKK take alphabetic variants to be identical (BKK Section 2.1); locally nameless makes that literally true — α-equivalent terms are *equal*, so capture-avoiding substitution and the entire renaming theory disappear. As in BKK, non-logical constants are *parameters* with names drawn from a type $'p$, and we include BKK's optional primitive equality *Eq* σ (BKK Section 2.1, Remark 7.9), alongside the always-expressible defined Leibniz equality (BKK Section 2.2). Beyond BKK's signature we deliberately carry a family of description operators *Iota* σ of type $(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma$, one for each type: BKK have no description operator (they decline to add one, BKK Section 2.3.1, pointing to Andrews 1972 for the semantic issues); we follow that reference instead and equip *Iota* σ with the description axiom for singletons — the rule $NK(\iota)$ of the calculus and the corresponding model condition *gm-descB*. Definitions and lemmas below that carry no BKK reference are locally-nameless infrastructure (opening, closing, freshness, renaming): they have no counterpart in the paper, where identification of alphabetic variants is handled informally (BKK Section 2.1).

1.1 Types and terms

datatype *ty* = *Ind* (‹ι›) | *Bool* (‹o›) | *Fun ty ty* (**infixr** ‹⇒› *65*)

datatype (*pars*: $'p$) *tm* =
  Bnd nat — bound variable (de Bruijn index)
| *Fre nat ty* — free variable: a name and its type
| *Par* $'p$ *ty* — parameter (typed constant)
| *Neg* | *Dis* | *Pi ty* | *Iota ty* | *Eq ty*
  — logical constants $\neg$, $\vee$, $\mathbf{\Pi}_\sigma$, $\boldsymbol{\iota}_\sigma$, $=_\sigma$
| *App* $'p$ *tm* $'p$ *tm* (**infixl** ‹·› *200*)
| *Abs ty* $'p$ *tm* — abstraction, domain type annotated (BKK's $\lambda X_\sigma.\ A$; nameless, so no binder variable)
for *map*: *prn* — We call the map function for the parameter type *prn* for parameter renaming.

BKK write disjunctions in infix and negations in prefix notation (BKK Section 2.2 declares the convention of writing $((\vee A)B)$ as $A \vee B$); we mirror this with input/output abbreviations for the applied connectives.

Abstraction is written $\mathbf{\Lambda}_\sigma\ b$ for BKK's $\lambda X_\sigma.\ A$ (bold Λ, since λ is reserved); being nameless, it displays the domain type but no binder variable.

notation *Abs* (‹**Λ**_ -› [*0*, *200*] *200*)

Decorated atoms: a free variable x of type σ is written ${x^f}_\sigma$ (BKK's X_σ),

a parameter w is written $w^p{}_\sigma$ (BKK's typed constants).

notation *Fre* ($\langle$-^{f}_$\rangle$ [*1000*, *0*] *1000*)
notation *Par* ($\langle$-^{p}_$\rangle$ [*1000*, *0*] *1000*)

abbreviation *NegA* :: *'p tm* ⇒ *'p tm* ($\langle\neg$ -$\rangle$ [*66*] *66*) **where**
$\neg\ A \equiv Neg \cdot A$
abbreviation *DisA* :: *'p tm* ⇒ *'p tm* ⇒ *'p tm* (**infixr** $\langle\vee\rangle$ *61*) **where**
$A \vee B \equiv (Dis \cdot A) \cdot B$

Summary of the term notation and its BKK Section 2 counterparts: application $s \cdot t$ (BKK juxtaposition), abstraction $\mathbf{\Lambda}_\sigma\ b$, free variables $x^f{}_\sigma$ (BKK's X_σ), parameters $w^p{}_\sigma$ (typed constants), and the applied connectives $\neg\ A$ and $A \vee B$. The defined layer adds $A \supset B$, $\mathbf{\Pi}_\sigma\ b$, Leibniz equality $A \doteq_\alpha B$ and applied primitive equality (below); opening is $b\langle u\rangle$ (below); on the semantic side, $(\!|A|\!)_\xi$ is the denotation and $\xi(x_\sigma := d)$ the assignment update (Section 2).

Types and terms (over a countable parameter type) are countable, so the language of sentences can be enumerated — the basis for our countable rendering of BKK's transfinite extension construction (BKK Section 6, Lemma 6.32).

instance *ty* :: *countable* ⟨*proof*⟩
instance *tm* :: (*countable*) *countable* ⟨*proof*⟩

1.2 Opening and free-variable substitution

$opn_k\ u\ t$ replaces the bound index k in t by u; $t(\!|u|\!)$ opens the top binder. Because bound variables are indices, substituting for a free variable (*fsub*) needs no renaming: it passes straight through *Abs*.

primrec *opn* :: *nat* ⇒ *'p tm* ⇒ *'p tm* ⇒ *'p tm* **where**
  opn k u (*Bnd i*) = (*if i = k then u else Bnd i*)
| *opn k u* ($n^f{}_\sigma$) = $n^f{}_\sigma$
| *opn k u* ($p^p{}_\sigma$) = $p^p{}_\sigma$
| *opn k u Neg* = *Neg*
| *opn k u Dis* = *Dis*
| *opn k u* (*Pi σ*) = *Pi σ*
| *opn k u* (*Iota τ*) = *Iota τ*
| *opn k u* (*Eq τ*) = *Eq τ*
| *opn k u* ($s \cdot t$) = (*opn k u s*) · (*opn k u t*)
| *opn k u* ($\mathbf{\Lambda}_\sigma\ b$) = $\mathbf{\Lambda}_\sigma$ (*opn* (*Suc k*) *u b*)

Opening the outermost binder is by far the most frequent operation, so it gets its own notation: $b\langle u\rangle$ is b with de Bruijn index *0* instantiated to u — BKK's $[u/X]B$ for the bound variable of the enclosing abstraction or quantifier.

abbreviation *opn0* ($\langle$-$\langle$-$\rangle\rangle$ [*1000*, *0*] *1000*) **where** $b\langle u\rangle \equiv opn\ 0\ u\ b$

primrec *fsub* :: *nat* ⇒ *ty* ⇒ *'p tm* ⇒ *'p tm* ⇒ *'p tm* **where**
  fsub x σ u (*Bnd i*) = *Bnd i*
| *fsub x σ u* ($n^f\,\tau$) = (*if n* = *x* ∧ *τ* = *σ then u else* $n^f\,\tau$)
| *fsub x σ u* ($p^p\,\tau$) = $p^p\,\tau$
| *fsub x σ u Neg* = *Neg*
| *fsub x σ u Dis* = *Dis*
| *fsub x σ u* (*Pi τ*) = *Pi τ*
| *fsub x σ u* (*Iota τ*) = *Iota τ*
| *fsub x σ u* (*Eq τ*) = *Eq τ*
| *fsub x σ u* (*s* · *t*) = (*fsub x σ u s*) · (*fsub x σ u t*)
| *fsub x σ u* (**Λ**$_\tau$ *b*) = **Λ**$_\tau$ (*fsub x σ u b*)

Substitution notation: $b[u/x_\sigma]$ substitutes u for the free variable $x^f{}_\sigma$ in b.

syntax *-fsub* :: *logic* ⇒ *logic* ⇒ *logic* ⇒ *logic* ⇒ *logic* (‹-[-′/-]› [*1000*, *0*, *0*, *0*] *1000*)
syntax-consts *-fsub* == *fsub*
translations $b[u/x_\sigma]$ ⇌ *CONST fsub x σ u b*

1.3 Free variables and parameters

primrec *fvs* :: *'p tm* ⇒ *nat set* **where**
  fvs (*Bnd i*) = {}
| *fvs* ($n^f\,\sigma$) = {*n*}
| *fvs* ($p^p\,\sigma$) = {}
| *fvs Neg* = {} | *fvs Dis* = {} | *fvs* (*Pi σ*) = {}
| *fvs* (*Iota σ*) = {} | *fvs* (*Eq σ*) = {}
| *fvs* (*s* · *t*) = *fvs s* ∪ *fvs t*
| *fvs* (**Λ**$_\sigma$ *b*) = *fvs b*

lemma *finite-fvs* [*simp*]: *finite* (*fvs t*) ⟨*proof*⟩
lemma *finite-pars* [*simp*]: *finite* (*pars t*) ⟨*proof*⟩

Renaming parameters. Eigen-parameters (BKK's w_α) must be chosen fresh; to weaken the context or extend a consistent set we move them out of the way with an injective renaming.

lemma *prn-opn*: *prn ϱ* (*opn k u t*) = *opn k* (*prn ϱ u*) (*prn ϱ t*) ⟨*proof*⟩
lemma *pars-prn*: *pars* (*prn ϱ t*) = *ϱ* ‘ *pars t* ⟨*proof*⟩
lemma *prn-prn*: *prn f* (*prn g t*) = *prn* (λ*p*. *f* (*g p*)) *t* ⟨*proof*⟩
lemma *prn-id* [*simp*]: *prn* (λ*p*. *p*) *t* = *t* ⟨*proof*⟩
lemma *prn-cong*: (⋀*p*. *p* ∈ *pars t* ⟹ *ϱ p* = *p*) ⟹ *prn ϱ t* = *t* ⟨*proof*⟩

The typed free occurrences — the (name, type) pairs a term reads from an assignment. A name can occur at several types, so this is finer than *fvs*.

primrec *occ* :: *'p tm* ⇒ (*nat* × *ty*) *set* **where**
  occ (*Bnd i*) = {}
| *occ* ($n^f\,\sigma$) = {(*n*, *σ*)}
| *occ* ($p^p\,\sigma$) = {}
| *occ Neg* = {} | *occ Dis* = {} | *occ* (*Pi σ*) = {}

| *occ* (*Iota* σ) = {} | *occ* (*Eq* σ) = {}
| *occ* ($s \cdot t$) = *occ* $s \cup$ *occ* t
| *occ* ($\mathbf{\Lambda}_\sigma$ b) = *occ* b

lemma *fvs-eq-fst-occ*: *fvs* t = *fst* ‘ *occ* t ⟨*proof*⟩
lemma *occ-opn*: *occ* (*opn* k u t) $\subseteq$ *occ* $t \cup$ *occ* u ⟨*proof*⟩

1.4 A fresh free variable exists

A deterministic fresh name for a finite set (used for the abstraction case of the denotation).

definition *fresh* :: *nat set* ⇒ *nat* **where** *fresh* $S \equiv$ *LEAST* n. $n \notin S$

lemma *fresh-notin*: *finite* $S \Longrightarrow$ *fresh* $S \notin S$
⟨*proof*⟩

1.5 Basic laws of opening and substitution

Free variables of a substitution. (A name may occur at several types, so x is only removed under the safe over-approximation.)

lemma *fvs-fsub*: *fvs* (*fsub* x σ u t) $\subseteq$ *fvs* $t \cup$ *fvs* u ⟨*proof*⟩
lemma *fvs-opn*: *fvs* (*opn* k u t) $\subseteq$ *fvs* $t \cup$ *fvs* u ⟨*proof*⟩

Opening with a free variable preserves size — the measure for the size-recursive denotation of an abstraction (its body is opened with a fresh variable of the same size).

lemma *size-opn-Fre* [*simp*]: *size* (*opn* k ($x^f{}_\sigma$) t) = *size* t ⟨*proof*⟩

1.6 Local closure

A term is *locally closed* if every bound index is captured by an enclosing binder. As is standard for the locally-nameless representation, the *Abs* rule uses a *cofinite* quantifier: opening the body with a fresh free variable is locally closed. A locally closed term denotes exactly an α-equivalence class of BKK's named terms (BKK Section 2.1).

inductive *lc* :: *'p tm* ⇒ *bool* **where**
  lc-Fre [*intro*]: *lc* ($n^f{}_\sigma$)
| *lc-Par* [*intro*]: *lc* ($p^p{}_\sigma$)
| *lc-Neg* [*intro*]: *lc Neg*
| *lc-Dis* [*intro*]: *lc Dis*
| *lc-Pi* [*intro*]: *lc* (*Pi* σ)
| *lc-Iota* [*intro*]: *lc* (*Iota* σ)
| *lc-Eq* [*intro*]: *lc* (*Eq* σ)
| *lc-App* [*intro*]: *lc* $s \Longrightarrow$ *lc* $t \Longrightarrow$ *lc* ($s \cdot t$)
| *lc-Abs*: *finite* $L \Longrightarrow$ ($\bigwedge x.\ x \notin L \Longrightarrow$ *lc* ($b\langle x^f{}_\sigma\rangle$)) $\Longrightarrow$ *lc* ($\mathbf{\Lambda}_\sigma$ b)

If opening at j is already fixed by a later opening at $i \neq j$, then opening at i alone was already the identity.

lemma *opn-core*: $i \neq j \Longrightarrow opn\ j\ v\ t = opn\ i\ u\ (opn\ j\ v\ t) \Longrightarrow opn\ i\ u\ t = t$
⟨*proof*⟩

A locally closed term ignores opening.

lemma *opn-lc* [*simp*]: $lc\ t \Longrightarrow opn\ k\ u\ t = t$
⟨*proof*⟩

fsub commutes with opening when the substituted term is locally closed.

lemma *fsub-opn*: $lc\ u \Longrightarrow fsub\ x\ \sigma\ u\ (opn\ k\ v\ t) = opn\ k\ (fsub\ x\ \sigma\ u\ v)\ (fsub\ x\ \sigma\ u\ t)$
⟨*proof*⟩

Opening with u equals opening with a fresh free variable and then substituting u for it.

lemma *fsub-intro*: $x \notin fvs\ t \Longrightarrow opn\ k\ u\ t = fsub\ x\ \sigma\ u\ (opn\ k\ ({x^f}_\sigma)\ t)$
⟨*proof*⟩

Openings at distinct indices commute (for locally closed fillers).

lemma *opn-opn-comm*: $i \neq j \Longrightarrow lc\ u \Longrightarrow lc\ v \Longrightarrow opn\ i\ u\ (opn\ j\ v\ t) = opn\ j\ v\ (opn\ i\ u\ t)$
⟨*proof*⟩

1.7 Typing: well-formed formulae

$wff_\sigma(t)$ is BKK's $t \in wff_\sigma(\Sigma)$ (BKK Section 2.1): t is a well-formed formula of type σ. The *Abs* rule uses the cofinite quantifier, so a well-formed formula is in particular locally closed.

inductive *wff* :: $ty \Rightarrow$ 'p $tm \Rightarrow$ *bool* (⟨*wff*_'(-')⟩ [0,0] 1000) **where**
  wff-Fre: $wff_\sigma({n^f}_\sigma)$
| *wff-Par*: $wff_\sigma({p^p}_\sigma)$
| *wff-Neg*: $wff_{o \Rightarrow o}(Neg)$
| *wff-Dis*: $wff_{o \Rightarrow o \Rightarrow o}(Dis)$
| *wff-Pi*: $wff_{(\sigma \Rightarrow o) \Rightarrow o}(Pi\ \sigma)$
| *wff-Iota*: $wff_{(\sigma \Rightarrow o) \Rightarrow \sigma}(Iota\ \sigma)$
| *wff-Eq*: $wff_{\sigma \Rightarrow \sigma \Rightarrow o}(Eq\ \sigma)$
| *wff-App*: $wff_{\sigma \Rightarrow \tau}(s) \Longrightarrow wff_\sigma(t) \Longrightarrow wff_\tau(s \cdot t)$
| *wff-Abs*: *finite* $L \Longrightarrow (\bigwedge x.\ x \notin L \Longrightarrow wff_\tau(b\langle{x^f}_\sigma\rangle)) \Longrightarrow wff_{\sigma \Rightarrow \tau}(\mathbf{\Lambda}_\sigma\ b)$

lemma *wff-lc*: $wff_\sigma(t) \Longrightarrow lc\ t$ ⟨*proof*⟩

inductive-cases *wff-FreE* [*elim*!]: $wff_\tau({n^f}_\sigma)$
inductive-cases *wff-ParE* [*elim*!]: $wff_\tau({p^p}_\sigma)$
inductive-cases *wff-NegE* [*elim*!]: $wff_\tau(Neg)$
inductive-cases *wff-DisE* [*elim*!]: $wff_\tau(Dis)$
inductive-cases *wff-PiE* [*elim*!]: $wff_\tau(Pi\ \sigma)$
inductive-cases *wff-IotaE* [*elim*!]: $wff_\tau(Iota\ \sigma)$
inductive-cases *wff-EqE* [*elim*!]: $wff_\tau(Eq\ \sigma)$
inductive-cases *wff-AppE* [*elim*]: $wff_\tau(s \cdot t)$

inductive-cases *wff-AbsE* [*elim*]: $wff_\tau(\mathbf{\Lambda}_\sigma\ b)$

Types are unique.

lemma *wff-unique*: $wff_\sigma(t) \Longrightarrow wff_\tau(t) \Longrightarrow \sigma = \tau$
⟨*proof*⟩

Well-typedness is preserved when a free variable is replaced by a term of the same type (BKK Section 2.1: substitution respects the typing; the paper leaves this implicit).

lemma *wff-fsub*: $wff_\tau(t) \Longrightarrow wff_\varrho(u) \Longrightarrow wff_\tau(fsub\ x\ \varrho\ u\ t)$
⟨*proof*⟩

Consequently an abstraction may be opened with *any* fresh free variable and stays well-typed — the locally-nameless counterpart of BKK's α-invariance (BKK Section 2.1).

lemma *wff-Abs-open*: **assumes** *w*: $wff_{\sigma \Rightarrow \tau}(\mathbf{\Lambda}_\sigma\ b)$ **and** *x*: $x \notin fvs\ b$
shows $wff_\tau(b\langle {x^f}_\sigma \rangle)$
⟨*proof*⟩

Opening a well-typed abstraction with a well-typed argument stays well-typed — the typing counterpart of β-reduction (implicit in BKK Section 2.1; the semantic analogue is BKK Lemma 3.20).

lemma *wff-opn*: $wff_{\sigma \Rightarrow \tau}(\mathbf{\Lambda}_\sigma\ b) \Longrightarrow wff_\sigma(a) \Longrightarrow wff_\tau(b\langle a \rangle)$
⟨*proof*⟩

Parameter renaming leaves the typing unchanged (there is no BKK counterpart: parameter renaming is part of the locally-nameless infrastructure).

lemma *wff-prn*[*intro*!]: $wff_\sigma(t) \Longrightarrow wff_\sigma(prn\ \varrho\ t)$
⟨*proof*⟩

1.8 Turning one parameter into a free variable

pvar $w\ \sigma\ x\ t$ replaces every occurrence of the parameter w at type σ by the free variable x. This realizes the evaluation variant of BKK's proof of Theorem 7.3 (case $NK(\Pi I)$): from an evaluation $\mathcal{E}$ "one can define another evaluation function $\mathcal{E}'$ such that $\mathcal{E}'(w) \equiv a$ and $\mathcal{E}'(A) \equiv \mathcal{E}(A)$ if w does not occur in A".

primrec *pvar* :: $'p \Rightarrow ty \Rightarrow nat \Rightarrow 'p\ tm \Rightarrow 'p\ tm$ **where**
  $pvar\ w\ \sigma\ x\ (Bnd\ i) = Bnd\ i$
| $pvar\ w\ \sigma\ x\ ({n^f}_\tau) = {n^f}_\tau$
| $pvar\ w\ \sigma\ x\ ({p^p}_\tau) = (if\ p = w \wedge \tau = \sigma\ then\ {x^f}_\sigma\ else\ {p^p}_\tau)$
| $pvar\ w\ \sigma\ x\ Neg = Neg$
| $pvar\ w\ \sigma\ x\ Dis = Dis$
| $pvar\ w\ \sigma\ x\ (Pi\ \tau) = Pi\ \tau$
| $pvar\ w\ \sigma\ x\ (Iota\ \tau) = Iota\ \tau$
| $pvar\ w\ \sigma\ x\ (Eq\ \tau) = Eq\ \tau$
| $pvar\ w\ \sigma\ x\ (s \cdot t) = (pvar\ w\ \sigma\ x\ s) \cdot (pvar\ w\ \sigma\ x\ t)$

| *pvar w σ x* (**Λ**$_\tau$ *b*) = **Λ**$_\tau$ (*pvar w σ x b*)

lemma *pvar-opn*: *pvar w σ x* (*opn k u t*) = *opn k* (*pvar w σ x u*) (*pvar w σ x t*)
⟨*proof*⟩
lemma *pvar-id* [*simp*]: *w* ∉ *pars t* ⟹ *pvar w σ x t* = *t*
⟨*proof*⟩
lemma *wff-pvar*: *wff*$_\tau$(*t*) ⟹ *wff*$_\tau$(*pvar w σ x t*)
⟨*proof*⟩

1.9 β-conversion (BKK Section 2)

BKK's β-equality $\equiv_\beta$ (BKK Section 2.1): the congruence closure of the β-redex $(\lambda x.\ b)\ a \to b[a/x]$. In the locally-nameless presentation the redex reduces to $b\langle a\rangle$, and the rule under an abstraction is stated cofinitely, as for typing and local closure.

The relation is *type-indexed*: $s \approx_\varrho t$ holds only for well-formed terms of type ϱ. Carrying the type keeps the symmetric/transitive rules type-preserving (a β-expansion does not otherwise determine the argument's type), which is exactly what the soundness proof of $NK(\beta)$ needs.

inductive *beq* :: *'p tm* ⇒ *ty* ⇒ *'p tm* ⇒ *bool* (‹- ≈_ -› [*51*, *0*, *51*] *50*) **where**
  beta: *wff*$_{\sigma\Rightarrow\varrho}$(**Λ**$_\sigma$ *b*) ⟹ *wff*$_\sigma$(*a*) ⟹ (**Λ**$_\sigma$ *b*) · *a* ≈$_\varrho$ *b*⟨*a*⟩
| *refl*: *wff*$_\varrho$(*t*) ⟹ *t* ≈$_\varrho$ *t*
| *sym*: *s* ≈$_\varrho$ *t* ⟹ *t* ≈$_\varrho$ *s*
| *trans*: *r* ≈$_\varrho$ *s* ⟹ *s* ≈$_\varrho$ *t* ⟹ *r* ≈$_\varrho$ *t*
| *appL*: *s* ≈$_{\sigma\Rightarrow\varrho}$ *s'* ⟹ *wff*$_\sigma$(*t*) ⟹ *s* · *t* ≈$_\varrho$ *s'* · *t*
| *appR*: *t* ≈$_\sigma$ *t'* ⟹ *wff*$_{\sigma\Rightarrow\varrho}$(*s*) ⟹ *s* · *t* ≈$_\varrho$ *s* · *t'*
| *abs*: *finite L* ⟹ (⋀*x*. *x* ∉ *L* ⟹ *b*⟨*x*$^f{}_\sigma$⟩ ≈$_\tau$ *b'*⟨*x*$^f{}_\sigma$⟩)
    ⟹ **Λ**$_\sigma$ *b* ≈$_{\sigma\Rightarrow\tau}$ **Λ**$_\sigma$ *b'*

β-conversion relates well-formed terms of the stated type.

lemma *beq-wff*: *s* ≈$_\varrho$ *t* ⟹ *wff*$_\varrho$(*s*) ∧ *wff*$_\varrho$(*t*)
⟨*proof*⟩

lemma *beq-wffL*: *s* ≈$_\varrho$ *t* ⟹ *wff*$_\varrho$(*s*) **and** *beq-wffR*: *s* ≈$_\varrho$ *t* ⟹ *wff*$_\varrho$(*t*)
⟨*proof*⟩

β-conversion is stable under parameter renaming.

lemma *beq-rename*[*intro*!]: *s* ≈$_\tau$ *t* ⟹ *prn π s* ≈$_\tau$ *prn π t*
⟨*proof*⟩

1.10 The defined logical layer (BKK Section 2.2)

Following BKK (BKK Section 2.2), everything beyond the primitive constants of the signature — $\neg$, $\vee$, Pi_σ, the primitive equality $Eq\ \sigma$ and the description operators $Iota\ \sigma$ (above) — is defined. The universal quantifier is BKK's shorthand $\forall X_\sigma.\ A \equiv \mathbf{\Pi}_\sigma(\lambda X_\sigma.\ A)$ and implication their $A \supset B \equiv (\neg A) \vee B$. For falsity we deviate mildly from BKK, who use $F_\mathrm{o} \equiv \neg\forall P_\mathrm{o}$.

