Abstract Consistency Properties

Asta Halkjær Boserup, Anders Schlichtkrull

April 9, 2026

Abstract

Smullyan used abstract consistency properties to great effect in unifying meta-theoretical results in first-order logic. By abstractly specifying when a set of formulas is consistent, he proved a single model existence result for all sets that met the specification, reusing this result in proving completeness, compactness, etc. Fitting later defined abstract consistency properties for a range of other logics.

In this work we use locales to mechanize abstract consistency properties without fixing a particular logic or syntax, generalizing the work of Smullyan and Fitting. We use Fitting's technique for closing a consistency property under limits to guarantee the existence of a maximal element. This yields a maximal consistent set for any notion of consistency expressible in the framework. The usual conjunctive, disjunctive, universal and existential conditions of abstract consistency properties —based on Smullyan's uniform notation— arise as special cases of our abstract development. Users of the framework can define an abstract consistency property for their own syntax and logic, prove that it is well behaved, and receive a corresponding maximal consistent set from which to prove model existence.

We provide three example instantiations. First, compactness and completeness for a first-order logic where we only instantiate universal quantifiers with already occurring terms. Second, completeness over general models for a second-order logic. Third, completeness of our own natural deduction system for Prior's Ideal Language, a recently developed hybrid logic with propositional quantification.

Bibliography

[1] S. Berghofer. First-order logic according to Fitting. *Archive of Formal Proofs*, 2007, August 2007. Formal proof development.

[2] P. Blackburn, T. Braüner, and J. L. Kofod. Prior's ideal language. *Mathematical Structures in Computer Science*, 35, 2025.

[3] M. Fitting. *First-Order Logic and Automated Theorem Proving, Second Edition*. Graduate Texts in Computer Science. Springer, 1996.

[4] A. H. From. Soundness and completeness of an axiomatic system for first-order logic. *Archive of Formal Proofs*, 2021, September 2021. Formal proof development.

[5] A. H. From. A succinct formalization of the completeness of first-order logic. In H. Basold, J. Cockx, and S. Ghilezan, editors, *27th International Conference on Types for Proofs and Programs, TYPES 2021, June 14-18, 2021, Leiden, The Netherlands (Virtual Conference)*, volume 239 of *LIPIcs*, pages 8:1–8:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[6] A. H. From and A. Schlichtkrull. Abstract, Compositional Consistency: Isabelle/HOL Locales for Completeness à la Fitting. In Y. Forster and C. Keller, editors, *16th International Conference on Interactive Theorem Proving (ITP 2025)*, volume 352 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:20, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[7] R. M. Smullyan. A unifying principal in quantification theory. *Proceedings of the National Academy of Sciences*, 49(6):828–832, 1963.

[8] R. M. Smullyan. *First-Order Logic*. Springer, Berlin, 1968.

[9] J. Väänänen. Second-order and Higher-order Logic. In E. N. Zalta and U. Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2024 edition, 2024.

Contents

Chapter 1

Abstract Consistency Properties

theory *Abstract-Consistency-Property* **imports**
HOL−Cardinals.Cardinal-Order-Relation
begin

1.1 Utility

lemma *Set-Diff-Un*: ‹$X - (Y \cup Z) = X - Y - Z$›
⟨*proof*⟩

lemma *infinite-diff-finite*: ‹*finite* $A \Longrightarrow$ *infinite* $(- B) \Longrightarrow$ *infinite* $(- (A \cup B))$›
⟨*proof*⟩

lemma *infinite-Diff-fin-Un*: ‹*infinite* $(X - Y) \Longrightarrow$ *finite* $Z \Longrightarrow$ *infinite* $(X - (Z \cup Y))$›
⟨*proof*⟩

lemma *infinite-Diff-subset*: ‹*infinite* $(X - A) \Longrightarrow B \subseteq A \Longrightarrow$ *infinite* $(X - B)$›
⟨*proof*⟩

lemma *finite-bound*:
fixes X :: ‹$('a$ :: *size*$)$ *set*›
assumes ‹*finite* X› ‹$X \neq \{\}$›
shows ‹$\exists x \in X. \forall y \in X.$ *size* $y \leq$ *size* x›
⟨*proof*⟩

lemma *infinite-UNIV-size*:
fixes f :: ‹$('a$ :: *size*$) \Rightarrow 'a$›
assumes ‹$\bigwedge x.$ *size* $x <$ *size* $(f\ x)$›
shows ‹*infinite* $(UNIV$ :: $'a$ *set*$)$›
⟨*proof*⟩

lemma *infinite-left*: ‹*finite* $C \Longrightarrow$ *infinite* $A \Longrightarrow |A| \leq o\ |-\ B| \Longrightarrow |A| \leq o\ |-\ (C \cup B)|$›
⟨*proof*⟩

lemma *card-of-infinite-smaller-Union*:
assumes ‹$\forall x.\ |f\ x| <o\ |X|$› ‹*infinite* X›
shows ‹$|\bigcup x \in X.\ f\ x| \leq o\ |X|$›
⟨*proof*⟩

context *wo-rel*
begin

lemma *underS-bound*: ‹$a \in$ *underS* $c \Longrightarrow b \in$ *underS* $c \Longrightarrow a \in$ *under* $b \vee b \in$ *under* a›
⟨*proof*⟩

lemma *finite-underS-bound*:
assumes ‹*finite* X› ‹$X \subseteq$ *underS* a› ‹$X \neq \{\}$›
shows ‹$\exists\, a \in X.\ \forall\, b \in X.\ b \in$ *under* a›
⟨*proof*⟩

lemma *finite-bound-under*:
assumes ‹*finite* p› ‹$p \subseteq (\bigcup a \in$ *Field* $r.\ f\ a)$›
shows ‹$\exists\, b.\ p \subseteq (\bigcup a \in$ *under* $b.\ f\ a)$›
⟨*proof*⟩

lemma *underS-trans*: ‹$a \in$ *underS* $b \Longrightarrow b \in$ *underS* $c \Longrightarrow a \in$ *underS* c›
⟨*proof*⟩

definition *is-chain* :: ‹$('a \Rightarrow 'a\ set) \Rightarrow bool$› **where**
‹*is-chain* $f \equiv \forall\, a \in$ *Field* $r.\ \forall\, b \in$ *Field* $r.\ b \in$ *under* $a \longrightarrow f\ b \subseteq f\ a$›

lemma *is-chainD*: ‹*is-chain* $f \Longrightarrow b \in$ *under* $a \Longrightarrow x \in f\ b \Longrightarrow x \in f\ a$›
⟨*proof*⟩

lemma *chain-index*:
assumes *ch*: ‹*is-chain* f› **and** *fin*: ‹*finite* F› **and** *ne*: ‹*Field* $r \neq \{\}$›
shows ‹$F \subseteq (\bigcup a \in$ *Field* $r.\ f\ a) \Longrightarrow \exists\, a \in$ *Field* $r.\ F \subseteq f\ a$›
⟨*proof*⟩

end

1.2 Finite Character

definition *close* :: ‹$'a\ set\ set \Rightarrow 'a\ set\ set$› **where**
‹*close* $C \equiv \{S.\ (\exists\, S' \in C.\ S \subseteq S')\}$›

definition *subset-closed* :: ‹$'a\ set\ set \Rightarrow bool$› **where**
‹*subset-closed* $C \equiv \forall\, S' \in C.\ \forall\, S \subseteq S'.\ S \in C$›

lemma *subset-in-close*: ‹$S \subseteq S' \Longrightarrow x \cup S' \in C \Longrightarrow x \cup S \in close\ C$›
⟨*proof*⟩

lemma *close-closed*: ‹*subset-closed* (*close C*)›
⟨*proof*⟩

lemma *close-subset*: ‹$C \subseteq close\ C$›
⟨*proof*⟩

definition *finite-char* :: ‹$'a\ set\ set \Rightarrow bool$› **where**
‹*finite-char* $C \equiv \forall S.\ S \in C \longleftrightarrow (\forall S' \subseteq S.\ finite\ S' \longrightarrow S' \in C)$›

definition *mk-finite-char* :: ‹$'a\ set\ set \Rightarrow 'a\ set\ set$› **where**
‹*mk-finite-char* $C \equiv \{S.\ (\forall S' \subseteq S.\ finite\ S' \longrightarrow S' \in C)\}$›

lemma *finite-char*: ‹*finite-char* (*mk-finite-char C*)›
⟨*proof*⟩

lemma *finite-char-closed*: ‹*finite-char* $C \Longrightarrow$ *subset-closed* C›
⟨*proof*⟩

lemma *finite-char-subset*: ‹*subset-closed* $C \Longrightarrow C \subseteq$ *mk-finite-char* C›
⟨*proof*⟩

lemma (**in** *wo-rel*) *chain-union-closed*:
assumes ‹*finite-char C*› ‹*is-chain f*› ‹$\forall a \in Field\ r.\ f\ a \in C$› ‹$Field\ r \neq \{\}$›
shows ‹$(\bigcup a \in Field\ r.\ f\ a) \in C$›
⟨*proof*⟩

definition *maximal* :: ‹$'a\ set\ set \Rightarrow 'a\ set \Rightarrow bool$› **where**
‹*maximal* $C\ S \longleftrightarrow (\forall S' \in C.\ S \subseteq S' \longrightarrow S = S')$›

1.3 Consistency Properties

locale *Params* =
fixes *map-fm* :: ‹$('x \Rightarrow 'x) \Rightarrow 'fm \Rightarrow 'fm$›
and *params-fm* :: ‹$'fm \Rightarrow 'x\ set$›
and *is-param* :: ‹$'x \Rightarrow bool$›
assumes *map-fm-id*: ‹*map-fm id* = *id*›
and *finite-params-fm* [*simp*]: ‹$\bigwedge p.\ finite\ (params\text{-}fm\ p)$›
and *map-params-fm*: ‹$\bigwedge f\ g\ p.\ (\forall x \in params\text{-}fm\ p.\ f\ x = g\ x) \Longrightarrow map\text{-}fm\ f\ p = map\text{-}fm\ g\ p$›
begin

definition *is-subst* :: ‹$('x \Rightarrow 'x) \Rightarrow bool$› **where**
‹*is-subst* $f \equiv \forall x.\ is\text{-}param\ x \longleftrightarrow is\text{-}param\ (f\ x)$›

lemma *is-subst-id* [*intro*]: ‹*is-subst id*›

⟨*proof*⟩

definition *mk-alt-consistency* :: ‹*'fm set set ⇒ 'fm set set*› **where**
‹*mk-alt-consistency C* ≡ {*S*. (∃ *f*. *is-subst f* ∧ *map-fm f* ‘ *S* ∈ *C*)}›

lemma *mk-alt-consistency-subset*: ‹*C* ⊆ *mk-alt-consistency C*›
⟨*proof*⟩

lemma *mk-alt-consistency-closed*:
assumes ‹*subset-closed C*›
shows ‹*subset-closed* (*mk-alt-consistency C*)›
⟨*proof*⟩

abbreviation *params* :: ‹*'fm set ⇒ 'x set*› **where**
‹*params S* ≡ ⋃ *p* ∈ *S*. *params-fm p*›

lemma *infinite-params*: ‹*infinite* (*U* − *params B*) ⟹ *infinite* (*U* − *params* (*set ps* ∪ *B*))›
⟨*proof*⟩

lemma *infinite-params-left*:
assumes ‹*infinite A*› ‹|*A*| ≤*o* |*U* − *params S*|›
shows ‹|*A*| ≤*o* |*U* − *params* (*set ps* ∪ *S*)|›
⟨*proof*⟩

definition *enough-new* :: ‹*'fm set ⇒ bool*› **where**
‹*enough-new S* ≡ |*UNIV* :: *'fm set*| ≤*o* |*Collect is-param* − *params S*|›

lemma *enough-new-countable*:
assumes ‹∃ *to-nat* :: *'fm ⇒ nat*. *inj to-nat*› ‹*infinite* (*Collect is-param* − *params S*)›
shows ‹*enough-new S*›
⟨*proof*⟩

lemma *enough-new-all-param*:
assumes ‹|*UNIV* :: *'fm set*| ≤*o* |*UNIV* − *params S*|› ‹⋀*x*. *is-param x*›
shows ‹*enough-new S*›
⟨*proof*⟩

end

datatype (*'x*, *'fm*) *kind*
= *Cond* ‹*'fm list ⇒* (*'fm set set ⇒ 'fm set ⇒ bool*) *⇒ bool*› ‹*'fm set ⇒ bool*›
| *Wits* ‹*'fm ⇒ 'x ⇒ 'fm list*›

inductive (**in** *Params*) *sat*$_E$ :: ‹(*'x*, *'fm*) *kind ⇒ 'fm set set ⇒ bool*› **where**
sat$_E$*-Cond* [*intro*!]: ‹(⋀*S ps Q*. *S* ∈ *C* ⟹ *set ps* ⊆ *S* ⟹ *P ps Q* ⟹ *Q C S*) ⟹ *sat*$_E$ (*Cond P H*) *C*›
| *sat*$_E$*-Wits* [*intro*!]: ‹(⋀*S p*. *S* ∈ *C* ⟹ *p* ∈ *S* ⟹ (∃ *x*. *is-param x* ∧ *set* (*W p x*)

$\cup\ S \in C)) \Longrightarrow sat_E\ (Wits\ W)\ C\rangle$

inductive-cases (**in** *Params*) sat_E-*CondE*[*elim*!]: $\langle sat_E\ (Cond\ P\ H)\ C\rangle$
inductive-cases (**in** *Params*) sat_E-*WitsE*[*elim*!]: $\langle sat_E\ (Wits\ W)\ C\rangle$

inductive (**in** *Params*) $sat_A :: \langle('x, 'fm)\ kind \Rightarrow 'fm\ set\ set \Rightarrow bool\rangle$ **where**
sat_A-*Cond* [*intro*!]: $\langle(\bigwedge S\ ps\ Q.\ S \in C \Longrightarrow set\ ps \subseteq S \Longrightarrow P\ ps\ Q \Longrightarrow Q\ C\ S) \Longrightarrow sat_A\ (Cond\ P\ H)\ C\rangle$
| sat_A-*Wits* [*intro*!]: $\langle(\bigwedge S\ p\ x.\ S \in C \Longrightarrow p \in S \Longrightarrow x \notin params\ S \Longrightarrow is\text{-}param\ x \Longrightarrow set\ (W\ p\ x) \cup S \in C) \Longrightarrow sat_A\ (Wits\ W)\ C\rangle$

inductive-cases (**in** *Params*) sat_A-*CondE*[*elim*!]: $\langle sat_A\ (Cond\ P\ H)\ C\rangle$
inductive-cases (**in** *Params*) sat_A-*WitsE*[*elim*!]: $\langle sat_A\ (Wits\ W)\ C\rangle$

definition (**in** *Params*) $prop_E :: \langle('x, 'fm)\ kind\ list \Rightarrow 'fm\ set\ set \Rightarrow bool\rangle$ **where**
$\langle prop_E\ Ks\ C \equiv \forall K \in set\ Ks.\ sat_E\ K\ C\rangle$

definition (**in** *Params*) $prop_A :: \langle('x, 'fm)\ kind\ list \Rightarrow 'fm\ set\ set \Rightarrow bool\rangle$ **where**
$\langle prop_A\ Ks\ C \equiv \forall K \in set\ Ks.\ sat_A\ K\ C\rangle$

inductive (**in** *Params*) $sat_H :: \langle('x, 'fm)\ kind \Rightarrow 'fm\ set \Rightarrow bool\rangle$ **where**
sat_H-*Cond* [*intro*!]: $\langle H\ S \Longrightarrow sat_H\ (Cond\ P\ H)\ S\rangle$
| sat_H-*Wits* [*intro*!]: $\langle(\bigwedge p.\ p \in S \Longrightarrow (\exists x.\ is\text{-}param\ x \wedge set\ (W\ p\ x) \subseteq S)) \Longrightarrow sat_H\ (Wits\ W)\ S\rangle$

inductive-cases (**in** *Params*) sat_H-*CondE*[*elim*!]: $\langle sat_H\ (Cond\ P\ H)\ C\rangle$
inductive-cases (**in** *Params*) sat_H-*WitsE*[*elim*!]: $\langle sat_H\ (Wits\ W)\ C\rangle$

definition (**in** *Params*) $prop_H :: \langle('x, 'fm)\ kind\ list \Rightarrow 'fm\ set \Rightarrow bool\rangle$ **where**
$\langle prop_H\ Ks\ S \equiv \forall K \in set\ Ks.\ sat_H\ K\ S\rangle$

theorem (**in** *Params*) sat_H-*Wits*: $\langle sat_E\ (Wits\ W)\ C \Longrightarrow S \in C \Longrightarrow maximal\ C\ S \Longrightarrow sat_H\ (Wits\ W)\ S\rangle$
⟨*proof*⟩

1.3.1 Consistency Kinds

locale *Consistency-Kind* = *Params map-fm params-fm is-param*
for
$map\text{-}fm :: \langle('x \Rightarrow 'x) \Rightarrow 'fm \Rightarrow 'fm\rangle$ **and**
$params\text{-}fm :: \langle'fm \Rightarrow 'x\ set\rangle$ **and**
$is\text{-}param :: \langle'x \Rightarrow bool\rangle$ +
fixes $K :: \langle('x, 'fm)\ kind\rangle$
assumes *respects-close*: $\langle\bigwedge C.\ sat_E\ K\ C \Longrightarrow sat_E\ K\ (close\ C)\rangle$
and *respects-alt*: $\langle\bigwedge C.\ sat_E\ K\ C \Longrightarrow subset\text{-}closed\ C \Longrightarrow sat_A\ K\ (mk\text{-}alt\text{-}consistency\ C)\rangle$
and *respects-fin*: $\langle\bigwedge C.\ subset\text{-}closed\ C \Longrightarrow sat_A\ K\ C \Longrightarrow sat_A\ K\ (mk\text{-}finite\text{-}char\ C)\rangle$
and *hintikka*: $\langle\bigwedge C\ S.\ sat_E\ K\ C \Longrightarrow S \in C \Longrightarrow maximal\ C\ S \Longrightarrow sat_H\ K\ S\rangle$

locale *Consistency-Kinds* = *Params map-fm params-fm is-param*
  for
    map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
    params-fm :: ‹'*fm* ⇒ '*x set*› **and**
    is-param :: ‹'*x* ⇒ *bool*› +
  fixes *Ks* :: ‹('*x*, '*fm*) *kind list*›
  assumes *all-kinds*: ‹⋀*K*. *K* ∈ *set Ks* ⟹ *Consistency-Kind map-fm params-fm is-param K*›
begin

lemma *sat*$_E$: ‹*K* ∈ *set Ks* ⟹ *prop*$_E$ *Ks C* ⟹ *sat*$_E$ *K C*›
  ⟨*proof*⟩

lemma *prop*$_E$*-close*: ‹*prop*$_E$ *Ks C* ⟹ *prop*$_E$ *Ks* (*close C*)›
  ⟨*proof*⟩

lemma *prop*$_E$*-alt*: ‹*prop*$_E$ *Ks C* ⟹ *subset-closed C* ⟹ *prop*$_A$ *Ks* (*mk-alt-consistency C*)›
  ⟨*proof*⟩

lemma *prop*$_E$*-fin*: ‹*subset-closed C* ⟹ *prop*$_A$ *Ks C* ⟹ *prop*$_A$ *Ks* (*mk-finite-char C*)›
  ⟨*proof*⟩

definition *mk-alt-fin* :: ‹'*fm set set* ⇒ '*fm set set*› **where**
  ‹*mk-alt-fin C* ≡ *mk-finite-char* (*mk-alt-consistency* (*close C*))›

lemma *mk-alt-fin-subset-closed*: ‹*subset-closed* (*mk-alt-fin C*)›
  ⟨*proof*⟩

lemma *mk-alt-fin-finite-char*: ‹*finite-char* (*mk-alt-fin C*)›
  ⟨*proof*⟩

lemma *mk-alt-fin-in*: ‹*S* ∈ *C* ⟹ *S* ∈ *mk-alt-fin C*›
  ⟨*proof*⟩

theorem *prop*$_E$: ‹*prop*$_E$ *Ks C* ⟹ *prop*$_A$ *Ks* (*mk-alt-fin C*)›
  ⟨*proof*⟩

end

fun (**in** *Params*) *witness-kinds* :: ‹('*x*, '*fm*) *kind list* ⇒ '*fm* ⇒ '*fm set* ⇒ '*fm set*›
where
  ‹*witness-kinds* [] *p S* = {}›
| ‹*witness-kinds* (*Cond* - - # *Ks*) *p S* = *witness-kinds Ks p S*›
| ‹*witness-kinds* (*Wits W* # *Ks*) *p S* =
  (*let*
    rest = *witness-kinds Ks p S*;