$P \vee \neg P$ (BKK Lemma 3.43, footnote 11): our $\bot$ is $\forall\, X_{\mathrm{o}}.\ X$ and $\top \equiv \neg\bot$. The two choices are interderivable in NK_β (cf. BKK Remark 7.2) and both falsa are unsatisfied in every Σ-model, so every use below is invariant under the exchange. Leibniz equality — always expressible, alongside the primitive *Eq* α of the signature — is BKK's Leibniz combinator $Q_\alpha \equiv \lambda X_\alpha\ Y_\alpha.\ \forall\, P_{\alpha\Rightarrow\mathrm{o}}.\ P\ X \supset P\ Y$ (BKK Section 2.2, the Leibniz formula for equality). Because bound variables are de Bruijn indices, *Leib* α is a genuinely *closed* term: BKK's reserved bound names X, Y, P are simply the indices *2*, *1*, *0*.

definition *Forall* :: *ty* ⇒ *'p tm* ⇒ *'p tm* (‹**Π**_ -› [*0*, *200*] *200*) **where**
  $\mathbf{\Pi}_\sigma\ b = (Pi\ \sigma) \cdot (\mathbf{\Lambda}_\sigma\ b)$
definition *FalseB* :: *'p tm* (‹⊥›) **where** $\bot = \mathbf{\Pi}_{\mathrm{o}}\ (Bnd\ 0)$
definition *TrueB* :: *'p tm* (‹⊤›) **where** $\top = \neg\ \bot$
definition *ImpB* :: *'p tm* ⇒ *'p tm* ⇒ *'p tm* (**infixr** ‹⊃› *60*) **where**
  $\varphi \supset \psi = (\neg\ \varphi) \vee \psi$
definition *Leib* :: *ty* ⇒ *'p tm* **where**
  $Leib\ \alpha = \mathbf{\Lambda}_\alpha\ (Abs\ \alpha\ (\mathbf{\Pi}_{\alpha\Rightarrow\mathrm{o}}\ ((Bnd\ 0) \cdot (Bnd\ 2) \supset (Bnd\ 0) \cdot (Bnd\ 1))))$
abbreviation *LeibA* :: *'p tm* ⇒ *ty* ⇒ *'p tm* ⇒ *'p tm* (‹- ≐_ -› [*66*,*0*,*66*] *65*) **where**
  $A \doteq_\alpha B \equiv ((Leib\ \alpha) \cdot A) \cdot B$

Primitive equality applied (BKK's optional $=_\alpha \in \Sigma_{\alpha\to\alpha\to\mathrm{o}}$, BKK Remark 7.9).

abbreviation *PEqA* :: *'p tm* ⇒ *ty* ⇒ *'p tm* ⇒ *'p tm* (‹- =_ -› [*66*,*0*,*66*] *65*) **where**
  $A =_\alpha B \equiv ((Eq\ \alpha) \cdot A) \cdot B$
lemma *wff-PEq* [*intro*]: $wff_\alpha(A) \Longrightarrow wff_\alpha(B) \Longrightarrow wff_{\mathrm{o}}(A =_\alpha B)$
  ⟨*proof*⟩

1.11 Opening distributes over the defined layer

lemma *opn-Forall* [*simp*]: $opn\ k\ u\ (\mathbf{\Pi}_\sigma\ b) = \mathbf{\Pi}_\sigma\ (opn\ (Suc\ k)\ u\ b)$
  ⟨*proof*⟩
lemma *opn-ImpB* [*simp*]: $opn\ k\ u\ (\varphi \supset \psi) = (opn\ k\ u\ \varphi) \supset (opn\ k\ u\ \psi)$
  ⟨*proof*⟩
lemma *opn-FalseB* [*simp*]: $opn\ k\ u\ (\bot :: 'p\ tm) = \bot$
  ⟨*proof*⟩
lemma *opn-Leib* [*simp*]: $opn\ k\ u\ (Leib\ \alpha :: 'p\ tm) = Leib\ \alpha$
  ⟨*proof*⟩

1.12 A convenient abstraction-typing rule

For the closed defined terms, opening the body with *any* free variable is well-typed, so the cofinite side-condition collapses to a universal one.

lemma *wff-AbsI*: $(\bigwedge x.\ wff_\tau(b\langle x^f{}_\sigma\rangle)) \Longrightarrow wff_{\sigma\Rightarrow\tau}(\mathbf{\Lambda}_\sigma\ b)$
  ⟨*proof*⟩

1.13 Typing of the defined layer

lemma *wff-Forall* [*intro*]: $(\bigwedge x.\ \mathit{wff}_{\mathrm{o}}(b\langle x^{f}{}_{\sigma}\rangle)) \Longrightarrow \mathit{wff}_{\mathrm{o}}(\mathbf{\Pi}_{\sigma}\ b)$
⟨*proof*⟩
lemma *wff-Not* [*intro*]: $\mathit{wff}_{\mathrm{o}}(\varphi) \Longrightarrow \mathit{wff}_{\mathrm{o}}(\neg\ \varphi)$
⟨*proof*⟩
lemma *wff-Or* [*intro*]: $\mathit{wff}_{\mathrm{o}}(\varphi) \Longrightarrow \mathit{wff}_{\mathrm{o}}(\psi) \Longrightarrow \mathit{wff}_{\mathrm{o}}(\varphi \vee \psi)$
⟨*proof*⟩
lemma *wff-ImpB* [*intro*]: $\mathit{wff}_{\mathrm{o}}(\varphi) \Longrightarrow \mathit{wff}_{\mathrm{o}}(\psi) \Longrightarrow \mathit{wff}_{\mathrm{o}}(\varphi \supset \psi)$
⟨*proof*⟩
lemma *wff-FalseB* [*intro*, *simp*]: $\mathit{wff}_{\mathrm{o}}(\bot)$ ⟨*proof*⟩
lemma *wff-TrueB* [*intro*, *simp*]: $\mathit{wff}_{\mathrm{o}}(\top)$ ⟨*proof*⟩
lemma *wff-App-FreFre* [*intro*]: $\mathit{wff}_{\mathrm{o}}((p^{f}{}_{\sigma} \Rightarrow \mathrm{o}) \cdot (x^{f}{}_{\sigma}))$
⟨*proof*⟩
lemma *wff-Leib* [*intro*, *simp*]: $\mathit{wff}_{\alpha\Rightarrow\alpha\Rightarrow\mathrm{o}}(\mathit{Leib}\ \alpha)$
⟨*proof*⟩
lemma *wff-LeibE* [*intro*]: $\mathit{wff}_{\alpha}(A) \Longrightarrow \mathit{wff}_{\alpha}(B) \Longrightarrow \mathit{wff}_{\mathrm{o}}(A \doteq_{\alpha} B)$
⟨*proof*⟩

1.14 Free variables and local closure of the defined layer

lemma *fvs-defs* [*simp*]: *fvs* $(\mathbf{\Pi}_{\sigma}\ b) = \mathit{fvs}\ b$ *fvs* $(\bot :: 'p\ tm) = \{\}$ *fvs* $(\top :: 'p\ tm) = \{\}$
fvs $(\varphi \supset \psi) = \mathit{fvs}\ \varphi \cup \mathit{fvs}\ \psi$ *fvs* $(\mathit{Leib}\ \alpha :: 'p\ tm) = \{\}$
⟨*proof*⟩

lemma *pars-defs* [*simp*]: *pars* $(\mathbf{\Pi}_{\sigma}\ b) = \mathit{pars}\ b$ *pars* $(\bot :: 'p\ tm) = \{\}$
pars $(\top :: 'p\ tm) = \{\}$ *pars* $(\varphi \supset \psi) = \mathit{pars}\ \varphi \cup \mathit{pars}\ \psi$ *pars* $(\mathit{Leib}\ \alpha :: 'p\ tm) = \{\}$
⟨*proof*⟩

1.15 Parameter renaming distributes over the defined layer

lemma *prn-Forall* [*simp*]: *prn* ϱ $(\mathbf{\Pi}_{\sigma}\ b) = \mathbf{\Pi}_{\sigma}$ $(\mathit{prn}\ \varrho\ b)$
⟨*proof*⟩
lemma *prn-ImpB* [*simp*]: *prn* ϱ $(\varphi \supset \psi) = (\mathit{prn}\ \varrho\ \varphi) \supset (\mathit{prn}\ \varrho\ \psi)$
⟨*proof*⟩
lemma *prn-FalseB* [*simp*]: *prn* ϱ $(\bot :: 'p\ tm) = \bot$
⟨*proof*⟩
lemma *prn-Leib* [*simp*]: *prn* ϱ $(\mathit{Leib}\ \alpha :: 'p\ tm) = \mathit{Leib}\ \alpha$
⟨*proof*⟩

1.16 The defined existential quantifier

abbreviation *ExistsB* :: *ty* ⇒ *'p tm* ⇒ *'p tm* (‹∃ _ -› [*0*, *200*] *200*) **where**
$\exists_{\sigma}\ b \equiv \neg\ (\mathbf{\Pi}_{\sigma}\ (\neg\ b))$

1.17 Named binders for the defined quantifiers

Named-binder input syntax for the defined quantifiers: *clos k x σ t* abstracts the free variable $x^f{}_\sigma$ to the de Bruijn index k (the converse of *opn*), so that $\exists\, x_\sigma.\ \varphi$ and $\mathbf{\Pi} x_\sigma.\ \varphi$ bind an ordinary named variable; the locally-nameless representation is recovered by computation (the *-eq* lemmas below).

primrec *clos* :: *nat* ⇒ *nat* ⇒ *ty* ⇒ *'p tm* ⇒ *'p tm* **where**
  clos k x σ (Bnd i) = Bnd i
| *clos k x σ* $(n^f{}_\tau)$ = (*if n = x ∧ τ = σ then Bnd k else* $n^f{}_\tau$)
| *clos k x σ* $(p^p{}_\tau)$ = $p^p{}_\tau$
| *clos k x σ Neg = Neg*
| *clos k x σ Dis = Dis*
| *clos k x σ (Pi τ) = Pi τ*
| *clos k x σ (Iota τ) = Iota τ*
| *clos k x σ (Eq τ) = Eq τ*
| *clos k x σ (s · t) = (clos k x σ s) · (clos k x σ t)*
| *clos k x σ* $(\mathbf{\Lambda}_\tau\ b) = \mathbf{\Lambda}_\tau$ *(clos (Suc k) x σ b)*

lemma *clos-Forall* [*simp*]: *clos k x σ* $(\mathbf{\Pi}_\tau\ b) = \mathbf{\Pi}_\tau$ *(clos (Suc k) x σ b)*
  ⟨*proof*⟩
lemma *clos-ImpB* [*simp*]: *clos k x σ (φ ⊃ ψ) = (clos k x σ φ) ⊃ (clos k x σ ψ)*
  ⟨*proof*⟩
lemma *clos-Leib* [*simp*]: *clos k x σ (Leib α :: 'p tm) = Leib α*
  ⟨*proof*⟩
definition *ExN* :: *nat* ⇒ *ty* ⇒ *'p tm* ⇒ *'p tm* (‹∃ -. -› [*1000*, *0*, *61*] *61*) **where**
  $\exists\, x_\sigma.\ b = \exists_\sigma$ *(clos 0 x σ b)*
definition *AllN* :: *nat* ⇒ *ty* ⇒ *'p tm* ⇒ *'p tm* (‹**Π**-. -› [*1000*, *0*, *61*] *61*) **where**
  $\mathbf{\Pi} x_\sigma.\ b = \mathbf{\Pi}_\sigma$ *(clos 0 x σ b)*
definition *LamN* :: *nat* ⇒ *ty* ⇒ *'p tm* ⇒ *'p tm* (‹**Λ**-. -› [*1000*, *0*, *61*] *61*) **where**
  $\mathbf{\Lambda} x_\sigma.\ b = \mathbf{\Lambda}_\sigma$ *(clos 0 x σ b)*

Fixed variable names for readable named-binder statements: the script letters $\mathcal{G}$, $\mathcal{F}$, $\mathcal{X}$, $\mathcal{I}$, $\mathcal{H}$ name the (numeric) free variables *0*, …, *4*.

definition $\mathcal{G}$ :: *nat* **where** $\mathcal{G}$ = *0*
definition $\mathcal{F}$ :: *nat* **where** $\mathcal{F}$ = *1*
definition $\mathcal{X}$ :: *nat* **where** $\mathcal{X}$ = *2*
definition $\mathcal{I}$ :: *nat* **where** $\mathcal{I}$ = *3*
definition $\mathcal{H}$ :: *nat* **where** $\mathcal{H}$ = *4*

Stronger size-based induction.

lemma *size-induct*[*case-names Bnd Fre Par Neg Dis Pi Iota Eq App Abs*]:
  assumes ‹$\bigwedge$*x. P (Bnd x)*›
    and ‹$\bigwedge$*x α. P* $x^f{}_\alpha$›
    and ‹$\bigwedge$*x α. P* $x^p{}_\alpha$›
    and ‹*P Neg*›
    and ‹*P Dis*›
    and ‹$\bigwedge$*α. P (Pi α)*›
    and ‹$\bigwedge$*α. P (Iota α)*›
    and ‹$\bigwedge$*α. P (Eq α)*›

and ‹$\bigwedge A\ B.\ (\bigwedge C.\ \mathit{size}\ C \leq \mathit{size}\ A \Longrightarrow P\ C) \Longrightarrow (\bigwedge C.\ \mathit{size}\ C \leq \mathit{size}\ B \Longrightarrow P\ C) \Longrightarrow P\ (A \cdot B)$›
and ‹$\bigwedge \alpha\ A.\ (\bigwedge C.\ \mathit{size}\ C \leq \mathit{size}\ A \Longrightarrow P\ C) \Longrightarrow P\ (\mathbf{\Lambda}_\alpha\ A)$›
shows ‹$P\ x$›
⟨*proof*⟩

2 Semantics: applicative structures, Henkin models and standard models

The semantics in BKK's own layered terminology, in their order: *applicative structures* $(D, @)$ (BKK Definition 3.1), Σ-*evaluations* $J = (D, @, E)$ (BKK Definition 3.18) with an *abstract* evaluation function subject to BKK's four conditions, Σ-*valuations* and Σ-*models* $M = (D, @, E, \upsilon)$ (BKK Definitions 3.40 and 3.41), and the model class $\mathcal{M}_{\beta f \flat}$ (BKK Definition 3.49, the Σ-Henkin models of Definition 3.50). The signature Σ is a static parameter of the whole development: the logical constants fixed by the term datatype plus the typed parameters drawn from $'p$, exactly as BKK fix Σ at the start of their Section 3.

After the abstract notions, the recursive denotation $(\!|t|\!)_\xi$ is introduced as the *canonical construction* of an evaluation function over frame-like structures (BKK's Σ-evaluations over frames); it is the vehicle for building concrete models (the standard models here, and the term model of the completeness proof in Section 3).

2.1 Assignment update

BKK's assignment update $\varphi,[a/X]$ (BKK Definition 3.17), written in function-update style with the variable's type subscripted: $\xi(x_\sigma := d)$.

definition *upd* :: $(\mathit{nat} \Rightarrow \mathit{ty} \Rightarrow 'u) \Rightarrow \mathit{nat} \Rightarrow \mathit{ty} \Rightarrow 'u \Rightarrow (\mathit{nat} \Rightarrow \mathit{ty} \Rightarrow 'u)$
($‹$-$'($-$:=$ -$')›$ [*1000*, *0*, *0*, *0*] *1000*) **where**
$\xi(x_\sigma := d) \equiv \lambda n\ \tau.\ \mathit{if}\ n = x \wedge \tau = \sigma\ \mathit{then}\ d\ \mathit{else}\ \xi\ n\ \tau$

lemma *upd-same* [*simp*]: $\xi(x_\sigma := d)\ x\ \sigma = d$ ⟨*proof*⟩
lemma *upd-comm*: $(x, \sigma) \neq (w, \tau) \Longrightarrow (\xi(x_\sigma := d))(w_\tau := e) = (\xi(w_\tau := e))(x_\sigma := d)$
⟨*proof*⟩

2.2 Applicative structures (BKK Definition 3.1)

A typed collection of non-empty domains with an application operator. BKK's typing discipline $@ : D_{\alpha\to\beta} \times D_\alpha \to D_\beta$ becomes a closure condition on the one abstract operator. BKK's *frames* (Definition 3.4, $D_{\alpha\to\beta} \subseteq F(D_\alpha;\ D_\beta)$) are a set-theoretic notion with no direct analogue over an abstract carrier; by BKK Remark 3.6 every frame is functional, and it is

functionality (BKK Definition 3.5; named property f in Definition 3.46) that all mathematical arguments consume, so the model class below is delineated by functionality.

locale *app-struct* = **fixes** $Dm :: ty \Rightarrow 'u \Rightarrow bool$ **and** $Ap :: 'u \Rightarrow 'u \Rightarrow 'u$
  assumes *as-nonempty*: $\exists\, a.\ Dm\ \alpha\ a$ **and** *as-appTy*: $Dm\ (\alpha \Rightarrow \beta)\ f \Longrightarrow Dm\ \alpha\ a \Longrightarrow Dm\ \beta\ (Ap\ f\ a)$
begin

Variable assignments into the structure (BKK Definition 3.17); the update $\xi(x_\sigma := d)$ is BKK's $\varphi,[d/X]$.

definition *asg* :: $(nat \Rightarrow ty \Rightarrow 'u) \Rightarrow bool$ **where** $asg\ \xi \equiv \forall\, n\ \tau.\ Dm\ \tau\ (\xi\ n\ \tau)$

lemma *asg-upd*: $asg\ \xi \Longrightarrow Dm\ \sigma\ d \Longrightarrow asg\ (\xi(x_\sigma := d))$
  ⟨*proof*⟩

Functionality: property f of BKK Definition 3.5.

definition *functional* :: *bool* **where**
  $functional \equiv \forall\, \alpha\ \beta\ f\ g.\ Dm\ (\alpha \Rightarrow \beta)\ f \longrightarrow Dm\ (\alpha \Rightarrow \beta)\ g \longrightarrow$
    $(\forall\, a.\ Dm\ \alpha\ a \longrightarrow Ap\ f\ a = Ap\ g\ a) \longrightarrow f = g$

end

2.3 Σ-evaluations (BKK Definition 3.18)

An evaluation function E maps assignments to typed functions from well-formed formulae into the domains, subject to BKK's four conditions: (1) it extends the assignment on variables, (2) it is homomorphic for application, (3) it depends only on the assignment's values at the free variables (coincidence), and (4) it respects β-conversion (BKK state this via β-normal forms; over our typed β-equality $\approx_\tau$ of Section 1 the two formulations coincide, cf. BKK Remark 3.19). In addition E is typed: well-formed formulae of type τ denote in D_τ.

locale *sigma-eval* = *app-struct Dm Ap* **for** $Dm :: ty \Rightarrow 'u \Rightarrow bool$ **and** $Ap :: 'u \Rightarrow 'u \Rightarrow 'u$ +
  fixes $Ee :: (nat \Rightarrow ty \Rightarrow 'u) \Rightarrow 'p\ tm \Rightarrow 'u$
  assumes *ev-type*: $wff_\tau(A) \Longrightarrow asg\ \xi \Longrightarrow Dm\ \tau\ (Ee\ \xi\ A)$
    and *ev-var*: $asg\ \xi \Longrightarrow Ee\ \xi\ ({n^f}_\sigma) = \xi\ n\ \sigma$
    and *ev-app*: $wff_{\sigma \Rightarrow \tau}(F) \Longrightarrow wff_\sigma(A) \Longrightarrow asg\ \xi \Longrightarrow Ee\ \xi\ (F \cdot A) = Ap\ (Ee\ \xi\ F)\ (Ee\ \xi\ A)$
    and *ev-coin*: $wff_\tau(A) \Longrightarrow asg\ \xi \Longrightarrow asg\ \xi' \Longrightarrow (\bigwedge n\ \sigma.\ (n, \sigma) \in occ\ A \Longrightarrow \xi\ n\ \sigma = \xi'\ n\ \sigma)$
      $\Longrightarrow Ee\ \xi\ A = Ee\ \xi'\ A$
    and *ev-beta*: $A \approx_\tau B \Longrightarrow asg\ \xi \Longrightarrow Ee\ \xi\ A = Ee\ \xi\ B$
begin

The derived β-application law: the denotation of an abstraction is determined applicatively by the openings of its body (from conditions (1), (2), (4) and coincidence; the vehicle for all abstraction reasoning below).

lemma *ev-abs-app*:
assumes *wb*: $wff_{\sigma \Rightarrow \tau}(\mathbf{\Lambda}_\sigma\ b)$ **and** *xi*: *asg* ξ **and** *x*: $x \notin fvs\ b$ **and** *d*: *Dm* σ *d*
shows *Ap* (*Ee* ξ ($\mathbf{\Lambda}_\sigma$ *b*)) *d* = *Ee* ($\xi(x_\sigma := d)$) ($b\langle x^f{}_\sigma \rangle$)
⟨*proof*⟩

end

2.4 Σ-valuations and Σ-models (BKK Definitions 3.40 and 3.41)