$a = SOME\ x.\ x \in Collect\ is\text{-}param - params\ (rest \cup \{p\} \cup S)$
$in\ set\ (W\ p\ a) \cup rest)$›

lemma (**in** *Params*) *witness-kinds-new*:
assumes ‹*infinite* (*UNIV* :: *'fm set*)› ‹*infinite* (*Collect is-param* − *params S*)›
shows ‹*infinite* (*Collect is-param* − *params* (*witness-kinds Ks p S* ∪ {*p*} ∪ *S*))›
⟨*proof*⟩

lemma (**in** *Params*) *witness-kinds*:
assumes *inf*: ‹*infinite* (*UNIV* :: *'fm set*)› **and** ‹*infinite* (*Collect is-param* − *params S*)› ‹*Wits W* ∈ *set Ks*›
shows ‹∃ *x*. *is-param x* ∧ *set* (*W p x*) ⊆ *witness-kinds Ks p S*›
⟨*proof*⟩

locale *Maximal-Consistency* = *wo-rel* ‹|*UNIV*| :: *'fm rel*› + *Consistency-Kinds map-fm params-fm is-param Ks*
for
map-fm :: ‹(*'x* ⇒ *'x*) ⇒ *'fm* ⇒ *'fm*› **and**
params-fm :: ‹*'fm* ⇒ *'x set*› **and**
is-param :: ‹*'x* ⇒ *bool*› **and**
Ks :: ‹(*'x*, *'fm*) *kind list*› +
assumes *inf-univ*: ‹*infinite* (*UNIV* :: *'fm set*)›
begin

lemma *Cinfinite-r*: ‹*Cinfinite* |*UNIV* :: *'fm set*|›
⟨*proof*⟩

lemma *isLimOrd*: *isLimOrd*
⟨*proof*⟩

lemma *aboveS-ne*: ‹*aboveS a* ≠ {}›
⟨*proof*⟩

lemma *params-left*: ‹*enough-new S* ⟹ *enough-new* (*set ps* ∪ *S*)›
⟨*proof*⟩

definition *witness* :: ‹*'fm* ⇒ *'fm set* ⇒ *'fm set*› **where**
‹*witness* ≡ *witness-kinds Ks*›

definition *extendS* :: ‹*'fm set set* ⇒ *'fm* ⇒ *'fm set* ⇒ *'fm set*› **where**
‹*extendS C a prev* ≡ *if* ({*a*} ∪ *prev* ∈ *C*) *then* (*witness a prev* ∪ {*a*} ∪ *prev*) *else prev*›

definition *extendL* :: ‹*'fm set set* ⇒ (*'fm* ⇒ *'fm set*) ⇒ *'fm* ⇒ *'fm set*› **where**
‹*extendL C rec a* ≡ ⋃ *b* ∈ *underS a*. *rec b*›

definition *extend* :: ‹*'fm set set* ⇒ *'fm set* ⇒ *'fm* ⇒ *'fm set*› **where**
‹*extend C S a* ≡ *worecZSL S* (*extendS C*) (*extendL C*) *a*›

lemma *adm-woL-extendL*: ‹*adm-woL* (*extendL C*)›
⟨*proof*⟩

definition *Extend* :: ‹*'fm set set* ⇒ *'fm set* ⇒ *'fm set*› **where**
‹*Extend C S* ≡ ⋃ *a*. *extend C S a*›

lemma *finite-witness-kinds*: ‹*finite* (*witness-kinds Qs p S*)›
⟨*proof*⟩

lemma *finite-witness*: ‹*finite* (*witness p S*)›
⟨*proof*⟩

lemma *finite-witness-kinds-params*: ‹*finite* (*params* (*witness-kinds Qs p S*))›
⟨*proof*⟩

lemma *finite-witness-params*: ‹*finite* (*params* (*witness p S*))›
⟨*proof*⟩

lemma *extend-zero* [*simp*]: ‹*extend C S zero* = *S*›
⟨*proof*⟩

lemma *extend-succ* [*simp*]: ‹*extend C S* (*succ a*) =
(*if* {*a*} ∪ *extend C S a* ∈ *C then witness a* (*extend C S a*) ∪ {*a*} ∪ *extend C S a else extend C S a*)›
⟨*proof*⟩

lemma *extend-isLim* [*simp*]:
assumes ‹*isLim a*› ‹*a* ≠ *zero*›
shows ‹*extend C S a* = (⋃ *b* ∈ *underS a*. *extend C S b*)›
⟨*proof*⟩

lemma *extend-subset*: ‹*S* ⊆ *extend C S a*›
⟨*proof*⟩

lemma *Extend-subset*: ‹*S* ⊆ *Extend C S*›
⟨*proof*⟩

lemma *extend-underS*: ‹*b* ∈ *underS a* ⟹ *extend C S b* ⊆ *extend C S a*›
⟨*proof*⟩

lemma *extend-under*: ‹*b* ∈ *under a* ⟹ *extend C S b* ⊆ *extend C S a*›
⟨*proof*⟩

lemma *params-origin*:
assumes ‹*x* ∈ *params* (*extend C S a*)›
shows ‹*x* ∈ *params S* ∨ (∃ *b* ∈ *underS a*. *x* ∈ *params* (*witness b* (*extend C S b*) ∪ {*b*}))›
⟨*proof*⟩

lemma *is-chain-extend*: ‹*is-chain* (*extend C S*)›
⟨*proof*⟩

lemma *extend-in-C-step*:
assumes ‹$prop_A$ *Ks C*› ‹{*a*} ∪ *extend C S a* ∈ *C*›
and *inf*: ‹*infinite* (*Collect is-param* − *params* ({*a*} ∪ *extend C S a*))›
shows ‹*extend C S* (*succ a*) ∈ *C*›
⟨*proof*⟩

lemma *extend-in-C-stop*:
assumes ‹*extend C S a* ∈ *C*›
and ‹{*a*} ∪ *extend C S a* ∉ *C*›
shows ‹*extend C S* (*succ a*) ∈ *C*›
⟨*proof*⟩

lemma *infinite-succ-extend*:
assumes ‹*S* ∈ *C*› ‹*enough-new S*› ‹*isSucc p*›
shows ‹*infinite* (*Collect is-param* − *params* (*extend C S p*))›
⟨*proof*⟩

lemma *extend-in-C*:
assumes ‹$prop_A$ *Ks C*› ‹*finite-char C*› ‹*S* ∈ *C*› ‹*enough-new S*›
shows ‹*extend C S a* ∈ *C*›
⟨*proof*⟩

lemma *Extend-in-C*:
assumes ‹$prop_A$ *Ks C*› ‹*finite-char C*› ‹*S* ∈ *C*› ‹*enough-new S*›
shows ‹*Extend C S* ∈ *C*›
⟨*proof*⟩

theorem *Extend-maximal*:
assumes ‹*subset-closed C*›
shows ‹*maximal C* (*Extend C S*)›
⟨*proof*⟩

definition *witnessed* :: ‹'*fm set* ⇒ *bool*› **where**
‹*witnessed S* ≡ ∀ *p* ∈ *S*. ∃ *S'*. *infinite* (*Collect is-param* − *params S'*) ∧ *witness p S'* ⊆ *S*›

theorem *Extend-witnessed*:
assumes ‹$prop_A$ *Ks C*› ‹*finite-char C*› ‹*S* ∈ *C*› ‹*enough-new S*›
shows ‹*witnessed* (*Extend C S*)›
⟨*proof*⟩

abbreviation *mk-mcs* :: ‹'*fm set set* ⇒ '*fm set* ⇒ '*fm set*› **where**
‹*mk-mcs C S* ≡ *Extend* (*mk-alt-fin C*) *S*›

theorem *mk-mcs-rmaximal*: ‹*maximal C* (*mk-mcs C S*)›

⟨*proof*⟩

theorem *mk-mcs-witnessed*:
assumes ‹$prop_E$ *Ks C*› ‹*S* ∈ *C*› ‹*enough-new S*›
shows ‹*witnessed* (*mk-mcs C S*)›
⟨*proof*⟩

1.4 Hintikka Sets

lemma *mk-mcs-hintikka*:
assumes ‹$prop_E$ *Ks C*› ‹*S* ∈ *C*› ‹*enough-new S*›
shows ‹$prop_H$ *Ks* (*mk-mcs C S*)›
⟨*proof*⟩

end

locale *Hintikka* = *Maximal-Consistency map-fm params-fm is-param Ks*
for
map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
params-fm :: ‹'*fm* ⇒ '*x set*› **and**
is-param :: ‹'*x* ⇒ *bool*› **and**
Ks :: ‹('*x*, '*fm*) *kind list*› +
fixes *H* :: ‹'*fm set*›
assumes *hintikka*: ‹$prop_H$ *Ks H*›
begin

lemma sat_H: ‹*K* ∈ *set Ks* ⟹ sat_H *K H*›
⟨*proof*⟩

end

context *Maximal-Consistency*
begin

theorem *mk-mcs-Hintikka*:
assumes ‹$prop_E$ *Ks C*› ‹*S* ∈ *C*› ‹*enough-new S*›
shows ‹*Hintikka map-fm params-fm is-param Ks* (*mk-mcs C S*)›
⟨*proof*⟩

end

1.5 Derivational Consistency

locale *Derivational-Kind* = *Consistency-Kind map-fm params-fm is-param K*
for
map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
params-fm :: ‹'*fm* ⇒ '*x set*› **and**
is-param :: ‹'*x* ⇒ *bool*› **and**

K :: ‹('*x*, '*fm*) *kind*› +
fixes *consistent* :: ‹'*fm set* ⇒ *bool*› (‹⊢ -› [*51*] *50*)
assumes *kind*: ‹*infinite* (*UNIV* :: '*fm set*) ⟹ sat_E *K* {*A*. *enough-new A* ∧ ⊢ *A*}›

locale *Derivational-Consistency* = *Maximal-Consistency map-fm params-fm is-param Ks*
for
map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
params-fm :: ‹'*fm* ⇒ '*x set*› **and**
is-param :: ‹'*x* ⇒ *bool*› **and**
Ks :: ‹('*x*, '*fm*) *kind list*› +
fixes *consistent* :: ‹'*fm set* ⇒ *bool*› (‹⊢ -› [*51*] *50*)
assumes *all-consistent*: ‹*infinite* (*UNIV* :: '*fm set*) ⟹ $prop_E$ *Ks* {*A*. *enough-new A* ∧ ⊢ *A*}›
begin

theorem *Consistency*: ‹$prop_E$ *Ks* {*A*. *enough-new A* ∧ ⊢ *A*}›
⟨*proof*⟩

end

1.6 Weak Derivational Consistency

locale *Weak-Derivational-Kind* = *Consistency-Kind map-fm params-fm is-param K*
for
map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
params-fm :: ‹'*fm* ⇒ '*x set*› **and**
is-param :: ‹'*x* ⇒ *bool*› **and**
K :: ‹('*x*, '*fm*) *kind*› +
fixes *consistent* :: ‹'*fm list* ⇒ *bool*› (‹⊢ -› [*51*] *50*)
assumes *kind*: ‹*infinite* (*Collect is-param*) ⟹ sat_E *K* {*S*. ∃ *A*. *set A* = *S* ∧ ⊢ *A*}›

locale *Weak-Derivational-Consistency* = *Maximal-Consistency map-fm params-fm is-param Ks*
for
map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
params-fm :: ‹'*fm* ⇒ '*x set*› **and**
is-param :: ‹'*x* ⇒ *bool*› **and**
Ks :: ‹('*x*, '*fm*) *kind list*› +
fixes *consistent* :: ‹'*fm list* ⇒ *bool*› (‹⊢ -› [*51*] *50*)
assumes *Consistency*: ‹*infinite* (*UNIV* :: '*x set*) ⟹ $prop_E$ *Ks* {*S*. ∃ *A*. *set A* = *S* ∧ ⊢ *A*}›

1.7 Conflicts

locale *Confl* = *Params map-fm params-fm is-param*
  for
    map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
    params-fm :: ‹'*fm* ⇒ '*x set*› **and**
    is-param :: ‹'*x* ⇒ *bool*› +
  fixes *classify* :: ‹'*fm list* ⇒ '*fm list* ⇒ *bool*› (**infix** ‹⇝✗› *50*)
  assumes *confl-map*: ‹⋀*ps qs f*. *ps* ⇝✗ *qs* ⟹ *map* (*map-fm f*) *ps* ⇝✗ *map* (*map-fm f*) *qs*›
begin

inductive *cond* **where**
  cond [*intro*!]: ‹*ps* ⇝✗ *qs* ⟹ *cond ps* (λ- *S*. *set qs* ∩ *S* = {})›

inductive-cases *condE*[*elim*!]: ‹*cond ps Q*›

inductive *hint* **where**
  hint [*intro*!]: ‹(⋀*ps qs q*. *ps* ⇝✗ *qs* ⟹ *set ps* ⊆ *H* ⟹ *q* ∈ *set qs* ⟹ *q* ∉ *H*) ⟹ *hint H*›

declare *hint.simps*[*simp*]

abbreviation *kind* :: ‹('*x*, '*fm*) *kind*› **where**
  ‹*kind* ≡ *Cond cond hint*›

end

sublocale *Confl* ⊆ *Consistency-Kind map-fm params-fm is-param kind*
⟨*proof*⟩

locale *Derivational-Confl* = *Confl map-fm params-fm is-param classify*
  for
    map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
    params-fm :: ‹'*fm* ⇒ '*x set*› **and**
    is-param :: ‹'*x* ⇒ *bool*› **and**
    classify :: ‹'*fm list* ⇒ '*fm list* ⇒ *bool*› (**infix** ‹⇝✗› *50*) +
  fixes *consistent* :: ‹'*fm set* ⇒ *bool*› (‹⊢ -› [*51*] *50*)
  assumes *consistent*: ‹⋀*S ps qs x*. *set ps* ⊆ *S* ⟹ *ps* ⇝✗ *qs* ⟹ *x* ∈ *set qs* ⟹ *x* ∈ *S* ⟹ ¬ ⊢ *S*›

sublocale *Derivational-Confl* ⊆ *Derivational-Kind map-fm params-fm is-param kind consistent*
  ⟨*proof*⟩

locale *Weak-Derivational-Confl* = *Confl map-fm params-fm is-param classify*
  for
    map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**

params-fm :: ‹*'fm* ⇒ *'x set*› **and**
is-param :: ‹*'x* ⇒ *bool*› **and**
classify :: ‹*'fm list* ⇒ *'fm list* ⇒ *bool*› (**infix** ‹$\rightsquigarrow_{\boldsymbol{\times}}$› *50*) +
fixes *consistent* :: ‹*'fm list* ⇒ *bool*› (‹⊢ -› [*51*] *50*)
assumes *consistent*: ‹$\bigwedge$*A ps qs x. set ps* ⊆ *set A* ⟹ *ps* $\rightsquigarrow_{\boldsymbol{\times}}$ *qs* ⟹ *x* ∈ *set qs* ⟹ *x* ∈ *set A* ⟹ ¬ ⊢ *A*›

sublocale *Weak-Derivational-Confl* ⊆ *Weak-Derivational-Kind map-fm params-fm is-param kind consistent*
⟨*proof*⟩

1.8 Alpha

locale *Alpha* = *Params map-fm params-fm is-param*
for
map-fm :: ‹(*'x* ⇒ *'x*) ⇒ *'fm* ⇒ *'fm*› **and**
params-fm :: ‹*'fm* ⇒ *'x set*› **and**
is-param :: ‹*'x* ⇒ *bool*› +
fixes *classify* :: ‹*'fm list* ⇒ *'fm list* ⇒ *bool*› (**infix** ‹$\rightsquigarrow_{\alpha}$› *50*)
assumes *alpha-map*: ‹$\bigwedge$*ps qs f. ps* $\rightsquigarrow_{\alpha}$ *qs* ⟶ *map* (*map-fm f*) *ps* $\rightsquigarrow_{\alpha}$ *map* (*map-fm f*) *qs*›
begin

inductive *cond* **where**
cond [*intro*!]: ‹*ps* $\rightsquigarrow_{\alpha}$ *qs* ⟹ *cond ps* (λ*C S. set qs* ∪ *S* ∈ *C*)›

inductive-cases *condE*[*elim*!]: ‹*cond ps Q*›

inductive *hint* **where**
hint [*intro*!]: ‹(⋀*ps qs q. ps* $\rightsquigarrow_{\alpha}$ *qs* ⟹ *set ps* ⊆ *H* ⟹ *q* ∈ *set qs* ⟹ *q* ∈ *H*) ⟹ *hint H*›

declare *hint.simps*[*simp*]

abbreviation *kind* :: ‹(*'x*, *'fm*) *kind*› **where**
‹*kind* ≡ *Cond cond hint*›

end

sublocale *Alpha* ⊆ *Consistency-Kind map-fm params-fm is-param kind*
⟨*proof*⟩

locale *Derivational-Alpha* = *Alpha map-fm params-fm is-param classify*
for
map-fm :: ‹(*'x* ⇒ *'x*) ⇒ *'fm* ⇒ *'fm*› **and**
params-fm :: ‹*'fm* ⇒ *'x set*› **and**
is-param :: ‹*'x* ⇒ *bool*› **and**
classify :: ‹*'fm list* ⇒ *'fm list* ⇒ *bool*› (**infix** ‹$\rightsquigarrow_{\alpha}$› *50*) +
fixes *consistent* :: ‹*'fm set* ⇒ *bool*› (‹⊢ -› [*51*] *50*)

assumes *consistent*: ‹$\bigwedge S\ ps\ qs.\ set\ ps \subseteq S \Longrightarrow ps \rightsquigarrow_\alpha qs \Longrightarrow \vdash S \Longrightarrow \vdash set\ qs \cup S$›

sublocale *Derivational-Alpha* $\subseteq$ *Derivational-Kind map-fm params-fm is-param kind consistent*
⟨*proof*⟩

locale *Weak-Derivational-Alpha* = *Alpha map-fm params-fm is-param classify*
for
map-fm :: ‹$('x \Rightarrow 'x) \Rightarrow 'fm \Rightarrow 'fm$› **and**
params-fm :: ‹$'fm \Rightarrow 'x\ set$› **and**
is-param :: ‹$'x \Rightarrow bool$› **and**
classify :: ‹$'fm\ list \Rightarrow 'fm\ list \Rightarrow bool$› (**infix** ‹$\rightsquigarrow_\alpha$› 50) +
fixes *consistent* :: ‹$'fm\ list \Rightarrow bool$› (‹$\vdash$ -› [51] 50)
assumes *consistent*: ‹$\bigwedge A\ ps\ qs.\ set\ ps \subseteq set\ A \Longrightarrow ps \rightsquigarrow_\alpha qs \Longrightarrow \vdash A \Longrightarrow \vdash qs\ @\ A$›

sublocale *Weak-Derivational-Alpha* $\subseteq$ *Weak-Derivational-Kind map-fm params-fm is-param kind consistent*
⟨*proof*⟩

1.9 Beta

locale *Beta* = *Params map-fm params-fm is-param*
for
map-fm :: ‹$('x \Rightarrow 'x) \Rightarrow 'fm \Rightarrow 'fm$› **and**
params-fm :: ‹$'fm \Rightarrow 'x\ set$› **and**
is-param :: ‹$'x \Rightarrow bool$› +
fixes *classify* :: ‹$'fm\ list \Rightarrow 'fm\ list \Rightarrow bool$› (**infix** ‹$\rightsquigarrow_\beta$› 50)
assumes *beta-map*: ‹$\bigwedge ps\ qs\ f.\ ps \rightsquigarrow_\beta qs \longrightarrow map\ (map\text{-}fm\ f)\ ps \rightsquigarrow_\beta map\ (map\text{-}fm\ f)\ qs$›
begin

inductive *cond* **where**
cond [*intro*!]: ‹$ps \rightsquigarrow_\beta qs \Longrightarrow cond\ ps\ (\lambda C\ S.\ \exists q \in set\ qs.\ \{q\} \cup S \in C)$›

inductive-cases *condE*[*elim*!]: ‹*cond ps Q*›

inductive *hint* **where**
hint [*intro*!]: ‹$(\bigwedge ps\ qs.\ ps \rightsquigarrow_\beta qs \Longrightarrow set\ ps \subseteq H \Longrightarrow \exists q \in set\ qs.\ q \in H) \Longrightarrow hint\ H$›

declare *hint.simps*[*simp*]

abbreviation *kind* :: ‹$('x,\ 'fm)\ kind$› **where**
‹*kind* ≡ *Cond cond hint*›

end

sublocale *Beta* $\subseteq$ *Consistency-Kind map-fm params-fm is-param kind*
⟨*proof*⟩

locale *Derivational-Beta* = *Beta map-fm params-fm is-param classify*
  for
    map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
    params-fm :: ‹'*fm* ⇒ '*x set*› **and**
    is-param :: ‹'*x* ⇒ *bool*› **and**
    classify :: ‹'*fm list* ⇒ '*fm list* ⇒ *bool*› (**infix** ‹$\rightsquigarrow_\beta$› 50) +
  fixes *consistent* :: ‹'*fm set* ⇒ *bool*› (‹⊢ -› [51] 50)
  assumes *consistent*: ‹$\bigwedge$*S ps qs*. *set ps* ⊆ *S* ⟹ *ps* $\rightsquigarrow_\beta$ *qs* ⟹ ⊢ *S* ⟹ ∃ *q* ∈ *set qs*. ⊢ {*q*} ∪ *S*›

sublocale *Derivational-Beta* $\subseteq$ *Derivational-Kind map-fm params-fm is-param kind consistent*
⟨*proof*⟩

locale *Weak-Derivational-Beta* = *Beta map-fm params-fm is-param classify*
  for
    map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
    params-fm :: ‹'*fm* ⇒ '*x set*› **and**
    is-param :: ‹'*x* ⇒ *bool*› **and**
    classify :: ‹'*fm list* ⇒ '*fm list* ⇒ *bool*› (**infix** ‹$\rightsquigarrow_\beta$› 50) +
  fixes *consistent* :: ‹'*fm list* ⇒ *bool*› (‹⊢ -› [51] 50)
  assumes *consistent*: ‹$\bigwedge$*A ps qs*. *set ps* ⊆ *set A* ⟹ *ps* $\rightsquigarrow_\beta$ *qs* ⟹ ⊢ *A* ⟹ ∃ *q* ∈ *set qs*. ⊢ *q* # *A*›

sublocale *Weak-Derivational-Beta* $\subseteq$ *Weak-Derivational-Kind map-fm params-fm is-param kind consistent*
⟨*proof*⟩

1.10 Gamma

locale *Gamma* = *Params map-fm params-fm is-param*
  for
    map-tm :: ‹('*x* ⇒ '*x*) ⇒ '*tm* ⇒ '*tm*› **and**
    map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
    params-fm :: ‹'*fm* ⇒ '*x set*› **and**
    is-param :: ‹'*x* ⇒ *bool*› +
  fixes *classify* :: ‹'*fm list* ⇒ ('*fm set* ⇒ '*tm set*) × ('*tm* ⇒ '*fm list*) ⇒ *bool*› (**infix** ‹$\rightsquigarrow_\gamma$› 50)
  assumes *gamma-map*: ‹$\bigwedge$*ps F qs f*. *ps* $\rightsquigarrow_\gamma$ (*F*, *qs*) ⟹ (∃ *G rs*. *map* (*map-fm f*) *ps* $\rightsquigarrow_\gamma$ (*G*, *rs*) ∧
      (∀ *S*. *map-tm f* ‘ *F S* ⊆ *G* (*map-fm f* ‘ *S*)) ∧
      (∀ *t*. *map* (*map-fm f*) (*qs t*) = *rs* (*map-tm f t*)))›
    and *gamma-mono*: ‹$\bigwedge$*ps F qs S S'*. *ps* $\rightsquigarrow_\gamma$ (*F*, *qs*) ⟹ *S* ⊆ *S'* ⟹ *F S* ⊆ *F S'*›
    and *gamma-fin*: ‹$\bigwedge$*ps F qs t A*. *ps* $\rightsquigarrow_\gamma$ (*F*, *qs*) ⟹ *t* ∈ *F A* ⟹ ∃ *B* ⊆ *A*. *finite*