A Σ-valuation is a (total) function $\upsilon : D_o \to \{T, F\}$ — rendered as a HOL predicate — satisfying the properties $L_\neg(E(\neg))$, $L_\vee(E(\vee))$ and $L^\alpha{}_\forall(E(\Pi_\alpha))$ of BKK's Figure 2. A Σ-evaluation together with such a valuation is a Σ-model. Following BKK Definition 3.41 (and Remark 3.42) we include primitive equality $L^\alpha{}_=(E(=_\alpha))$, and — extending BKK Definition 3.41, whose Σ-models have no description operator — a description property for $E(\iota_\alpha)$, matching $NK(\iota)$. Since the logical constants are closed, their denotations are assignment- independent (coincidence); we fix a canonical assignment to name them.

locale *sigma-model* = *sigma-eval Dm Ap Ee* **for**
Dm :: *ty* ⇒ *'u* ⇒ *bool* **and** *Ap* :: *'u* ⇒ *'u* ⇒ *'u* **and** *Ee* :: (*nat* ⇒ *ty* ⇒ *'u*) ⇒ *'p tm* ⇒ *'u* +
fixes *vl* :: *'u* ⇒ *bool*
assumes *vl-neg*: *asg* ξ ⟹ *Dm* o *a* ⟹ *vl* (*Ap* (*Ee* ξ *Neg*) *a*) ⟷ ¬ *vl a*
and *vl-dis*: *asg* ξ ⟹ *Dm* o *a* ⟹ *Dm* o *b* ⟹ *vl* (*Ap* (*Ap* (*Ee* ξ *Dis*) *a*) *b*) ⟷ *vl a* ∨ *vl b*
and *vl-pi*: *asg* ξ ⟹ *Dm* (σ ⇒ o) *f* ⟹
vl (*Ap* (*Ee* ξ (*Pi* σ)) *f*) ⟷ (∀ *d*. *Dm* σ *d* ⟶ *vl* (*Ap f d*))
and *vl-eq*: *asg* ξ ⟹ *Dm* σ *a* ⟹ *Dm* σ *b* ⟹ *vl* (*Ap* (*Ap* (*Ee* ξ (*Eq* σ)) *a*) *b*) ⟷ *a* = *b*
and *vl-iota*: *asg* ξ ⟹ *Dm* (σ ⇒ o) *f* ⟹ *Dm* σ *a* ⟹ (⋀*b*. *Dm* σ *b* ⟹ *vl* (*Ap f b*) ⟷ *b* = *a*)
⟹ *Ap* (*Ee* ξ (*Iota* σ)) *f* = *a*
begin

Satisfaction and validity (BKK Definition 3.41): $M \models_\varphi A$ iff $\upsilon(E_\varphi(A)) \equiv T$.

definition *satisfies* :: (*nat* ⇒ *ty* ⇒ *'u*) ⇒ *'p tm* ⇒ *bool* (‹⊨_ -› [0, 40] 40) **where**
$\models_\xi A \equiv vl\ (Ee\ \xi\ A)$
definition *valid* :: *'p tm* ⇒ *bool* (‹⊨ -› [40] 40) **where**
$\models A \equiv \forall \xi.\ asg\ \xi \longrightarrow \models_\xi A$

end

2.5 The model class $\mathcal{M}_{\beta f b}$ (BKK Definition 3.49)

BKK's completeness class for *NK* is $\mathcal{M}_{\beta f b}$: Σ-models satisfying properties q, f and b (with primitive equality, property q holds automatically, BKK

Definition 3.49 — the q-witness at type α is the denotation $E(=_\alpha)$, cf. the satisfaction lemma for Leibniz equality; with property b the valuation is two-valued on D_o). This class coincides with the Σ-Henkin models of BKK Definition 3.50 up to isomorphism (BKK Lemma 3.67 and Theorem 3.68).

locale *bkk-model* = *sigma-model Dm Ap Ee vl*
  for *Dm* :: *ty* ⇒ *'u* ⇒ *bool* **and** *Ap* :: *'u* ⇒ *'u* ⇒ *'u*
  and *Ee* :: (*nat* ⇒ *ty* ⇒ *'u*) ⇒ *'p tm* ⇒ *'u* **and** *vl* :: *'u* ⇒ *bool* +
  assumes *prop-f*: *functional* **and** *prop-b*: *Dm* ∘ *a* ⟹ *Dm* ∘ *b* ⟹ *vl a* ⟷ *vl b* ⟹ *a* = *b*

2.6 Truth values in a Σ-model (BKK Lemma 3.43)

context *sigma-model*
begin

The canonical assignment, from non-emptiness of the domains.

definition *xi0* :: *nat* ⇒ *ty* ⇒ *'u* **where** *xi0* ≡ λ*n* τ. *SOME a*. *Dm* τ *a*
lemma *asg-xi0*: *asg xi0* ⟨*proof*⟩

Closed terms evaluate independently of the assignment; the canonical assignment serves as the reference.

lemma *Ee-closed*: *wff*$_\tau$(*A*) ⟹ *occ A* = {} ⟹ *asg* ξ ⟹ *Ee* ξ *A* = *Ee xi0 A*
  ⟨*proof*⟩

Evaluating $\bot = \mathbf{\Pi}_o(Bnd\ 0)$: its truth means every boolean object is true.

lemma *vl-FalseB*: **assumes** *xi*: *asg* ξ **shows** *vl* (*Ee* ξ ⊥) ⟷ (∀ *d*. *Dm* ∘ *d* ⟶ *vl d*)
⟨*proof*⟩

BKK Lemma 3.43: $v(E(\top)) \equiv T$ and $v(E(\bot)) \equiv F$; in particular D_o contains a true and a false object (BKK Remark 3.44).

lemma *vl-TF*: **assumes** *xi*: *asg* ξ **shows** *vl* (*Ee* ξ ⊤) ∧ ¬ *vl* (*Ee* ξ ⊥)
  ⟨*proof*⟩

end

2.7 Model-relative truth and validity at a carrier

Truth of A in a model $\langle D,@,E,v\rangle$ under an assignment, and validity over *all* models of the class $\mathcal{M}_{\beta fb}$ at a given value carrier $'u$ — the carrier appears explicitly in the notation $\models('u)\ A$.

definition *rel-truth* ::
  ((*nat* ⇒ *ty* ⇒ *'u*) ⇒ *'p tm* ⇒ *'u*) ⇒ (*'u* ⇒ *bool*) ⇒ (*nat* ⇒ *ty* ⇒ *'u*) ⇒ *'p tm* ⇒ *bool*
  (‹⟨-,-⟩,- ⊨ -› [*0*, *0*, *0*, *61*] *60*) **where**
  ⟨*Ee*,*vl*⟩,ξ ⊨ *A* ≡ *vl* (*Ee* ξ *A*)

definition *bkk-valid* :: *'u itself* ⇒ *'p tm* ⇒ *bool* **where**
  bkk-valid u A ≡ ∀ *Dm Ap Ee* (*vl* :: *'u* ⇒ *bool*) *ξ*.
    bkk-model Dm Ap Ee vl ⟶ *app-struct.asg Dm ξ* ⟶ ⟨*Ee*,*vl*⟩,*ξ* ⊨ *A*

syntax *-bkk-valid* :: *type* ⇒ *logic* ⇒ *logic* (‹⊨'(-') -› [*1000*, *61*] *60*)
syntax-consts *-bkk-valid* == *bkk-valid*
translations *-bkk-valid t A* == *CONST bkk-valid* (*-TYPE t*) *A*

definition *bkk-consequence* :: *'u itself* ⇒ *'p tm set* ⇒ *'p tm* ⇒ *bool* **where**
  bkk-consequence u Γ *A* ≡ ∀ *Dm Ap Ee* (*vl* :: *'u* ⇒ *bool*) *ξ*.
    bkk-model Dm Ap Ee vl ⟶ *app-struct.asg Dm ξ* ⟶
    (∀ *B* ∈ Γ . *wff* o *B* ∧ ⟨*Ee*,*vl*⟩,*ξ* ⊨ *B*) ⟶ ⟨*Ee*,*vl*⟩,*ξ* ⊨ *A*

syntax *-bkk-consequence* :: *logic* ⇒ *type* ⇒ *logic* ⇒ *logic* (‹- ⊨'(-') -› [*61*, *1000*, *61*] *60*)
syntax-consts *-bkk-consequence* == *bkk-consequence*
translations *-bkk-consequence* Γ *t A* == *CONST bkk-consequence* (*-TYPE t*) Γ *A*

2.8 Σ-evaluations over frames: the canonical construction

A *frame signature* fixes the applicative structure (BKK Definition 3.1) and the denotation objects of the logical constants and parameters. It carries no conditions; the denotation and its purely semantic properties (coincidence, BKK Definition 3.18(3)) live here.

locale *frame-sig* =
  fixes *Dm* :: *ty* ⇒ *'u* ⇒ *bool* (‹𝒟_›)
    and *Ap* :: *'u* ⇒ *'u* ⇒ *'u* (**infixl** ‹@› *200*)
    and *Lm* :: *ty* ⇒ (*'u* ⇒ *'u*) ⇒ *'u* **and** *Tv* :: *'u* **and** *Fv* :: *'u*
    and *Ngv* :: *'u* **and** *Dsv* :: *'u* **and** *Iv* :: *ty* ⇒ *'u*
    and *Ev* :: *ty* ⇒ *'u* **and** *Piv* :: *ty* ⇒ *'u* **and** *Jv* :: *'p* ⇒ *ty* ⇒ *'u*
begin

The denotation $(\!|A|\!)_\xi$ of a term under an assignment ξ — BKK's evaluation function (BKK Definition 3.18).

fun *den* :: *'p tm* ⇒ (*nat* ⇒ *ty* ⇒ *'u*) ⇒ *'u* (‹(|-|)_› [*0*,*0*] *1000*) **where**
    $(\!|Bnd\ i|\!)_\xi$ = *Tv* — junk: no free bound index in a locally closed term
  | $(\!|n^f{}_\sigma|\!)_\xi$ = *ξ n σ*
  | $(\!|p^p{}_\sigma|\!)_\xi$ = *Jv p σ*
  | $(\!|Neg|\!)_\xi$ = *Ngv*
  | $(\!|Dis|\!)_\xi$ = *Dsv*
  | $(\!|Pi\ \sigma|\!)_\xi$ = *Piv σ*
  | $(\!|Iota\ \sigma|\!)_\xi$ = *Iv σ*
  | $(\!|s \cdot t|\!)_\xi$ = $((\!|s|\!)_\xi)$ @ $((\!|t|\!)_\xi)$
  | $(\!|\mathbf{\Lambda}_\sigma\ b|\!)_\xi$ = *Lm σ* (λ*d*. $(\!|b\langle(fresh\ (fvs\ b))^f{}_\sigma\rangle|\!)_{\xi((fresh\ (fvs\ b))_\sigma := d)}$)
  | $(\!|Eq\ \sigma|\!)_\xi$ = *Ev σ*

The recursive equations must be kept out of the default simpset: the *Abs* equation unfolds under a λ and would loop.

declare *den.simps*(*8*,*9*) [*simp del*]

Coincidence (BKK Definition 3.18(3)): the denotation depends only on the assignment at the (typed) free occurrences.

lemma *den-coincidence*: $(\bigwedge n\ \tau.\ (n,\ \tau) \in occ\ t \Longrightarrow \xi\ n\ \tau = \xi'\ n\ \tau) \Longrightarrow (\!|t|\!)_{\xi} = (\!|t|\!)_{\xi'}$
⟨*proof*⟩

α-invariance: opening with either of two fresh free variables gives the same denotation. This makes the fresh choice built into *den* irrelevant, and is the key to the substitution-value lemma below. The abstraction case renames both internal fresh choices to a common fresh w (inner induction hypothesis), commutes the openings ($[\![?i \neq ?j;\ lc\ ?u;\ lc\ ?v]\!] \Longrightarrow opn\ ?i\ ?u\ (opn\ ?j\ ?v\ ?t) = opn\ ?j\ ?v\ (opn\ ?i\ ?u\ ?t)$), then swaps the two names (outer induction hypothesis).

lemma *den-rename*: $x \notin fvs\ t \Longrightarrow y \notin fvs\ t \Longrightarrow (\!|opn\ k\ (x^f\sigma)\ t|\!)_{\xi(x_\sigma := d)}$
$= (\!|opn\ k\ (y^f\sigma)\ t|\!)_{\xi(y_\sigma := d)}$
⟨*proof*⟩

Any sufficiently fresh variable may be used to compute an abstraction's denotation.

lemma *den-Abs*: $x \notin fvs\ b \Longrightarrow (\!|\mathbf{\Lambda}_\sigma\ b|\!)_{\xi} = Lm\ \sigma\ (\lambda d.\ (\!|b\langle x^f\sigma\rangle|\!)_{\xi(x_\sigma := d)})$
⟨*proof*⟩

The substitution-value lemma (BKK Lemma 3.20).

lemma *den-fsub*: $lc\ u \Longrightarrow (\!|fsub\ x\ \sigma\ u\ t|\!)_{\xi} = (\!|t|\!)_{\xi(x_\sigma := (\!|u|\!)_{\xi})}$
⟨*proof*⟩

The β form of the substitution-value lemma (BKK Lemma 3.20): opening an abstraction body with a (locally closed) argument u is computed by evaluating the body under the assignment updated with the denotation of u. This is the semantic counterpart of β-reduction and the workhorse of soundness.

lemma *den-beta*: **assumes** *u*: $lc\ u$ **and** *x*: $x \notin fvs\ b$
shows $(\!|b\langle u\rangle|\!)_{\xi} = (\!|b\langle x^f\sigma\rangle|\!)_{\xi(x_\sigma := (\!|u|\!)_{\xi})}$
⟨*proof*⟩

end

Satisfaction of the connectives (the Σ-valuation conditions of BKK Definition 3.41 and Figure 2; for the *defined* connectives cf. BKK Remark 3.47 and Lemma 3.48): model-theoretic facts, stated here so that the calculus can use them for soundness.

context *sigma-model*
begin

lemma *sat-Neg*: **assumes** $wff_{\mathrm{o}}(A)$ **and** *asg* ξ
shows *vl* (*Ee* ξ ($\neg$ A)) $\longleftrightarrow$ $\neg$ *vl* (*Ee* ξ A)
⟨*proof*⟩
lemma *sat-Dis*: **assumes** $wff_{\mathrm{o}}(A)$ **and** $wff_{\mathrm{o}}(B)$ **and** *asg* ξ
shows *vl* (*Ee* ξ ($A \vee B$)) $\longleftrightarrow$ *vl* (*Ee* ξ A) $\vee$ *vl* (*Ee* ξ B)
⟨*proof*⟩
lemma *sat-ImpB*: **assumes** $wff_{\mathrm{o}}(A)$ **and** $wff_{\mathrm{o}}(B)$ **and** *asg* ξ
shows *vl* (*Ee* ξ ($A \supset B$)) $\longleftrightarrow$ (*vl* (*Ee* ξ A) $\longrightarrow$ *vl* (*Ee* ξ B))
⟨*proof*⟩
lemma *sat-Pi*: **assumes** $wff_{\sigma \Rightarrow \mathrm{o}}(G)$ **and** *asg* ξ
shows *vl* (*Ee* ξ (*Pi* σ · G)) $\longleftrightarrow$ ($\forall$ *d*. *Dm* σ *d* $\longrightarrow$ *vl* (*Ap* (*Ee* ξ G) *d*))
⟨*proof*⟩
lemma *sat-Forall*: **assumes** *wA*: $wff_{\sigma \Rightarrow \mathrm{o}}(\mathbf{\Lambda}_\sigma\ b)$ **and** *x*: $x \notin$ *fvs* b **and** *xi*: *asg* ξ
shows *vl* (*Ee* ξ ($\mathbf{\Pi}_\sigma$ b)) $\longleftrightarrow$ ($\forall$ *d*. *Dm* σ *d* $\longrightarrow$ *vl* (*Ee* ($\xi(x_\sigma := d)$) ($b\langle x^f{}_\sigma \rangle$)))
⟨*proof*⟩

BKK Lemma 4.2: in a Σ-model with primitive equality (which gives property q with witness $E(=_\alpha)$), Leibniz equality is satisfied exactly by equal denotations.

end

2.9 Σ-Henkin models: the general-model locale (BKK Definition 3.50)

BKK's soundness and completeness theorems (BKK Theorem 7.3, Corollary 7.7) cover all eight model classes $\mathcal{M}_*$; this development instantiates the most specialised one, the class $\mathcal{M}_{\beta f b}$ of Σ-*Henkin models* (BKK Definition 3.50): Σ-models (BKK Definition 3.41) satisfying property b, property f (functionality), and property q (BKK Definitions 3.46 and 3.49). Crucially the function domains need not be full (BKK Definition 3.5): following Henkin — in BKK's words, it is sufficient to require that $\mathcal{D}_{\alpha \Rightarrow \beta}$ "has enough members that any well-formed formula can be evaluated" (BKK Section 2.3.1). We therefore require the λ-conditions — λ-comprehension *gm-lamTy* and the β-condition *gm-beta* of the Σ-evaluation (BKK Definition 3.18) — only for the functions that are *denotations of λ-terms*; equivalently, every wff denotes. A term model cannot be full: with property b the domain $\mathcal{D}_{\iota \Rightarrow \mathrm{o}}$ of a full frame over infinite $\mathcal{D}_\iota$ would be uncountable, whereas the term model is countable. (BKK avoid Andrews' term *general models* for this notion; we keep it in the locale name *general-model*, in Andrews' sense.) Standard models — the full case, BKK Definition 3.51 — appear at the end of this section as a sublocale. Beyond BKK, the locale carries the description condition *gm-descB* for the typed description operators *Iota* σ (Andrews 1972, BKK's reference [3]), matched by the rule $NK(\iota)$ of the calculus.

Note that we render the applicative structure abstractly (an application operation @ on a carrier $'u$) rather than literally over a frame of functions (BKK Definition 3.4); by functionality and BKK Theorem 3.68 the two presentations describe the same class of models up to isomorphism.

locale *general-model* = *frame-sig* +
— every domain is inhabited (part of BKK Definition 3.1, applicative structures)
assumes *gm-nonempty*: $\exists\, d.\ \mathcal{D}_\sigma\ d$
— property b (BKK Definition 3.46): $\mathcal{D}_\mathrm{o} = \{Tv, Fv\}$
and *gm-TF*: $Tv \neq Fv$ **and** *gm-boolean*: $\mathcal{D}_\mathrm{o}\ a \longleftrightarrow a = Tv \vee a = Fv$
— application stays in the codomain, and the constants inhabit their domains
and *gm-appTy*: $[\![\mathcal{D}_{\sigma\Rightarrow\tau}\ f;\ \mathcal{D}_\sigma\ a]\!] \Longrightarrow \mathcal{D}_\tau\ (f\ @\ a)$
and *gm-negTy*: $\mathcal{D}_{\mathrm{o}\Rightarrow\mathrm{o}}\ Ngv$ **and** *gm-disTy*: $\mathcal{D}_{\mathrm{o}\Rightarrow\mathrm{o}\Rightarrow\mathrm{o}}\ Dsv$
and *gm-piTy*: $\mathcal{D}_{(\sigma\Rightarrow\mathrm{o})\Rightarrow\mathrm{o}}\ (Piv\ \sigma)$
and *gm-iotaTy*: $\mathcal{D}_{(\sigma\Rightarrow\mathrm{o})\Rightarrow\sigma}\ (Iv\ \sigma)$ **and** *gm-parTy*: $\mathcal{D}_\sigma\ (Jv\ p\ \sigma)$
— Σ-valuation (BKK Figure 2 / Definition 3.41) with $v = (\lambda a.\ a = Tv)$
and *gm-negB*: $\mathcal{D}_\mathrm{o}\ a \Longrightarrow (Ngv\ @\ a = Tv) = (a \neq Tv)$
and *gm-disB*: $[\![\mathcal{D}_\mathrm{o}\ a;\ \mathcal{D}_\mathrm{o}\ b]\!] \Longrightarrow (Dsv\ @\ a\ @\ b = Tv) = (a = Tv \vee b = Tv)$
and *gm-piB*: $\mathcal{D}_{\sigma\Rightarrow\mathrm{o}}\ f \Longrightarrow (Piv\ \sigma\ @\ f = Tv) = (\forall\, d.\ \mathcal{D}_\sigma\ d \longrightarrow f\ @\ d = Tv)$
— property f (functionality, BKK Definition 3.46) and property q (BKK Definitions 3.46 and 3.49)
and *gm-funct*: $[\![\mathcal{D}_{\sigma\Rightarrow\tau}\ g;\ \mathcal{D}_{\sigma\Rightarrow\tau}\ k;\ \bigwedge a.\ \mathcal{D}_\sigma\ a \Longrightarrow g\ @\ a = k\ @\ a]\!] \Longrightarrow g = k$
— primitive equality (BKK Remark 7.9): $Ev\ \sigma$ satisfies $L^\sigma{}_=$ of BKK Figure 2; property q (BKK Definitions 3.46 and 3.49) follows with witness $Ev\ \sigma$
and *gm-eqTy*: $\mathcal{D}_{\sigma\Rightarrow\sigma\Rightarrow\mathrm{o}}\ (Ev\ \sigma)$
and *gm-eqB*: $[\![\mathcal{D}_\sigma\ a;\ \mathcal{D}_\sigma\ b]\!] \Longrightarrow (Ev\ \sigma\ @\ a\ @\ b = Tv) = (a = b)$
— the description condition (beyond BKK, cf. Andrews 1972): if f behaves as the singleton $\{a\}$, description picks out a
and *gm-descB*: $[\![\mathcal{D}_{\sigma\Rightarrow\mathrm{o}}\ f;\ \mathcal{D}_\sigma\ a;\ \forall\, b.\ \mathcal{D}_\sigma\ b \longrightarrow (f\ @\ b = Tv) = (b = a)]\!] \Longrightarrow Iv\ \sigma\ @\ f = a$
— Henkin λ-conditions: λ-comprehension (the Σ-evaluation is total on wffs, BKK Definition 3.18) and the β-condition (BKK Definition 3.18(4)), at the *denotation* functions only
and *gm-lamTy*: $[\![wff_{\sigma\Rightarrow\tau}(\mathbf{\Lambda}_\sigma\ bd);\ \forall\, n\ \varrho.\ \mathcal{D}_\varrho\ (\xi\ n\ \varrho)]\!] \Longrightarrow \mathcal{D}_{\sigma\Rightarrow\tau}\ (\llparenthesis \mathbf{\Lambda}_\sigma\ bd \rrparenthesis_\xi)$
and *gm-beta*: $[\![wff_\tau(bd\langle x^f{}_\sigma\rangle);\ x \notin fvs\ bd;\ \forall\, n\ \varrho.\ \mathcal{D}_\varrho\ (\xi\ n\ \varrho);\ \mathcal{D}_\sigma\ a]\!]$
$\Longrightarrow \llparenthesis \mathbf{\Lambda}_\sigma\ bd \rrparenthesis_\xi\ @\ a = \llparenthesis bd\langle x^f{}_\sigma\rangle \rrparenthesis_{\xi(x_\sigma\ :=\ a)}$
begin

An assignment is *total* (*resp*, respects the domains everywhere) if it maps every typed variable into the matching domain — BKK's assignment *into* the structure.

definition *resp* **where** $resp\ \xi \longleftrightarrow (\forall\, (n{::}nat)\ \tau.\ \mathcal{D}_\tau\ (\xi\ n\ \tau))$
lemma *Tv-dom* [*simp*]: $\mathcal{D}_\mathrm{o}\ Tv$ **and** *Fv-dom* [*simp*]: $\mathcal{D}_\mathrm{o}\ Fv$
⟨*proof*⟩

Every well-formed term denotes in the domain of its type (BKK Definition 3.18): the λ-comprehension condition *gm-lamTy* is exactly what makes the abstraction case go through.

lemma *den-dom*: $wff_\sigma(t) \Longrightarrow resp\ \xi \Longrightarrow \mathcal{D}_\sigma\ (\llparenthesis t \rrparenthesis_\xi)$

⟨*proof*⟩

The semantic β rule at the term level (BKK Remark 3.19): applying an abstraction to a well-formed argument evaluates the opened body.

lemma *den-App-Abs*:
  assumes *wb*: $wff_{\sigma\Rightarrow\tau}(\mathbf{\Lambda}_\sigma\ b)$ **and** *wa*: $wff_\sigma(a)$ **and** *r*: *resp* ξ
  shows $(\!|(\mathbf{\Lambda}_\sigma\ b)\cdot a|\!)_\xi = (\!|b\langle a\rangle|\!)_\xi$
⟨*proof*⟩

end

2.10 Closed well-formed terms and simultaneous substitution

A *closed* well-formed term, BKK's $cwff_\sigma$ (BKK Section 2.2; BKK reserve *sentence* for closed formulae of type o — parameters are allowed, they play the role of BKK's constants). These are the carriers of the term model.

definition *cwff* :: *ty* ⇒ *'p tm* ⇒ *bool* **where** $cwff\ \sigma\ A \longleftrightarrow wff_\sigma(A) \wedge fvs\ A = \{\}$
lemma *cwffI*: $wff_\sigma(A) \Longrightarrow fvs\ A = \{\} \Longrightarrow cwff\ \sigma\ A$ ⟨*proof*⟩
lemma *cwff-wff*: $cwff\ \sigma\ A \Longrightarrow wff_\sigma(A)$ ⟨*proof*⟩
lemma *cwff-closed*: $cwff\ \sigma\ A \Longrightarrow fvs\ A = \{\}$ ⟨*proof*⟩
lemma *cwff-lc*: $cwff\ \sigma\ A \Longrightarrow lc\ A$ ⟨*proof*⟩
lemma *cwff-Neg*: $cwff$ (o⇒o) *Neg*
  and *cwff-Dis*: $cwff$ (o⇒o⇒o) *Dis*
  and *cwff-Pi*: $cwff$ ((σ⇒o)⇒o) (*Pi* σ)
  and *cwff-Iota*: $cwff$ ((σ⇒o)⇒σ) (*Iota* σ)
  and *cwff-TrueB*: $cwff$ o ⊤
  and *cwff-FalseB*: $cwff$ o ⊥
  ⟨*proof*⟩
lemma *cwff-Eq*: $cwff$ (σ⇒σ⇒o) (*Eq* σ) ⟨*proof*⟩
lemma *cwff-Par*: $cwff\ \sigma\ (p^p{}_\sigma)$ ⟨*proof*⟩
lemma *cwff-App*: $cwff\ (\sigma\Rightarrow\tau)\ s \Longrightarrow cwff\ \sigma\ t \Longrightarrow cwff\ \tau\ (s\cdot t)$
  ⟨*proof*⟩
lemma *cwff-opn*: $cwff\ (\sigma\Rightarrow\tau)\ (\mathbf{\Lambda}_\sigma\ b) \Longrightarrow cwff\ \sigma\ a \Longrightarrow cwff\ \tau\ (b\langle a\rangle)$
  ⟨*proof*⟩
lemma *cwff-unique*: $cwff\ \sigma\ A \Longrightarrow cwff\ \tau\ A \Longrightarrow \sigma = \tau$
  ⟨*proof*⟩

Converse of $[\![wff_{?\sigma} \Rightarrow {}_{?\tau}(\mathbf{\Lambda}_{?\sigma}\ ?b);\ ?x \notin fvs\ ?b]\!] \Longrightarrow wff_{?\tau}(?b\langle ?x^f{}_{?\sigma}\rangle)$: a body that is well-typed when opened with *one* fresh variable yields a well-typed abstraction (all fresh openings are α-variants).

lemma *wff-Abs-open-rev*: $wff_\tau(b\langle x^f{}_\sigma\rangle) \Longrightarrow x \notin fvs\ b \Longrightarrow wff_{\sigma\Rightarrow\tau}(\mathbf{\Lambda}_\sigma\ b)$
  ⟨*proof*⟩

Simultaneous substitution of a (closed) term for every free variable — the analogue of the closing substitution σ in BKK's term evaluation (BKK Definition 3.35). Because the replacements are closed, it commutes with

opening — the locally-nameless analogue of BKK's parallel substitution, with no binder renaming.

primrec *msub* :: $(nat \Rightarrow ty \Rightarrow 'p\ tm) \Rightarrow 'p\ tm \Rightarrow 'p\ tm$ **where**
  $msub\ \varrho\ (Bnd\ i) = Bnd\ i$
| $msub\ \varrho\ (n^f{}_\tau) = \varrho\ n\ \tau$
| $msub\ \varrho\ (p^p{}_\tau) = p^p{}_\tau$
| $msub\ \varrho\ Neg = Neg$
| $msub\ \varrho\ Dis = Dis$
| $msub\ \varrho\ (Pi\ \tau) = Pi\ \tau$
| $msub\ \varrho\ (Iota\ \tau) = Iota\ \tau$
| $msub\ \varrho\ (Eq\ \tau) = Eq\ \tau$
| $msub\ \varrho\ (s \cdot t) = (msub\ \varrho\ s) \cdot (msub\ \varrho\ t)$
| $msub\ \varrho\ (\mathbf{\Lambda}_\tau\ b) = \mathbf{\Lambda}_\tau\ (msub\ \varrho\ b)$

Parallel-substitution notation: $u[\![\varrho]\!]$ applies the substitution ϱ to every free variable of u.

syntax *-msub* :: $logic \Rightarrow logic \Rightarrow logic$ $(\langle\text{-}[\![\text{-}]\!]\rangle\ [1000,\ 0]\ 1000)$
syntax-consts *-msub* == *msub*
translations $u[\![\varrho]\!] \rightleftharpoons CONST\ msub\ \varrho\ u$

lemma *msub-opn*: $(\bigwedge n\ \tau.\ lc\ (\varrho\ n\ \tau)) \Longrightarrow msub\ \varrho\ (opn\ k\ u\ t) = opn\ k\ (msub\ \varrho\ u)\ (msub\ \varrho\ t)$
  ⟨*proof*⟩
lemma *msub-cong*: $(\bigwedge n\ \tau.\ (n, \tau) \in occ\ t \Longrightarrow \varrho\ n\ \tau = \varrho'\ n\ \tau) \Longrightarrow msub\ \varrho\ t = msub\ \varrho'\ t$
  ⟨*proof*⟩
lemma *fvs-msub*: $(\bigwedge n\ \tau.\ fvs\ (\varrho\ n\ \tau) = \{\}) \Longrightarrow fvs\ (msub\ \varrho\ t) = \{\}$
  ⟨*proof*⟩
lemma *wff-msub*:
  $wff_\sigma(t) \Longrightarrow (\bigwedge n\ \tau.\ wff_\tau(\varrho\ n\ \tau)) \Longrightarrow wff_\sigma(msub\ \varrho\ t)$
⟨*proof*⟩

A well-formed term becomes a *closed* well-formed term under a closed simultaneous substitution — the instance used to build the term model.

lemma *cwff-msub*: $wff_\sigma(t) \Longrightarrow (\bigwedge n\ \tau.\ cwff\ \tau\ (\varrho\ n\ \tau)) \Longrightarrow cwff\ \sigma\ (msub\ \varrho\ t)$
  ⟨*proof*⟩

A closed term is untouched by simultaneous substitution.

lemma *msub-closed*: $fvs\ t = \{\} \Longrightarrow msub\ \varrho\ t = t$ ⟨*proof*⟩

2.11 The valuation locale

The *valuation* locale axiomatises what a term model provides: a carrier $'u$ with an application @ and an evaluation $\mathcal{V}$ of closed well-formed terms. It abstracts the *quotient* of BKK's term evaluation (BKK Definition 3.35) by Leibniz equality, as constructed in the model-existence proof (BKK Theorem 6.33) — note that the bare term evaluation $\mathcal{TE}(\Sigma)^\beta$ is *not* functional (BKK Remark 3.37); functionality only holds after the quotient.

The axioms: *v-app*/*v-beta* are the evaluation conditions (BKK Definition 3.18(2),(4)); *v-neg*, *v-dis*, *v-pi* are $L_\neg$, $L_\vee$, $L^\sigma{}_\forall$ (BKK Figure 2); *v-ext* is functionality (property f), *v-eq* is primitive equality $L^\sigma{}_=$ (whence property q), *v-desc* the description condition, *v-type* type-disjointness of the domains, and *v-TF*/*v-bool* are property b (BKK Definition 3.46).

locale *valuation* =
  fixes *Dv* :: *ty* ⇒ *'u* ⇒ *bool* (‹$\mathcal{D}$_›)
    and *Vap* :: *'u* ⇒ *'u* ⇒ *'u* (**infixl** ‹@› *200*)
    and *Val* :: *'p tm* ⇒ *'u* (‹$\mathcal{V}$›)
  assumes *v-dom*: $\mathcal{D}_\sigma\ d \longleftrightarrow (\exists\, t.\ cwff\ \sigma\ t \wedge d = \mathcal{V}\ t)$
      and *v-app*: $cwff\ (\sigma{\Rightarrow}\tau)\ s \Longrightarrow cwff\ \sigma\ t \Longrightarrow \mathcal{V}\ (s \cdot t) = \mathcal{V}\ s\ @\ \mathcal{V}\ t$
      and *v-beta*: $cwff\ (\sigma{\Rightarrow}\tau)\ (\mathbf{\Lambda}_\sigma\ b) \Longrightarrow cwff\ \sigma\ a \Longrightarrow \mathcal{V}\ (\mathbf{\Lambda}_\sigma\ b)\ @\ \mathcal{V}\ a = \mathcal{V}\ (b\langle a\rangle)$
      and *v-TF*: $\mathcal{V}\ \top \neq \mathcal{V}\ \bot$
      and *v-bool*: $cwff\ \mathrm{o}\ \varphi \Longrightarrow \mathcal{V}\ \varphi = \mathcal{V}\ \top \vee \mathcal{V}\ \varphi = \mathcal{V}\ \bot$
      and *v-neg*: $cwff\ \mathrm{o}\ \varphi \Longrightarrow \mathcal{V}\ (\neg\ \varphi) = (if\ \mathcal{V}\ \varphi = \mathcal{V}\ \top\ then\ \mathcal{V}\ \bot\ else\ \mathcal{V}\ \top)$
      and *v-dis*: $cwff\ \mathrm{o}\ \varphi \Longrightarrow cwff\ \mathrm{o}\ \psi \Longrightarrow$
            $\mathcal{V}\ (\varphi \vee \psi) = (if\ \mathcal{V}\ \varphi = \mathcal{V}\ \top \vee \mathcal{V}\ \psi = \mathcal{V}\ \top\ then\ \mathcal{V}\ \top\ else\ \mathcal{V}\ \bot)$
      and *v-pi*: $cwff\ (\sigma{\Rightarrow}\mathrm{o})\ f \Longrightarrow \mathcal{V}\ ((Pi\ \sigma) \cdot f) =$
          $(if\ (\forall\, a.\ cwff\ \sigma\ a \longrightarrow \mathcal{V}\ f\ @\ \mathcal{V}\ a\ = \mathcal{V}\ \top)\ then\ \mathcal{V}\ \top\ else\ \mathcal{V}\ \bot)$
      and *v-ext*: $cwff\ (\sigma{\Rightarrow}\tau)\ g \Longrightarrow cwff\ (\sigma{\Rightarrow}\tau)\ h$
            $\Longrightarrow (\bigwedge a.\ cwff\ \sigma\ a \Longrightarrow \mathcal{V}\ g\ @\ \mathcal{V}\ a = \mathcal{V}\ h\ @\ \mathcal{V}\ a) \Longrightarrow \mathcal{V}\ g = \mathcal{V}\ h$
      and *v-eq*: $cwff\ \sigma\ a \Longrightarrow cwff\ \sigma\ b \Longrightarrow (\mathcal{V}\ (Eq\ \sigma)\ @\ \mathcal{V}\ a\ @\ \mathcal{V}\ b = \mathcal{V}\ \top) = (\mathcal{V}\ a = \mathcal{V}\ b)$
      and *v-desc*: $cwff\ (\sigma{\Rightarrow}\mathrm{o})\ f \Longrightarrow cwff\ \sigma\ a$
            $\Longrightarrow (\forall\, b.\ cwff\ \sigma\ b \longrightarrow (\mathcal{V}\ f\ @\ \mathcal{V}\ b = \mathcal{V}\ \top) = (\mathcal{V}\ b = \mathcal{V}\ a))$
            $\Longrightarrow \mathcal{V}\ ((Iota\ \sigma) \cdot f) = \mathcal{V}\ a$
      and *v-type*: $cwff\ \sigma\ s \Longrightarrow cwff\ \tau\ t \Longrightarrow \mathcal{V}\ s = \mathcal{V}\ t \Longrightarrow \sigma = \tau$
begin

lemma *v-domI*: $cwff\ \sigma\ t \Longrightarrow \mathcal{D}_\sigma\ (\mathcal{V}\ t)$ ⟨*proof*⟩

end

2.12 The term evaluation (BKK Section 6)

A valuation extends to an evaluation function by simultaneous substitution of representatives: $E_\xi(A) := \mathcal{V}([\varrho_\xi]A)$, BKK's evaluation for the term structure. β-respect is inherited from *v-beta* under closing substitutions, functionality is *v-ext*, and the remaining valuation conditions supply the L-properties — so every valuation is directly a Σ-model in the class $\mathcal{M}_{\beta f b}$, with no detour through the recursive denotation.

context *valuation*
begin

definition *vresp* **where** $vresp\ \xi \longleftrightarrow (\forall\, (n{::}nat)\ \tau.\ \mathcal{D}_\tau\ (\xi\ n\ \tau))$
definition *rep-of* **where** $rep\text{-}of\ \xi\ (n{::}nat)\ \tau = (SOME\ t.\ cwff\ \tau\ t \wedge \xi\ n\ \tau = \mathcal{V}\ t)$
lemma *rep-of-spec*: **assumes** *vresp* ξ **shows** $cwff\ \tau\ (rep\text{-}of\ \xi\ n\ \tau) \wedge \xi\ n\ \tau = \mathcal{V}$

(*rep-of* ξ n τ)
⟨*proof*⟩

The valuation respects β-conversion under closing substitutions (BKK's quotient of the term structure by β, Section 6): the abstraction case is pointwise by *v-beta* and closed by *v-ext*.

lemma *beq-V*: $s \approx_{\varrho'} t \Longrightarrow (\bigwedge n\ \tau.\ cwff\ \tau\ (\varrho\ n\ \tau)) \Longrightarrow \mathcal{V}\ (msub\ \varrho\ s) = \mathcal{V}\ (msub\ \varrho\ t)$
⟨*proof*⟩

BKK's evaluation function for the term model.

definition *Ev* :: $(nat \Rightarrow ty \Rightarrow 'u) \Rightarrow 'p\ tm \Rightarrow 'u$ **where** $Ev\ \xi\ A = \mathcal{V}\ (msub\ (rep\text{-}of\ \xi)\ A)$

end

Every valuation is a Σ-model in the class $\mathcal{M}_{\beta fb}$ (the direct construction of BKK Section 6).

sublocale *valuation* ⊆ *bkkA*: *app-struct Dv Vap*
⟨*proof*⟩

sublocale *valuation* ⊆ *bkkM*: *bkk-model Dv Vap Ev* $\lambda a.\ a = \mathcal{V}\ \top$
⟨*proof*⟩

2.13 Standard models (BKK Definition 3.51)

A Σ-*standard model* (BKK Definition 3.51) is a Σ-Henkin model over a *full* frame (BKK Definition 3.5): every set-function between domains has a representative. We record fullness by the universal abstraction laws *Lm-dom* and *beta-Lm*, quantified over *all* functions $h :: 'u \Rightarrow 'u$ — strictly stronger than the Henkin conditions of *general-model*. On top of the full frame we assume the Σ-valuation conditions (BKK Definition 3.41, Figure 2) with property b, property f (functionality) and property q (BKK Definition 3.46).

locale *standard-model* = *frame-sig* +
— fullness (BKK Definition 3.5): every function has a representative, @ computes it
assumes *beta-Lm*: $[\![\bigwedge d.\ \mathcal{D}_\sigma\ d \Longrightarrow \mathcal{D}_\tau\ (h\ d);\ \mathcal{D}_\sigma\ a]\!] \Longrightarrow Lm\ \sigma\ h\ @\ a = h\ a$
and *Lm-dom*: $(\bigwedge d.\ \mathcal{D}_\sigma\ d \Longrightarrow \mathcal{D}_\tau\ (h\ d)) \Longrightarrow \mathcal{D}_{\sigma\Rightarrow\tau}\ (Lm\ \sigma\ h)$
and *Ap-dom*: $[\![\mathcal{D}_{\sigma\Rightarrow\tau}\ f;\ \mathcal{D}_\sigma\ a]\!] \Longrightarrow \mathcal{D}_\tau\ (f\ @\ a)$
— the logical constants and parameters inhabit their domains
and *Ngv-dom*: $\mathcal{D}_{o\Rightarrow o}\ Ngv$
and *Dsv-dom*: $\mathcal{D}_{o\Rightarrow o\Rightarrow o}\ Dsv$
and *Piv-dom*: $\mathcal{D}_{(\sigma\Rightarrow o)\Rightarrow o}\ (Piv\ \sigma)$
and *Iv-dom*: $\mathcal{D}_{(\sigma\Rightarrow o)\Rightarrow\sigma}\ (Iv\ \sigma)$
and *Ev-dom*: $\mathcal{D}_{\sigma\Rightarrow\sigma\Rightarrow o}\ (Ev\ \sigma)$ **and** *Jv-dom*: $\mathcal{D}_\sigma\ (Jv\ p\ \sigma)$
— property b (BKK Definition 3.46): $\mathcal{D}_o = \{Tv, Fv\}$
and *TF*: $Tv \neq Fv$ **and** *boolean*: $\mathcal{D}_o\ a \longleftrightarrow a = Tv \vee a = Fv$

— Σ-valuation (BKK Definition 3.41, Figure 2) with $\upsilon = (\lambda a.\ a = Tv)$
and *Lneg*: $\mathcal{D}_{\mathrm{o}}\ a \Longrightarrow (Ngv\ \mathbin{@}\ a = Tv) = (a \neq Tv)$
and *Ldis*: $[\![\mathcal{D}_{\mathrm{o}}\ a;\ \mathcal{D}_{\mathrm{o}}\ b]\!] \Longrightarrow (Dsv\ \mathbin{@}\ a\ \mathbin{@}\ b = Tv) = (a = Tv \vee b = Tv)$
and *Lall*: $\mathcal{D}_{\sigma \Rightarrow \mathrm{o}}\ f \Longrightarrow (Piv\ \sigma\ \mathbin{@}\ f = Tv) = (\forall d.\ \mathcal{D}_\sigma\ d \longrightarrow f\ \mathbin{@}\ d = Tv)$
and *Leq*: $[\![\mathcal{D}_\sigma\ a;\ \mathcal{D}_\sigma\ b]\!] \Longrightarrow (Ev\ \sigma\ \mathbin{@}\ a\ \mathbin{@}\ b = Tv) = (a = b)$
— properties f and q (BKK Definition 3.46)
and *funct*: $[\![\mathcal{D}_{\sigma \Rightarrow \tau}\ g;\ \mathcal{D}_{\sigma \Rightarrow \tau}\ k;\ \bigwedge a.\ \mathcal{D}_\sigma\ a \Longrightarrow g\ \mathbin{@}\ a = k\ \mathbin{@}\ a]\!] \Longrightarrow g = k$
— the description condition (beyond BKK, cf. Andrews 1972)
and *descB*: $[\![\mathcal{D}_{\sigma \Rightarrow \mathrm{o}}\ f;\ \mathcal{D}_\sigma\ a;\ \forall b.\ \mathcal{D}_\sigma\ b \longrightarrow (f\ \mathbin{@}\ b = Tv) = (b = a)]\!]$
$\Longrightarrow Iv\ \sigma\ \mathbin{@}\ f = a$
begin

As in *general-model*, membership of the truth values follows from property b.

lemma *Tv-dom* [*simp*]: $\mathcal{D}_{\mathrm{o}}\ Tv$ **and** *Fv-dom* [*simp*]: $\mathcal{D}_{\mathrm{o}}\ Fv$
⟨*proof*⟩

In a full frame every domain is inhabited (for o by *Tv*, for ι by applying *Iv* to a representable predicate, for function types by a constant function). BKK build non-emptiness into the applicative structure (BKK Definition 3.1).

lemma *dom-nonempty*: $\exists d.\ \mathcal{D}_\tau\ d$ ⟨*proof*⟩

In a full frame every well-formed term denotes in the domain of its type (the standard homomorphic construction, BKK Section 2.3.1): the abstraction case is immediate from fullness.

lemma *std-den-dom*: $wff_\sigma(t) \Longrightarrow \forall n\ \varrho.\ \mathcal{D}_\varrho\ (\xi\ n\ \varrho) \Longrightarrow \mathcal{D}_\sigma\ (\!|t|\!)_\xi$
⟨*proof*⟩

end

Every standard model is a Σ-Henkin general model (BKK Definition 3.51 is a special case of Definition 3.50): the universal abstraction laws specialise to the denotation functions.

sublocale *standard-model* ⊆ *general-model Dm Ap Lm Tv Fv Ngv Dsv Iv Ev Piv Jv*
⟨*proof*⟩

3 The natural-deduction calculus NK

We formalise the calculus $NK_{\beta f b}$ of BKK (BKK Definition 7.1, Figures 6 and 7) — the base system NK_β together with the extensional rules $NK(f)$ and $NK(b)$ — extended by BKK's rules $NK(=_r)$ and $NK(=_l)$ for the primitive equality of our signature (BKK Figure 9, Remark 7.9) and by the description rule $NK(\iota)$, a premise-free axiom scheme describing Leibniz singletons (beyond BKK, following Andrews 1972, their reference [3]). BKK

prove $NK_{\beta fb}$ sound and complete for the class of Σ-Henkin models $\mathcal{M}_{\beta fb}$ (BKK Theorem 7.3, Theorem 7.6, Corollary 7.7), and sketch the extension by primitive equality in Remark 7.9; we prove the fully extended calculus sound and complete for the correspondingly enriched class (with primitive equality and description, Section 2). The provability judgement $\Phi \vdash A$ relates a set of sentences Φ to a sentence A. Eigenvariables are *parameters* (BKK's w_α), which — unlike the free variables used only during evaluation — never occur bound.