$B \wedge t \in F\ B\rangle$
begin

inductive *cond* **where**
cond [*intro*!]: ‹$ps \rightsquigarrow_\gamma (F,\ qs) \Longrightarrow cond\ ps\ (\lambda C\ S.\ \forall\, t \in F\ S.\ set\ (qs\ t) \cup S \in C)$›

inductive-cases *condE*[*elim*!]: ‹*cond ps Q*›

inductive *hint* **where**
hint [*intro*!]: ‹$(\bigwedge ps\ F\ qs.\ ps \rightsquigarrow_\gamma (F,\ qs) \Longrightarrow set\ ps \subseteq H \Longrightarrow (\forall\, t \in F\ H.\ set\ (qs\ t) \subseteq H)) \Longrightarrow hint\ H$›

declare *hint.simps*[*simp*]

abbreviation *kind* :: ‹$('x,\ 'fm)\ kind$› **where**
‹*kind* ≡ *Cond cond hint*›

end

sublocale *Gamma* ⊆ *Consistency-Kind map-fm params-fm is-param kind*
⟨*proof*⟩

locale *Derivational-Gamma* = *Gamma map-tm map-fm params-fm is-param classify*
for
map-tm :: ‹$('x \Rightarrow 'x) \Rightarrow 'tm \Rightarrow 'tm$› **and**
map-fm :: ‹$('x \Rightarrow 'x) \Rightarrow 'fm \Rightarrow 'fm$› **and**
params-fm :: ‹$'fm \Rightarrow 'x\ set$› **and**
is-param :: ‹$'x \Rightarrow bool$› **and**
classify :: ‹$'fm\ list \Rightarrow ('fm\ set \Rightarrow 'tm\ set) \times ('tm \Rightarrow 'fm\ list) \Rightarrow bool$› (**infix** ‹$\rightsquigarrow_\gamma$› 50) +
fixes *consistent* :: ‹$'fm\ set \Rightarrow bool$› (‹⊢ -› [51] 50)
assumes *consistent*: ‹$\bigwedge S\ ps\ F\ qs\ t.\ set\ ps \subseteq S \Longrightarrow ps \rightsquigarrow_\gamma (F,\ qs) \Longrightarrow t \in F\ S \Longrightarrow \vdash S \Longrightarrow \vdash set\ (qs\ t) \cup S$›

sublocale *Derivational-Gamma* ⊆ *Derivational-Kind map-fm params-fm is-param kind consistent*
⟨*proof*⟩

locale *Weak-Derivational-Gamma* = *Gamma map-tm map-fm params-fm is-param classify*
for
map-tm :: ‹$('x \Rightarrow 'x) \Rightarrow 'tm \Rightarrow 'tm$› **and**
map-fm :: ‹$('x \Rightarrow 'x) \Rightarrow 'fm \Rightarrow 'fm$› **and**
params-fm :: ‹$'fm \Rightarrow 'x\ set$› **and**
is-param :: ‹$'x \Rightarrow bool$› **and**
classify :: ‹$'fm\ list \Rightarrow ('fm\ set \Rightarrow 'tm\ set) \times ('tm \Rightarrow 'fm\ list) \Rightarrow bool$› (**infix** ‹$\rightsquigarrow_\gamma$› 50) +
fixes *consistent* :: ‹$'fm\ list \Rightarrow bool$› (‹⊢ -› [51] 50)

assumes *consistent*: ‹$\bigwedge A\ ps\ F\ qs\ t.\ set\ ps \subseteq set\ A \Longrightarrow ps \rightsquigarrow_\gamma (F,\ qs) \Longrightarrow t \in F\ (set\ A) \Longrightarrow \vdash A \Longrightarrow \vdash qs\ t\ @\ A$›

sublocale *Weak-Derivational-Gamma* ⊆ *Weak-Derivational-Kind map-fm params-fm is-param kind consistent*
⟨*proof*⟩

locale *Gamma-UNIV* = *Params map-fm params-fm is-param*
for
map-tm :: ‹$('x \Rightarrow 'x) \Rightarrow 'tm \Rightarrow 'tm$› **and**
map-fm :: ‹$('x \Rightarrow 'x) \Rightarrow 'fm \Rightarrow 'fm$› **and**
params-fm :: ‹$'fm \Rightarrow 'x\ set$› **and**
is-param :: ‹$'x \Rightarrow bool$› +
fixes *classify* :: ‹$'fm\ list \Rightarrow ('tm \Rightarrow 'fm\ list) \Rightarrow bool$› (**infix** ‹$\rightsquigarrow_\gamma''$› 50)
assumes *gamma-map-UNIV*: ‹$\bigwedge ps\ qs\ f.\ ps \rightsquigarrow_\gamma' qs \Longrightarrow \exists rs.\ map\ (map\text{-}fm\ f)\ ps \rightsquigarrow_\gamma' rs \wedge$
$(\forall t.\ map\ (map\text{-}fm\ f)\ (qs\ t) = rs\ (map\text{-}tm\ f\ t))$›
begin

abbreviation (*input*) *classify-UNIV* **where**
‹$classify\text{-}UNIV \equiv \lambda ps\ (F,\ qs).\ (F = (\lambda\text{-}.\ UNIV)) \wedge ps \rightsquigarrow_\gamma' qs$›

end

sublocale *Gamma-UNIV* ⊆ *Gamma map-tm map-fm params-fm is-param classify-UNIV*
⟨*proof*⟩

1.11 Delta

locale *Delta* = *Params map-fm params-fm is-param*
for
map-fm :: ‹$('x \Rightarrow 'x) \Rightarrow 'fm \Rightarrow 'fm$› **and**
params-fm :: ‹$'fm \Rightarrow 'x\ set$› **and**
is-param :: ‹$'x \Rightarrow bool$› +
fixes δ :: ‹$'fm \Rightarrow 'x \Rightarrow 'fm\ list$›
assumes *delta-map*: ‹$\bigwedge p\ f\ x.\ is\text{-}param\ x \Longrightarrow is\text{-}subst\ f \Longrightarrow \delta\ (map\text{-}fm\ f\ p)\ (f\ x) = map\ (map\text{-}fm\ f)\ (\delta\ p\ x)$›
begin

abbreviation ‹$kind \equiv Wits\ \delta$›
end

sublocale *Delta* ⊆ *Consistency-Kind map-fm params-fm is-param kind*
⟨*proof*⟩

locale *Derivational-Delta* = *Delta map-fm params-fm is-param* δ
for
map-fm :: ‹$('x \Rightarrow 'x) \Rightarrow 'fm \Rightarrow 'fm$› **and**

params-fm :: ‹*'fm* ⇒ *'x set*› **and**
is-param :: ‹*'x* ⇒ *bool*› **and**
δ :: ‹*'fm* ⇒ *'x* ⇒ *'fm list*› +
fixes *consistent* :: ‹*'fm set* ⇒ *bool*› (‹⊢ -› [*51*] *50*)
assumes *consistent*: ‹⋀*S p x*. *p* ∈ *S* ⟹ *is-param x* ⟹ *x* ∉ *params S* ⟹ ⊢ *S* ⟹ ⊢ *set* (*δ p x*) ∪ *S*›

sublocale *Derivational-Delta* ⊆ *Derivational-Kind map-fm params-fm is-param kind consistent*
⟨*proof*⟩

locale *Weak-Derivational-Delta* = *Delta map-fm params-fm is-param δ*
for
map-fm :: ‹(*'x* ⇒ *'x*) ⇒ *'fm* ⇒ *'fm*› **and**
params-fm :: ‹*'fm* ⇒ *'x set*› **and**
is-param :: ‹*'x* ⇒ *bool*› **and**
δ :: ‹*'fm* ⇒ *'x* ⇒ *'fm list*› +
fixes *consistent* :: ‹*'fm list* ⇒ *bool*› (‹⊢ -› [*51*] *50*)
assumes *consistent*: ‹⋀*A p x*. *p* ∈ *set A* ⟹ *is-param x* ⟹ *x* ∉ *params* (*set A*) ⟹ ⊢ *A* ⟹ ⊢ *δ p x* @ *A*›

sublocale *Weak-Derivational-Delta* ⊆ *Weak-Derivational-Kind map-fm params-fm is-param kind consistent*
⟨*proof*⟩

1.12 Modal

The Hintikka property you want depends on the concrete logic. See Term-Modal Logics by Fitting, Thalmann and Voronkov, p. 156 bottom.

locale *Modal* = *Params map-fm params-fm is-param*
for
map-fm :: ‹(*'x* ⇒ *'x*) ⇒ *'fm* ⇒ *'fm*› **and**
params-fm :: ‹*'fm* ⇒ *'x set*› **and**
is-param :: ‹*'x* ⇒ *bool*› +
fixes *classify* :: ‹*'fm list* ⇒ (*'fm set* ⇒ *'fm set*) × *'fm list* ⇒ *bool*› (**infix** ‹⇝$_{\square}$›
50)
and *hint* :: ‹*'fm set* ⇒ *bool*›
assumes *modal-map*: ‹⋀*ps F qs f*. *ps* ⇝$_{\square}$ (*F*, *qs*) ⟹ ∃ *G*. *map* (*map-fm f*) *ps* ⇝$_{\square}$ (*G*, *map* (*map-fm f*) *qs*) ∧
(∀ *S*. *map-fm f* ‘ *F S* ⊆ *G* (*map-fm f* ‘ *S*))›
and *modal-mono*: ‹⋀*ps F qs S S'*. *ps* ⇝$_{\square}$ (*F*, *qs*) ⟹ *S* ⊆ *S'* ⟹ *F S* ⊆ *F S'*›
and *modal-fin*: ‹⋀*ps F qs S A*. *ps* ⇝$_{\square}$ (*F*, *qs*) ⟹ *finite A* ⟹ *A* ⊆ *F S* ⟹ ∃ *S'* ⊆ *S*. *finite S'* ∧ *A* ⊆ *F S'*›
begin

inductive *cond* **where**
cond [*intro*!]: ‹*ps* ⇝$_{\square}$ (*F*, *qs*) ⟹ *cond ps* (λ*C S*. *set qs* ∪ *F S* ∈ *C*)›

inductive-cases *condE*[*elim*!]: ‹*cond ps Q*›

abbreviation *kind* :: ‹('*x*, '*fm*) *kind*› **where**
‹*kind* ≡ *Cond cond hint*›

end

locale *ModalH* = *Modal map-fm params-fm is-param classify hint*
for
map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
params-fm :: ‹'*fm* ⇒ '*x set*› **and**
is-param :: ‹'*x* ⇒ *bool*› **and**
classify :: ‹'*fm list* ⇒ ('*fm set* ⇒ '*fm set*) × '*fm list* ⇒ *bool*› (**infix** ‹⇝$_\Box$› *50*)
and
hint :: ‹'*fm set* ⇒ *bool*› +
assumes *modal-hintikka*: ‹⋀*C S*. *sat*$_E$ *kind C* ⟹ *S* ∈ *C* ⟹ *maximal C S* ⟹ *sat*$_H$ *kind S*›

sublocale *ModalH* ⊆ *Consistency-Kind map-fm params-fm is-param kind*
⟨*proof*⟩

locale *Derivational-Modal* = *ModalH map-fm params-fm is-param classify*
for
map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
params-fm :: ‹'*fm* ⇒ '*x set*› **and**
is-param :: ‹'*x* ⇒ *bool*› **and**
classify :: ‹'*fm list* ⇒ ('*fm set* ⇒ '*fm set*) × '*fm list* ⇒ *bool*› (**infix** ‹⇝$_\Box$› *50*)
+
fixes *consistent* :: ‹'*fm set* ⇒ *bool*› (‹⊢ -› [*51*] *50*)
assumes *consistent*: ‹⋀*S ps F qs*. *set ps* ⊆ *S* ⟹ *ps* ⇝$_\Box$ (*F*, *qs*) ⟹ ⊢ *S* ⟹ ⊢ *set qs* ∪ *F S*›
and *params-subset*: ‹⋀*ps F qs S*. *ps* ⇝$_\Box$ (*F*, *qs*) ⟹ *params* (*F S*) ⊆ *params S*›

sublocale *Derivational-Modal* ⊆ *Derivational-Kind map-fm params-fm is-param kind consistent*
⟨*proof*⟩

locale *Weak-Derivational-Modal* = *ModalH map-fm params-fm is-param classify*
for
map-fm :: ‹('*x* ⇒ '*x*) ⇒ '*fm* ⇒ '*fm*› **and**
params-fm :: ‹'*fm* ⇒ '*x set*› **and**
is-param :: ‹'*x* ⇒ *bool*› **and**
classify :: ‹'*fm list* ⇒ ('*fm set* ⇒ '*fm set*) × '*fm list* ⇒ *bool*› (**infix** ‹⇝$_\Box$› *50*)
+
fixes *consistent* :: ‹'*fm list* ⇒ *bool*› (‹⊢ -› [*51*] *50*)
assumes *consistent*: ‹⋀*S ps F qs S'*. *set ps* ⊆ *set S* ⟹ *ps* ⇝$_\Box$ (*F*, *qs*) ⟹ ⊢ *S* ⟹ *set S'* = *F* (*set S*) ⟹ ⊢ *qs* @ *S'*›
and *params-subset*: ‹⋀*ps F qs S*. *ps* ⇝$_\Box$ (*F*, *qs*) ⟹ *params* (*F S*) ⊆ *params*

S›
and *F-size*: ‹$\bigwedge ps\ F\ qs\ S.\ ps \rightsquigarrow_{\square} (F,\ qs) \Longrightarrow |F\ S| \leq o\ |S|$›

sublocale *Weak-Derivational-Modal* $\subseteq$ *Weak-Derivational-Kind map-fm params-fm is-param kind consistent*
⟨*proof*⟩

end

Chapter 2

Example: First-Order Logic with Restricted Instantiation

theory *Example-Bounded-FOL* **imports**
Abstract-Consistency-Property
begin

2.1 Syntax

datatype (*params-tm*: *'f*) *tm*
= *Var nat* (‹#›)
| *Fun 'f ‹'f tm list›* (‹◯›)

abbreviation *Const* (‹⋆›) **where** ‹⋆*a* ≡ ◯*a* []›

datatype (*params-fm*: *'f*, *'p*) *fm*
= *Fls* (‹⊥›)
| *Pre 'p ‹'f tm list›* (‹•›)
| *Imp ‹('f, 'p) fm›* ‹*('f, 'p) fm›* (**infixr** ‹⟶› *55*)
| *Uni ‹('f, 'p) fm›* (‹∀›)

abbreviation *Neg* :: ‹*('f, 'p) fm* ⇒ *('f, 'p) fm›* (‹¬ -› [*70*] *70*) **where**
‹¬ *p* ≡ *p* ⟶ ⊥›

abbreviation *has-subterm* :: ‹*('f, 'p) fm›* **where**
‹*has-subterm* ≡ •*undefined* [#*0*] ⟶ •*undefined* [#*0*]›

abbreviation *with-subterm* :: ‹*('f, 'p) fm* ⇒ *('f, 'p) fm›* **where**
‹*with-subterm p* ≡ *has-subterm* ⟶ *p*›

2.2 Semantics

datatype (*'a*, *'f*, *'p*) *model* = *Model ‹'a set› ‹nat* ⇒ *'a› ‹'f* ⇒ *'a list* ⇒ *'a› ‹'p* ⇒ *'a list* ⇒ *bool›*

primrec *wf-model* :: ‹('a, 'f, 'p) *model* ⇒ *bool*› **where**
  ‹*wf-model* (*Model U E F G*) ⟷ (∀ *n*. *E n* ∈ *U*) ∧ (∀ *f ts*. *F f ts* ∈ *U*)›

fun *semantics-tm* :: ‹(*nat* ⇒ 'a) × ('f ⇒ 'a *list* ⇒ 'a) ⇒ 'f *tm* ⇒ 'a› (‹⦇-⦈›)
where
  ‹⦇(*E*, -)⦈ (#*n*) = *E n*›
| ‹⦇(*E*, *F*)⦈ (○*f ts*) = *F f* (*map* ⦇(*E*, *F*)⦈ *ts*)›

primrec *add-env* :: ‹'a ⇒ (*nat* ⇒ 'a) ⇒ *nat* ⇒ 'a› (**infix** ‹≫› 0) **where**
  ‹(*t* ≫ *s*) 0 = *t*›
| ‹(*t* ≫ *s*) (*Suc n*) = *s n*›

fun *semantics-fm* :: ‹('a, 'f, 'p) *model* ⇒ ('f, 'p) *fm* ⇒ *bool*› (**infix** ‹⊨› 50) **where**
  ‹(- ⊨ ⊥) = *False*›
| ‹(*Model* - *E F G* ⊨ ∘*P ts*) = *G P* (*map* ⦇(*E*, *F*)⦈ *ts*)›
| ‹(*Model U E F G* ⊨ *p* ⟶ *q*) = (*Model U E F G* ⊨ *p* ⟶ *Model U E F G* ⊨ *q*)›
| ‹(*Model U E F G* ⊨ ∀ *p*) = (∀ *x* ∈ *U*. *Model U* (*x* ≫ *E*) *F G* ⊨ *p*)›

2.3 Operations

primrec *lift-tm* :: ‹'f *tm* ⇒ 'f *tm*› **where**
  ‹*lift-tm* (#*n*) = #(*n*+1)›
| ‹*lift-tm* (○*f ts*) = ○*f* (*map lift-tm ts*)›

primrec *sub-tm* :: ‹(*nat* ⇒ 'f *tm*) ⇒ 'f *tm* ⇒ 'f *tm*› **where**
  ‹*sub-tm s* (#*n*) = *s n*›
| ‹*sub-tm s* (○*f ts*) = ○*f* (*map* (*sub-tm s*) *ts*)›

primrec *sub-fm* :: ‹(*nat* ⇒ 'f *tm*) ⇒ ('f, 'p) *fm* ⇒ ('f, 'p) *fm*› **where**
  ‹*sub-fm* - ⊥ = ⊥›
| ‹*sub-fm s* (∘*P ts*) = ∘*P* (*map* (*sub-tm s*) *ts*)›
| ‹*sub-fm s* (*p* ⟶ *q*) = *sub-fm s p* ⟶ *sub-fm s q*›
| ‹*sub-fm s* (∀ *p*) = ∀ (*sub-fm* (#0 ≫ λ*n*. *lift-tm* (*s n*)) *p*)›

abbreviation *inst-single* :: ‹'f *tm* ⇒ ('f, 'p) *fm* ⇒ ('f, 'p) *fm*› (‹⟨-⟩›) **where**
  ‹⟨*t*⟩ ≡ *sub-fm* (*t* ≫ #)›

abbreviation *psub* :: ‹('f ⇒ 'g) ⇒ ('f, 'p) *fm* ⇒ ('g, 'p) *fm*› **where**
  ‹*psub f* ≡ *map-fm f id*›

primrec *size-fm* :: ‹('f, 'p) *fm* ⇒ *nat*› **where**
  ‹*size-fm* ⊥ = 1›
| ‹*size-fm* (∘- -) = 1›
| ‹*size-fm* (*p* ⟶ *q*) = 1 + *size-fm p* + *size-fm q*›
| ‹*size-fm* (∀ *p*) = 1 + *size-fm p*›

2.3.1 Lemmas

lemma *wf-model-tm* [*simp*]: ‹*wf-model* (*Model U E F G*) ⟹ ⦇(*E*, *F*)⦈ *t* ∈ *U*›
⟨*proof*⟩

lemma *size-sub-fm* [*simp*]: ‹*size-fm* (*sub-fm s p*) = *size-fm p*›
⟨*proof*⟩

lemma *upd-params-tm* [*simp*]: ‹*f* ∉ *params-tm t* ⟹ ⦇(*E*, *F*(*f* := *x*))⦈ *t* = ⦇(*E*, *F*)⦈ *t*›
⟨*proof*⟩

lemma *upd-params-fm* [*simp*]:
assumes ‹*f* ∉ *params-fm p*›
shows ‹*Model U E* (*F*(*f* := *x*)) *G* ⊨ *p* ⟷ *Model U E F G* ⊨ *p*›
⟨*proof*⟩

lemma *finite-params-tm* [*simp*]: ‹*finite* (*params-tm t*)›
⟨*proof*⟩

lemma *finite-params-fm′* [*simp*]: ‹*finite* (*params-fm p*)›
⟨*proof*⟩

lemma *env* [*simp*]: ‹*P* ((*x* ≫ *E*) *n*) = (*P x* ≫ λ*n*. *P* (*E n*)) *n*›
⟨*proof*⟩

lemma *lift-lemma*: ‹⦇(*x* ≫ *E*, *F*)⦈ (*lift-tm t*) = ⦇(*E*, *F*)⦈ *t*›
⟨*proof*⟩

lemma *sub-tm-semantics*: ‹⦇(*E*, *F*)⦈ (*sub-tm s t*) = ⦇(λ*n*. ⦇(*E*, *F*)⦈ (*s n*), *F*)⦈ *t*›
⟨*proof*⟩

lemma *sub-fm-semantics* [*simp*]:
‹*Model U E F G* ⊨ *sub-fm s p* ⟷ *Model U* (λ*n*. ⦇(*E*, *F*)⦈ (*s n*)) *F G* ⊨ *p*›
⟨*proof*⟩

lemma *map-tm-sub-tm* [*simp*]: ‹*map-tm f* (*sub-tm g t*) = *sub-tm* (*map-tm f o g*) (*map-tm f t*)›
⟨*proof*⟩

lemma *map-tm-lift-tm* [*simp*]: ‹*map-tm f* (*lift-tm t*) = *lift-tm* (*map-tm f t*)›
⟨*proof*⟩

lemma *psub-sub-fm*: ‹*psub f* (*sub-fm g p*) = *sub-fm* (*map-tm f o g*) (*psub f p*)›
⟨*proof*⟩

lemma *map-tm-inst-single*: ‹(*map-tm f o* (*u* ≫ #)) *t* = (*map-tm f u* ≫ #) *t*›
⟨*proof*⟩

lemma *psub-inst-single* [*simp*]: ‹*psub f* (⟨*t*⟩*p*) = ⟨*map-tm f t*⟩(*psub f p*)›

⟨*proof*⟩

2.4 Terms

primrec *terms-tm* :: ‹*'f tm ⇒ 'f tm set*› **where**
  ‹*terms-tm* (#*n*) = {#*n*}›
| ‹*terms-tm* (◯*f ts*) = {◯*f ts*} ∪ ⋃(*set* (*map terms-tm ts*))›

primrec *terms-fm* :: ‹(*'f*, *'p*) *fm ⇒ 'f tm set*› **where**
  ‹*terms-fm* ⊥ = {}›
| ‹*terms-fm* (•- *ts*) = ⋃(*set* (*map terms-tm ts*))›
| ‹*terms-fm* (*p* ⟶ *q*) = *terms-fm p* ∪ *terms-fm q*›
| ‹*terms-fm* (∀ *p*) = *terms-fm p*›

definition *terms* :: ‹(*'f*, *'p*) *fm set ⇒ 'f tm set*› **where**
  ‹*terms S* ≡ ⋃ *p* ∈ *S*. *terms-fm p*›