3.1 The inference rules of $NK_{\beta fb}$ (BKK Figures 6, 7 and 9)

Following BKK we work with the primitive constants $\neg$, $\vee$, Π_α, $=_\alpha$ and $\boldsymbol{\iota}_\alpha$; the remaining operators ($\supset$, $\bot$, Leibniz equality $\doteq$) are defined. The rule $NK(\Pi I)$ discharges an eigen-parameter w that must not occur in the context Φ or in the quantified predicate.

inductive *bprov* :: *'p tm set* ⇒ *'p tm* ⇒ *bool* (**infix** ‹⊢› *40*) **where**
  Hyp: $A \in \Phi \Longrightarrow \Phi \vdash A$ — BKK $NK(Hyp)$
| *Beta*: $A \approx_o B \Longrightarrow \Phi \vdash A \Longrightarrow \Phi \vdash B$ — BKK $NK(\beta)$
| *NegI*: $\Phi \cup \{A\} \vdash \bot \Longrightarrow wff_o(A) \Longrightarrow \Phi \vdash \neg A$ — BKK $NK(\neg I)$
| *NegE*: $\Phi \vdash \neg A \Longrightarrow \Phi \vdash A \Longrightarrow wff_o(C) \Longrightarrow \Phi \vdash C$ — BKK $NK(\neg E)$
| *DisIL*: $\Phi \vdash A \Longrightarrow wff_o(B) \Longrightarrow \Phi \vdash A \vee B$ — BKK $NK(\vee I_L)$
| *DisIR*: $\Phi \vdash B \Longrightarrow wff_o(A) \Longrightarrow \Phi \vdash A \vee B$ — BKK $NK(\vee I_R)$
| *DisE*: $\Phi \vdash A \vee B \Longrightarrow \Phi \cup \{A\} \vdash C \Longrightarrow \Phi \cup \{B\} \vdash C$
  $\Longrightarrow wff_o(A) \Longrightarrow wff_o(B) \Longrightarrow \Phi \vdash C$ — BKK $NK(\vee E)$
| *PiI*: $\Phi \vdash G \cdot (w^p \alpha) \Longrightarrow wff_{\alpha \Rightarrow o}(G) \Longrightarrow w \notin pars\ G$
  $\Longrightarrow (\forall D \in \Phi.\ w \notin pars\ D)$
  $\Longrightarrow \Phi \vdash (Pi\ \alpha) \cdot G$ — BKK $NK(\Pi I)$
| *PiE*: $\Phi \vdash (Pi\ \alpha) \cdot G \Longrightarrow wff_\alpha(A) \Longrightarrow \Phi \vdash G \cdot A$ — BKK $NK(\Pi E)$
| *Contr*: $\Phi \cup \{\neg A\} \vdash \bot \Longrightarrow wff_o(A) \Longrightarrow \Phi \vdash A$ — BKK $NK(Contr)$
| *FuncE*: $\Phi \vdash \mathbf{\Pi}_\alpha\ (G \cdot (Bnd\ 0) \doteq_\beta H \cdot (Bnd\ 0))$
  $\Longrightarrow wff_{\alpha \Rightarrow \beta}(G) \Longrightarrow wff_{\alpha \Rightarrow \beta}(H)$
  $\Longrightarrow \Phi \vdash G \doteq_{\alpha \Rightarrow \beta} H$ — BKK $NK(f)$
| *BoolE*: $\Phi \cup \{A\} \vdash B \Longrightarrow \Phi \cup \{B\} \vdash A \Longrightarrow wff_o(A) \Longrightarrow wff_o(B)$
  $\Longrightarrow \Phi \vdash A \doteq_o B$ — BKK $NK(b)$
| *Desc*: $wff_\alpha(A) \Longrightarrow \Phi \vdash (Iota\ \alpha) \cdot ((Leib\ \alpha) \cdot A) \doteq_\alpha A$
  — $NK(\iota)$, beyond BKK: Leibniz-singleton description (Andrews 1972)
| *EqR*: $wff_\alpha(A) \Longrightarrow \Phi \vdash A =_\alpha A$ — BKK $NK(=_r)$, Figure 9
| *EqL*: $\Phi \vdash C =_\alpha D \Longrightarrow \Phi \vdash C \doteq_\alpha D$ — BKK $NK(=_l)$, Figure 9

Provability from the empty hypothesis set, with its own turnstile.

abbreviation *provable* :: *'p tm* ⇒ *bool* (‹⊢ -› [*61*] *60*) **where**
  provable A ≡ {} ⊢ *A*

Everything derivable from a set of propositions is a proposition.

lemma *bprov-wff*: $\Phi \vdash C \Longrightarrow (\bigwedge A\ .\ A \in \Phi \Longrightarrow wff_o(A)) \Longrightarrow wff_o(C)$
⟨*proof*⟩

Derivability is stable under injective parameter renaming — injectivity keeps the eigen-parameter side-conditions of $NK(\Pi I)$. This lets us move eigen-parameters out of the way, which is what makes weakening (and later, extension of consistent sets) admissible.

lemma *bprov-rename*: **assumes** π: *inj* π **shows** $\Phi \vdash C \Longrightarrow prn\ \pi\ `\ \Phi \vdash prn\ \pi\ C$ ⟨*proof*⟩

3.2 Weakening

Weakening is admissible. BKK leave this structural property implicit (their contexts are sets and the rules mention the context only via membership and extension); in the formalisation the eigen-parameter condition of $NK(\Pi I)$ makes it a genuine lemma, proved by transposing the eigen-parameter out of the way of the enlarged context.

Transposing two parameters — an involutive, injective renaming — lets us shift an eigen-parameter to a fresh one when weakening the context.

definition *swp* :: *'p* ⇒ *'p* ⇒ *'p* ⇒ *'p* **where**
  swp a b = (λ*x*. *if x* = *a then b else if x* = *b then a else x*)
lemma *swp-inj*: *inj* (*swp a b*) ⟨*proof*⟩
lemma *swp-swp* [*simp*]: *swp a b* (*swp a b x*) = *x* ⟨*proof*⟩
lemma *swp-apply*: *swp a b a* = *b* ⟨*proof*⟩
lemma *prn-swp-swp* [*simp*]: *prn* (*swp a b*) (*prn* (*swp a b*) *t*) = *t* ⟨*proof*⟩
lemma *image-prn-swp-swp* [*simp*]: *prn* (*swp a b*) ` (*prn* (*swp a b*) ` *S*) = *S*
  ⟨*proof*⟩

The parameters used by a context, and the property of leaving infinitely many free. *freep* is our rendering of BKK's *sufficiently* Σ*-pure* (BKK Definition 6.3): since a parameter name may be used at every type, infinitely many unused names provide, for each type, a witness reservoir of the cardinality of the (countable) language.

definition *usedp* :: *'p tm set* ⇒ *'p set* **where** *usedp* Φ ≡ (⋃ *D* ∈ Φ. *pars D*)
definition *freep* :: *'p tm set* ⇒ *bool* **where** *freep* Φ ≡ *infinite* (− *usedp* Φ)
lemma *usedp-insert*: *usedp* (*insert A* Φ) = *pars A* ∪ *usedp* Φ ⟨*proof*⟩
lemma *usedp-prn*: *usedp* (*prn* π ` Φ) = π ` *usedp* Φ ⟨*proof*⟩
lemma *bij-swp*: *bij* (*swp a b*) ⟨*proof*⟩
lemma *infinite-inj-image*: *inj f* ⟹ *infinite A* ⟹ *infinite* (*f* ` *A*)
  ⟨*proof*⟩
lemma *freep-add*: *freep* Φ ⟹ *freep* (*insert A* Φ)
  ⟨*proof*⟩
lemma *freep-fresh*: *freep* Φ ⟹ *finite F* ⟹ ∃ *w*. *w* ∉ *usedp* Φ ∧ *w* ∉ *F*
  ⟨*proof*⟩
lemma *freep-prn*: *freep* Φ ⟹ *freep* (*prn* (*swp a b*) ` Φ)
  ⟨*proof*⟩

lemma *bprov-weaken*: Φ ⊢ *C* ⟹ Φ ⊆ Ψ ⟹ *freep* Ψ ⟹ Ψ ⊢ *C*
⟨*proof*⟩

3.3 Compactness

Every derivation uses only finitely many hypotheses — BKK: "since every NK^*-proof is finite" (used in the proof of BKK Corollary 7.8). With set-based contexts this is again a genuine lemma.

When the parameter type is infinite, every finite context leaves infinitely many parameters free, and every derivation uses only a finite part of its context.

lemma *freep-finite*: **fixes** Φ :: *'p::infinite tm set* **assumes** *finite* Φ
shows *freep* Φ
⟨*proof*⟩

lemma *bprov-finite*: $\Phi \vdash (C :: 'p::\mathit{infinite}\ \mathit{tm}) \Longrightarrow \exists \Phi 0.\ \mathit{finite}\ \Phi 0 \wedge \Phi 0 \subseteq \Phi \wedge \Phi 0 \vdash C$
⟨*proof*⟩

4 Soundness

This section proves *NK* sound for the model class $\mathcal{M}_{\beta fb}$ (BKK Theorem 7.3, including BKK's evaluation-variant argument).

4.1 The canonical construction respects β-conversion (BKK Remark 3.19)

In a Σ-Henkin model, β-convertible terms denote the same object under any total assignment; the abstraction-congruence case uses functionality (property f).

context *general-model*
begin

lemma *beq-den*: $s \approx_\varrho t \Longrightarrow (\forall\, n\ \tau.\ \mathcal{D}_\tau\ (\xi\ n\ \tau)) \Longrightarrow (\!|s|\!)_\xi = (\!|t|\!)_\xi$
⟨*proof*⟩

end

4.2 Abstract soundness (BKK Theorem 7.3)

Soundness over the abstract Σ-models of Section 2, following BKK's proof of Theorem 7.3 case by case. The $NK(\Pi I)$ case uses BKK's device verbatim: "from the evaluation function E, one can define another evaluation function E' such that $E'(w) \equiv a$ and $E'_\varphi(A) \equiv E_\varphi(A)$ if w does not occur in A" — realised below by reading the parameter as a fresh variable after an injective shift of all free variables. The extensionality cases $NK(f)$ and $NK(b)$ rest on BKK's Lemma 4.2 on Leibniz equality, proven here abstractly (BKK

themselves route them through Theorem 4.3(3) and Lemma 3.48/Theorem 4.3(4), which rest on Lemma 4.2).

4.2.1 The variable shift

vshift renames every free variable n to $n + 1$, freeing the name 0 at every type; β-equality and typing are stable under it and under the parameter-to-variable rename *pvar*.

definition *vshift* :: *'p tm* ⇒ *'p tm* **where** *vshift* = *msub* (λ*n* τ. (*Suc n*)$^f{}_\tau$)
lemma *occ-vshift*: *occ* (*vshift t*) = (λ(*n*, τ). (*Suc n*, τ)) ‘ *occ t*
⟨*proof*⟩
lemma *pars-vshift*: *pars* (*vshift t*) = *pars t*
⟨*proof*⟩
lemma *wff-vshift*: *wff*$_\tau$(*t*) ⟹ *wff*$_\tau$(*vshift t*)
⟨*proof*⟩
lemma *vshift-opn*: *vshift* (*t*⟨*x*$^f{}_\sigma$⟩) = (*vshift t*)⟨(*Suc x*)$^f{}_\sigma$⟩
⟨*proof*⟩
lemma *beq-vshift*: *s* ≈$_\varrho$ *t* ⟹ *vshift s* ≈$_\varrho$ *vshift t*
⟨*proof*⟩

lemma *beq-pvar*: *s* ≈$_\varrho$ *t* ⟹ *pvar w* σ*w x s* ≈$_\varrho$ *pvar w* σ*w x t*
⟨*proof*⟩

lemma *occ-pvar*: *occ* (*pvar w* σ*w x t*) ⊆ *occ t* ∪ {(*x*, σ*w*)}
⟨*proof*⟩

4.2.2 Satisfaction of Leibniz equality (BKK Lemma 4.2)

Leibniz equality β-reduces to its $\forall$-form (the syntactic prelude to Lemma 4.2).

lemma *Leib-beq*: **assumes** *wA*: *wff*$_\alpha$(*A*) **and** *wB*: *wff*$_\alpha$(*B*)
shows (*A* �doteq$_\alpha$ *B*) ≈$_\mathrm{o}$ **Π**$_{\alpha \Rightarrow \mathrm{o}}$ ((*Bnd 0* · *A*) ⊃ (*Bnd 0* · *B*))
⟨*proof*⟩

context *sigma-model*
begin

lemma *sat-Leib*: **assumes** *wA*: *wff*$_\alpha$(*A*) **and** *wB*: *wff*$_\alpha$(*B*) **and** *xi*: *asg* ξ
shows *vl* (*Ee* ξ (*A* �doteq$_\alpha$ *B*)) ⟷ *Ee* ξ *A* = *Ee* ξ *B*
⟨*proof*⟩

end

4.2.3 The evaluation variant at a parameter

context *bkk-model*
begin

Agreement under the variable shift: shifting all free variables and the assignment in step leaves denotations unchanged (property f resolves the abstraction case applicatively).

lemma *Ee-vshift*:
assumes ‹$wff_\tau(A)$› ‹*asg* ξ› ‹*asg* ξ'› ‹$\bigwedge n\ \tau'.\ (n, \tau') \in occ\ A \Longrightarrow \xi'\ (Suc\ n)\ \tau' = \xi\ n\ \tau'$›
shows ‹*Ee* ξ' (*vshift* A) = *Ee* ξ A›
⟨*proof*⟩

BKK's E' (the $NK(\Pi I)$ case of Theorem 7.3): the parameter w is read as the freshly freed variable *0*, assigned a.

definition *upshift* :: (*nat* ⇒ *ty* ⇒ 'u) ⇒ *ty* ⇒ 'u ⇒ *nat* ⇒ *ty* ⇒ 'u **where**
upshift ξ σ a = ($\lambda n\ \tau$. *case n of 0* ⇒ (*if* $\tau = \sigma$ *then a else* ξ *0* τ) | *Suc m* ⇒ ξ *m* τ)
definition *Evar* :: 'p ⇒ *ty* ⇒ 'u ⇒ (*nat* ⇒ *ty* ⇒ 'u) ⇒ 'p *tm* ⇒ 'u **where**
Evar w σ *a* ξ *A* = *Ee* (*upshift* ξ σ *a*) (*pvar w* σ *0* (*vshift A*))
lemma *asg-upshift*: *asg* ξ ⟹ *Dm* σ *a* ⟹ *asg* (*upshift* ξ σ *a*)
⟨*proof*⟩
lemma *Evar-par*: *asg* ξ ⟹ *Dm* σ *a* ⟹ *Evar w* σ *a* ξ ($w^p{}_\sigma$) = *a*
⟨*proof*⟩
lemma *Evar-agree*: $wff_\tau(A)$ ⟹ *asg* ξ ⟹ *Dm* σ *a* ⟹ *w* ∉ *pars A* ⟹ *Evar w* σ *a* ξ *A* = *Ee* ξ *A*
⟨*proof*⟩
lemma *Evar-bkk*: **assumes** *a*: *Dm* σ *a* **shows** *bkk-model Dm Ap* (*Evar w* σ *a*) *vl*
⟨*proof*⟩

end

4.2.4 Soundness

BKK Theorem 7.3, for the class $\mathcal{M}_{\beta f\flat}$ (with primitive equality and description): each case mirrors BKK's proof text.

theorem *soundness-bkk*:
assumes Φ ⊢ *C bkk-model Dm Ap Ee vl app-struct.asg Dm* ξ
∀ *A* ∈ Φ. $wff_{\mathrm{o}}(A)$ ∀ *A* ∈ Φ. *vl* (*Ee* ξ *A*)
shows *vl* (*Ee* ξ *C*)
⟨*proof*⟩

4.2.5 The canonical construction is a BKK model

Every Σ-Henkin general model in the sense of Section 2 — the frame-based canonical construction — is a BKK model: take $E := den$ and $\upsilon := (\lambda a.\ a = Tv)$. The four evaluation conditions are the denotation lemmas, and the L-properties are the *gm*-conditions.

sublocale *general-model* ⊆ *bkkA*: *app-struct Dm Ap*
⟨*proof*⟩

sublocale *general-model* $\subseteq$ *bkk*: *bkk-model Dm Ap* $\lambda\xi\ A.\ (\!|A|\!)_\xi$ $\lambda a.\ a = Tv$
⟨*proof*⟩

4.2.6 Derived rules for the defined quantifiers

$NK(\Pi E)$ and $NK(\Pi I)$ for the folded quantifier $\mathbf{\Pi}_\sigma$.

lemma *PiE-Forall*: $\Phi \vdash \mathbf{\Pi}_\alpha\ b \Longrightarrow \mathit{wff}_\alpha(A) \Longrightarrow \Phi \vdash (\mathbf{\Lambda}_\alpha\ b) \cdot A$
⟨*proof*⟩
lemma *PiI-Forall*: $\Phi \vdash (\mathbf{\Lambda}_\alpha\ b) \cdot (w^p{}_\alpha) \Longrightarrow \mathit{wff}_\alpha \Rightarrow {}_\mathrm{o}(\mathbf{\Lambda}_\alpha\ b) \Longrightarrow w \notin \mathit{pars}\ (\mathbf{\Lambda}_\alpha\ b)$
$\Longrightarrow (\forall D \in \Phi.\ w \notin \mathit{pars}\ D) \Longrightarrow \Phi \vdash \mathbf{\Pi}_\alpha\ b$
⟨*proof*⟩

Existential elimination at a fresh eigen-parameter, for the defined quantifier $\exists\ = \neg\mathbf{\Pi}\neg$: the derived counterpart of the paper-style step "obtain a witness".

lemma *ExE*: **assumes** *ex*: $\Phi \vdash \exists_\sigma\ b$ **and** *step*: $\Phi \cup \{b\langle w^p{}_\sigma\rangle\} \vdash C$
and *wb*: $\mathit{wff}_\sigma \Rightarrow {}_\mathrm{o}(\mathbf{\Lambda}_\sigma\ b)$
and *wC*: $\mathit{wff}_\mathrm{o}(C)$ **and** *wpb*: $w \notin \mathit{pars}\ b$ **and** *wpC*: $w \notin \mathit{pars}\ C$
and *wpΦ*: $\forall D \in \Phi.\ w \notin \mathit{pars}\ D$ **and** *fp*: *freep* Φ
shows $\Phi \vdash C$
⟨*proof*⟩

Universal instantiation directly at the β-reduced instance.

lemma *PiE-open*: $\Phi \vdash \mathbf{\Pi}_\alpha\ b \Longrightarrow \mathit{wff}_\alpha \Rightarrow {}_\mathrm{o}(\mathbf{\Lambda}_\alpha\ b) \Longrightarrow \mathit{wff}_\alpha(A) \Longrightarrow \Phi \vdash b\langle A\rangle$
⟨*proof*⟩

Soundness, repackaged in the validity and satisfaction notation: the two forms used by the closing summary.

theorem *soundness-sat*:
$\Phi \vdash C \Longrightarrow \Phi \models('u)\ C$
⟨*proof*⟩

theorem *soundness-valid*: $\vdash A \Longrightarrow \models('u)\ A$
⟨*proof*⟩

5 Completeness

Henkin completeness via the model-existence / abstract-consistency method of BKK Section 6 (Corollary 7.7), with the term model realised as a term evaluation (BKK Definition 3.35) — then strengthened to arbitrary infinite value carriers, signatures with infinitely many parameters, and open formulas.

5.1 Model existence and completeness (BKK Section 6, Corollary 7.7)

We build a Henkin term model from a maximal consistent, saturated set of sentences, following BKK's model-existence route (BKK Section 6). This file develops *NK*-consistency and its closure properties (BKK Lemma 7.5), the maximal saturated extension (BKK Lemma 6.32), the term model with its truth lemma (BKK Theorem 6.33), and the completeness theorem itself (BKK Corollary 7.7).

5.1.1 Consistency (BKK Definition 7.4)

A set of sentences is *NK-consistent* (BKK Definition 7.4) if falsity is not derivable from it.

definition *con* :: *'p tm set* ⇒ *bool* **where** *con* Φ ⟷ ¬ (Φ ⊢ ⊥)
lemma *con-I*: (Φ ⊢ ⊥ ⟹ *False*) ⟹ *con* Φ ⟨*proof*⟩

Subsets of a consistent set are consistent (using weakening).

lemma *con-mono*: *con* Ψ ⟹ Φ ⊆ Ψ ⟹ *freep* Ψ ⟹ *con* Φ ⟨*proof*⟩

Consistency is of finite character (compactness, BKK Definition 6.1): a set is consistent as soon as all its finite subsets are.

lemma *con-compact*:
(⋀Φ0::'p::*infinite tm set*. *finite* Φ0 ⟹ Φ0 ⊆ Φ ⟹ *con* Φ0) ⟹ *con* Φ
⟨*proof*⟩

The central step of a maximal-consistent extension (the ∇_{sat} case of BKK Lemma 7.5; property ∇_{sat} is BKK Definition 6.5): from a consistent set, adding a proposition or its negation keeps it consistent.

lemma *con-split*: **assumes** *con* Φ **and** *wff*$_{\mathrm{o}}$(*A*)
shows *con* (*insert A* Φ) ∨ *con* (*insert* (¬ *A*) Φ)
⟨*proof*⟩

A consistent set does not contain both a proposition and its negation.

lemma *con-not-both*: *con* Φ ⟹ *A* ∈ Φ ⟹ ¬ *A* ∈ Φ ⟹ *wff*$_{\mathrm{o}}$(*A*) ⟹ *False*
⟨*proof*⟩

5.1.2 Some admissible rules

lemma *freep-un*: *freep* Φ ⟹ *freep* (Φ ∪ {*A*})
⟨*proof*⟩

Double-negation elimination (derivable from the classical rule *NK*(*Contr*)).

lemma *dneg*: Φ ⊢ ¬ (*Neg* ⋅ *X*) ⟹ *wff*$_{\mathrm{o}}$(*X*) ⟹ *freep* Φ ⟹ Φ ⊢ *X*
⟨*proof*⟩

Excluded middle and implication introduction (derivable in the classical calculus).

lemma *bprov-em*: **assumes** *wA*: $wff_{\mathrm{o}}(A)$ **shows** $\Phi \vdash (\neg\ A) \vee A$
⟨*proof*⟩
lemma *bprov-ImpE*: **assumes** *AB*: $\Phi \vdash A \supset B$ **and** *A*: $\Phi \vdash A$
and *fp*: *freep* Φ
and *wA*: $wff_{\mathrm{o}}(A)$ **and** *wB*: $wff_{\mathrm{o}}(B)$ **shows** $\Phi \vdash B$
⟨*proof*⟩
lemma *bprov-TrueB*: $\Phi \vdash \top$ ⟨*proof*⟩

5.1.3 Witnessing a false universal (BKK property $\nabla_\exists$, Definition 6.5; the $\nabla_\exists$ case of Lemma 7.5)

If $\neg\Pi_\alpha\ G$ is consistent with Φ, then it stays consistent when we add a witness $\neg\ (G\ c)$ for a fresh parameter *c*. This is the key step for making the extension saturated.

lemma *con-witness*:
assumes *con*: *con* (*insert* ($\neg$ ((*Pi* α) · *G*)) Φ)
and *wG*: $wff_{\alpha \Rightarrow \mathrm{o}}(G)$ **and** *fp*: *freep* Φ
and *c*: *c* $\notin$ *usedp* Φ *c* $\notin$ *pars G*
shows *con* (*insert* ($\neg$ (*G* · ($c^p{}_\alpha$))) (*insert* ($\neg$ ((*Pi* α) · *G*)) Φ))
⟨*proof*⟩

5.1.4 Maximal consistent, saturated extension (BKK's abstract extension lemma 6.32)

We enumerate the (countably many) sentences and, step by step, decide each one or its negation (*con-split*), immediately adding a Henkin witness for a decided negated universal (*con-witness*). The union of the chain is a maximal consistent, saturated set.

definition *freshc* :: *'p tm set* ⇒ *'p tm* ⇒ *'p* **where**
freshc S G ≡ *SOME c*. *c* $\notin$ *usedp S* ∧ *c* $\notin$ *pars G*

lemma *freshc-fresh*: *freep S* ⟹ *freshc S G* $\notin$ *usedp S* ∧ *freshc S G* $\notin$ *pars G*
⟨*proof*⟩

fun *wit* :: *'p tm set* ⇒ *'p tm* ⇒ *'p tm set* **where**
‹*wit S* ($\neg$*App* (*Pi* α) *G*) = {$\neg$ (*G* · ((*freshc S G*)$^p{}_\alpha$))}›
| ‹*wit* - - = {}›

lemma *wit-cases*: *wit S A* = {} ∨ (∃ α *G*. *A* = $\neg$ ((*Pi* α) · *G*) ∧
wit S A = {$\neg$ (*G* · ((*freshc S G*)$^p{}_\alpha$))})
⟨*proof*⟩
lemma *finite-wit*: *finite* (*wit S A*)
⟨*proof*⟩
definition *step* :: *'p tm set* ⇒ *'p tm* ⇒ *'p tm set* **where**
step S A ≡ *if cwff* ∘ *A*
then (*if con* (*insert A S*) *then insert A S* ∪ *wit S A else insert* ($\neg$ *A*) *S*)
else S

primrec *ext* :: *'p*::{*countable*,*infinite*} *tm set* ⇒ *nat* ⇒ *'p tm set* **where**
  ext Φ *0* = Φ
| *ext* Φ (*Suc n*) = *step* (*ext* Φ *n*) (*from-nat n*)

definition *Hset* :: *'p*::{*countable*,*infinite*} *tm set* ⇒ *'p tm set* **where**
  Hset Φ = (⋃ *n*. *ext* Φ *n*)

The chain is increasing and each stage stays finite-in-parameters and consistent.

lemma *ext-mono*: *ext* Φ *n* ⊆ *ext* Φ (*Suc n*)
  ⟨*proof*⟩
lemma *ext-mono'*: *m* ≤ *n* ⟹ *ext* Φ *m* ⊆ *ext* Φ *n*
  ⟨*proof*⟩
lemma *freep-un-finite*: *freep S* ⟹ *finite T* ⟹ *freep* (*S* ∪ *T*)
⟨*proof*⟩
lemma *freep-step*: *freep S* ⟹ *freep* (*step S A*)
  ⟨*proof*⟩
lemma *freep-ext*: *freep* Φ ⟹ *freep* (*ext* Φ *n*)
  ⟨*proof*⟩
lemma *con-step*: **assumes** *con S* **and** *freep S* **shows** *con* (*step S A*)
⟨*proof*⟩
lemma *con-ext*: *con* Φ ⟹ *freep* Φ ⟹ *con* (*ext* Φ *n*)
  ⟨*proof*⟩

Hset Φ extends Φ and, by compactness, is consistent.

lemma *Phi-sub-Hset*: Φ ⊆ *Hset* Φ ⟨*proof*⟩
lemma *finite-sub-ext*: *finite F* ⟹ *F* ⊆ *Hset* Φ ⟹ ∃ *N*. *F* ⊆ *ext* Φ *N*
⟨*proof*⟩
lemma *con-Hset*: **fixes** Φ :: *'p*::{*countable*,*infinite*} *tm set*
  assumes *con* Φ **and** *freep* Φ
  shows *con* (*Hset* Φ)
  ⟨*proof*⟩

Crucially, *freep* (*Hset* Φ) fails (a maximal set uses every parameter), so we never weaken *to Hset*. Instead we use that every *finite* subset of *Hset* is consistent — finite contexts are always *freep*, which is all the proof rules need.

lemma *Hset-finite-con*: **fixes** Φ :: *'p*::{*countable*,*infinite*} *tm set*
  assumes *con* Φ **and** *freep* Φ **and** *finite F* **and** *F* ⊆ *Hset* Φ
  shows *con F*
  ⟨*proof*⟩

Hset decides every sentence — BKK call this *saturated*, property $\sim\nabla_{sat}$ (BKK Definition 6.24) — and has the Henkin witness property $\sim\nabla_{\exists}$ (BKK Definition 6.19). We call the former *maximality* and reserve *saturation* for the witness property; the lemma names below follow this convention.

lemma *Hset-maximal*: **fixes** Φ :: *'p*::{*countable*,*infinite*} *tm set*
  assumes *c*: *cwff* ∘ *A* **shows** *A* ∈ *Hset* Φ ∨ ¬ *A* ∈ *Hset* Φ

⟨*proof*⟩
lemma *Hset-saturated*: **fixes** Φ :: *'p*::{*countable*,*infinite*} *tm set*
assumes *con*Φ: *con* Φ **and** *fp*: *freep* Φ
and *cA*: *cwff* o $(\neg\ ((Pi\ \alpha)\cdot G))$ **and** *inH*: $\neg\ ((Pi\ \alpha)\cdot G) \in Hset\ \Phi$
shows $\exists\, c.\ \neg\ (G\cdot(c^p{}_\alpha)) \in Hset\ \Phi$
⟨*proof*⟩

5.1.5 Leibniz equality is reflexive (BKK property ∇_r, Lemma 6.25)

lemma *leib-refl*: **assumes** *fp*: *freep* Φ **and** *wa*: $wff_\alpha(A)$
shows $\Phi \vdash A \doteq_\alpha A$ ⟨*proof*⟩

Leibniz substitution (the substitutivity built into BKK's Leibniz equality, Section 2.2; cf. the ∇-properties of BKK Lemma 6.12): equals may replace equals in any predicate. Instantiating P with suitable predicates yields symmetry, transitivity and congruence.

lemma *leib-subst*:
assumes *AB*: $\Phi \vdash A \doteq_\alpha B$ **and** *fp*: *freep* Φ **and** *wa*: $wff_\alpha(A)$ **and** *wb*: $wff_\alpha(B)$
and *wP*: $wff_{\alpha\Rightarrow o}(P)$ **and** *PA*: $\Phi \vdash P\cdot A$
shows $\Phi \vdash P\cdot B$
⟨*proof*⟩
lemma *leib-sym*:
assumes *AB*: $\Phi \vdash A \doteq_\alpha B$ **and** *fp*: *freep* Φ **and** *wa*: $wff_\alpha(A)$ **and** *wb*: $wff_\alpha(B)$
shows $\Phi \vdash B \doteq_\alpha A$
⟨*proof*⟩
lemma *leib-trans*:
assumes *AB*: $\Phi \vdash A \doteq_\alpha B$ **and** *BC*: $\Phi \vdash B \doteq_\alpha C$ **and** *fp*: *freep* Φ
and *wa*: $wff_\alpha(A)$ **and** *wb*: $wff_\alpha(B)$ **and** *wc*: $wff_\alpha(C)$
shows $\Phi \vdash A \doteq_\alpha C$
⟨*proof*⟩
lemma *leib-cong2*:
assumes *AA*: $\Phi \vdash A \doteq_\alpha A'$ **and** *fp*: *freep* Φ
and *wa*: $wff_\alpha(A)$ **and** *wa'*: $wff_\alpha(A')$ **and** *wC*: $wff_{\alpha\Rightarrow\beta}(C)$
shows $\Phi \vdash (C\cdot A) \doteq_\beta (C\cdot A')$
⟨*proof*⟩
lemma *leib-cong1*:
assumes *CC*: $\Phi \vdash C \doteq_{\alpha\Rightarrow\beta} C'$ **and** *fp*: *freep* Φ
and *wC*: $wff_{\alpha\Rightarrow\beta}(C)$ **and** *wC'*: $wff_{\alpha\Rightarrow\beta}(C')$ **and** *wa*: $wff_\alpha(A)$
shows $\Phi \vdash (C\cdot A) \doteq_\beta (C'\cdot A)$
⟨*proof*⟩

5.1.6 Deductive closure of the Hintikka set

Since *Hset* is maximal and each of its finite subsets is consistent, every sentence provable from a finite subset already belongs to *Hset* (deductive closure — a consequence of the maximality of the extension, BKK Lemma 6.32).

lemma *Hset-deduct*: **fixes** Φ :: *'p*::{*countable*,*infinite*} *tm set*
assumes *con*Φ: *con* Φ **and** *fp*Φ: *freep* Φ
and *finF*: *finite* F **and** *subF*: $F \subseteq$ *Hset* Φ **and** *FS*: $F \vdash S$
and *cS*: *cwff* o S **shows** $S \in$ *Hset* Φ
⟨*proof*⟩

In particular, Leibniz equality is reflexive on *Hset* — BKK's property ∇_r for saturated Hintikka sets (Lemma 6.25; Lemma 6.23 gives the negative form $\sim\nabla_{=r}$); symmetry, transitivity and congruence follow below.

lemma *Hset-leib-refl*: **fixes** Φ :: *'p*::{*countable*,*infinite*} *tm set*
assumes *con*Φ: *con* Φ **and** *fp*Φ: *freep* Φ **and** *ca*: *cwff* α A
shows $(A \doteq_\alpha A) \in$ *Hset* Φ
⟨*proof*⟩
lemma *Hset-leib-sym*: **fixes** Φ :: *'p*::{*countable*,*infinite*} *tm set*
assumes *con*Φ: *con* Φ **and** *fp*Φ: *freep* Φ
and *ca*: *cwff* α A **and** *cb*: *cwff* α B **and** *AB*: $(A \doteq_\alpha B) \in$ *Hset* Φ
shows $(B \doteq_\alpha A) \in$ *Hset* Φ
⟨*proof*⟩
lemma *Hset-leib-trans*: **fixes** Φ :: *'p*::{*countable*,*infinite*} *tm set*
assumes *con*Φ: *con* Φ **and** *fp*Φ: *freep* Φ
and *ca*: *cwff* α A **and** *wb*: *wff*$_\alpha(B)$ **and** *cc*: *cwff* α C
and *AB*: $(A \doteq_\alpha B) \in$ *Hset* Φ **and** *BC*: $(B \doteq_\alpha C) \in$ *Hset* Φ
shows $(A \doteq_\alpha C) \in$ *Hset* Φ
⟨*proof*⟩
lemma *Hset-leib-cong*: **fixes** Φ :: *'p*::{*countable*,*infinite*} *tm set*
assumes *con*Φ: *con* Φ **and** *fp*Φ: *freep* Φ
and *cc*: *cwff* $(\alpha{\Rightarrow}\beta)$ C **and** *cc'*: *cwff* $(\alpha{\Rightarrow}\beta)$ C' **and** *ca*: *cwff* α A
and *ca'*: *cwff* α A'
and *CC*: $(C \doteq_{\alpha\Rightarrow\beta} C') \in$ *Hset* Φ **and** *AA*: $(A \doteq_\alpha A') \in$ *Hset* Φ
shows $((C \cdot A) \doteq_\beta (C' \cdot A')) \in$ *Hset* Φ
⟨*proof*⟩

Leibniz-equal propositions have the same truth (membership transfers along $\sim$ at type o); proved by Leibniz substitution with the identity predicate.

lemma *Hset-leib-mp*: **fixes** Φ :: *'p*::{*countable*,*infinite*} *tm set*
assumes *con*Φ: *con* Φ **and** *fp*Φ: *freep* Φ
and *ca*: *cwff* o A **and** *cb*: *cwff* o B **and** *AB*: $(A \doteq_o B) \in$ *Hset* Φ
and *A*: $A \in$ *Hset* Φ
shows $B \in$ *Hset* Φ
⟨*proof*⟩

5.2 The term model (BKK Section 6, the Hintikka lemma)

Fix a consistent, parameter-rich set Φ; its Hintikka extension H is maximal, consistent and saturated. The *term model* has as its domain the closed well-formed terms quotiented by provable Leibniz equality $A \sim B \equiv (A \doteq B) \in H$; this quotient is what forces property q.

locale *hintikka-model* = **fixes** Φ :: *'p*::{*countable*,*infinite*} *tm set*
  assumes *con*Φ: *con* Φ **and** *fp*Φ: *freep* Φ
begin

The $\sim$-equivalence classes of closed well-formed terms (*cwff* from Section 2): $A \sim B \equiv (A \doteq_\sigma B) \in H$ — the Leibniz quotient of BKK's model-existence proof (BKK Theorem 6.33); the $\sim\nabla$-properties of Leibniz equality in H are BKK Lemma 6.23.

definition *cls* :: *'p tm* $\Rightarrow$ *'p tm set* **where**
  cls A $\equiv$ $\{B.\ \exists\sigma.\ cwff\ \sigma\ A \wedge cwff\ \sigma\ B \wedge (A \doteq_\sigma B) \in Hset\ \Phi\}$
lemma *cls-self*: *cwff* σ *A* $\Longrightarrow$ *A* $\in$ *cls A*
  ⟨*proof*⟩
lemma *leib-sym'*:
  cwff σ *A* $\Longrightarrow$ *cwff* σ *B* $\Longrightarrow$ $(A \doteq_\sigma B) \in Hset\ \Phi \Longrightarrow (B \doteq_\sigma A) \in Hset\ \Phi$
  ⟨*proof*⟩
lemma *leib-trans'*:
  cwff σ *A* $\Longrightarrow$ *cwff* σ *B* $\Longrightarrow$ *cwff* σ *C* $\Longrightarrow$ $(A \doteq_\sigma B) \in Hset\ \Phi$
    $\Longrightarrow (B \doteq_\sigma C) \in Hset\ \Phi \Longrightarrow (A \doteq_\sigma C) \in Hset\ \Phi$
  ⟨*proof*⟩

The quotient is faithful: two classes coincide exactly when the terms are Leibniz-equal in H. This is what will give property q in the model.

lemma *cls-eq-iff*:
  assumes *wA*: *cwff* σ *A* **and** *wB*: *cwff* σ *B*
    shows (*cls A* = *cls B*) $\longleftrightarrow$ $(A \doteq_\sigma B) \in Hset\ \Phi$
  ⟨*proof*⟩

lemma *cls-rep*:
  assumes *cwff* σ *A*
  shows *cwff* σ (*SOME B. B* $\in$ *cls A*) $\wedge$ $(A \doteq_\sigma (SOME\ B.\ B \in cls\ A)) \in Hset\ \Phi$
⟨*proof*⟩

The applicative structure of the term model (BKK Definition 3.1): the domain of type σ consists of the classes of closed terms of type σ, and application is term application.

definition *Dm* :: *ty* $\Rightarrow$ *'p tm set* $\Rightarrow$ *bool* **where** *Dm* σ *X* $\equiv \exists A.\ cwff\ \sigma\ A \wedge X =$ *cls A*
definition *Ap* :: *'p tm set* $\Rightarrow$ *'p tm set* $\Rightarrow$ *'p tm set* **where**
  Ap X Y $\equiv$ *cls* ((*SOME A. A* $\in$ *X*) $\cdot$ (*SOME B. B* $\in$ *Y*))
lemma *Ap-cls*:
  assumes *wA*: *cwff* $(\alpha \Rightarrow \beta)$ *A* **and** *wB*: *cwff* α *B*
    shows *Ap* (*cls A*) (*cls B*) = *cls* (*A* $\cdot$ *B*)
⟨*proof*⟩
lemma *Dm-cls*: *cwff* σ *A* $\Longrightarrow$ *Dm* σ (*cls A*) ⟨*proof*⟩
lemma *Ap-dom*: *Dm* $(\alpha \Rightarrow \beta)$ *X* $\Longrightarrow$ *Dm* α *Y* $\Longrightarrow$ *Dm* β (*Ap X Y*)
  ⟨*proof*⟩

Truth and falsity behave (the analogue of BKK Lemma 3.43 / property b): $\top \in H$, $\bot \notin H$, so *cls* $\top \neq$ *cls* $\bot$.

lemma *Hset-TrueB*: $\top \in \mathit{Hset}\ \Phi$
⟨*proof*⟩
lemma *Hset-not-FalseB*: $\bot \notin \mathit{Hset}\ \Phi$
⟨*proof*⟩
lemma *TF*: $\mathit{cls}\ \top \neq \mathit{cls}\ \bot$
⟨*proof*⟩

5.2.1 Satisfaction bridges: membership in H as truth (BKK Lemma 6.21, Lemmas 6.25–6.26)

H never contains both φ and $\neg\varphi$ (BKK's property $\sim\nabla_c$, Definition 6.19, here for arbitrary sentences via Lemma 6.10).

lemma *Hset-notboth*: $\mathit{wff}_{\mathrm{o}}(\varphi) \Longrightarrow \varphi \in \mathit{Hset}\ \Phi \Longrightarrow \neg\ \varphi \notin \mathit{Hset}\ \Phi$
⟨*proof*⟩

Boolean extensionality inside H (via the rule $NK(b)$; the saturated-sets lemma for property b, BKK Lemma 6.26): a member of H is Leibniz-equal to $\top$, a non-member to $\bot$.

lemma *Hset-eq-TrueB*:
assumes *p*: $\varphi \in \mathit{Hset}\ \Phi$ **and** *w*: $\mathit{cwff}\ \mathrm{o}\ \varphi$
shows $(\varphi \doteq_{\mathrm{o}} \top) \in \mathit{Hset}\ \Phi$
⟨*proof*⟩

lemma *Hset-eq-FalseB*: **assumes** *np*: $\neg\ \varphi \in \mathit{Hset}\ \Phi$ **and** *w*: $\mathit{cwff}\ \mathrm{o}\ \varphi$
shows $(\varphi \doteq_{\mathrm{o}} \bot) \in \mathit{Hset}\ \Phi$
⟨*proof*⟩

Sentences with the same truth value in H are Leibniz-equal in H (the general form of the $NK(b)$ bridge, cf. BKK Lemma 6.26).

lemma *Hset-iff-eq*:
assumes *wp*: $\mathit{cwff}\ \mathrm{o}\ \varphi$ **and** *wq*: $\mathit{cwff}\ \mathrm{o}\ \psi$
and *iff*: $\varphi \in \mathit{Hset}\ \Phi \longleftrightarrow \psi \in \mathit{Hset}\ \Phi$
shows $(\varphi \doteq_{\mathrm{o}} \psi) \in \mathit{Hset}\ \Phi$
⟨*proof*⟩

THE satisfaction bridge: a sentence's class is the class of $\top$ exactly when it is in H (the v-valuation of the term model in the proof of BKK Theorem 6.33).

lemma *cls-TrueB-iff*: $\mathit{cwff}\ \mathrm{o}\ \varphi \Longrightarrow \mathit{cls}\ \varphi = \mathit{cls}\ \top \longleftrightarrow \varphi \in \mathit{Hset}\ \Phi$
⟨*proof*⟩

Property b at the class level: the boolean domain has exactly the classes of $\top$ and $\bot$ (BKK Definition 3.46).

lemma *cls-bool*: $\mathit{cwff}\ \mathrm{o}\ \varphi \Longrightarrow \mathit{cls}\ \varphi = \mathit{cls}\ \top \vee \mathit{cls}\ \varphi = \mathit{cls}\ \bot$
⟨*proof*⟩
lemma *cls-FalseB-iff*: $\mathit{cwff}\ \mathrm{o}\ \varphi \Longrightarrow \mathit{cls}\ \varphi = \mathit{cls}\ \bot \longleftrightarrow \varphi \notin \mathit{Hset}\ \Phi$
⟨*proof*⟩

5.2.2 The evaluation and logical conditions of the term model

The β-condition (BKK Definition 3.18(4)) inside H: a β-redex is Leibniz-equal to its reduct, via the rule $NK(\beta)$ applied to reflexivity.

lemma *Hset-beta*:
assumes *wAbs*: *cwff* $(\sigma \Rightarrow \tau)$ $(\mathbf{\Lambda}_\sigma\ b)$ **and** *wa*: *cwff* $\sigma\ a$
shows $((\mathbf{\Lambda}_\sigma\ b) \cdot a \doteq_\tau b\langle a \rangle) \in \mathit{Hset}\ \Phi$
⟨*proof*⟩

The $L_\neg$ condition (BKK Figure 2) at the class level, from maximality and consistency of H.

lemma *cls-neg*: *cwff* o $\varphi \Longrightarrow \mathit{cls}\ (\neg\ \varphi) = (\mathit{if}\ \mathit{cls}\ \varphi = \mathit{cls}\ \top\ \mathit{then}\ \mathit{cls}\ \bot\ \mathit{else}\ \mathit{cls}\ \top)$
⟨*proof*⟩

The $L_\vee$ condition: H treats disjunction disjunctively (BKK $\nabla_\vee$, Definition 6.19).