2.4.1 Lemmas

lemma *terms-mono*: ‹*S* ⊆ *S'* ⟹ *terms S* ⊆ *terms S'*›
  ⟨*proof*⟩

lemma *terms-tm-refl* [*intro*]: ‹*t* ∈ *terms-tm t*›
  ⟨*proof*⟩

lemma *terms-tm-trans* [*trans*]: ‹*s* ∈ *terms-tm t* ⟹ *r* ∈ *terms-tm s* ⟹ *r* ∈ *terms-tm t*›
  ⟨*proof*⟩

lemma *map-terms-tm* [*simp*]: ‹*terms-tm* (*map-tm f t*) = *map-tm f* ‘ *terms-tm t*›
  ⟨*proof*⟩

lemma *map-terms-fm* [*simp*]: ‹*terms-fm* (*map-fm f g p*) = *map-tm f* ‘ *terms-fm p*›
  ⟨*proof*⟩

lemma *terms-tm-closed* [*dest*]: ‹*t* ∈ *terms-tm s* ⟹ *terms-tm t* ⊆ *terms-tm s*›
  ⟨*proof*⟩

lemma *terms-fm-closed*: ‹*t* ∈ *terms-fm p* ⟹ *terms-tm t* ⊆ *terms-fm p*›
  ⟨*proof*⟩

lemma *terms-source*: ‹*t* ∈ *terms S* ⟹ ∃ *S'* ⊆ *S*. *finite S'* ∧ *t* ∈ *terms S'*›
  ⟨*proof*⟩

lemma *terms-tm-Fun*: ‹◯*f ts* ∈ *terms-tm s* ⟹ *t* ∈ *set ts* ⟹ *t* ∈ *terms-tm s*›
  ⟨*proof*⟩

lemma *terms-Fun*: ‹◯*f ts* ∈ *terms S* ⟹ *set ts* ⊆ *terms S*›
  ⟨*proof*⟩

2.5 Guard

definition *guard* :: ‹'*a* ⇒ '*a set* ⇒ '*a*› (**infix** ‹∈?› *50*) **where**
‹*x* ∈? *S* ≡ *if x* ∈ *S then x else SOME y*. *y* ∈ *S*›

lemma *guard-in*: ‹*S* ≠ {} ⟹ (*x* ∈? *S*) ∈ *S*›
⟨*proof*⟩

lemma *guard-refl* [*simp*]: ‹*t* ∈ *S* ⟹ *t* ∈? *S* = *t*›
⟨*proof*⟩

2.6 Model Existence

inductive
confl-class :: ‹('*f*, '*p*) *fm list* ⇒ ('*f*, '*p*) *fm list* ⇒ *bool*› (**infix** ‹⇝✗› *50*) **and**
alpha-class :: ‹('*f*, '*p*) *fm list* ⇒ ('*f*, '*p*) *fm list* ⇒ *bool*› (**infix** ‹⇝$_\alpha$› *50*) **and**
beta-class :: ‹('*f*, '*p*) *fm list* ⇒ ('*f*, '*p*) *fm list* ⇒ *bool*› (**infix** ‹⇝$_\beta$› *50*) **and**
gamma-class :: ‹('*f*, '*p*) *fm list* ⇒ (('*f*, '*p*) *fm set* ⇒ -) × ('*f tm* ⇒ -) ⇒ *bool*›
(**infix** ‹⇝$_\gamma$› *50*)
where
CFls [*intro*]: ‹[⊥] ⇝✗ [⊥]›
| *CNeg* [*intro*]: ‹[¬ (·*P ts*)] ⇝✗ [·*P ts*]›
| *CImpN* [*intro*]: ‹[¬ (*p* ⟶ *q*)] ⇝$_\alpha$ [*p*, ¬ *q*]›
| *CImpP* [*intro*]: ‹[*p* ⟶ *q*] ⇝$_\beta$ [¬ *p*, *q*]›
| *CAllP* [*intro*]: ‹[∀ *p*] ⇝$_\gamma$ (*terms*, λ*t*. [⟨*t*⟩*p*])›

fun δ :: ‹('*f*, '*p*) *fm* ⇒ '*f* ⇒ ('*f*, '*p*) *fm list*› **where**
‹δ (¬ ∀ *p*) *x* = [¬ ⟨⋆*x*⟩*p*]›
| ‹δ - - = []›

interpretation *P*: *Params psub params-fm* ‹λ-. *True*›
⟨*proof*⟩

interpretation *C*: *Confl psub params-fm* ‹λ-. *True*› *confl-class*
⟨*proof*⟩

interpretation *A*: *Alpha psub params-fm* ‹λ-. *True*› *alpha-class*
⟨*proof*⟩

interpretation *B*: *Beta psub params-fm* ‹λ-. *True*› *beta-class*
⟨*proof*⟩

interpretation *G*: *Gamma map-tm psub params-fm* ‹λ-. *True*› *gamma-class*
⟨*proof*⟩

interpretation *D*: *Delta psub params-fm* ‹λ-. *True*› δ
⟨*proof*⟩

abbreviation *Kinds* :: ‹('*f*, ('*f*, '*p*) *fm*) *kind list*› **where**

‹*Kinds* ≡ [*C.kind*, *A.kind*, *B.kind*, *G.kind*, *D.kind*]›

lemma *prop$_E$-Kinds* [*intro*]:
assumes ‹*P.sat$_E$ C.kind C*› ‹*P.sat$_E$ A.kind C*› ‹*P.sat$_E$ B.kind C*› ‹*P.sat$_E$ G.kind C*› ‹*P.sat$_E$ D.kind C*›
shows ‹*P.prop$_E$ Kinds C*›
⟨*proof*⟩

interpretation *Consistency-Kinds psub params-fm* ‹λ-. *True*› *Kinds*
⟨*proof*⟩

interpretation *Maximal-Consistency psub params-fm* ‹λ-. *True*› *Kinds*
⟨*proof*⟩

abbreviation *canonical* :: ‹('*f*, '*p*) *fm set* ⇒ ('*f tm*, '*f*, '*p*) *model*› **where**
‹*canonical H* ≡
Model (*terms H*) (λ*n*. #*n* ∈*?* *terms H*) (λ*f ts*. ◯*f ts* ∈*?* *terms H*) (λ*P ts*. •*P ts* ∈ *H*)›

lemma *wf-canonical*:
assumes ‹*terms H* ≠ {}›
shows ‹*wf-model* (*canonical H*)›
⟨*proof*⟩

lemma *canonical-tm-id* [*simp*]:
‹*t* ∈ *terms H* ⟹ ⦇(λ*n*. #*n* ∈*?* *terms H*, λ*P ts*. ◯*P ts* ∈*?* *terms H*)⦈ *t* = *t*›
⟨*proof*⟩

lemma *canonical-tm-id-map* [*simp*]:
‹*set ts* ⊆ *terms H* ⟹ *map* ⦇(λ*n*. #*n* ∈*?* *terms H*, λ*P ts*. ◯*P ts* ∈*?* *terms H*)⦈ *ts* = *ts*›
⟨*proof*⟩

locale *MyHintikka* = *Hintikka psub params-fm* ‹λ-. *True*› *Kinds H*
for *H* :: ‹('*f*, '*p*) *fm set*› +
assumes *terms-ne*: ‹*terms H* ≠ {}›
begin

lemmas
confl = *sat$_H$*[*of C.kind*] **and**
alpha = *sat$_H$*[*of A.kind*] **and**
beta = *sat$_H$*[*of B.kind*] **and**
gamma = *sat$_H$*[*of G.kind*] **and**
delta = *sat$_H$*[*of D.kind*]

theorem *model*: ‹(*p* ∈ *H* ⟶ *canonical H* ⊨ *p*) ∧ (¬ *p* ∈ *H* ⟶ ¬ *canonical H* ⊨ *p*)›
⟨*proof*⟩

end

theorem *model-existence*:
fixes $S :: \langle('f, 'p)\ fm\ set\rangle$
assumes $\langle P.prop_E\ Kinds\ C\rangle$
and $\langle S \in C\rangle$
and $\langle P.enough\text{-}new\ S\rangle$
and $\langle terms\ S \neq \{\}\rangle$
and $\langle p \in S\rangle$
shows $\langle canonical\ (mk\text{-}mcs\ C\ S) \models p\rangle$
⟨*proof*⟩

2.7 Compactness

abbreviation *semantics-set* :: $\langle('a, 'f, 'p)\ model \Rightarrow ('f, 'p)\ fm\ set \Rightarrow bool\rangle$ (**infix** $\langle\models\rangle$ *50*) **where**
$\langle M \models S \equiv \forall p \in S.\ M \models p\rangle$

lemma *compact-C*: $\langle P.sat_E\ C.kind\ \{S :: ('f, 'p)\ fm\ set.\ P.enough\text{-}new\ S \wedge$
$(\forall S' \subseteq S.\ finite\ S' \longrightarrow (\exists (U :: 'a\ set)\ E\ F\ G.\ wf\text{-}model\ (Model\ U\ E\ F\ G) \wedge$
$Model\ U\ E\ F\ G \models S'))\}\rangle$
⟨*proof*⟩

lemma *compact-A*: $\langle P.sat_E\ A.kind\ \{S :: ('f, 'p)\ fm\ set.\ P.enough\text{-}new\ S \wedge$
$(\forall S' \subseteq S.\ finite\ S' \longrightarrow (\exists (U :: 'a\ set)\ E\ F\ G.\ wf\text{-}model\ (Model\ U\ E\ F\ G) \wedge$
$Model\ U\ E\ F\ G \models S'))\}\rangle$
⟨*proof*⟩

lemma *compact-B*: $\langle P.sat_E\ B.kind\ \{S :: ('f, 'p)\ fm\ set.\ P.enough\text{-}new\ S \wedge$
$(\forall S' \subseteq S.\ finite\ S' \longrightarrow (\exists (U :: 'a\ set)\ E\ F\ G.\ wf\text{-}model\ (Model\ U\ E\ F\ G) \wedge$
$Model\ U\ E\ F\ G \models S'))\}\rangle$
⟨*proof*⟩

lemma *compact-G*: $\langle P.sat_E\ G.kind\ \{S :: ('f, 'p)\ fm\ set.\ P.enough\text{-}new\ S \wedge$
$(\forall S' \subseteq S.\ finite\ S' \longrightarrow (\exists (U :: 'a\ set)\ E\ F\ G.\ wf\text{-}model\ (Model\ U\ E\ F\ G) \wedge$
$Model\ U\ E\ F\ G \models S'))\}\rangle$
⟨*proof*⟩

lemma *compact-D*: $\langle P.sat_E\ D.kind\ \{S :: ('f, 'p)\ fm\ set.\ P.enough\text{-}new\ S \wedge$
$(\forall S' \subseteq S.\ finite\ S' \longrightarrow (\exists (U :: 'a\ set)\ E\ F\ G.\ wf\text{-}model\ (Model\ U\ E\ F\ G) \wedge$
$Model\ U\ E\ F\ G \models S'))\}\rangle$
⟨*proof*⟩

lemma *compact-prop*: $\langle P.prop_E\ Kinds\ \{S :: ('f, 'p)\ fm\ set.\ P.enough\text{-}new\ S \wedge$
$(\forall S' \subseteq S.\ finite\ S' \longrightarrow (\exists (U :: 'a\ set)\ E\ F\ G.\ wf\text{-}model\ (Model\ U\ E\ F\ G) \wedge$
$Model\ U\ E\ F\ G \models S'))\}\rangle$
⟨*proof*⟩

theorem *compactness*:

fixes $S :: \langle('f, 'p)\ fm\ set\rangle$
assumes $\langle P.enough\text{-}new\ S\rangle$
shows $\langle(\exists(U :: 'f\ tm\ set)\ E\ F\ G.\ wf\text{-}model\ (Model\ U\ E\ F\ G) \wedge Model\ U\ E\ F\ G \models S) \longleftrightarrow (\forall S' \subseteq S.\ finite\ S' \longrightarrow (\exists(U :: 'f\ tm\ set)\ E\ F\ G.\ wf\text{-}model\ (Model\ U\ E\ F\ G) \wedge Model\ U\ E\ F\ G \models S'))\rangle$
⟨*proof*⟩

2.8 Natural Deduction

locale *Natural-Deduction*
begin

inductive *ND-Set* :: $\langle('f, 'p)\ fm\ set \Rightarrow ('f, 'p)\ fm \Rightarrow bool\rangle$ (**infix** $\langle\vdash\rangle$ *50*) **where**
  Assm [*dest*]: $\langle p \in A \Longrightarrow A \vdash p\rangle$
| *FlsE* [*elim*]: $\langle A \vdash \bot \Longrightarrow A \vdash p\rangle$
| *ImpI* [*intro*]: $\langle\{p\} \cup A \vdash q \Longrightarrow A \vdash (p \longrightarrow q)\rangle$
| *ImpE* [*dest*]: $\langle A \vdash (p \longrightarrow q) \Longrightarrow A \vdash p \Longrightarrow A \vdash q\rangle$
| *UniI* [*intro*]: $\langle A \vdash \langle\star a\rangle p \Longrightarrow a \notin P.params\ (\{p\} \cup A) \Longrightarrow A \vdash \forall p\rangle$
| *UniE* [*dest*]: $\langle A \vdash \forall p \Longrightarrow t \in terms\ (\{p\} \cup A) \Longrightarrow A \vdash \langle t\rangle p\rangle$
| *Clas*: $\langle\{p \longrightarrow q\} \cup A \vdash p \Longrightarrow A \vdash p\rangle$

2.8.1 Soundness

theorem *soundness-set*:
  assumes $\langle A \vdash p\rangle$ $\langle wf\text{-}model\ (Model\ U\ E\ F\ G)\rangle$
  shows $\langle\forall q \in A.\ Model\ U\ E\ F\ G \models q \Longrightarrow Model\ U\ E\ F\ G \models p\rangle$
  ⟨*proof*⟩

2.8.2 Derivational Consistency

lemma *Boole*: $\langle\{\neg p\} \cup A \vdash \bot \Longrightarrow A \vdash p\rangle$
  ⟨*proof*⟩

sublocale *DC*: *Derivational-Confl psub params-fm* $\langle\lambda\text{-}.\ True\rangle$ *confl-class* $\langle\lambda A.\ \neg A \vdash \bot\rangle$
⟨*proof*⟩

sublocale *DA*: *Derivational-Alpha psub params-fm* $\langle\lambda\text{-}.\ True\rangle$ *alpha-class* $\langle\lambda A.\ \neg A \vdash \bot\rangle$
⟨*proof*⟩

sublocale *DB*: *Derivational-Beta psub params-fm* $\langle\lambda\text{-}.\ True\rangle$ *beta-class* $\langle\lambda A.\ \neg A \vdash \bot\rangle$
⟨*proof*⟩

sublocale *DG*: *Derivational-Gamma map-tm psub params-fm* $\langle\lambda\text{-}.\ True\rangle$ *gamma-class* $\langle\lambda A.\ \neg A \vdash \bot\rangle$
⟨*proof*⟩

sublocale *DD*: *Derivational-Delta psub params-fm* ‹λ-. *True*› δ ‹λA. ¬ A ⊩ ⊥›
⟨*proof*⟩

sublocale *Derivational-Consistency psub params-fm* ‹λ-. *True*› *Kinds* ‹λA. ¬ A ⊩ ⊥›
⟨*proof*⟩

2.8.3 Strong Completeness

lemma *with-subterm-elim*: ‹A ⊩ *with-subterm* p ⟹ A ⊩ p›
⟨*proof*⟩

theorem *strong-completeness*:
fixes p :: ‹('f, 'p) fm›
assumes *mod*: ‹⋀(U :: 'f tm set) E F G. wf-model (Model U E F G) ⟹
(∀ q ∈ A. Model U E F G ⊨ q) ⟹ Model U E F G ⊨ p›
and ‹P.enough-new A›
shows ‹A ⊩ p›
⟨*proof*⟩

2.8.4 Natural Deduction with Lists

inductive *ND-List* :: ‹('f, 'p) fm list ⇒ ('f, 'p) fm ⇒ bool› (**infix** ‹⊢› 50) **where**
Assm [*simp*]: ‹p ∈ set A ⟹ A ⊢ p›
| *FlsE* [*elim*]: ‹A ⊢ ⊥ ⟹ A ⊢ p›
| *ImpI* [*intro*]: ‹p # A ⊢ q ⟹ A ⊢ p ⟶ q›
| *ImpE* [*dest*]: ‹A ⊢ p ⟶ q ⟹ A ⊢ p ⟹ A ⊢ q›
| *UniI* [*intro*]: ‹A ⊢ ⟨⋆a⟩p ⟹ a ∉ P.params ({p} ∪ set A) ⟹ A ⊢ ∀ p›
| *UniE* [*dest*]: ‹A ⊢ ∀ p ⟹ t ∈ terms ({p} ∪ set A) ⟹ A ⊢ ⟨t⟩p›
| *Clas*: ‹(p ⟶ q) # A ⊢ p ⟹ A ⊢ p›

definition *bounded* :: ‹'a list ⇒ 'a set ⇒ ('a list ⇒ bool) ⇒ bool› **where**
‹*bounded* K A P ≡ set K ⊆ A ∧ (∀ B. set K ⊆ set B ⟶ set B ⊆ A ⟶ P B)›

lemma *bounded-one* [*elim*]:
assumes ‹*bounded* K A P› ‹⋀A. P A ⟹ Q A›
shows ‹*bounded* K A Q›
⟨*proof*⟩

lemma *bounded-two* [*elim*]:
assumes ‹*bounded* K A P› ‹*bounded* K' A Q› ‹⋀A. P A ⟹ Q A ⟹ R A›
shows ‹*bounded* (K @ K') A R›
⟨*proof*⟩

lemma *bounded-removeAll* [*dest*]:
assumes ‹*bounded* K ({p} ∪ A) P›
shows ‹*bounded* (*removeAll* p K) A (λB. P (p # B))›
⟨*proof*⟩

lemma *bounded-terms*:

assumes ‹$t \in$ *terms* $(\{p\} \cup A)$›
shows ‹$t \in$ *terms-fm* $p \wedge$ *bounded* $[]$ A $(\lambda B.\ t \in$ *terms* $(set\ (p\ \#\ B))) \vee$
$(\exists\, q \in A.\ t \in$ *terms-fm* $q \wedge$ *bounded* $[q]$ A $(\lambda B.\ t \in$ *terms* $(set\ (p\ \#\ B))))$›
⟨*proof*⟩

lemma *bounded-params*:
assumes ‹$a \notin$ *P.params* $(\{p\} \cup A)$› ‹*bounded* K A P›
shows ‹*bounded* K A $(\lambda B.\ a \notin$ *P.params* $(set\ (p\ \#\ B)))$›
⟨*proof*⟩

lemma *finite-kernel*: ‹$A \Vdash p \Longrightarrow \exists\, K.$ *bounded* K A $(\lambda B.\ B \vdash p)$›
⟨*proof*⟩

corollary *finite-assumptions*: ‹$A \Vdash p \Longrightarrow \exists\, B.$ *set* $B \subseteq A \wedge B \vdash p$›
⟨*proof*⟩

lemma *to-set*: ‹$A \vdash p \Longrightarrow$ *set* $A \Vdash p$›
⟨*proof*⟩

corollary *soundness-list*:
assumes ‹$A \vdash p$› ‹*wf-model* $(Model\ U\ E\ F\ G)$› ‹$\forall\, q \in$ *set* $A.$ *Model* $U\ E\ F\ G \models q$›
shows ‹*Model* $U\ E\ F\ G \models p$›
⟨*proof*⟩

corollary *soundness-nil*: ‹$[] \vdash p \Longrightarrow$ *wf-model* $(Model\ U\ E\ F\ G) \Longrightarrow$ *Model* $U\ E\ F\ G \models p$›
⟨*proof*⟩

corollary ‹$\neg\ ([] \vdash \bot)$›
⟨*proof*⟩

corollary *strong-completeness-list*:
fixes p :: ‹$('f, 'p)$ *fm*›
assumes *mod*: ‹$\bigwedge(U :: 'f\ tm\ set)\ E\ F\ G.$ *wf-model* $(Model\ U\ E\ F\ G) \Longrightarrow (\forall\, q \in A.$ *Model* $U\ E\ F\ G \models q) \Longrightarrow$ *Model* $U\ E\ F\ G \models p$›
and ‹*P.enough-new* A›
shows ‹$\exists\, B.$ *set* $B \subseteq A \wedge B \vdash p$›
⟨*proof*⟩

theorem *main*:
fixes p :: ‹$('f, 'p)$ *fm*›
assumes ‹$|UNIV :: ('f, 'p)\ fm\ set| \leq o\ |UNIV :: 'f\ set|$›
shows ‹$[] \vdash p \longleftrightarrow (\forall\, (U :: 'f\ tm\ set)\ E\ F\ G.$ *wf-model* $(Model\ U\ E\ F\ G) \longrightarrow$ *Model* $U\ E\ F\ G \models p)$›
⟨*proof*⟩

end

2.9 Tableau

locale *Tableau*
begin

inductive *TC* :: ‹('*f*, '*p*) *fm set* ⇒ *bool*› (‹⊢ -› [*51*] *50*) **where**
  Axiom [*simp*]: ‹¬ ·*P ts* ∈ *A* ⟹ ·*P ts* ∈ *A* ⟹ ⊢ *A*›
| *FlsP* [*simp*]: ‹⊥ ∈ *A* ⟹ ⊢ *A*›
| *FlsN* [*intro*]: ‹⊢ *A* ⟹ ⊢ {¬ ⊥} ∪ *A*›
| *ImpP* [*intro*]: ‹⊢ {¬ *p*} ∪ *A* ⟹ ⊢ {*q*} ∪ *A* ⟹ ⊢ {*p* ⟶ *q*} ∪ *A*›
| *ImpN* [*intro*]: ‹⊢ {*p*, ¬ *q*} ∪ *A* ⟹ ⊢ {¬ (*p* ⟶ *q*)} ∪ *A*›
| *UniP* [*intro*]: ‹⊢ {⟨*t*⟩*p*} ∪ *A* ⟹ *t* ∈ *terms* ({*p*} ∪ *A*) ⟹ ⊢ {∀ *p*} ∪ *A*›
| *UniN* [*intro*]: ‹⊢ {¬ ⟨⋆*a*⟩*p*} ∪ *A* ⟹ *a* ∉ *P.params* ({*p*} ∪ *A*) ⟹ ⊢ {¬ ∀ *p*} ∪ *A*›

2.9.1 Soundness

theorem *soundness*:
  assumes ‹⊢ *A*› ‹*wf-model* (*Model U E F G*)›
  shows ‹∃ *q* ∈ *A*. ¬ *Model U E F G* ⊨ *q*›
  ⟨*proof*⟩

2.9.2 Derivational Consistency

sublocale *DC*: *Derivational-Confl psub params-fm* ‹λ-. *True*› *confl-class* ‹λ*A*. ¬ ⊢ *A*›
⟨*proof*⟩

sublocale *DA*: *Derivational-Alpha psub params-fm* ‹λ-. *True*› *alpha-class* ‹λ*A*. ¬ ⊢ *A*›
⟨*proof*⟩

sublocale *DB*: *Derivational-Beta psub params-fm* ‹λ-. *True*› *beta-class* ‹λ*A*. ¬ ⊢ *A*›
⟨*proof*⟩

sublocale *DG*: *Derivational-Gamma map-tm psub params-fm* ‹λ-. *True*› *gamma-class* ‹λ*A*. ¬ ⊢ *A*›
⟨*proof*⟩

sublocale *DD*: *Derivational-Delta psub params-fm* ‹λ-. *True*› *δ* ‹λ*A*. ¬ ⊢ *A*›
⟨*proof*⟩

sublocale *Derivational-Consistency psub params-fm* ‹λ-. *True*› *Kinds* ‹λ*A*. ¬ ⊢ *A*›
  ⟨*proof*⟩

2.9.3 Strong Completeness

theorem *strong-completeness*:

fixes $p :: \langle('f, 'p)\ fm\rangle$
assumes *mod*: $\langle\bigwedge(U :: 'f\ tm\ set)\ E\ F\ G.\ wf\text{-}model\ (Model\ U\ E\ F\ G) \Longrightarrow (\forall\, q \in A.\ Model\ U\ E\ F\ G \models q) \Longrightarrow Model\ U\ E\ F\ G \models p\rangle$
and $\langle P.enough\text{-}new\ A\rangle$
shows $\langle\vdash \{\neg\ with\text{-}subterm\ p\} \cup A\rangle$
⟨*proof*⟩

end

end

Chapter 3

Example: Second-Order Logic

Generalizes [4] and [5] from first-order logic to second-order logic

theory *Example-SOL* **imports**
Abstract-Consistency-Property
begin

3.1 Syntax

datatype (*params-sym*:'*f*) *sym*
= *VarS nat* (‹#$_2$›)
| *SymS* '*f* (‹◯$_2$›)

datatype (*params-tm*: '*f*) *tm*
= *Var nat* (‹#›)
| *Fun* ‹'*f sym*› ‹'*f tm list*› (‹◯›)
| *Cst* '*f* (‹⋆›)

datatype (*params-fm*: '*f*) *fm*
= *Falsity* (‹⊥›)
| *is-Pre*: *Pre* ‹'*f sym*› ‹'*f tm list*› (‹•›)
| *Imp* ‹'*f fm*› ‹'*f fm*› (**infixr** ‹⟶› *55*)
| *Uni* ‹'*f fm*› (‹∀ ›)
| *UniP* ‹'*f fm*› (‹∀ $_P$›)
| *UniF* ‹'*f fm*› (‹∀ $_F$›)

abbreviation *Neg* (‹¬ -› [*70*] *70*) **where** ‹¬ *p* ≡ *p* ⟶ ⊥›

abbreviation *And* (**infix** ‹∧› *50*) **where** ‹*p* ∧ *q* ≡ ¬ (*p* ⟶ ¬ *q*)›

abbreviation *Iff* (**infix** ‹⟷› *50*) **where** ‹*p* ⟷ *q* ≡ (*p* ⟶ *q*) ∧ (*q* ⟶ *p*)›

abbreviation *Eql* (‹- = -›) **where** ‹*t1* = *t2* ≡ (∀ $_P$ ((•(#$_2$ *0*) [*t1*]) ⟷ (•(#$_2$

0) [*t2*]]))›

abbreviation *ExiF* (‹∃ $_F$›) **where** ‹∃ $_F$ $p \equiv \neg(\forall\,_F(\neg p))$›

abbreviation *ExiP* (‹∃ $_P$›) **where** ‹∃ $_P$ $p \equiv \neg(\forall\,_P(\neg p))$›

term ‹∀ (⊥ ⟶ (•(○$_2$ ''*P*'') [○(○$_2$ ''*f*'') [#*0*]]))›

3.2 Semantics

definition *shift* (‹-⟨-:-⟩›) **where**
‹*E*⟨*n*:*x*⟩ *m* ≡ *if m* < *n then E m else if m* = *n then x else E* (*m*−*1*)›

primrec *semantics-fn* (‹⦇-, -⦈$_2$›) **where**
‹⦇*E*$_F$, *F*⦈$_2$ (#$_2$ *n*) = *E*$_F$ *n*›
| ‹⦇*E*$_F$, *F*⦈$_2$ (○$_2$ *f*) = *F f*›

primrec *semantics-tm* (‹⦇-,-, -, -⦈›) **where**
‹⦇*E*, *E*$_F$, *C*, *F*⦈ (#*n*) = *E n*›
| ‹⦇*E*, *E*$_F$, *C*, *F*⦈ (○*f ts*) = (⦇*E*$_F$, *F*⦈$_2$ *f*) (*map* ⦇*E*, *E*$_F$, *C*, *F*⦈ *ts*)›
| ‹⦇*E*, *E*$_F$, *C*, *F*⦈ (⋆ *c*) = *C c*›

fun *semantics-fm* (**infix** ‹⊨› *50*) **where**
‹((-, -, -, -, -, -, -, -) ⊨ ⊥) = *False*›
| ‹((*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ •*P ts*) = ⦇*E*$_P$, *G*⦈$_2$ *P* (*map* ⦇*E*, *E*$_F$, *C*, *F*⦈ *ts*)›
| ‹((*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p* ⟶ *q*) = ((*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p* ⟶ (*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ *q*)›
| ‹((*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ ∀ *p*) = (∀ *x*. (*E*⟨*0*:*x*⟩, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*)›
| ‹((*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ ∀ $_P$*p*) = (∀ *x* ∈ *PS*. (*E*, *E*$_F$, *E*$_P$⟨*0*:*x*⟩, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*)›
| ‹((*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ ∀ $_F$*p*) = (∀ *x* ∈ *FS*. (*E*, *E*$_F$⟨*0*:*x*⟩, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*)›

proposition ‹(*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ (∀ (•*P* [# *0*]) ⟶ •*P* [⋆*a*])›
⟨*proof*⟩

3.3 Operations

3.3.1 Shift

context fixes *n m* :: *nat* **begin**

lemma *shift-eq* [*simp*]: ‹*n* = *m* ⟹ *E*⟨*n*:*x*⟩ *m* = *x*›
⟨*proof*⟩

lemma *shift-gt* [*simp*]: ‹*m* < *n* ⟹ *E*⟨*n*:*x*⟩ *m* = *E m*›

⟨*proof*⟩

lemma *shift-lt* [*simp*]: ‹$n < m \Longrightarrow E\langle n{:}x\rangle\ m = E\ (m-1)$›
⟨*proof*⟩

lemma *shift-commute* [*simp*]: ‹$(E\langle n{:}y\rangle\langle 0{:}x\rangle) = (E\langle 0{:}x\rangle\langle n+1{:}y\rangle)$›
⟨*proof*⟩

end

3.3.2 Parameters

abbreviation ‹*params* $S \equiv \bigcup p \in S.$ *params-fm* p›

lemma *upd-params-sym* [*simp*]: ‹$f \notin$ *params-sym* $fn \Longrightarrow \langle\!|E_F, F(f := x)|\!\rangle_2\ fn = \langle\!|E_F, F|\!\rangle_2\ fn$›
⟨*proof*⟩

lemma *upd-params-tm* [*simp*]: ‹$f \notin$ *params-tm* $t \Longrightarrow \langle\!|E, E_F, C, F(f := x)|\!\rangle\ t = \langle\!|E, E_F, C, F|\!\rangle\ t$›
⟨*proof*⟩

lemma *upd-params-tm-c* [*simp*]: ‹$c \notin$ *params-tm* $t \Longrightarrow \langle\!|E, E_F, C(c := x), F|\!\rangle\ t = \langle\!|E, E_F, C, F|\!\rangle\ t$›
⟨*proof*⟩

lemma *upd-params-fm* [*simp*]: ‹$f \notin$ *params-fm* $p \Longrightarrow (E, E_F, E_P, C, F(f := x), G, PS, FS) \models p \longleftrightarrow (E, E_F, E_P, C, F, G, PS, FS) \models p$›
⟨*proof*⟩

lemma *upd-params-fm-c* [*simp*]: ‹$c \notin$ *params-fm* $p \Longrightarrow (E, E_F, E_P, C(c := x), F, G, PS, FS) \models p \longleftrightarrow (E, E_F, E_P, C, F, G, PS, FS) \models p$›
⟨*proof*⟩

lemma *upd-params-fm-G* [*simp*]: ‹$P \notin$ *params-fm* $p \Longrightarrow (E, E_F, E_P, C, F, G(P := x), PS, FS) \models p \longleftrightarrow (E, E_F, E_P, C, F, G, PS, FS) \models p$›
⟨*proof*⟩

lemma *finite-params-sym* [*simp*]: ‹*finite* (*params-sym* *fn*)›
⟨*proof*⟩

lemma *finite-params-tm* [*simp*]: ‹*finite* (*params-tm* *t*)›
⟨*proof*⟩

lemma *finite-params-fm* [*simp*]: ‹*finite* (*params-fm* *p*)›
⟨*proof*⟩

3.3.3 Instantiation

primrec *lift-tm* (‹↑›) **where**

$\langle\uparrow(\#n) = \#(n{+}1)\rangle$
| $\langle\uparrow(\bigcirc f\ ts) = \bigcirc f\ (map \uparrow\ ts)\rangle$
| $\langle\uparrow(\star c) = \star\ c\rangle$

primrec *lift-sym* ($\langle\uparrow_2\rangle$) **where**
$\langle\uparrow_2(\#_2\ n) = \#_2\ (n{+}1)\rangle$
| $\langle\uparrow_2(\bigcirc_2\ p) = \bigcirc_2\ p\rangle$

primrec *lift-fn* ($\langle\uparrow_F\rangle$) **where**
$\langle\uparrow_F(\#n) = \#n\rangle$
| $\langle\uparrow_F(\bigcirc f\ ts) = \bigcirc(\uparrow_2 f)\ (map \uparrow_F\ ts)\rangle$
| $\langle\uparrow_F(\star\ c) = \star\ c\rangle$

primrec *inst-tm* ($\langle\langle\!\langle \text{-}'/\text{-}\rangle\!\rangle\rangle$) **where**
$\langle\langle\!\langle s/m\rangle\!\rangle(\#n) = (if\ n < m\ then\ \#n\ else\ if\ n = m\ then\ s\ else\ \#(n{-}1))\rangle$
| $\langle\langle\!\langle s/m\rangle\!\rangle(\bigcirc f\ ts) = \bigcirc f\ (map\ \langle\langle\!\langle s/m\rangle\!\rangle\ ts)\rangle$
| $\langle\langle\!\langle s/m\rangle\!\rangle(\star c) = \star c\rangle$

primrec *inst-sym* ($\langle\langle\!\langle \text{-}'/\text{-}\rangle\!\rangle_2\rangle$) **where**
$\langle\langle\!\langle s/m\rangle\!\rangle_2\ (\#_2\ n) = (if\ n < m\ then\ \#_2\ n\ else\ if\ n = m\ then\ s\ else\ \#_2\ (n{-}1))\rangle$
| $\langle\langle\!\langle s/m\rangle\!\rangle_2\ (\bigcirc_2\ p) = \bigcirc_2\ p\rangle$

primrec *inst-fn* ($\langle\langle\!\langle \text{-}'/\text{-}\rangle\!\rangle_F\rangle$) **where**
$\langle\langle\!\langle s/m\rangle\!\rangle_F(\#n) = (\#n)\rangle$
| $\langle\langle\!\langle s/m\rangle\!\rangle_F(\bigcirc f\ ts) = \bigcirc(\langle\langle\!\langle s/m\rangle\!\rangle_2\ f)\ (map\ \langle\langle\!\langle s/m\rangle\!\rangle_F\ ts)\rangle$
| $\langle\langle\!\langle s/m\rangle\!\rangle_F(\star c) = (\star c)\rangle$

primrec *inst-fm* ($\langle\langle\text{-}'/\text{-}\rangle\rangle$) **where**
$\langle\langle\text{-}/\text{-}\rangle\bot = \bot\rangle$
| $\langle\langle s/m\rangle(\cdot P\ ts) = \cdot P\ (map\ \langle\langle\!\langle s/m\rangle\!\rangle\ ts)\rangle$
| $\langle\langle s/m\rangle(p \longrightarrow q) = \langle s/m\rangle p \longrightarrow \langle s/m\rangle q\rangle$
| $\langle\langle s/m\rangle(\forall\, p) = \forall\, (\langle\uparrow s/m{+}1\rangle p)\rangle$
| $\langle\langle s/m\rangle(\forall_P p) = \forall_P(\langle s/m\rangle p)\rangle$
| $\langle\langle s/m\rangle(\forall_F p) = \forall_F(\langle\uparrow_F\ s/m\rangle p)\rangle$

primrec *inst-fm-P* ($\langle\langle\text{-}'/\text{-}\rangle_P\rangle$) **where**
$\langle\langle\text{-}/\text{-}\rangle_P\bot = \bot\rangle$
| $\langle\langle s/m\rangle_P(\cdot P\ ts) = \cdot(\langle\langle\!\langle s/m\rangle\!\rangle_2\ P)\ ts\rangle$
| $\langle\langle s/m\rangle_P(p \longrightarrow q) = \langle s/m\rangle_P p \longrightarrow \langle s/m\rangle_P q\rangle$
| $\langle\langle s/m\rangle_P(\forall\, p) = \forall\, (\langle s/m\rangle_P p)\rangle$
| $\langle\langle s/m\rangle_P(\forall_P p) = \forall_P(\langle\uparrow_2\ s/m{+}1\rangle_P p)\rangle$
| $\langle\langle s/m\rangle_P(\forall_F p) = \forall_F(\langle s/m\rangle_P p)\rangle$

primrec *inst-fm-F* ($\langle\langle\text{-}'/\text{-}\rangle_F\rangle$) **where**
$\langle\langle\text{-}/\text{-}\rangle_F\bot = \bot\rangle$
| $\langle\langle s/m\rangle_F(\cdot P\ ts) = \cdot P\ (map\ \langle\langle\!\langle s/m\rangle\!\rangle_F\ ts)\rangle$
| $\langle\langle s/m\rangle_F(p \longrightarrow q) = \langle s/m\rangle_F p \longrightarrow \langle s/m\rangle_F q\rangle$
| $\langle\langle s/m\rangle_F(\forall\, p) = \forall\, (\langle s/m\rangle_F p)\rangle$
| $\langle\langle s/m\rangle_F(\forall_P p) = \forall_P(\langle s/m\rangle_F p)\rangle$

| ‹$\langle s/m\rangle_F(\forall_F p) = \forall_F(\langle\uparrow_2 s/m{+}1\rangle_F p)$›

lemma *lift-lemma* [*simp*]: ‹$\langle\!|E\langle 0{:}x\rangle, E_F, C, F|\!\rangle\ (\uparrow t) = \langle\!|E, E_F, C, F|\!\rangle\ t$›
⟨*proof*⟩

lemma *lift-lemma-P* [*simp*]: ‹$\langle\!|E_P\langle 0{:}x\rangle, G|\!\rangle_2\ (\uparrow_2 P) = \langle\!|E_P, G|\!\rangle_2\ P$›
⟨*proof*⟩

lemma *lift-lemma-F* [*simp*]: ‹$\langle\!|E, E_F\langle 0{:}x\rangle, C, F|\!\rangle\ (\uparrow_F tm) = \langle\!|E, E_F, C, F|\!\rangle\ tm$›
⟨*proof*⟩

lemma *inst-tm-semantics* [*simp*]: ‹$\langle\!|E, E_F, C, F|\!\rangle\ (\langle\!|s/m|\!\rangle t) = \langle\!|E\langle m{:}\langle\!|E, E_F, C, F|\!\rangle\ s\rangle, E_F, C, F|\!\rangle\ t$›
⟨*proof*⟩

lemma *inst-sym-semantics* [*simp*]: ‹$\langle\!|E_F, G|\!\rangle_2\ (\langle\!|s/m|\!\rangle_2\ fn) = \langle\!|E_F\langle m{:}\langle\!|E_F, G|\!\rangle_2\ s\rangle, G|\!\rangle_2\ fn$›
⟨*proof*⟩

lemma *inst-tm-semantics-F* [*simp*]: ‹$\langle\!|E, E_F, C, F|\!\rangle\ (\langle\!|s/m|\!\rangle_F\ t) = \langle\!|E, E_F\langle m{:}\langle\!|E_F, F|\!\rangle_2\ s\rangle, C, F|\!\rangle\ t$›
⟨*proof*⟩

lemma *inst-fm-semantics-F* [*simp*]:
‹$(E, E_F, E_P, C, F, G, PS, FS) \models (\langle t/m\rangle_F\ p) \longleftrightarrow (E, E_F\langle m{:}\langle\!|E_F, F|\!\rangle_2\ t\rangle, E_P, C, F, G, PS, FS) \models p$›
⟨*proof*⟩

lemma *inst-fm-semantics* [*simp*]:
‹$(E, E_F, E_P, C, F, G, PS, FS) \models (\langle t/m\rangle p) \longleftrightarrow (E\langle m{:}\langle\!|E, E_F, C, F|\!\rangle\ t\rangle, E_F, E_P, C, F, G, PS, FS) \models p$›
⟨*proof*⟩

lemma *inst-fm-semantics-P* [*simp*]: ‹$(E, E_F, E_P, C, F, G, PS, FS) \models (\langle P/m\rangle_P p) \longleftrightarrow (E, E_F, E_P\langle m{:}\langle\!|E_P, G|\!\rangle_2\ P\rangle, C, F, G, PS, FS) \models p$›
⟨*proof*⟩

3.3.4 Size

The built-in *size* is not invariant under substitution.

primrec *size-fm* **where**
‹*size-fm* $\bot = 1$›
| ‹*size-fm* $(\bullet\text{-}\ \text{-}) = 1$›
| ‹*size-fm* $(p \longrightarrow q) = 1 + \textit{size-fm}\ p + \textit{size-fm}\ q$›
| ‹*size-fm* $(\forall\, p) = 1 + \textit{size-fm}\ p$›
| ‹*size-fm* $(\forall_P p) = 1 + \textit{size-fm}\ p$›
| ‹*size-fm* $(\forall_F p) = 1 + \textit{size-fm}\ p$›

lemma *size-inst-fm* [*simp*]: ‹*size-fm* $(\langle t/m\rangle p) = \textit{size-fm}\ p$›

⟨*proof*⟩

lemma *size-inst-fm-P* [*simp*]: ‹*size-fm* (⟨*t*/*m*⟩$_P$ *p*) = *size-fm p*›
⟨*proof*⟩

lemma *size-inst-fm-F* [*simp*]: ‹*size-fm* (⟨*t*/*m*⟩$_F$ *p*) = *size-fm p*›
⟨*proof*⟩

3.4 Model Existence

inductive *confl-class* :: ‹'*f fm list* ⇒ '*f fm list* ⇒ *bool*› (**infix** ‹⇝✗› *50*) **where**
CFls: ‹[⊥] ⇝✗ [⊥]›
| *CNeg*: ‹[¬ (•*P ts*)] ⇝✗ [•*P ts*]›

inductive *alpha-class* :: ‹'*f fm list* ⇒ '*f fm list* ⇒ *bool*› (**infix** ‹⇝$_\alpha$› *50*) **where**
CImpN: ‹[¬ (*p* ⟶ *q*)] ⇝$_\alpha$ [*p*, ¬ *q*]›

inductive *beta-class* :: ‹'*f fm list* ⇒ '*f fm list* ⇒ *bool*› (**infix** ‹⇝$_\beta$› *50*) **where**
CImpP: ‹[*p* ⟶ *q*] ⇝$_\beta$ [¬ *p*, *q*]›

inductive *gamma-class* :: ‹'*f fm list* ⇒ ('*f tm* ⇒ '*f fm list*) ⇒ *bool*› (**infix** ‹⇝$_\gamma$› *50*) **where**
CAllP: ‹[∀ *p*] ⇝$_\gamma$ (λ*t*. [⟨*t*/*0*⟩*p*])›

inductive *gamma-class-P* :: ‹'*f fm list* ⇒ ('*f sym* ⇒ '*f fm list*) ⇒ *bool*› (**infix** ‹⇝$_{\gamma P}$› *50*) **where**
CAllPP: ‹[∀$_P$ *p*] ⇝$_{\gamma P}$ (λ*s*. [⟨*s*/*0*⟩$_P$ *p*])›

inductive *gamma-class-F* :: ‹'*f fm list* ⇒ ('*f sym* ⇒ '*f fm list*) ⇒ *bool*› (**infix** ‹⇝$_{\gamma F}$› *50*) **where**
CAllFP: ‹[∀$_F$ *p*] ⇝$_{\gamma F}$ (λ*s*. [⟨*s*/*0*⟩$_F$ *p*])›

fun δ :: ‹'*f fm* ⇒ '*f* ⇒ '*f fm list*› **where**
CAllN: ‹δ (¬ ∀ *p*) *x* = [¬ ⟨⋆*x*/*0*⟩ *p*]›
| *CAll2PN*: ‹δ (¬ ∀$_P$ *p*) *x* = [¬ ⟨◯$_2$ *x*/*0*⟩$_P$ *p*]›
| *CAll2FN*: ‹δ (¬ ∀$_F$ *p*) *x* = [¬ ⟨◯$_2$ *x*/*0*⟩$_F$ *p*]›
| *NOMATCH*: ‹δ - - = []›

interpretation *P*: *Params map-fm params-fm* ‹λ-. *True*›
⟨*proof*⟩

interpretation *C*: *Confl map-fm params-fm* ‹λ-. *True*› *confl-class*
⟨*proof*⟩

interpretation *A*: *Alpha map-fm params-fm* ‹λ-. *True*› *alpha-class*
⟨*proof*⟩

interpretation *B*: *Beta map-fm params-fm* ‹λ-. *True*› *beta-class*
⟨*proof*⟩

lemma *map-tm-inst-tm* [*simp*]:
map-tm f (⟪*t*/*n*⟫ *x*) = ⟪*map-tm f t*/*n*⟫ (*map-tm f x*)
⟨*proof*⟩

lemma *map-tm-lift-tm* [*simp*]: *map-tm f* (↑ *t*) = ↑ (*map-tm f t*)
⟨*proof*⟩

lemma *map-sym-lift-fn* [*simp*]: ‹*map-sym f* (↑$_2$ *t*) = ↑$_2$ (*map-sym f t*)›
⟨*proof*⟩

lemma *map-tm-lift-fn* [*simp*]: *map-tm f* (↑$_F$ *t*) = ↑$_F$ (*map-tm f t*)
⟨*proof*⟩

lemma *map-fm-inst-single* [*simp*]: ‹*map-fm f* (⟨*t*/*m*⟩*p*) = ⟨*map-tm f t*/*m*⟩(*map-fm f p*)›
⟨*proof*⟩