lemma *Hset-dis-iff*:
assumes *wp*: *cwff* o φ **and** *wq*: *cwff* o ψ
shows $\varphi \vee \psi \in \mathit{Hset}\ \Phi \longleftrightarrow \varphi \in \mathit{Hset}\ \Phi \vee \psi \in \mathit{Hset}\ \Phi$
⟨*proof*⟩

lemma *cls-dis*:
assumes *wp*: *cwff* o φ **and** *wq*: *cwff* o ψ
shows $\mathit{cls}\ (\varphi \vee \psi) = (\mathit{if}\ \mathit{cls}\ \varphi = \mathit{cls}\ \top \vee \mathit{cls}\ \psi = \mathit{cls}\ \top\ \mathit{then}\ \mathit{cls}\ \top\ \mathit{else}\ \mathit{cls}\ \bot)$
⟨*proof*⟩

The $L^\sigma{}_\forall$ condition: H treats the quantifier universally — $NK(\Pi E)$ gives one direction, the witness property $\sim\nabla_\exists$ (BKK Definition 6.19) together with maximality the other.

lemma *Hset-pi-iff*:
assumes *wf*: *cwff* $(\sigma \Rightarrow \mathrm{o})$ *f*
shows $(\mathit{Pi}\ \sigma) \cdot f \in \mathit{Hset}\ \Phi \longleftrightarrow (\forall\, a.\ \mathit{cwff}\ \sigma\ a \longrightarrow f \cdot a \in \mathit{Hset}\ \Phi)$
⟨*proof*⟩

lemma *cls-pi*:
assumes *wf*: *cwff* $(\sigma \Rightarrow \mathrm{o})$ *f*
shows $\mathit{cls}\ ((\mathit{Pi}\ \sigma) \cdot f) = (\mathit{if}\ (\forall\, a.\ \mathit{cwff}\ \sigma\ a \longrightarrow \mathit{Ap}\ (\mathit{cls}\ f)\ (\mathit{cls}\ a) = \mathit{cls}\ \top)$
$\mathit{then}\ \mathit{cls}\ \top\ \mathit{else}\ \mathit{cls}\ \bot)$
⟨*proof*⟩

The β-condition transfers membership: a redex of type o is in H exactly when its reduct is (via the Leibniz bridge).

lemma *Hset-beta-iff*:
assumes *cwff* $(\sigma \Rightarrow \mathrm{o})$ $(\mathbf{\Lambda}_\sigma\ b)$ **and** *cwff* $\sigma\ a$
shows $(\mathbf{\Lambda}_\sigma\ b) \cdot a \in \mathit{Hset}\ \Phi \longleftrightarrow b\langle a \rangle \in \mathit{Hset}\ \Phi$
⟨*proof*⟩

Type uniqueness of the classes (BKK: the domains are disjoint by type).

lemma *cls-type*:
cwff σ $s \Longrightarrow$ *cwff* τ $t \Longrightarrow$ *cls* s = *cls* $t \Longrightarrow \sigma = \tau$
⟨*proof*⟩

Property q (BKK Definition 3.46): the Leibniz combinator itself is the identity relation of the term model, by ⟦*cwff* *?σ* *?A*; *cwff* *?σ* *?B*⟧ ⟹ (*cls* *?A* = *cls* *?B*) = (*?A* $\doteq_{?\sigma}$ *?B* ∈ *Hset* Φ).

lemma *cwff-Leib*: *cwff* (σ⇒σ⇒o) (*Leib* σ) ⟨*proof*⟩

Property f (functionality, BKK Definition 3.46) via the rule $NK(f)$: two functions that agree on every class are Leibniz-equal in H. Saturation enters through *cwff* (*?σ* ⇒ o) *?f* ⟹ (*Pi* *?σ* · *?f* ∈ *Hset* Φ) = (∀ *a*. *cwff* *?σ* *a* ⟶ *?f* · *a* ∈ *Hset* Φ) to establish the Π-premise of $NK(f)$.

lemma *cls-ext*:
assumes *wg*: *cwff* (σ⇒τ) *g* **and** *wh*: *cwff* (σ⇒τ) *h*
and *ag*: ⋀*a*. *cwff* σ *a* ⟹ *Ap* (*cls g*) (*cls a*) = *Ap* (*cls h*) (*cls a*)
shows *cls g* = *cls h*
⟨*proof*⟩

The description condition of the term model (beyond BKK; cf. Andrews 1972): a class that provably behaves as the singleton of a is mapped by *Iota* σ to the class of a. Functional extensionality $NK(f)$ reduces the pointwise hypothesis to a Leibniz equation with the literal singleton (*Leib* σ) · a, which the axiom $NK(\iota)$ describes.

lemma *cls-desc*:
assumes *wf*: *cwff* (σ⇒o) *f* **and** *wa*: *cwff* σ *a*
and *sing*: ⋀*b*. *cwff* σ *b* ⟹ (*Ap* (*cls f*) (*cls b*) = *cls* ⊤) = (*cls b* = *cls a*)
shows *cls* ((*Iota* σ) · *f*) = *cls a*
⟨*proof*⟩

Primitive equality in H (BKK Remark 7.9): reflexivity is $\nabla_{=_r}$ via $NK(=_r)$, and primitive and Leibniz equality coincide in H — $\rightarrow$ via $NK(=_l)$ ($\nabla_{=_{\doteq}}$), $\leftarrow$ by Leibniz substitution into $\lambda x.\ a =_\sigma x$ from reflexivity.

lemma *Hset-peq-iff*:
assumes *wa*: *cwff* σ *a* **and** *wb*: *cwff* σ *b*
shows (*a* =σ *b*) ∈ *Hset* Φ ⟷ (*a* $\doteq_\sigma$ *b*) ∈ *Hset* Φ
⟨*proof*⟩

end

5.2.3 Model existence (BKK Lemma 7.5 / Theorem 7.6)

The Hintikka set induces a term evaluation: the class map *cls* together with the term-model domains and application satisfies every *valuation* condition. Via the bridge *valuation* ⊆ *bkk-model* of Section 2, every NK-consistent set of sentences therefore has a model in the class $\mathcal{M}_{\beta f b}$ — BKK's model-existence theorem.

sublocale *hintikka-model* $\subseteq$ *V*: *valuation Dm Ap cls*
⟨*proof*⟩

5.2.4 The truth lemma

context *hintikka-model*
begin

THE TRUTH LEMMA (the satisfaction claim of BKK Theorem 6.33; the underlying Hintikka properties are BKK Lemma 6.21): under any domain-respecting assignment a sentence denotes its own class, and it denotes the truth value *cls* $\top$ of the term model exactly when it belongs to *H*.

theorem *truth-lemma*:
cwff o $\varphi \Longrightarrow$ *V.vresp* $\xi \Longrightarrow$ *V.Ev* ξ φ = *cls* $\top \longleftrightarrow \varphi \in$ *Hset* Φ
⟨*proof*⟩

end

5.2.5 Completeness (BKK Corollary 7.7)

$NK_{\beta f b}$ with $NK(\iota)$ is complete for the class $\mathcal{M}_{\beta f b}$ of Σ-models with description (the description-enriched $\mathcal{M}_{\beta f b}$, cf. Andrews 1972): a sentence that is valid in every such model of a sufficiently Σ-pure (*freep*, BKK Definition 6.3) set of sentences Φ (it suffices to assume validity over the models carried by $'p$ *tm set* — in particular the term model) is derivable from Φ. Unlike BKK, who allow signatures of any infinite cardinality $\aleph_s$ (BKK Remark 3.16), we fix a countable type of parameter names ($'p$ :: {*countable*, *infinite*}); the enumeration of sentences in the extension lemma rests on it. The proof is by contraposition, BKK's argument: if $\Phi \vdash A$ fails then $\Phi \cup \{\neg A\}$ is *NK*-consistent (*NK*(*Contr*)), extends to a maximal saturated set (BKK's abstract extension lemma 6.32), and its term model — a general model by model existence — satisfies Φ but refutes *A*, contradicting validity.

lemma *fp0*: *freep* ({} :: $'p$::{*countable*,*infinite*} *tm set*)
⟨*proof*⟩

theorem *completeness*:
fixes Φ :: $'p$::{*countable*,*infinite*} *tm set* **and** *A* :: $'p$ *tm*
assumes *c*: *cwff* o *A* **and** *fp*: *freep* Φ
and *valid*: $\models$($'p$ *tm set*) *A*
and *sen*: $\bigwedge B.\ B \in \Phi \Longrightarrow$ *cwff* o *B*
shows $\Phi \vdash A$
⟨*proof*⟩

Soundness (BKK Theorem 7.3) and completeness (BKK Corollary 7.7) are statements about the *same* class of models, so together they characterise derivability semantically: a sentence is derivable from Φ exactly when it holds in every model of Φ in the class $\mathcal{M}_{\beta f b}$ over the term carrier.

The semantic characterisation of derivability from the empty hypothesis set (BKK Corollary 7.7 at $\Phi = \{\}$, combined with Theorem 7.3): a sentence is derivable iff it is valid in every Σ-model of the class $\mathcal{M}_{\beta f b}$ (over the carrier of the term model; the general hypothesis-set form is *completeness*). The soundness direction is *soundness-bkk*; the completeness direction only shrinks the quantification to the canonical models via the sublocale bridge.

theorem *derivable-iff-valid*:
fixes $A :: 'p::\{countable,infinite\}\ tm$
assumes *cwff* o A
shows $\vdash A \longleftrightarrow \models('p\ tm\ set)\ A$
⟨*proof*⟩

5.2.6 Further derived Leibniz rules

Leibniz modus ponens: transport a theorem along a proven Leibniz equation (via $NK(\beta)$ and Leibniz substitution into the identity predicate).

lemma *leib-mp*:
assumes *AB*: $\Phi \vdash A \doteq_o B$ **and** *A*: $\Phi \vdash A$ **and** *fp*: *freep* Φ
and *wA*: $wff_o(A)$ **and** *wB*: $wff_o(B)$
shows $\Phi \vdash B$
⟨*proof*⟩

From Leibniz to primitive equality (BKK Remark 7.9; by Leibniz substitution into $\mathbf{\Lambda}x.\ A =_\alpha x$ from $NK(=_r)$-reflexivity; the converse is the rule $NK(=_l)$).

lemma *leib-to-peq*:
assumes *AB*: $\Phi \vdash A \doteq_\alpha B$ **and** *fp*: *freep* Φ
and *wA*: $wff_\alpha(A)$ **and** *wB*: $wff_\alpha(B)$
shows $\Phi \vdash A =_\alpha B$
⟨*proof*⟩

The diagonal contradiction: a proposition Leibniz-equal to its own negation refutes the context (by excluded middle and Leibniz modus ponens).

lemma *leib-neg-contra*:
assumes *E*: $\Phi \vdash A \doteq_o (\neg A)$ **and** *fp*: *freep* Φ **and** *wA*: $wff_o(A)$
shows $\Phi \vdash \bot$
⟨*proof*⟩

Application of a primitive equation, and β-reduction on the right of $\doteq$.

lemma *peq-app*: $\Phi \vdash C =_{\alpha \Rightarrow \beta} D \Longrightarrow$ *freep* $\Phi \Longrightarrow wff_{\alpha \Rightarrow \beta}(C) \Longrightarrow wff_{\alpha \Rightarrow \beta}(D)$
$\Longrightarrow wff_\alpha(A) \Longrightarrow \Phi \vdash (C \cdot A) \doteq_\beta (D \cdot A)$
⟨*proof*⟩
lemma *leib-reduce-right*: $\Phi \vdash A \doteq_o B \Longrightarrow B \approx_o B' \Longrightarrow wff_o(A) \Longrightarrow \Phi \vdash A \doteq_o B'$
⟨*proof*⟩

The same three steps for *primitive* equality (via $NK(=_l)$ in and *leib-to-peq* out): application to an argument, β-reduction on the right, and the diagonal contradiction $A = \neg\ A \Longrightarrow \bot$.

5.3 Completeness at every signature and carrier

This part strengthens the completeness theorem from the term-model carrier to *arbitrary* infinite value carriers, by an explicit model-embedding construction: every Σ-model of the class $\mathcal{M}_{\beta f b}$ whose total domain embeds injectively into a carrier $'u$ has an isomorphic copy on $'u$, and satisfaction is invariant under the embedding. Since the term model has a countable total domain (the language is countable), it embeds into every infinite carrier, so validity over any infinite carrier suffices for derivability. Completeness is then extended further, to open formulas and to signatures with infinitely many parameters, and the development closes with the main theorems stated in self-contained notation.

5.3.1 A countermodel for every underivable sentence

The completeness construction, packaged as an explicit countermodel: if a sentence is not derivable, the Hintikka term model refutes it. The total domain of the term model is countable — the domains are $\sim$-classes of closed wffs.

lemma *refuting-term-model*:
fixes $A :: 'p::\{countable,infinite\}\ tm$
assumes *c*: *cwff* ∘ A **and** *nd*: $\neg\vdash A$
obtains *Dm Ap* **and** $Ee :: (nat \Rightarrow ty \Rightarrow 'p\ tm\ set) \Rightarrow 'p\ tm \Rightarrow 'p\ tm\ set$ **and** *vl* ξ
where *bkk-model Dm Ap Ee vl countable* $\{x.\ \exists\tau.\ Dm\ \tau\ x\}$
app-struct.asg Dm ξ $\neg$ *vl* (*Ee* ξ A)
⟨*proof*⟩

5.3.2 Model embedding: satisfaction is carrier-independent

A Σ-model over a carrier $'v$ whose total domain maps injectively into a carrier $'u$ has an isomorphic copy over $'u$; every refutation transfers. All model conditions are pointwise, so the transfer needs no induction on terms.

lemma *bkk-model-embed*:
fixes $Dm :: ty \Rightarrow 'v \Rightarrow bool$ **and** $i :: 'v \Rightarrow 'u$
and $A :: 'p\ tm$
assumes *M*: *bkk-model Dm Ap Ee vl*
and *inj*: *inj-on i* $\{x.\ \exists\tau.\ Dm\ \tau\ x\}$
and *wA*: $wff_o(A)$
and *xi*: *app-struct.asg Dm* ξ **and** *nA*: $\neg$ *vl* (*Ee* ξ A)
obtains $Dm'\ Ap'$ **and** $Ee' :: (nat \Rightarrow ty \Rightarrow 'u) \Rightarrow 'p\ tm \Rightarrow 'u$ **and** $vl'\ \xi'$

where *bkk-model Dm′ Ap′ Ee′ vl′ app-struct.asg Dm′ ξ′ ¬ vl′ (Ee′ ξ′ A)*
⟨*proof*⟩

5.3.3 Completeness at every infinite carrier

The strengthened form of BKK Corollary 7.7: validity over the models of $\mathcal{M}_{\beta f b}$ at *any* infinite value carrier implies derivability. The term carrier plays no special role: it only needs to embed into the given carrier, which its countability guarantees.

theorem *completeness-at-any-carrier*:
fixes *A* :: *'p*::{*countable*,*infinite*} *tm*
assumes *c*: *cwff* o *A*
and *valid*: ⊨(*'u*::*infinite*) *A*
shows ⊢ *A*
⟨*proof*⟩

5.3.4 Completeness for open formulas

Completeness does not require closed sentences: assignments interpret the free variables. The bridge is a substitution-value law for *abstract* Σ-evaluations (the abstract form of BKK Lemma 3.20, one variable at a time), proved without any term induction: substitution is expressed through closing, β-conversion and the evaluation conditions. On the syntactic side a free variable is generalised through a fresh parameter by the rule chain $NK(\beta)$–$NK(\Pi I)$–$NK(\Pi E)$–$NK(\beta)$.

lemma *opn-clos-sub*: *opn k v t* = *t* ⟹ *opn k v* (*clos k x σ t*) = *fsub x σ v t*
⟨*proof*⟩

lemma *fsub-id*: *fsub x σ* (*x*^{f} *σ*) *t* = *t*
⟨*proof*⟩

lemma *occ-clos*: (*x*, *σ*) ∉ *occ* (*clos k x σ t*)
⟨*proof*⟩

lemma *pars-clos* [*simp*]: *pars* (*clos k x σ t*) = *pars t*
⟨*proof*⟩

lemma *finite-occ*: *finite* (*occ t*)
⟨*proof*⟩

lemma *occ-fsub-closed*: *occ u* = {} ⟹ *occ* (*fsub x σ u t*) = *occ t* − {(*x*, *σ*)}
⟨*proof*⟩

The sharpened β-application law: the fresh-name condition only concerns the typed occurrence (*x*, *σ*), not the bare name (a name may occur at several types).

lemma (**in** *sigma-eval*) *ev-abs-app-occ*:
assumes *wb*: *wff*$_{\sigma\Rightarrow\tau}$(**Λ**$_\sigma$ *b*) **and** *xi*: *asg ξ*

and x: $(x, \sigma) \notin occ\ b$ **and** d: $Dm\ \sigma\ d$
shows $Ap\ (Ee\ \xi\ (\mathbf{\Lambda}_\sigma\ b))\ d = Ee\ (\xi(x_\sigma := d))\ (b\langle x^f{}_\sigma\rangle)$
⟨*proof*⟩

The substitution-value law for abstract Σ-evaluations (BKK Lemma 3.20 for a single free variable): substituting *any* well-formed term equals updating the assignment with its value.

lemma (**in** *sigma-eval*) *ev-fsub-one*:
assumes wA: $wff_\tau(A)$ **and** wu: $wff_\sigma(u)$ **and** xi: $asg\ \xi$
shows $Ee\ \xi\ (fsub\ x\ \sigma\ u\ A) = Ee\ (\xi(x_\sigma := Ee\ \xi\ u))\ A$
⟨*proof*⟩

Syntactic generalisation: a fresh parameter substituted for a free variable can be quantified away and re-instantiated, recovering the open formula.

lemma *bprov-generalize-par*:
assumes wA: $wff_{\mathrm{O}}(A)$ **and** p: $p \notin pars\ A$
and d: $\vdash fsub\ x\ \sigma\ (p^p{}_\sigma)\ A$
shows $\vdash A$
⟨*proof*⟩

Completeness for arbitrary (locally closed) well-formed formulas, by induction on the number of typed free occurrences: each occurrence is generalised through a fresh parameter, semantically justified by the substitution-value law.

lemma *completeness-open-aux*:
fixes $A :: 'p::\{countable,infinite\}\ tm$
shows $card\ (occ\ A) \leq n \Longrightarrow wff_{\mathrm{O}}(A) \Longrightarrow (\models('p\ tm\ set)\ A) \Longrightarrow \vdash A$
⟨*proof*⟩

theorem *completeness-open*:
fixes $A :: 'p::\{countable,infinite\}\ tm$
assumes $wff_{\mathrm{O}}(A)$
and $\models('p\ tm\ set)\ A$
shows $\vdash A$
⟨*proof*⟩

lemma *completeness-open-any-aux*:
fixes $A :: 'p::\{countable,infinite\}\ tm$
shows $card\ (occ\ A) \leq n \Longrightarrow wff_{\mathrm{O}}(A) \Longrightarrow \models('u::infinite)\ A \Longrightarrow \vdash A$
⟨*proof*⟩

theorem *completeness-open-at-any-carrier*:
fixes $A :: 'p::\{countable,infinite\}\ tm$
assumes $wff_{\mathrm{O}}(A)$
and $\models('u::infinite)\ A$
shows $\vdash A$
⟨*proof*⟩

5.3.5 Signature transport

On the semantic side, maps of parameter names need *no* injectivity: any $h :: 'p \Rightarrow 'q$ turns a model for $'q$ into a model for $'p$ by evaluating through *prn h* — the value conditions $vl_\neg, \ldots, vl_\iota$ only inspect *Ee* at the logical constants, which *prn* fixes.

lemma *prn-occ* [*simp*]: *occ* (*prn f t*) = *occ t*
⟨*proof*⟩

lemma *bkk-model-reduct*:
fixes *Ee* :: (*nat* ⇒ *ty* ⇒ *'u*) ⇒ *'q tm* ⇒ *'u* **and** *h* :: *'p* ⇒ *'q*
assumes *bkk-model Dm Ap Ee vl*
shows *bkk-model Dm Ap* (λξ *t*. *Ee* ξ (*prn h t*)) *vl*
⟨*proof*⟩

lemma *bkk-valid-map*:
fixes *h* :: *'p* ⇒ *'q*
assumes *v*: ⊨(*'u*) (*A* :: *'p tm*)
shows ⊨(*'u*) (*prn h A* :: *'q tm*)
⟨*proof*⟩

Completeness for every signature with infinitely many parameters: the finitely many parameters of A are relocated into a copy of $\mathbb{N}$ inside $'p$, completeness over $\mathbb{N}$ applies, and *bprov-rename* transports the derivation back. (With only finitely many parameters this route is barred: $NK(\Pi I)$ consumes fresh eigen-parameters, and an injection $\mathbb{N} \Rightarrow 'p$ is exactly what supplies them.)

theorem *completeness-at-any-signature*:
fixes *A* :: *'p*::*infinite tm*
assumes *wA*: $wff_{\mathrm{o}}(A)$ **and** *v*: ⊨(*'u*::*infinite*) *A*
shows ⊢ *A*
⟨*proof*⟩

6 Consistency

Consistency, in the typical variants: a concrete Σ-standard model over finite domains is exhibited (so the model class $\mathcal{M}_{\beta fb}$ is non-empty), whence by soundness *NK* does not derive ⊥, and no sentence is derivable together with its negation.