lemma *map-sym-inst-sym* [*simp*]: ‹*map-sym f* (⟪*t*/*m*⟫$_2$ *p*) = ⟪*map-sym f t*/*m*⟫$_2$ (*map-sym f p*)›
⟨*proof*⟩

lemma *psub-inst-single′* [*simp*]: ‹*map-fm f* (⟨*t*/*m*⟩$_P$ *p*) = ⟨*map-sym f t*/*m*⟩$_P$(*map-fm f p*)›
⟨*proof*⟩

lemma *map-tm-inst-fn* [*simp*]: ‹*map-tm f* (⟪*t*/*m*⟫$_F$ *s*) = ⟪*map-sym f t*/*m*⟫$_F$ (*map-tm f s*)›
⟨*proof*⟩

lemma *psub-inst-single″* [*simp*]: ‹*map-fm f* (⟨*t*/*m*⟩$_F$ *p*) = ⟨*map-sym f t*/*m*⟩$_F$(*map-fm f p*)›
⟨*proof*⟩

interpretation *G*: *Gamma-UNIV map-tm map-fm params-fm* ‹λ-. *True*› *gamma-class*
⟨*proof*⟩

interpretation G_P: *Gamma-UNIV map-sym map-fm params-fm* ‹λ-. *True*› *gamma-class-P*
⟨*proof*⟩

interpretation G_F: *Gamma-UNIV map-sym map-fm params-fm* ‹λ-. *True*› *gamma-class-F*
⟨*proof*⟩

interpretation *D*: *Delta map-fm params-fm* ‹λ-. *True*› *δ*
⟨*proof*⟩

abbreviation *Kinds* :: ‹('*x*, '*x fm*) *kind list*› **where**
‹*Kinds* ≡ [*C.kind*, *A.kind*, *B.kind*, *G.kind*, G_P.*kind*, G_F.*kind*, *D.kind*]›

lemma *prop$_E$-Kinds*:
assumes ‹*P.sat$_E$ C.kind C*› ‹*P.sat$_E$ A.kind C*› ‹*P.sat$_E$ B.kind C*› ‹*P.sat$_E$ G.kind C*› ‹*P.sat$_E$ G$_P$.kind C*›
‹*P.sat$_E$ G$_F$.kind C*› ‹*P.sat$_E$ D.kind C*›
shows ‹*P.prop$_E$ Kinds C*›
⟨*proof*⟩

interpretation *Consistency-Kinds map-fm params-fm* ‹λ-. *True*› *Kinds*
⟨*proof*⟩

interpretation *Maximal-Consistency map-fm params-fm* ‹λ-. *True*› *Kinds*
⟨*proof*⟩

abbreviation *henv$_P$* **where** *henv$_P$ H* == λ*n ts*. ·(#$_2$ *n*) *ts* ∈ *H*

abbreviation *hpred* **where** *hpred H* == λ*P ts*. ·(○$_2$ *P*) *ts* ∈ *H*

abbreviation *hdom$_P$* **where** *hdom$_P$ H* == *range* (*henv$_P$ H*) ∪ *range* (*hpred H*)

abbreviation *henv$_F$* **where** *henv$_F$* == λ*f*. ○ (#$_2$ *f*)

abbreviation *hfun* **where** *hfun* == λ*f*. ○ (○$_2$ *f*)

definition *hdom$_F$* **where** *hdom$_F$* == *range henv$_F$* ∪ *range hfun*

abbreviation (*input*) *hmodel* (‹⟦-⟧›) **where** ‹⟦*H*⟧ ≡ (#, *henv$_F$*, *henv$_P$ H*, ⋆, *hfun*, *hpred H*, *hdom$_P$ H*, *hdom$_F$*)›

lemma *semantics-tm-id* [*simp*]: ‹⦇#, *henv$_F$* , ⋆ , λ*f*. ○ (○$_2$ *f*)⦈ *t* = *t*›
⟨*proof*⟩

lemma *semantics-tm-id-map* [*simp*]: ‹*map* ⦇#, λ*f*. ○ (#$_2$ *f*) , ⋆, λ*f*. ○ (○$_2$ *f*)⦈ *ts* = *ts*›
⟨*proof*⟩

lemma *semantics-fn-h* [*simp*]: ‹⦇*henv$_P$ S*, *hpred S*⦈$_2$ *P ts* ⟷ ·*P ts* ∈ *S*›
⟨*proof*⟩

lemma *canonical-henv$_P$*:
‹∀ *t*. (#, *henv$_F$*, (*henv$_P$ S*)⟨*0*:⦇*henv$_P$ S*, *hpred S*⦈$_2$ *t*⟩, ⋆, *hfun*, *hpred S*, *hdom$_P$ S*, *hdom$_F$*) ⊨ *p* ⟹
(#, *henv$_F$*, (*henv$_P$ S*)⟨*0*:*henv$_P$ S n*⟩, ⋆, *hfun*, *hpred S*, *hdom$_P$ S*, *hdom$_F$*) ⊨ *p*›
⟨*proof*⟩

lemma *canonical-hpred*:
‹∀ *t*. (#, *henv$_F$*, (*henv$_P$ S*)⟨*0*:⦇*henv$_P$ S*, *hpred S*⦈$_2$ *t*⟩, ⋆, *hfun*, *hpred S*, *hdom$_P$ S*, *hdom$_F$*) ⊨ *p* ⟹
(#, *henv$_F$*, (*henv$_P$ S*)⟨*0*:*hpred S P*⟩, ⋆, *hfun*, *hpred S*, *hdom$_P$ S*, *hdom$_F$*) ⊨

p›
⟨*proof*⟩

lemma *canonical-henv$_F$*:
‹∀ t. (#, *henv$_F$*⟨*0*:⦇*henv$_F$*, *hfun*⦈$_2$ t⟩, *henv$_P$* S, ⋆, *hfun*, *hpred* S, *hdom$_P$* S, *hdom$_F$*) ⊨ p ⟹
(#, *henv$_F$*⟨*0*:*henv$_F$* f⟩, *henv$_P$* S, ⋆, *hfun*, *hpred* S, *hdom$_P$* S, *hdom$_F$*) ⊨ p›
⟨*proof*⟩

lemma *canonical-hfun*:
‹∀ t. (#, *henv$_F$*⟨*0*:⦇*henv$_F$*, *hfun*⦈$_2$ t⟩, *henv$_P$* S, ⋆, *hfun*, *hpred* S, *hdom$_P$* S, *hdom$_F$*) ⊨ p ⟹
(#, *henv$_F$*⟨*0*:*hfun* f⟩, *henv$_P$* S, ⋆, *hfun*, *hpred* S, *hdom$_P$* S, *hdom$_F$*) ⊨ p›
⟨*proof*⟩

locale *MyHintikka* = *Hintikka map-fm params-fm* ‹λ-. *True*› *Kinds* S **for** S :: ‹'*x fm set*›
begin

lemmas
confl = *sat$_H$*[*of C.kind*] **and**
alpha = *sat$_H$*[*of A.kind*] **and**
beta = *sat$_H$*[*of B.kind*] **and**
gammaFO = *sat$_H$*[*of G.kind*] **and**
gamma2P = *sat$_H$*[*of G$_P$.kind*] **and**
gamma2F = *sat$_H$*[*of G$_F$.kind*] **and**
delta = *sat$_H$*[*of D.kind*]

theorem *model*: ‹($p \in S \longrightarrow ⟦S⟧ \models p$) ∧ ($\neg\ p \in S \longrightarrow \neg\ ⟦S⟧ \models p$)›
⟨*proof*⟩

end

theorem *model-existence*:
fixes S :: ‹'*x fm set*›
assumes ‹*P.prop$_E$ Kinds C*›
and ‹$S \in C$›
and ‹*P.enough-new S*›
and ‹$p \in S$›
shows ‹⟦*mk-mcs C S*⟧ ⊨ p›
⟨*proof*⟩

3.5 Propositional Semantics

primrec *boolean* **where**
‹*boolean* - - ⊥ = *False*›
| ‹*boolean G* - (·*P ts*) = *G P ts*›
| ‹*boolean G A* ($p \longrightarrow q$) = (*boolean G A p* ⟶ *boolean G A q*)›
| ‹*boolean* - *A* (∀ p) = *A* (∀ p)›

| ‹*boolean* - *A* ($\forall_P p$) = *A* ($\forall_P p$)›
| ‹*boolean* - *A* ($\forall_F p$) = *A* ($\forall_F p$)›

abbreviation ‹*tautology p* ≡ ∀ *G A*. *boolean G A p*›

proposition ‹*tautology* (∀ (·*P* [#*0*]) ⟶ ∀ (·*P* [#*0*]))›
⟨*proof*⟩

lemma *boolean-semantics*: ‹*boolean* (λ*a*. ⦇E_P,*G*⦈$_2$ *a* ∘ *map* ⦇*E*, E_F, *C*, *F*⦈) (λ*p*. (*E*, E_F, E_P, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*) = (λ*p*. (*E*, E_F, E_P, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*)›
⟨*proof*⟩

lemma *tautology*[*simp*]: ‹*tautology p* ⟹ (*E*, E_F, E_P, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*›
⟨*proof*⟩

proposition ‹∃ *p*. (∀ *E* E_F E_P *C F G PS FS*. (*E*, E_F, E_P, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*) ∧ ¬ *tautology p*›
⟨*proof*⟩

3.6 Calculus

Adapted from System Q1 by Smullyan in First-Order Logic (1968).

inductive *Axiomatic* (‹⊢ -› [*50*] *50*) **where**
TA: ‹*tautology p* ⟹ ⊢ *p*›
| *IA*: ‹⊢ ∀ *p* ⟶ ⟨*t*/*0*⟩*p*›
| IA_P: ‹⊢ $\forall_P p$ ⟶ $\langle s/0\rangle_P p$›
| IA_F: ‹⊢ $\forall_F p$ ⟶ $\langle s/0\rangle_F p$›
| *MP*: ‹⊢ *p* ⟶ *q* ⟹ ⊢ *p* ⟹ ⊢ *q*›
| *GR*: ‹⊢ *q* ⟶ ⟨⋆*a*/*0*⟩*p* ⟹ *a* ∉ *params* {*p*, *q*} ⟹ ⊢ *q* ⟶ ∀ *p*›
| GR_P: ‹⊢ *q* ⟶ $\langle \bigcirc_2\ a/0\rangle_P p$ ⟹ *a* ∉ *params* {*p*, *q*} ⟹ ⊢ *q* ⟶ $\forall_P p$›
| GR_F: ‹⊢ *q* ⟶ $\langle \bigcirc_2\ a/0\rangle_F p$ ⟹ *a* ∉ *params* {*p*, *q*} ⟹ ⊢ *q* ⟶ $\forall_F p$›

We simulate assumptions on the lhs of ⊢ with a chain of implications on the rhs.

primrec *imply* (**infixr** ‹⇝› *56*) **where**
‹([] ⇝ *q*) = *q*›
| ‹(*p* # *ps* ⇝ *q*) = (*p* ⟶ *ps* ⇝ *q*)›

abbreviation *Axiomatic-assms* (‹- ⊢ -› [*50*, *50*] *50*) **where**
‹*ps* ⊢ *q* ≡ ⊢ *ps* ⇝ *q*›

3.7 Soundness

fun *wf-model* **where**
‹*wf-model* (*E*, E_F, E_P, *C*, *F*, *G*, *PS*, *FS*) ⟷ *range G* ⊆ *PS* ∧ *range* E_P ⊆ *PS* ∧ *range F* ⊆ *FS* ∧ *range* E_F ⊆ *FS*›

theorem *soundness*:
shows ‹$\vdash p \Longrightarrow$ *wf-model* (E, E_F, E_P, C, F, G, PS, FS) $\Longrightarrow$ (E, E_F, E_P, C, F, G, PS, FS) $\models p$›
⟨*proof*⟩

corollary ‹$\neg$ ($\vdash \bot$)›
⟨*proof*⟩

3.8 Derived Rules

lemma *Imp1*: ‹$\vdash q \longrightarrow p \longrightarrow q$›
and *Imp2*: ‹$\vdash (p \longrightarrow q \longrightarrow r) \longrightarrow (p \longrightarrow q) \longrightarrow p \longrightarrow r$›
and *Neg*: ‹$\vdash \neg \neg p \longrightarrow p$›
⟨*proof*⟩

The tautology axiom TA is not used directly beyond this point.

lemma *Tran′*: ‹$\vdash (q \longrightarrow r) \longrightarrow (p \longrightarrow q) \longrightarrow p \longrightarrow r$›
⟨*proof*⟩

lemma *Swap*: ‹$\vdash (p \longrightarrow q \longrightarrow r) \longrightarrow q \longrightarrow p \longrightarrow r$›
⟨*proof*⟩

lemma *Tran*: ‹$\vdash (p \longrightarrow q) \longrightarrow (q \longrightarrow r) \longrightarrow p \longrightarrow r$›
⟨*proof*⟩

Note that contraposition in the other direction is an instance of the lemma Tran.

lemma *contraposition*: ‹$\vdash (\neg q \longrightarrow \neg p) \longrightarrow p \longrightarrow q$›
⟨*proof*⟩

lemma *GR′*: ‹$\vdash \neg \langle \star a/0 \rangle p \longrightarrow q \Longrightarrow a \notin$ *params* $\{p, q\} \Longrightarrow \vdash \neg (\forall p) \longrightarrow q$›
⟨*proof*⟩

lemma *GR*$_P$*′*: ‹$\vdash \neg \langle \bigcirc_2 P/0 \rangle_P p \longrightarrow q \Longrightarrow P \notin$ *params* $\{p, q\} \Longrightarrow \vdash \neg (\forall_P p) \longrightarrow q$›
⟨*proof*⟩

lemma *GR*$_F$*′*: ‹$\vdash \neg \langle \bigcirc_2 F/0 \rangle_F p \longrightarrow q \Longrightarrow F \notin$ *params* $\{p, q\} \Longrightarrow \vdash \neg (\forall_F p) \longrightarrow q$›
⟨*proof*⟩

lemma *imply-ImpE*: ‹$\vdash ps \rightsquigarrow p \longrightarrow ps \rightsquigarrow (p \longrightarrow q) \longrightarrow ps \rightsquigarrow q$›
⟨*proof*⟩

lemma *MP′*: ‹$ps \vdash p \longrightarrow q \Longrightarrow ps \vdash p \Longrightarrow ps \vdash q$›
⟨*proof*⟩

lemma *imply-Cons* [*intro*]: ‹$ps \vdash q \Longrightarrow p \# ps \vdash q$›
⟨*proof*⟩

lemma *imply-head* [*intro*]: ‹$p \# ps \vdash p$›
⟨*proof*⟩

lemma *add-imply* [*simp*]: ‹$\vdash q \Longrightarrow ps \vdash q$›
⟨*proof*⟩

lemma *imply-mem* [*simp*]: ‹$p \in set\ ps \Longrightarrow ps \vdash p$›
⟨*proof*⟩

lemma *deduct1*: ‹$ps \vdash p \longrightarrow q \Longrightarrow p \# ps \vdash q$›
⟨*proof*⟩

lemma *imply-append* [*iff*]: ‹$(ps\ @\ qs \rightsquigarrow r) = (ps \rightsquigarrow qs \rightsquigarrow r)$›
⟨*proof*⟩

lemma *imply-swap-append*: ‹$ps\ @\ qs \vdash r \Longrightarrow qs\ @\ ps \vdash r$›
⟨*proof*⟩

lemma *deduct2*: ‹$p \# ps \vdash q \Longrightarrow ps \vdash p \longrightarrow q$›
⟨*proof*⟩

lemmas *deduct* [*iff*] = *deduct1 deduct2*

lemma *cut*: ‹$p \# ps \vdash r \Longrightarrow q \# ps \vdash p \Longrightarrow q \# ps \vdash r$›
⟨*proof*⟩

lemma *Boole*: ‹$(\neg p) \# ps \vdash \bot \Longrightarrow ps \vdash p$›
⟨*proof*⟩

lemma *imply-weaken*: ‹$ps \vdash q \Longrightarrow set\ ps \subseteq set\ ps' \Longrightarrow ps' \vdash q$›
⟨*proof*⟩

3.9 Derivational Consistency

interpretation *DC*: *Weak-Derivational-Confl map-fm params-fm* ‹λ-. *True*› *confl-class* ‹$\lambda A.\ \neg\ A \vdash \bot$›
⟨*proof*⟩

interpretation *DA*: *Weak-Derivational-Alpha map-fm params-fm* ‹λ-. *True*› *alpha-class* ‹$\lambda A.\ \neg\ A \vdash \bot$›
⟨*proof*⟩

interpretation *DB*: *Weak-Derivational-Beta map-fm params-fm* ‹λ-. *True*› *beta-class* ‹$\lambda A.\ \neg\ A \vdash \bot$›
⟨*proof*⟩

interpretation *DG*: *Weak-Derivational-Gamma map-tm map-fm params-fm* ‹λ-. *True*› *G.classify-UNIV* ‹λ*A*. ¬ *A* ⊢ ⊥›
⟨*proof*⟩

interpretation DG_P: *Weak-Derivational-Gamma map-sym map-fm params-fm* ‹λ-. *True*› G_P*.classify-UNIV* ‹λ*A*. ¬ *A* ⊢ ⊥›
⟨*proof*⟩

interpretation DG_F: *Weak-Derivational-Gamma map-sym map-fm params-fm* ‹λ-. *True*› G_F*.classify-UNIV* ‹λ*A*. ¬ *A* ⊢ ⊥›
⟨*proof*⟩

lemma *imply-params-fm*: ‹*params-fm* (*ps* ⇝ *q*) = *params-fm q* ∪ (⋃*p* ∈ *set ps*. *params-fm p*)›
⟨*proof*⟩

interpretation *DD*: *Weak-Derivational-Delta map-fm params-fm* ‹λ-. *True*› *δ* ‹λ*A*. ¬ *A* ⊢ ⊥›
⟨*proof*⟩

term *P.params*

interpretation *Weak-Derivational-Consistency map-fm params-fm* ‹λ-. *True*› *Kinds* ‹λ*A*. ¬ *A* ⊢ ⊥›
⟨*proof*⟩

theorem *weak-completeness*:
fixes *p* :: ‹'*x fm*›
assumes *mod*: ‹∀(*E* :: - ⇒ '*x tm*) E_F E_P *C F G PS FS*. *wf-model* (*E*, E_F, E_P, *C*, *F*, *G*, *PS*, *FS*) ⟶ (∀*q* ∈ *set ps*. (*E*, E_F, E_P, *C*, *F*, *G*, *PS*, *FS*) ⊨ *q*) ⟶ (*E*, E_F, E_P, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*›
and ‹*P.enough-new* (*set ps*)›
shows ‹*ps* ⊢ *p*›
⟨*proof*⟩

theorem *completeness*:
fixes *p* :: ‹'*x fm*›
assumes ‹∀(*E* :: *nat* ⇒ '*x tm*) E_P E_F *C F G PS FS*. *wf-model* (*E*, E_F, E_P, *C*, *F*, *G*, *PS*, *FS*) ⟶ (*E*, E_F, E_P, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*›
and ‹|*UNIV* :: '*x fm set*| ≤*o* |*UNIV* :: '*x set*|›
shows ‹⊢ *p*›
⟨*proof*⟩

3.10 Natural Deduction

locale *Natural-Deduction*
begin

inductive *ND-Set* :: ‹'*x fm set* ⇒ '*x fm* ⇒ *bool*› (**infix** ‹⊩› *50*) **where**

Assm [*dest*]: ‹$p \in A \Longrightarrow A \Vdash p$›
| *FlsE* [*elim*]: ‹$A \Vdash \bot \Longrightarrow A \Vdash p$›
| *ImpI* [*intro*]: ‹$\{p\} \cup A \Vdash q \Longrightarrow A \Vdash p \longrightarrow q$›
| *ImpE* [*dest*]: ‹$A \Vdash p \longrightarrow q \Longrightarrow A \Vdash p \Longrightarrow A \Vdash q$›
| *UniI* [*intro*]: ‹$A \Vdash \langle\star a/0\rangle p \Longrightarrow a \notin P.params\ (\{p\} \cup A) \Longrightarrow A \Vdash \forall\, p$›
| *UniE* [*dest*]: ‹$A \Vdash \forall\, p \Longrightarrow A \Vdash \langle t/0\rangle p$›
| $UniI_P$ [*intro*]: ‹$A \Vdash \langle\bigcirc_2\ a/0\rangle_P\ p \Longrightarrow a \notin P.params\ (\{p\} \cup A) \Longrightarrow A \Vdash \forall_P\ p$›
| $UniE_P$ [*dest*]: ‹$A \Vdash \forall_P\ p \Longrightarrow A \Vdash \langle s/0\rangle_P\ p$›
| $UniI_F$ [*intro*]: ‹$A \Vdash \langle\bigcirc_2\ a/0\rangle_F\ p \Longrightarrow a \notin P.params\ (\{p\} \cup A) \Longrightarrow A \Vdash \forall_F\ p$›
| $UniE_F$ [*dest*]: ‹$A \Vdash \forall_F\ p \Longrightarrow A \Vdash \langle s/0\rangle_F\ p$›
| *Clas*: ‹$\{p \longrightarrow q\} \cup A \Vdash p \Longrightarrow A \Vdash p$›

3.10.1 Soundness

theorem *soundness-set*:
assumes ‹$A \Vdash p$› ‹*wf-model* (E, E_F, E_P, C, F, G, PS, FS)›
shows ‹$\forall\, q \in A.\ (E, E_F, E_P, C, F, G, PS, FS) \models q \Longrightarrow (E, E_F, E_P, C, F, G, PS, FS) \models p$›
⟨*proof*⟩

3.10.2 Derivational Consistency

lemma *Boole*: ‹$\{\neg\ p\} \cup A \Vdash \bot \Longrightarrow A \Vdash p$›
⟨*proof*⟩

sublocale *DC*: *Derivational-Confl map-fm params-fm* ‹λ-. *True*› *confl-class* ‹$\lambda A.\ \neg\ A \Vdash \bot$›
⟨*proof*⟩

sublocale *DA*: *Derivational-Alpha map-fm params-fm* ‹λ-. *True*› *alpha-class* ‹$\lambda A.\ \neg\ A \Vdash \bot$›
⟨*proof*⟩

sublocale *DB*: *Derivational-Beta map-fm params-fm* ‹λ-. *True*› *beta-class* ‹$\lambda A.\ \neg\ A \Vdash \bot$›
⟨*proof*⟩

sublocale *DG*: *Derivational-Gamma map-tm map-fm params-fm* ‹λ-. *True*› *G.classify-UNIV* ‹$\lambda A.\ \neg\ A \Vdash \bot$›
⟨*proof*⟩

sublocale *DGP*: *Derivational-Gamma map-sym map-fm params-fm* ‹λ-. *True*› G_P*.classify-UNIV* ‹$\lambda A.\ \neg\ A \Vdash \bot$›
⟨*proof*⟩

sublocale *DGF*: *Derivational-Gamma map-sym map-fm params-fm* ‹λ-. *True*› G_F*.classify-UNIV* ‹$\lambda A.\ \neg\ A \Vdash \bot$›
⟨*proof*⟩

sublocale *DD*: *Derivational-Delta map-fm params-fm* ‹λ-. *True*› δ ‹$\lambda A.\ \neg\ A \Vdash \bot$›