6.1 A concrete Σ-standard model over finite domains

The carrier: an individual, the two truth values, and functions as finite graphs. With a one-element domain of individuals every domain D_τ is finite, so the full function spaces of BKK Definition 3.5 are representable by graphs; *en* τ enumerates D_τ without repetition.

datatype *mval* = *MI* | *MB bool* | *MF* (*mval* × *mval*) *list*

fun *en* :: *ty* ⇒ *mval list* **where**
  en ι = [*MI*]
| *en* o = [*MB True*, *MB False*]
| *en* (σ ⇒ τ) = *map* (λ*vs*. *MF* (*zip* (*en* σ) *vs*))
                  (*List.n-lists* (*length* (*en* σ)) (*en* τ))

definition *cD* :: *ty* ⇒ *mval* ⇒ *bool* **where** *cD* τ *v* ≡ *v* ∈ *set* (*en* τ)

fun *cAp* :: *mval* ⇒ *mval* ⇒ *mval* **where**
  cAp (*MF G*) *x* = (*case map-of G x of Some y* ⇒ *y* | *None* ⇒ *MI*)
| *cAp v x* = *MI*

definition *cLm* :: *ty* ⇒ (*mval* ⇒ *mval*) ⇒ *mval* **where**
 cLm σ *h* ≡ *MF* (*zip* (*en* σ) (*map h* (*en* σ)))

lemma *en-nonempty*: *en* τ ≠ []
⟨*proof*⟩

lemma *distinct-en*: *distinct* (*en* τ)
⟨*proof*⟩

lemma *cD-fun*: *cD* (σ ⇒ τ) *v* ⟷ (∃ *vs*.
  length vs = *length* (*en* σ) ∧ (∀ *x* ∈ *set vs*. *cD* τ *x*) ∧ *v* = *MF* (*zip* (*en* σ) *vs*))
  ⟨*proof*⟩

lemma *cAp-cLm* [*simp*]: *cD* σ *a* ⟹ *cAp* (*cLm* σ *h*) *a* = *h a*
  ⟨*proof*⟩

lemma *cLm-dom*: (⋀*d*. *cD* σ *d* ⟹ *cD* τ (*h d*)) ⟹ *cD* (σ ⇒ τ) (*cLm* σ *h*)
  ⟨*proof*⟩

lemma *cAp-dom*: *cD* (σ ⇒ τ) *f* ⟹ *cD* σ *a* ⟹ *cD* τ (*cAp f a*)
⟨*proof*⟩

lemma *cAp-ext*:
  assumes *f*: *cD* (σ ⇒ τ) *f* **and** *g*: *cD* (σ ⇒ τ) *g*
     and *ag*: ⋀*a*. *cD* σ *a* ⟹ *cAp f a* = *cAp g a*
   shows *f* = *g*
⟨*proof*⟩

The denotations of the logical constants, and a canonical parameter and assignment interpretation.

definition *cNg* :: *mval* **where** *cNg* ≡ *cLm* o (λ*a*. *MB* (*a* = *MB False*))
definition *cDs* :: *mval* **where**
  cDs ≡ *cLm* o (λ*a*. *cLm* o (λ*b*. *MB* (*a* = *MB True* ∨ *b* = *MB True*)))
definition *cPi* :: *ty* ⇒ *mval* **where**
  cPi σ ≡ *cLm* (σ ⇒ o) (λ*f*. *MB* (∀ *d*. *cD* σ *d* ⟶ *cAp f d* = *MB True*))

definition *cEv* :: *ty* ⇒ *mval* **where**
  cEv σ ≡ *cLm σ* (λ*a*. *cLm σ* (λ*b*. *MB* (*a* = *b*)))
definition *cIv* :: *ty* ⇒ *mval* **where**
  cIv σ ≡ *cLm* (*σ* ⇛ o)
    (λ*f*. *if* ∃ *a*. *cD σ a* ∧ (∀ *b*. *cD σ b* ⟶ (*cAp f b* = *MB True*) = (*b* = *a*))
      then SOME a. *cD σ a* ∧ (∀ *b*. *cD σ b* ⟶ (*cAp f b* = *MB True*) = (*b* = *a*))
      else hd (*en σ*))

abbreviation *cJv* :: *'p* ⇒ *ty* ⇒ *mval* **where** *cJv p σ* ≡ *hd* (*en σ*)
abbreviation *cXi* :: *nat* ⇒ *ty* ⇒ *mval* **where** *cXi n σ* ≡ *hd* (*en σ*)

lemma *cD-bool*: *cD* o *v* ⟷ *v* = *MB True* ∨ *v* = *MB False*
  ⟨*proof*⟩

lemma *hd-en-dom*: *cD σ* (*hd* (*en σ*)) ⟨*proof*⟩

lemma *cIv-desc*:
  assumes *f*: *cD* (*σ* ⇛ o) *f* **and** *a*: *cD σ a*
    and *sing*: ∀ *b*. *cD σ b* ⟶ (*cAp f b* = *MB True*) = (*b* = *a*)
  shows *cAp* (*cIv σ*) *f* = *a*
⟨*proof*⟩

theorem *concrete-standard-model*:
  standard-model cD cAp cLm (*MB True*) (*MB False*) *cNg cDs cIv cEv cPi* (*cJv* :: *'p* ⇒ *ty* ⇒ *mval*)
⟨*proof*⟩

6.2 Consistency of *NK*

The Σ-model predicate of the canonical construction, exported from the sublocale chain *standard-model* ⊆ *general-model* ⊆ *bkk-model*.

lemma (**in** *general-model*) *bkk-model-pred*:
  bkk-model Dm Ap (λξ *A*. ⟦*A*⟧ξ) (λ*a*. *a* = *Tv*) ⟨*proof*⟩

Consistency of *NK* (the deep embedding): ⊥ is not derivable — by soundness (BKK Theorem 7.3) it would have to be *v*-true in the concrete model, contradicting BKK Lemma 3.43.

theorem *nk-consistent*: ¬ (⊢ (⊥ :: *'p tm*))
⟨*proof*⟩

No sentence is derivable together with its negation.

theorem *nk-not-both*: **assumes** ⊢ (*A* :: *'p tm*) **shows** ¬⊢ ¬ *A*
  ⟨*proof*⟩

In the terminology of the completeness development: the empty set is consistent (BKK Definition 7.4).

corollary *con-empty*: *con* ({} :: *'p tm set*)
  ⟨*proof*⟩

7 Example: Cantor's theorem

The surjective and the injective Cantor theorem, at every type σ: there is no surjection from σ onto $\sigma \Rightarrow \mathrm{o}$, and no injection from $\sigma \Rightarrow \mathrm{o}$ into σ. Both are derived *inside the calculus*: genuine *NK*-derivations via the diagonal predicate (for the injective version via the description operator $NK(\iota)$, following Andrews 1972). The Cantor sentences are stated as in the *Stanford Encyclopedia of Philosophy* entry on Church's type theory (Benzmüller and Andrews 2024), generalised from ι to every type σ.

7.1 The defined existential quantifier, and the Cantor sentences

The Cantor sentences are stated literally, as in the cited encyclopedia entry: no surjection $\mathcal{G} : \sigma \Rightarrow \sigma \Rightarrow \mathrm{o}$ onto $\sigma \Rightarrow \mathrm{o}$, and no injection $\mathcal{I} : (\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma$. The following lemmas record their locally-nameless normal forms (by computation) and the closedness facts used by the derivations. The right-hand sides deliberately show the machine-level de Bruijn normal form (indices *Bnd 0*, *Bnd* (*Suc 0*) and so on); the named-binder left-hand sides are the human-facing statements.

lemma *surj-norm*:
$(\exists\, \mathcal{G}_{\sigma \Rightarrow \sigma \Rightarrow \mathrm{o}}.\ \mathbf{\Pi}\mathcal{F}_{\sigma \Rightarrow \mathrm{o}}.\ \exists\, \mathcal{X}_{\sigma}.\ ((\mathcal{G}^f\sigma \Rightarrow \sigma \Rightarrow \mathrm{o} \cdot \mathcal{X}^f{}_{\sigma}) =_{\sigma \Rightarrow \mathrm{o}} \mathcal{F}^f\sigma \Rightarrow \mathrm{o}))$
$= \exists\, \sigma \Rightarrow \sigma \Rightarrow \mathrm{o}\ (\mathbf{\Pi}\sigma \Rightarrow \mathrm{o}\ (\exists\, \sigma$
$((\mathit{Bnd}\ (\mathit{Suc}\ (\mathit{Suc}\ 0)) \cdot \mathit{Bnd}\ 0) =_{\sigma \Rightarrow \mathrm{o}} \mathit{Bnd}\ (\mathit{Suc}\ 0))))$
⟨*proof*⟩

lemma *inj-norm*:
$(\exists\, \mathcal{I}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma}.\ \mathbf{\Pi}\mathcal{F}_{\sigma \Rightarrow \mathrm{o}}.\ \mathbf{\Pi}\mathcal{H}_{\sigma \Rightarrow \mathrm{o}}.$
$(((\mathcal{I}^f{}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma} \cdot \mathcal{F}^f\sigma \Rightarrow \mathrm{o}) =_{\sigma} (\mathcal{I}^f{}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma} \cdot \mathcal{H}^f\sigma \Rightarrow \mathrm{o}))$
$\supset (\mathcal{F}^f\sigma \Rightarrow \mathrm{o} =_{\sigma \Rightarrow \mathrm{o}} \mathcal{H}^f\sigma \Rightarrow \mathrm{o})))$
$= \exists_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma}\ (\mathbf{\Pi}\sigma \Rightarrow \mathrm{o}\ (\mathbf{\Pi}\sigma \Rightarrow \mathrm{o}$
$(((\mathit{Bnd}\ (\mathit{Suc}\ (\mathit{Suc}\ 0)) \cdot \mathit{Bnd}\ (\mathit{Suc}\ 0)) =_{\sigma} (\mathit{Bnd}\ (\mathit{Suc}\ (\mathit{Suc}\ 0)) \cdot \mathit{Bnd}$
$0))$
$\supset (\mathit{Bnd}\ (\mathit{Suc}\ 0) =_{\sigma \Rightarrow \mathrm{o}} \mathit{Bnd}\ 0))))$
⟨*proof*⟩

lemma *wff-surj*:
$\mathit{wff}_{\mathrm{o}}(\exists\, \mathcal{G}_{\sigma \Rightarrow \sigma \Rightarrow \mathrm{o}}.\ \mathbf{\Pi}\mathcal{F}_{\sigma \Rightarrow \mathrm{o}}.\ \exists\, \mathcal{X}_{\sigma}.$
$((\mathcal{G}^f\sigma \Rightarrow \sigma \Rightarrow \mathrm{o} \cdot \mathcal{X}^f{}_{\sigma}) =_{\sigma \Rightarrow \mathrm{o}} \mathcal{F}^f\sigma \Rightarrow \mathrm{o}))$
⟨*proof*⟩

lemma *wff-inj*:
$\mathit{wff}_{\mathrm{o}}(\exists\, \mathcal{I}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma}.\ \mathbf{\Pi}\mathcal{F}_{\sigma \Rightarrow \mathrm{o}}.\ \mathbf{\Pi}\mathcal{H}_{\sigma \Rightarrow \mathrm{o}}.$
$(((\mathcal{I}^f{}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma} \cdot \mathcal{F}^f\sigma \Rightarrow \mathrm{o}) =_{\sigma} (\mathcal{I}^f{}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma} \cdot \mathcal{H}^f\sigma \Rightarrow \mathrm{o}))$
$\supset (\mathcal{F}^f\sigma \Rightarrow \mathrm{o} =_{\sigma \Rightarrow \mathrm{o}} \mathcal{H}^f\sigma \Rightarrow \mathrm{o})))$
⟨*proof*⟩

lemma *fvs-surj* [*simp*]:
$fvs\ (\neg\ (\exists\, \mathcal{G}_\sigma \Rightarrow \sigma \Rightarrow \mathrm{o}.\ \mathbf{\Pi}\mathcal{F}_\sigma \Rightarrow \mathrm{o}.\ \exists\, \mathcal{X}_\sigma.$
$((\mathcal{G}^f\,\sigma \Rightarrow \sigma \Rightarrow \mathrm{o} \cdot \mathcal{X}^f\,\sigma) =_\sigma \Rightarrow \mathrm{o}\ \mathcal{F}^f\,\sigma \Rightarrow \mathrm{o}))) = \{\}$
⟨*proof*⟩

lemma *fvs-inj* [*simp*]:
$fvs\ (\neg\ (\exists\, \mathcal{I}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma}.\ \mathbf{\Pi}\mathcal{F}_\sigma \Rightarrow \mathrm{o}.\ \mathbf{\Pi}\mathcal{H}_\sigma \Rightarrow \mathrm{o}.$
$(((\mathcal{I}^f{}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma} \cdot \mathcal{F}^f\,\sigma \Rightarrow \mathrm{o}) =_\sigma (\mathcal{I}^f{}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma} \cdot \mathcal{H}^f\,\sigma \Rightarrow \mathrm{o}))$
$\supset (\mathcal{F}^f\,\sigma \Rightarrow \mathrm{o} =_\sigma \Rightarrow \mathrm{o}\ \mathcal{H}^f\,\sigma \Rightarrow \mathrm{o}))))) = \{\}$
⟨*proof*⟩

lemma *pars-surj* [*simp*]:
$pars\ (\exists\, \mathcal{G}_\sigma \Rightarrow \sigma \Rightarrow \mathrm{o}.\ \mathbf{\Pi}\mathcal{F}_\sigma \Rightarrow \mathrm{o}.\ \exists\, \mathcal{X}_\sigma.$
$((\mathcal{G}^f\,\sigma \Rightarrow \sigma \Rightarrow \mathrm{o} \cdot \mathcal{X}^f\,\sigma) =_\sigma \Rightarrow \mathrm{o}\ \mathcal{F}^f\,\sigma \Rightarrow \mathrm{o})) = \{\}$
⟨*proof*⟩

lemma *pars-inj* [*simp*]:
$pars\ (\exists\, \mathcal{I}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma}.\ \mathbf{\Pi}\mathcal{F}_\sigma \Rightarrow \mathrm{o}.\ \mathbf{\Pi}\mathcal{H}_\sigma \Rightarrow \mathrm{o}.$
$(((\mathcal{I}^f{}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma} \cdot \mathcal{F}^f\,\sigma \Rightarrow \mathrm{o}) =_\sigma (\mathcal{I}^f{}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma} \cdot \mathcal{H}^f\,\sigma \Rightarrow \mathrm{o}))$
$\supset (\mathcal{F}^f\,\sigma \Rightarrow \mathrm{o} =_\sigma \Rightarrow \mathrm{o}\ \mathcal{H}^f\,\sigma \Rightarrow \mathrm{o}))) = \{\}$
⟨*proof*⟩

Typing of the terms occurring in the derivations, once and for all.

lemma *wff-surj-body*:
$wff_{(\sigma \Rightarrow \sigma \Rightarrow \mathrm{o}) \Rightarrow \mathrm{o}}(\mathbf{\Lambda}\sigma \Rightarrow \sigma \Rightarrow \mathrm{o}\ (\mathbf{\Pi}\sigma \Rightarrow \mathrm{o}\ (\exists\,\sigma$
$((Bnd\ (Suc\ (Suc\ 0)) \cdot Bnd\ 0) =_\sigma \Rightarrow \mathrm{o}\ Bnd\ (Suc\ 0)))))$
⟨*proof*⟩

lemma *wff-inst-body*:
$wff_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \mathrm{o}}(\mathbf{\Lambda}\sigma \Rightarrow \mathrm{o}\ (\exists\,\sigma$
$((Par\ g\ (\sigma \Rightarrow \sigma \Rightarrow \mathrm{o}) \cdot Bnd\ 0) =_\sigma \Rightarrow \mathrm{o}\ Bnd\ (Suc\ 0))))$
⟨*proof*⟩

lemma *wff-diag*:
$wff_\sigma \Rightarrow \mathrm{o}(\mathbf{\Lambda}_\sigma\ (\neg\ ((Par\ g\ (\sigma \Rightarrow \sigma \Rightarrow \mathrm{o}) \cdot Bnd\ 0) \cdot Bnd\ 0)))$
⟨*proof*⟩

lemma *wff-wit-body*:
assumes $wff_\sigma \Rightarrow \mathrm{o}(D)$ **and** *lc D*
shows $wff_\sigma \Rightarrow \mathrm{o}(\mathbf{\Lambda}_\sigma\ ((Par\ g\ (\sigma \Rightarrow \sigma \Rightarrow \mathrm{o}) \cdot Bnd\ 0) =_\sigma \Rightarrow \mathrm{o}\ D))$
⟨*proof*⟩

7.2 The surjective Cantor theorem in *NK*

theorem *nk-surjective-cantor*: **fixes** σ :: *ty* **shows**
$\vdash (\neg\ (\exists\, \mathcal{G}_\sigma \Rightarrow \sigma \Rightarrow \mathrm{o}.\ \mathbf{\Pi}\mathcal{F}_\sigma \Rightarrow \mathrm{o}.\ \exists\, \mathcal{X}_\sigma.$
$((\mathcal{G}^f\,\sigma \Rightarrow \sigma \Rightarrow \mathrm{o} \cdot \mathcal{X}^f\,\sigma) =_\sigma \Rightarrow \mathrm{o}\ \mathcal{F}^f\,\sigma \Rightarrow \mathrm{o})) :: {'p}{::}infinite\ tm)$

(**is** $\vdash \neg$ *?S*)
⟨*proof*⟩

7.3 The injective Cantor theorem in *NK*

Typing of the description-based diagonal predicate.

lemma *wff-desc-diag*:
$\mathit{wff}_\sigma \Rightarrow {}_{\mathrm{o}}(\mathbf{\Lambda}_\sigma\ (\neg\ (((\mathit{Iota}\ (\sigma \Rightarrow \mathrm{o})) \cdot (\mathbf{\Lambda}_\sigma \Rightarrow \mathrm{o}\ (((\mathit{Par}\ i\ ((\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma)) \cdot \mathit{Bnd}\ 0) \doteq_\sigma \mathit{Bnd}\ (\mathit{Suc}\ 0)))) \cdot \mathit{Bnd}\ 0)))$
⟨*proof*⟩

theorem *nk-injective-cantor*: **fixes** σ :: *ty* **shows**
$\vdash (\neg\ (\exists \mathcal{I}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma}.\ \mathbf{\Pi}\mathcal{F}_\sigma \Rightarrow \mathrm{o}.\ \mathbf{\Pi}\mathcal{H}_\sigma \Rightarrow \mathrm{o}.$
$(((\mathcal{I}^f{}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma} \cdot \mathcal{F}^f{}_\sigma \Rightarrow \mathrm{o}) =_\sigma (\mathcal{I}^f{}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma} \cdot \mathcal{H}^f{}_\sigma \Rightarrow \mathrm{o}))$
$\supset (\mathcal{F}^f{}_\sigma \Rightarrow {}_{\mathrm{o}} =_\sigma \Rightarrow {}_{\mathrm{o}}\ \mathcal{H}^f{}_\sigma \Rightarrow {}_{\mathrm{o}})))$ :: $'p$::*infinite tm*)
(**is** $\vdash \neg$ *?S*)
⟨*proof*⟩

8 Main results

The central results of this entry, restated in one place. Soundness and completeness of *NK* for the class $\mathcal{M}_{\beta f b}$ (BKK Theorem 7.3 and a strengthening of BKK Corollary 7.7): for well-formed (open) formulas over every signature with infinitely many parameters, derivability coincides with validity at *every* infinite value carrier — no cardinality link between the parameter type $'p$ and the carrier $'u$ is needed. Consistency holds for arbitrary signatures, by the concrete standard model — note the asymmetry: consistency needs *no* constraint on $'p$ at all, whereas completeness requires $'p$ infinite, since the rule $NK(\Pi I)$ consumes fresh eigen-parameters. Cantor's theorem, surjective and injective, is derived inside *NK* at every type.

theorem *NK-soundness*:
$\Phi \vdash C \Longrightarrow \Phi \models('u)\ C$
⟨*proof*⟩

theorem *NK-completeness*:
assumes $\mathit{wff}_{\mathrm{o}}(A$::$'p$::*infinite tm*) **and** $\models('u$::*infinite*) A
shows $\vdash A$
⟨*proof*⟩

theorem *sound-and-complete*:
assumes $\mathit{wff}_{\mathrm{o}}(A$::$'p$::*infinite tm*)
shows $\vdash A \longleftrightarrow \models('u$::*infinite*) A
⟨*proof*⟩

theorem *consistency*: $\neg \vdash \bot$
⟨*proof*⟩

corollary *no-contradiction*: $\vdash A \Longrightarrow \neg \vdash \neg\, A$
⟨*proof*⟩

theorem *cantor-surjective*: **fixes** σ :: *ty* **shows**
$\vdash (\neg\, (\exists\, \mathcal{G}_{\sigma} \Rightarrow \sigma \Rightarrow \mathrm{o}.\ \mathbf{\Pi}\mathcal{F}_{\sigma} \Rightarrow \mathrm{o}.\ \exists\, \mathcal{X}_{\sigma}.$
$((\mathcal{G}^{f}{}_{\sigma} \Rightarrow \sigma \Rightarrow \mathrm{o} \cdot \mathcal{X}^{f}{}_{\sigma}) =_{\sigma \Rightarrow \mathrm{o}} \mathcal{F}^{f}{}_{\sigma} \Rightarrow \mathrm{o})) :: {'p}{::}\mathit{infinite}\ \mathit{tm})$
⟨*proof*⟩

theorem *cantor-injective*: **fixes** σ :: *ty* **shows**
$\vdash (\neg\, (\exists\, \mathcal{I}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma}.\ \mathbf{\Pi}\mathcal{F}_{\sigma} \Rightarrow \mathrm{o}.\ \mathbf{\Pi}\mathcal{H}_{\sigma} \Rightarrow \mathrm{o}.$
$(((\mathcal{I}^{f}{}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma} \cdot \mathcal{F}^{f}{}_{\sigma} \Rightarrow \mathrm{o}) =_{\sigma} (\mathcal{I}^{f}{}_{(\sigma \Rightarrow \mathrm{o}) \Rightarrow \sigma} \cdot \mathcal{H}^{f}{}_{\sigma} \Rightarrow \mathrm{o}))$
$\supset (\mathcal{F}^{f}{}_{\sigma} \Rightarrow \mathrm{o} =_{\sigma \Rightarrow \mathrm{o}} \mathcal{H}^{f}{}_{\sigma} \Rightarrow \mathrm{o}))) :: {'p}{::}\mathit{infinite}\ \mathit{tm})$
⟨*proof*⟩

References

[1] P. B. Andrews. General models, descriptions, and choice in type theory. *Journal of Symbolic Logic*, 37(2):385–394, 1972.

[2] P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, volume 27 of *Applied Logic Series*. Kluwer Academic Publishers, 2nd edition, 2002.

[3] C. Benzmüller and P. Andrews. Church's type theory. In E. N. Zalta and U. Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, 2024. Summer 2024 Edition, https://plato.stanford.edu/archives/sum2024/entries/type-theory-church/.

[4] C. Benzmüller, C. E. Brown, and M. Kohlhase. Higher-order semantics and extensionality. *Journal of Symbolic Logic*, 69(4):1027–1088, 2004.

[5] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, 1940.

[6] J. Díaz. Metatheory of $\mathcal{Q}_0$. *Archive of Formal Proofs*, November 2023. https://isa-afp.org/entries/Q0_Metatheory.html, Formal proof development.

[7] A. H. From. An Isabelle/HOL framework for synthetic completeness proofs. In *Proceedings of the 14th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2025)*. ACM, 2025.

[8] J. Harrison. Towards self-verification of HOL Light. In *Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Computer Science*, pages 177–191. Springer, 2006.

[9] M. Koch and D. Kirst. Undecidability, incompleteness, and completeness of second-order logic in Coq. In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2022)*, pages 274–290. ACM, 2022.