⟨*proof*⟩

sublocale *Derivational-Consistency map-fm params-fm* ‹λ-. *True*› *Kinds* ‹λ*A*. ¬ *A* ⊩ ⊥›
⟨*proof*⟩

3.10.3 Strong Completeness

theorem *strong-completeness*:
fixes *p* :: ‹'*x fm*›
assumes *mod*: ‹⋀(*E* :: *nat* ⇒ '*x tm*) *E*$_F$ *E*$_P$ *C F G PS FS*. *wf-model* (*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⟹
(∀ *q* ∈ *A*. (*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ *q*) ⟹ (*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*›
and ‹*P.enough-new A*›
shows ‹*A* ⊩ *p*›
⟨*proof*⟩

proposition ‹∃ *M*. *wf-model M*›
⟨*proof*⟩

end

end

Chapter 4

Example: Prior's Ideal Logic

theory *Example-PIL* **imports**
Abstract-Consistency-Property
HOL−Number-Theory.Number-Theory
begin

no-syntax
-Pi :: *pttrn* ⇒ *'a set* ⇒ *'b set* ⇒ (*'a* ⇒ *'b*) *set*
(‹(‹*indent=3 notation=‹binder* Π∈››Π -∈-./ -)› *10*)

4.1 Syntax

datatype (*symbols-tm*: *'x*) *tm*
= *Var nat* (‹#›)
| *Sym 'x* (‹◯›)

datatype (*symbols-fm*: *'x*) *fm*
= *TmI* ‹*'x tm*› (‹•›)
| *TmP* ‹*'x tm*› (‹•›)
| *Neg* ‹*'x fm*› (‹¬ -› [*70*] *70*)
| *Con* ‹*'x fm*› ‹*'x fm*› (**infixr** ‹∧› *35*)
| *Box* ‹*'x fm*› (‹□›)
| *Sat* ‹*'x tm*› ‹*'x fm*› (‹**@**›)
| *Glo* ‹*'x fm*› (‹**A**›)
| *Dwn* ‹*'x fm*› (‹↓›)
| *All* ‹*'x fm*› (‹∀ ›)

abbreviation *Fls* :: ‹*'x fm*› (‹⊥›) **where**
‹⊥ ≡ *undefined* ∧ ¬ *undefined*›

abbreviation *Imp* :: ‹*'x fm* ⇒ *'x fm* ⇒ *'x fm*› (**infixr** ‹⟶› *55*) **where**
‹*p* ⟶ *q* ≡ ¬ (*p* ∧ ¬ *q*)›

abbreviation *Dia* :: ‹*'x fm* ⇒ *'x fm*› (‹◇›) **where**
‹◇ *p* ≡ ¬ (□ (¬ *p*))›

type-synonym *'x lbd* = ‹*'x tm* × *'x fm*›

4.2 Semantics

record *'w frame* =
  $\mathcal{W}$ :: ‹*'w set*›
  $\mathcal{R}$ :: ‹*'w* ⇒ *'w set*›

record *'w gframe* = ‹*'w frame*› +
  Π :: ‹*'w set set*›

record (*'x*, *'w*) *model* = ‹*'w gframe*› +
  $\mathcal{N}$ :: ‹*'x* ⇒ *'w*›
  $\mathfrak{N}$ :: ‹*nat* ⇒ *'w*›
  $\mathcal{V}$ :: ‹*'x* ⇒ *'w set*›
  $\mathfrak{V}$ :: ‹*nat* ⇒ *'w set*›

abbreviation ‹*Model W R PI N e V f* ≡ (|$\mathcal{W}$ = *W*, $\mathcal{R}$ = *R*, Π = *PI*, $\mathcal{N}$ = *N*, $\mathfrak{N}$ = *e*, $\mathcal{V}$ = *V*, $\mathfrak{V}$ = *f*|)›

definition *admissible* :: ‹*'w frame* ⇒ *'w set set* ⇒ *bool*› **where**
  ‹*admissible M PI* ≡
    (∀ *X* ∈ *PI*. $\mathcal{W}$ *M* − *X* ∈ *PI*) ∧
    (∀ *X* ∈ *PI*. ∀ *Y* ∈ *PI*. *X* ∩ *Y* ∈ *PI*) ∧
    (∀ *X* ∈ *PI*. {*w* ∈ $\mathcal{W}$ *M*. ∀ *v* ∈ $\mathcal{R}$ *M w*. *v* ∈ *X*} ∈ *PI*)›

definition *wf-frame* :: ‹*'w frame* ⇒ *bool*› **where**
  ‹*wf-frame M* ≡ $\mathcal{W}$ *M* ≠ {} ∧ $\mathcal{R}$ *M* ‘ $\mathcal{W}$ *M* ⊆ *Pow* ($\mathcal{W}$ *M*)›

definition *wf-gframe* :: ‹*'w gframe* ⇒ *bool*› **where**
  ‹*wf-gframe M* ≡ *wf-frame* (*frame.truncate M*) ∧ Π *M* ≠ {} ∧ Π *M* ⊆ *Pow* ($\mathcal{W}$ *M*) ∧ *admissible* (*frame.truncate M*) (Π *M*)›

definition *wf-env* :: ‹(*'x*, *'w*) *model* ⇒ *bool*› **where**
  ‹*wf-env M* ≡ *range* ($\mathcal{N}$ *M*) ⊆ $\mathcal{W}$ *M* ∧ *range* ($\mathfrak{N}$ *M*) ⊆ $\mathcal{W}$ *M* ∧ *range* ($\mathcal{V}$ *M*) ⊆ Π *M* ∧ *range* ($\mathfrak{V}$ *M*) ⊆ Π *M*›

definition *wf-model* :: ‹(*'x*, *'w*) *model* ⇒ *bool*› **where**
  ‹*wf-model M* ≡ *wf-gframe* (*gframe.truncate M*) ∧ *wf-env M*›

lemmas *unfolds* =
  model.defs gframe.defs frame.defs
  model.select-convs gframe.select-convs frame.select-convs

context
  fixes *M* :: ‹(*'x*, *'w*) *model*›
  assumes *wf*: ‹*wf-model M*›
begin

lemma *wf-compl*: ‹$X \in \Pi\ M \Longrightarrow \mathcal{W}\ M - X \in \Pi\ M$›
⟨*proof*⟩

lemma *wf-inter*: ‹$X \in \Pi\ M \Longrightarrow Y \in \Pi\ M \Longrightarrow X \cap Y \in \Pi\ M$›
⟨*proof*⟩

lemma *wf-modal*: ‹$X \in \Pi\ M \Longrightarrow \{w \in \mathcal{W}\ M.\ \forall\, v \in \mathcal{R}\ M\ w.\ v \in X\} \in \Pi\ M$›
⟨*proof*⟩

lemma *wf-empty*: ‹$\{\} \in \Pi\ M$›
⟨*proof*⟩

lemma *wf-univ*: ‹$\mathcal{W}\ M \in \Pi\ M$›
⟨*proof*⟩

lemma *wf-*Π: ‹$P \in \Pi\ M \Longrightarrow P \subseteq \mathcal{W}\ M$›
⟨*proof*⟩

lemma *wf-*$\mathcal{N}$: ‹$\mathcal{N}\ M\ i \in \mathcal{W}\ M$›
⟨*proof*⟩

lemma *wf-*$\mathfrak{N}$: ‹$\mathfrak{N}\ M\ n \in \mathcal{W}\ M$›
⟨*proof*⟩

lemma *wf-*$\mathcal{V}$: ‹$\mathcal{V}\ M\ P \in \Pi\ M$›
⟨*proof*⟩

lemma *wf-*$\mathcal{V}'$: ‹$\mathcal{V}\ M\ n \subseteq \mathcal{W}\ M$›
⟨*proof*⟩

lemma *wf-*$\mathfrak{V}$: ‹$\mathfrak{V}\ M\ n \in \Pi\ M$›
⟨*proof*⟩

lemma *wf-*$\mathfrak{V}'$: ‹$\mathfrak{V}\ M\ n \subseteq \mathcal{W}\ M$›
⟨*proof*⟩

end

type-synonym $('x, 'w)\ ctx$ = ‹$('x, 'w)\ model \times 'w$›

primrec *add-env* :: ‹$'a \Rightarrow (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a$› (**infixr** ‹$\gg$› *0*) **where**
⟨$(t \gg e)\ 0 = t$›
| ⟨$(\text{-} \gg e)\ (Suc\ n) = e\ n$›

fun *semantics* :: ‹$('x, 'w)\ ctx \Rightarrow 'x\ fm \Rightarrow bool$› (**infix** ‹$\models$› *50*) **where**
⟨$((M, w) \models \cdot t) = (case\text{-}tm\ (\mathfrak{N}\ M)\ (\mathcal{N}\ M)\ t = w)$›
| ⟨$((M, w) \models \cdot P) = (w \in case\text{-}tm\ (\mathfrak{V}\ M)\ (\mathcal{V}\ M)\ P)$›
| ⟨$((M, w) \models (\neg\ p)) = (\neg\ (M, w) \models p)$›

| ‹((M, w) ⊨ (p ∧ q)) = ((M, w) ⊨ p ∧ (M, w) ⊨ q)›
| ‹((M, w) ⊨ □ p) = (∀ v ∈ ℛ M w. (M, v) ⊨ p)›
| ‹((M, -) ⊨ @i p) = ((M, case-tm (𝔑 M) (𝒩 M) i) ⊨ p)›
| ‹((M, -) ⊨ **A** p) = (∀ v ∈ 𝒲 M. (M, v) ⊨ p)›
| ‹((M, w) ⊨ ↓ p) = ((M(|𝔑 := (w ≫ 𝔑 M)|), w) ⊨ p)›
| ‹((M, w) ⊨ ∀ p) = (∀ P ∈ Π M. (M(|𝔙 := (P ≫ 𝔙 M)|), w) ⊨ p)›

lemma ‹(M, w) ⊨ p ⟶ q ⟷ (M, w) ⊨ p ⟶ (M, w) ⊨ q›
⟨*proof*⟩

lemma ‹(M, w) ⊨ ↓(•(#0))›
⟨*proof*⟩

lemma ‹(M, w) ⊨ ∀ (@(○k) (•(#0))) ⟶ @(○k) (∀ (•(#0)))›
⟨*proof*⟩

lemma ‹(M, w) ⊨ @(○k) (∀ (•(#0))) ⟶ ∀ (@(○k) (•(#0)))›
⟨*proof*⟩

lemma ‹*wf-model* M ⟹ (M, w) ⊨ ∀ (•(#0)) ⟶ •P›
⟨*proof*⟩

lemma ‹*wf-model* M ⟹ (M, w) ⊨ ∀ (@(○k) (•(#0))) ⟶ ↓(•(○k) ⟶ @(#0) (•P))›
⟨*proof*⟩

4.3 Operations

abbreviation *map-lbd* :: ‹('x ⇒ 'k) ⇒ 'x lbd ⇒ 'k lbd› **where**
‹*map-lbd* f ≡ *map-prod* (*map-tm* f) (*map-fm* f)›

primrec *symbols-lbd* :: ‹'x lbd ⇒ 'x set› **where**
‹*symbols-lbd* (i, p) = *symbols-tm* i ∪ *symbols-fm* p›

abbreviation *symbols* :: ‹'x lbd set ⇒ 'x set› **where**
‹*symbols* S ≡ ⋃ ip ∈ S. *symbols-lbd* ip›

abbreviation *lift-tm* :: ‹nat ⇒ 'x tm ⇒ 'x tm› **where**
‹*lift-tm* m ≡ *case-tm* (λn. *if* n < m *then* #n *else* #(n + m + 1)) ○›

primrec *vars-tm* :: ‹nat ⇒ 'x tm ⇒ nat set› **where**
‹*vars-tm* m (#n) = (*if* n ≤ m *then* {} *else* {n})›
| ‹*vars-tm* - (○-) = {}›

primrec *vars-fm* :: ‹nat ⇒ 'x fm ⇒ nat set› **where**
‹*vars-fm* m (•t) = *vars-tm* m t›
| ‹*vars-fm* m (•P) = *vars-tm* m P›
| ‹*vars-fm* m (¬ p) = *vars-fm* m p›
| ‹*vars-fm* m (p ∧ q) = *vars-fm* m p ∪ *vars-fm* m q›

| ‹*vars-fm m* (□ *p*) = *vars-fm m p*›
| ‹*vars-fm m* (@*i p*) = *vars-tm m i* ∪ *vars-fm m p*›
| ‹*vars-fm m* (**A** *p*) = *vars-fm m p*›
| ‹*vars-fm m* (↓ *p*) = *vars-fm* (*m* + *1*) *p*›
| ‹*vars-fm m* (∀ *p*) = *vars-fm* (*m* + *1*) *p*›

Nominals

primrec *sub-tm* :: ‹(*nat* ⇒ *'x tm*) ⇒ *'x tm* ⇒ *'x tm*› **where**
‹*sub-tm s* (#*n*) = *s n*›
| ‹*sub-tm* - (○*k*) = ○*k*›

primrec *sub-nom* :: ‹(*nat* ⇒ *'x tm*) ⇒ *'x fm* ⇒ *'x fm*› **where**
‹*sub-nom s* (•*t*) = •(*sub-tm s t*)›
| ‹*sub-nom* - (•*P*) = •*P*›
| ‹*sub-nom s* (¬ *p*) = ¬ *sub-nom s p*›
| ‹*sub-nom s* (*p* ∧ *q*) = (*sub-nom s p* ∧ *sub-nom s q*)›
| ‹*sub-nom s* (□ *p*) = □(*sub-nom s p*)›
| ‹*sub-nom s* (@*i p*) = @(*sub-tm s i*) (*sub-nom s p*)›
| ‹*sub-nom s* (**A** *p*) = **A** (*sub-nom s p*)›
| ‹*sub-nom s* (↓ *p*) = ↓ (*sub-nom* (#*0* ≫ λ*n*. *lift-tm 0* (*s n*)) *p*)›
| ‹*sub-nom s* (∀ *p*) = ∀ (*sub-nom s p*)›

abbreviation *inst-single-nom* :: ‹*'x tm* ⇒ *'x fm* ⇒ *'x fm*› (‹⟨-⟩$_i$›) **where**
‹⟨*t*⟩$_i$ ≡ *sub-nom* (*t* ≫ #)›

Propositions

fun *softqdf* :: ‹*'x fm* ⇒ *bool*› **where**
‹*softqdf* (•-) = *False*›
| ‹*softqdf* (•*P*) = *True*›
| ‹*softqdf* (¬ *p*) = *softqdf p*›
| ‹*softqdf* (*p* ∧ *q*) = (*softqdf p* ∧ *softqdf q*)›
| ‹*softqdf* (□ *p*) = *softqdf p*›
| ‹*softqdf* (@*i p*) = *softqdf p*›
| ‹*softqdf* (**A** *p*) = *softqdf p*›
| ‹*softqdf* (↓ *p*) = *False*›
| ‹*softqdf* (∀ *p*) = *False*›

abbreviation *softqdf-sub* :: ‹(*nat* ⇒ *'x fm*) ⇒ *bool*› **where**
‹*softqdf-sub s* ≡ ∀ *n*. *softqdf* (*s n*)›

primrec *lift-fm-nom* :: ‹*nat* ⇒ *'x fm* ⇒ *'x fm*› **where**
‹*lift-fm-nom m* (•*t*) = •(*lift-tm m t*)›
| ‹*lift-fm-nom* - (•*P*) = •*P*›
| ‹*lift-fm-nom m* (¬ *p*) = ¬ *lift-fm-nom m p*›
| ‹*lift-fm-nom m* (*p* ∧ *q*) = (*lift-fm-nom m p* ∧ *lift-fm-nom m q*)›
| ‹*lift-fm-nom m* (□ *p*) = □(*lift-fm-nom m p*)›
| ‹*lift-fm-nom m* (@*i p*) = @(*lift-tm m i*) (*lift-fm-nom m p*)›
| ‹*lift-fm-nom m* (**A** *p*) = **A** (*lift-fm-nom m p*)›

| ‹*lift-fm-nom m* (↓ *p*) = ↓ (*lift-fm-nom* (*m* + *1*) *p*)›
| ‹*lift-fm-nom m* (∀ *p*) = ∀ (*lift-fm-nom m p*)›

primrec *lift-fm-pro* :: ‹*nat* ⇒ *'x fm* ⇒ *'x fm*› **where**
‹*lift-fm-pro* - (•*t*) = •*t*›
| ‹*lift-fm-pro m* (•*P*) = •(*lift-tm m P*)›
| ‹*lift-fm-pro m* (¬ *p*) = ¬ *lift-fm-pro m p*›
| ‹*lift-fm-pro m* (*p* ∧ *q*) = (*lift-fm-pro m p* ∧ *lift-fm-pro m q*)›
| ‹*lift-fm-pro m* (□ *p*) = □(*lift-fm-pro m p*)›
| ‹*lift-fm-pro m* (@*i p*) = @*i* (*lift-fm-pro m p*)›
| ‹*lift-fm-pro m* (**A** *p*) = **A** (*lift-fm-pro m p*)›
| ‹*lift-fm-pro m* (↓ *p*) = ↓ (*lift-fm-pro m p*)›
| ‹*lift-fm-pro m* (∀ *p*) = ∀ (*lift-fm-pro* (*m* + *1*) *p*)›

primrec *sub-pro* :: ‹(*nat* ⇒ *'x fm*) ⇒ *'x fm* ⇒ *'x fm*› **where**
‹*sub-pro* - (•*t*) = •*t*›
| ‹*sub-pro s* (•*P*) = *case-tm s* (• *o* ◯) *P*›
| ‹*sub-pro s* (¬ *p*) = ¬ *sub-pro s p*›
| ‹*sub-pro s* (*p* ∧ *q*) = (*sub-pro s p* ∧ *sub-pro s q*)›
| ‹*sub-pro s* (□ *p*) = □(*sub-pro s p*)›
| ‹*sub-pro s* (@*i p*) = @*i* (*sub-pro s p*)›
| ‹*sub-pro s* (**A** *p*) = **A** (*sub-pro s p*)›
| ‹*sub-pro s* (↓ *p*) = ↓ (*sub-pro* (*lift-fm-nom 0 o s*) *p*)›
| ‹*sub-pro s* (∀ *p*) = ∀ (*sub-pro* (•(#*0*) ≫ λ*n*. *lift-fm-pro 0* (*s n*)) *p*)›

abbreviation *inst-single-pro* :: ‹*'x fm* ⇒ *'x fm* ⇒ *'x fm*› (‹⟨-⟩$_p$›) **where**
‹⟨*q*⟩$_p$ ≡ *sub-pro* (*q* ≫ • *o* #)›

primrec *sz-fm* :: ‹*'x fm* ⇒ *nat*› **where**
‹*sz-fm* (•*t*) = *1*›
| ‹*sz-fm* (•*P*) = *1*›
| ‹*sz-fm* (¬ *p*) = *sz-fm p* + *1*›
| ‹*sz-fm* (*p* ∧ *q*) = *sz-fm p* + *sz-fm q* + *1*›
| ‹*sz-fm* (□ *p*) = *sz-fm p* + *1*›
| ‹*sz-fm* (@*i p*) = *sz-fm p* + *1*›
| ‹*sz-fm* (**A** *p*) = *sz-fm p* + *1*›
| ‹*sz-fm* (↓ *p*) = *sz-fm p* + *1*›
| ‹*sz-fm* (∀ *p*) = *sz-fm p* + *1*›

primrec *qs-fm* :: ‹*'x fm* ⇒ *nat*› **where**
‹*qs-fm* (•*t*) = *0*›
| ‹*qs-fm* (•*P*) = *0*›
| ‹*qs-fm* (¬ *p*) = *qs-fm p*›
| ‹*qs-fm* (*p* ∧ *q*) = *max* (*qs-fm p*) (*qs-fm q*)›
| ‹*qs-fm* (□ *p*) = *qs-fm p*›
| ‹*qs-fm* (@*i p*) = *qs-fm p*›
| ‹*qs-fm* (**A** *p*) = *qs-fm p*›
| ‹*qs-fm* (↓ *p*) = *qs-fm p*›
| ‹*qs-fm* (∀ *p*) = *qs-fm p* + *1*›

4.3.1 Lemmas

Finite

lemma *finite-symbols-tm* [*simp*]: ‹*finite* (*symbols-tm t*)›
⟨*proof*⟩

lemma *finite-symbols-fm* [*simp*]: ‹*finite* (*symbols-fm p*)›
⟨*proof*⟩

lemma *finite-symbols-lbd*: ‹*finite* (*symbols-lbd p*)›
⟨*proof*⟩

Terms

lemma *env* [*simp*]: ‹$P\ ((x \gg E)\ n) = (P\ x \gg \lambda n.\ P\ (E\ n))\ n$›
⟨*proof*⟩

lemma *lift-lemma* [*simp*]: ‹*case-tm* $(x \gg e)$ *s* (*lift-tm 0 t*) = *case-tm e s t*›
⟨*proof*⟩

lemma *sub-tm-semantics* [*simp*]: ‹*case-tm e g* (*sub-tm s t*) = *case-tm* ($\lambda n.$ *case-tm e g* (*s n*)) *g t*›
⟨*proof*⟩

lemma *semantics-tm-id* [*simp*]: ‹*case-tm* # ○ *t* = *t*›
⟨*proof*⟩

lemma *semantics-tm-id-map* [*simp*]: ‹*map* (*case-tm* # ○) *ts* = *ts*›
⟨*proof*⟩

lemma *map-tm-sub-tm* [*simp*]: ‹*map-tm f* (*sub-tm g t*) = *sub-tm* (*map-tm f o g*) (*map-tm f t*)›
⟨*proof*⟩

lemma *map-tm-lift-tm* [*simp*]: ‹*map-tm f* (*lift-tm m t*) = *lift-tm m* (*map-tm f t*)›
⟨*proof*⟩

lemma *semantics-tm-fresh* [*simp*]: ‹$x \notin$ *symbols-tm t* $\Longrightarrow$ *case-tm e* ($g(x := a)$) *t* = *case-tm e g t*›
⟨*proof*⟩

lemma *map-tm-inst-single*: ‹(*map-tm f o* $(u \gg \#)$) *t* = (*map-tm f u* $\gg$ #) *t*›
⟨*proof*⟩

Nominals

lemma *size-sub-nom* [*simp*]: ‹*sz-fm* (*sub-nom s p*) = *sz-fm p*›
⟨*proof*⟩

lemma *semantics-symbols-cong-nom*:
assumes ‹$\forall\, i \in$ *symbols-fm p*. $N\ i = N'\ i$›
shows ‹(*Model W R PI N e V f*, *w*) $\models p \longleftrightarrow$ (*Model W R PI N' e V f*, *w*) $\models p$›
⟨*proof*⟩

corollary *semantics-symbols-other-nom* [*simp*]:
assumes ‹$i \notin$ *symbols-fm p*›
shows ‹(*Model W R PI* (*N*(*i* := *x*)) *e V f*, *w*) $\models p \longleftrightarrow$ (*Model W R PI N e V f*, *w*) $\models p$›
⟨*proof*⟩

lemma *sub-nom-semantics* [*simp*]:
‹(*Model W R PI N e V f*, *w*) $\models$ *sub-nom s p* $\longleftrightarrow$ (*Model W R PI N* (λ*n*. *case-tm e N* (*s n*)) *V f*, *w*) $\models p$›
⟨*proof*⟩

lemma *map-fm-sub-nom*: ‹*map-fm f* (*sub-nom s p*) = *sub-nom* (*map-tm f o s*) (*map-fm f p*)›
⟨*proof*⟩

lemma *map-fm-inst-single-nom* [*simp*]: ‹*map-fm f* ($\langle t \rangle_i$ *p*) = $\langle$*map-tm f t*$\rangle_i$ (*map-fm f p*)›
⟨*proof*⟩

Propositions

lemma *semantics-symbols-cong-pro*:
‹$\forall\, P \in$ *symbols-fm p*. *V P* = *V' P* $\Longrightarrow$
(*Model W R PI N e V f*, *w*) $\models p \longleftrightarrow$ (*Model W R PI N e V' f*, *w*) $\models p$›
⟨*proof*⟩

corollary *semantics-symbols-other-pro* [*simp*]:
assumes ‹$P \notin$ *symbols-fm p*›
shows ‹(*Model W R PI N e* (*V*(*P* := *x*)) *f*, *w*) $\models p \longleftrightarrow$ (*Model W R PI N e V f*, *w*) $\models p$›
⟨*proof*⟩

lemma *semantics-symbols-lbd-cong-pro*:
‹$\forall\, P \in$ *symbols-lbd* (*i*, *p*). *V P* = *V' P* $\Longrightarrow$
(*Model W R PI N e V f*, *w*) $\models p \longleftrightarrow$ (*Model W R PI N e V' f*, *w*) $\models p$›
⟨*proof*⟩

lemma *map-lift-fm-pro* [*simp*]: ‹*map-fm f* (*lift-fm-pro m p*) = *lift-fm-pro m* (*map-fm f p*)›
⟨*proof*⟩

lemma *map-lift-fm-nom* [*simp*]: ‹*map-fm f* (*lift-fm-nom m p*) = *lift-fm-nom m* (*map-fm f p*)›
⟨*proof*⟩

lemma *map-fm-sub-pro*: ‹*map-fm f* (*sub-pro s p*) = *sub-pro* (*map-fm f o s*) (*map-fm f p*)›
⟨*proof*⟩

lemma *map-fm-inst-single*: ‹(*map-fm f o* (*q* ≫ · *o* #)) *p* = (*map-fm f q* ≫ · *o* #) *p*›
⟨*proof*⟩

lemma *map-fm-inst-single-pro* [*simp*]: ‹*map-fm f* (⟨*q*⟩$_p$ *p*) = ⟨*map-fm f q*⟩$_p$ (*map-fm f p*)›
⟨*proof*⟩

4.3.2 softqdf

lemma *softqdf-map-fm* [*simp*]: ‹*softqdf* (*map-fm f p*) ⟷ *softqdf p*›
⟨*proof*⟩

lemma *softqdf-lift-fm-nom* [*simp*]: ‹*softqdf* (*lift-fm-nom m q*) ⟷ *softqdf q*›
⟨*proof*⟩

lemma *softqdf-lift-fm-pro* [*simp*]: ‹*softqdf* (*lift-fm-pro m q*) ⟷ *softqdf q*›
⟨*proof*⟩

4.3.3 Add env

lemma *range-add-env*:
assumes ‹*range f* ⊆ *A*› ‹*a* ∈ *A*›
shows ‹*range* (*a* ≫ *f*) ⊆ *A*›
⟨*proof*⟩

lemma *softqdf-add-env*: ‹*softqdf q* ⟹ *softqdf-sub* (*q* ≫ · *o* #)›
⟨*proof*⟩

lemma *wf-env-add-nom*: ‹*wf-env* (*Model W R PI N e V f*) ⟹ *w* ∈ *W* ⟹
wf-env (⦇$\mathcal{W}$ = *W*, $\mathcal{R}$ = *R*, Π = *PI*, $\mathcal{N}$ = *N*, $\mathfrak{N}$ = *w* ≫ *e*, $\mathcal{V}$ = *V*, $\mathfrak{V}$ = *f*⦈)›
⟨*proof*⟩

lemma *wf-model-add-nom*: ‹*wf-model* (*Model W R PI N e V f*) ⟹ *w* ∈ *W* ⟹
wf-model (⦇$\mathcal{W}$ = *W*, $\mathcal{R}$ = *R*, Π = *PI*, $\mathcal{N}$ = *N*, $\mathfrak{N}$ = *w* ≫ *e*, $\mathcal{V}$ = *V*, $\mathfrak{V}$ = *f*⦈)›
⟨*proof*⟩

lemma *wf-env-add-pro*: ‹*wf-env* (*Model W R PI N e V f*) ⟹ *P* ∈ *PI* ⟹
wf-env (⦇$\mathcal{W}$ = *W*, $\mathcal{R}$ = *R*, Π = *PI*, $\mathcal{N}$ = *N*, $\mathfrak{N}$ = *e*, $\mathcal{V}$ = *V*, $\mathfrak{V}$ = *P* ≫ *f*⦈)›
⟨*proof*⟩

lemma *wf-model-add-pro*:
‹*wf-model* (*Model W R PI N e V f*) ⟹ *P* ∈ *PI* ⟹ *wf-model* (*Model W R PI N e V* (*P* ≫ *f*))›
⟨*proof*⟩

lemma *softqdf-lift-fm-nom-add-env* [*simp*]:
‹*softqdf p* ⟹ (*Model W R PI N* (*v* ≫ *e*) *V f*, *w*) ⊨ *lift-fm-nom 0 p* ⟷ (*Model W R PI N e V f*, *w*) ⊨ *p*›
⟨*proof*⟩

lemma *softqdf-lift-fm-pro-add-env* [*simp*]:
‹*softqdf p* ⟹ (*Model W R PI N e V* (*P* ≫ *f*), *w*) ⊨ *lift-fm-pro 0 p* ⟷ (*Model W R PI N e V f*, *w*) ⊨ *p*›
⟨*proof*⟩

4.3.4 Sizes

lemma *qs-fm-sub-nom* [*simp*]: ‹*qs-fm* (*sub-nom s p*) = *qs-fm p*›
⟨*proof*⟩

lemma *softqdf-qs-fm* [*simp*]: ‹*softqdf q* ⟹ *qs-fm q* = *0*›
⟨*proof*⟩

lemma *softqdf-sub-pro*: ‹*softqdf-sub s* ⟹ *softqdf-sub* (•(#*0*) ≫ λ*n*. *lift-fm-pro 0* (*s n*))›
⟨*proof*⟩

lemma *qs-fm-sub-pro* [*simp*]: ‹*softqdf-sub s* ⟹ *qs-fm* (*sub-pro s p*) = *qs-fm p*›
⟨*proof*⟩

4.4 Propositional Quantification

definition *worlds* :: ‹('*x*, '*w*) *model* ⇒ '*x fm* ⇒ '*w set*› **where**
‹*worlds M p* ≡ { *w* ∈ *W M*. (*M*, *w*) ⊨ *p* }›

lemma *worlds-op* [*simp*]:
assumes ‹*wf-model M*›
shows
‹*worlds M* (¬ *p*) = *W M* − *worlds M p*›
‹*worlds M* (*p* ∧ *q*) = *worlds M p* ∩ *worlds M q*›
‹*worlds M* (□ *p*) = {*w* ∈ *W M*. ∀ *v* ∈ *R M w*. *v* ∈ *worlds M p*}›
⟨*proof*⟩

lemma *softqdf-worlds*:
assumes ‹*wf-model M*› ‹*softqdf p*›
shows ‹*worlds M p* ∈ Π *M*›
⟨*proof*⟩

definition *sqdfs* :: ‹('*x*, '*w*) *model* ⇒ '*w set set*› **where**
‹*sqdfs M* ≡ { *worlds M p* |*p*. *softqdf p* }›

lemma *sqdfs*:
assumes ‹*wf-model M*›

shows ‹*sqdfs M* ⊆ Π *M*›
⟨*proof*⟩

lemma *sqdfs-admissible*:
assumes ‹*wf-model M*›
shows ‹*admissible* (*frame.truncate M*) (*sqdfs M*)›
⟨*proof*⟩

definition *with-worlds* :: ‹('*x*, '*w*) *model* ⇒ (*nat* ⇒ '*x fm*) ⇒ ('*x*, '*w*) *model*›
where
‹*with-worlds M s* ≡ *M*(|𝔚 := *worlds M o s*|)›

lemma *sub-pro-with-worlds*:
assumes ‹*wf-model* (*Model W R PI N e V f*)› ‹*w* ∈ *W*› ‹*softqdf-sub s*›
shows ‹(*Model W R PI N e V f*, *w*) ⊨ *sub-pro s p* ⟷ (*with-worlds* (*Model W R PI N e V f*) *s*, *w*) ⊨ *p*›
⟨*proof*⟩

lemma *worlds-id-sub*:
assumes ‹*wf-model* (*Model W R PI N e V f*)›
shows ‹*worlds* (*Model W R PI N e V f*) (· (# *n*)) = *f n*›
⟨*proof*⟩

lemma *worlds-inst-single-pro*:
assumes ‹*wf-model* (*Model W R PI N e V f*)›
shows ‹*worlds* (*Model W R PI N e V f*) *o* (*q* ≫ · *o* #) = (*worlds* (*Model W R PI N e V f*) *q* ≫ *f*)›
⟨*proof*⟩

corollary *inst-single-worlds*:
assumes ‹*wf-model* (*Model W R PI N e V f*)› ‹*w* ∈ *W*› ‹*softqdf q*›
shows
‹(*Model W R PI N e V f*, *w*) ⊨ ⟨*q*⟩$_p$ *p* ⟷
(*Model W R PI N e V* (*worlds* (*Model W R PI N e V f*) *q* ≫ *f*), *w*) ⊨ *p*›
⟨*proof*⟩

4.5 Model Existence

inductive *confl-class* :: ‹'*x lbd list* ⇒ '*x lbd list* ⇒ *bool*› (**infix** ‹⇝✗› *50*) **where**
CNegP: ‹[(*i*, ¬ ·*P*)] ⇝✗ [(*i*, ·*P*)]›
| *CNegI*: ‹[(*i*, ¬ ·*k*)] ⇝✗ [(*i*, ·*k*)]›

inductive *alpha-class* :: ‹'*x lbd list* ⇒ '*x lbd list* ⇒ *bool*› (**infix** ‹⇝$_\alpha$› *50*) **where**
CSym: ‹[(*i*, ·*k*)] ⇝$_\alpha$ [(*k*, ·*i*)]›
| *CNom*: ‹[(*i*, ·*k*), (*i*, *p*)] ⇝$_\alpha$ [(*k*, *p*)]›
| *CNegN*: ‹[(*i*, ¬ ¬ *p*)] ⇝$_\alpha$ [(*i*, *p*)]›
| *CConP*: ‹[(*i*, *p* ∧ *q*)] ⇝$_\alpha$ [(*i*, *p*), (*i*, *q*)]›
| *CSatP*: ‹[(*i*, @*k p*)] ⇝$_\alpha$ [(*k*, *p*)]›
| *CSatN*: ‹[(*i*, ¬ @*k p*)] ⇝$_\alpha$ [(*k*, ¬ *p*)]›

| *CBoxP*: ‹[$(i, \Box p)$, $(i, \Diamond(\cdot k))$] $\rightsquigarrow_\alpha$ $[(k, p)]$›
| *CDwnP*: ‹[$(i, \downarrow p)$] $\rightsquigarrow_\alpha$ [$(i, \langle i \rangle_i\ p)$]›
| *CDwnN*: ‹[$(i, \neg \downarrow p)$] $\rightsquigarrow_\alpha$ [$(i, \neg \langle i \rangle_i\ p)$]›

inductive *beta-class* :: ‹*'x lbd list* ⇒ *'x lbd list* ⇒ *bool*› (**infix** ‹$\rightsquigarrow_\beta$› *50*) **where**
CConN: ‹[$(i, \neg (p \wedge q))$] $\rightsquigarrow_\beta$ $[(i, \neg p), (i, \neg q)]$›

inductive *gamma-class-nom* :: ‹*'x lbd list* ⇒ (*'x tm* ⇒ -) ⇒ *bool*› (**infix** ‹$\rightsquigarrow_{\gamma i}$› *50*) **where**
CRefl: ‹[] $\rightsquigarrow_{\gamma i}$ $(\lambda i.$ [$(i, \cdot i)$])›
| *CGloP*: ‹[$(i, \mathbf{A}\ p)$] $\rightsquigarrow_{\gamma i}$ $(\lambda k.$ [(k, p)])›

inductive *gamma-class-fm* :: ‹*'x lbd list* ⇒ (*'x lbd set* ⇒ *'x fm set*) × (*'x fm* ⇒ -) ⇒ *bool*› (**infix** ‹$\rightsquigarrow_{\gamma p}$› *50*) **where**
CAllP: ‹[$(i, \forall\ p)$] $\rightsquigarrow_{\gamma p}$ $(\lambda\text{-}.\ \{q.\ softqdf\ q\}, \lambda q.$ [$(i, \langle q \rangle_p\ p)$])›

fun δ :: ‹*'x lbd* ⇒ *'x* ⇒ *'x lbd list*› **where**
CBoxN: ‹δ $(i, \neg \Box p)$ $k = [(\bigcirc k, \neg p), (i, \Diamond (\cdot (\bigcirc k)))]$›
| *CGloN*: ‹δ $(i, \neg \mathbf{A}\ p)$ $k =$ [$(\bigcirc k, \neg p)$]›
| *CAllN*: ‹δ $(i, \neg \forall\ p)$ $P =$ [$(i, \neg \langle \cdot(\bigcirc P) \rangle_p\ p)$]›
| ‹δ - - = []›

interpretation *P*: *Params map-lbd symbols-lbd* ‹λ-. *True*›
⟨*proof*⟩

interpretation *C*: *Confl map-lbd symbols-lbd* ‹λ-. *True*› *confl-class*
⟨*proof*⟩

interpretation *A*: *Alpha map-lbd symbols-lbd* ‹λ-. *True*› *alpha-class*
⟨*proof*⟩

interpretation *B*: *Beta map-lbd symbols-lbd* ‹λ-. *True*› *beta-class*
⟨*proof*⟩

interpretation *GI*: *Gamma-UNIV map-tm map-lbd symbols-lbd* ‹λ-. *True*› *gamma-class-nom*
⟨*proof*⟩

interpretation *GP*: *Gamma map-fm map-lbd symbols-lbd* ‹λ-. *True*› *gamma-class-fm*
⟨*proof*⟩

interpretation *D*: *Delta map-lbd symbols-lbd* ‹λ-. *True*› δ
⟨*proof*⟩

abbreviation *Kinds* :: ‹(*'x*, *'x lbd*) *kind list*› **where**
‹*Kinds* ≡ [*C.kind*, *A.kind*, *B.kind*, *GI.kind*, *GP.kind*, *D.kind*]›

lemma *prop$_E$-Kinds*:
assumes ‹*P.sat$_E$ C.kind C*› ‹*P.sat$_E$ A.kind C*› ‹*P.sat$_E$ B.kind C*› ‹*P.sat$_E$ GI.kind C*› ‹*P.sat$_E$ GP.kind C*› ‹*P.sat$_E$ D.kind C*›

shows ‹$P.prop_E$ *Kinds* C›
⟨*proof*⟩

interpretation *Consistency-Kinds map-lbd symbols-lbd* ‹λ-. *True*› *Kinds*
⟨*proof*⟩

interpretation *Maximal-Consistency map-lbd symbols-lbd* ‹λ-. *True*› *Kinds*
⟨*proof*⟩

context begin

lemma ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \cdot P) \notin S \vee (i, \neg \cdot P) \notin S$›
⟨*proof*⟩

lemma ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i,\ p \wedge q) \in S \Longrightarrow \{(i,\ p),\ (i,\ q)\} \cup S \in C$›
⟨*proof*⟩

lemma ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \neg\ (p \wedge q)) \in S \Longrightarrow \{(i, \neg\ p)\} \cup S \in C \vee \{(i, \neg\ q)\} \cup S \in C$›
⟨*proof*⟩

lemma ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \Box\ p) \in S \Longrightarrow (i, \Diamond(\cdot k)) \in S \Longrightarrow \{(k,\ p)\} \cup S \in C$›
⟨*proof*⟩

lemma ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \neg\ \Box p) \in S \Longrightarrow \exists\, k.\ \{(k, \neg\ p),\ (i, \Diamond\ (\cdot k))\} \cup S \in C$›
⟨*proof*⟩

lemma ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow \{\ (i, \cdot i)\ \} \cup S \in C$›
⟨*proof*⟩

lemma ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \mathbf{A}\ p) \in S \Longrightarrow \{\ (k,\ p)\ \} \cup S \in C$›
⟨*proof*⟩

lemma ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \neg\ \mathbf{A}\ p) \in S \Longrightarrow \exists\, k.\ \{\ (k, \neg\ p)\ \} \cup S \in C$›
⟨*proof*⟩

lemma ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \forall\ \ p) \in S \Longrightarrow$ *softqdf* $q \Longrightarrow \{\ (i, \langle q \rangle_p\ p)\ \} \cup S \in C$›
⟨*proof*⟩

lemma ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \neg\ \forall\ \ p) \in S \Longrightarrow \exists P.\ \{\ (i, \neg\ \langle \cdot(\bigcirc P) \rangle_p\ p)\ \} \cup S \in C$›
⟨*proof*⟩

end

definition *equiv-nom* :: ‹*'x lbd set* ⇒ *'x tm* ⇒ *'x tm* ⇒ *bool*› **where**
‹*equiv-nom S i k* ≡ (*i*, •*k*) ∈ *S*›

definition *assign* :: ‹*'x tm* ⇒ *'x lbd set* ⇒ *'x tm*› (‹[-]_› [*0*, *100*] *100*) **where**
‹$[i]_S$ ≡ *wo-rel.minim* (|*UNIV*|) {*k*. *equiv-nom S i k*}›

definition *reach* :: ‹*'x lbd set* ⇒ *'x tm* ⇒ *'x tm set*› **where**
‹*reach S i* ≡ {$[k]_S$ |*k*. (*i*, ◇ (•*k*)) ∈ *S*}›

definition *val* :: ‹*'x lbd set* ⇒ *'x tm* ⇒ *'x tm set*› **where**
‹*val S P* ≡ {$[i]_S$ |*i*. (*i*, •*P*) ∈ *S*}›

lemma *range-val-ne*: ‹*range* (*val S*) ≠ {}›
⟨*proof*⟩

lemma *admissible-Pow*: ‹*admissible F* (*Pow* (𝒲 *F*))›
⟨*proof*⟩

definition *admits* :: ‹*'w frame* ⇒ *'w set set* ⇒ *'w set set* ⇒ *bool*› **where**
‹*admits F B PI* ≡ *PI* ≠ {} ∧ *PI* ⊆ *Pow* (𝒲 *F*) ∧ *B* ⊆ *PI* ∧ *admissible F PI*›

definition *adm-δ* :: ‹*'w frame* ⇒ *'w set set* ⇒ *'w set set*› **where**
‹*adm-δ M PI* ≡
{ 𝒲 *M* − *X* | *X*. *X* ∈ *PI* } ∪
{ *X* ∩ *Y* | *X Y*. *X* ∈ *PI* ∧ *Y* ∈ *PI* } ∪
{ {*w* ∈ 𝒲 *M*. ∀ *v* ∈ ℛ *M w*. *v* ∈ *X*} | *X*. *X* ∈ *PI* }›

abbreviation ‹*grow F B* ≡ λ*PI*. *B* ∪ *PI* ∪ *adm-δ F* (*B* ∪ *PI*)›

definition *admit* :: ‹*'w frame* ⇒ *'w set set* ⇒ *'w set set*› **where**
‹*admit F B* ≡ *lfp* (*grow F B*)›

lemma *mono-grow*: ‹*mono* (*grow F B*)›
⟨*proof*⟩

lemma *admissible-δ*: ‹*admissible F B* ⟷ *adm-δ F B* ⊆ *B*›
⟨*proof*⟩

lemma *admit-B*: ‹*B* ⊆ *admit F B*›
⟨*proof*⟩

lemma *admit-Pow*: ‹*B* ⊆ *Pow* (𝒲 *F*) ⟹ *admit F B* ⊆ *Pow* (𝒲 *F*)›
⟨*proof*⟩

lemma *admissible-grow*: ‹*admissible F B* ⟷ *grow F B B* = *B*›
⟨*proof*⟩

lemma *lfp-grow*: ‹*grow F B* (*lfp* (*grow F B*)) = *lfp* (*grow F B*)›

⟨*proof*⟩

lemma *admit-admissible*: ‹*admissible F* (*admit F B*)›
⟨*proof*⟩

lemma *admits-admit*: ‹$B \neq \{\} \Longrightarrow B \subseteq Pow\ (\mathcal{W}\ F) \Longrightarrow$ *admits F B* (*admit F B*)›
⟨*proof*⟩

lemma *admissible-admit*:
assumes ‹$B \neq \{\}$› ‹$B \subseteq Pow\ (\mathcal{W}\ F)$›
shows
‹*admit F B* $\neq \{\}$›
‹*admit F B* $\subseteq Pow\ (\mathcal{W}\ F)$›
‹*admissible F* (*admit F B*)›
⟨*proof*⟩

abbreviation *canonical-frame* :: ‹'*x lbd set* ⇒ ('*x tm*) *frame*› **where**
‹*canonical-frame S* ≡ (‖ $\mathcal{W} = \{[k]_S \mid k.\ True\}$, $\mathcal{R}$ = *reach S* ‖)›

abbreviation *canonical-gframe* :: ‹'*x lbd set* ⇒ ('*x tm*) *gframe*› **where**
‹*canonical-gframe S* ≡ *frame.extend* (*canonical-frame S*)
(‖ Π = *admit* (*canonical-frame S*) (*range* (*val S*)) ‖)›

definition *canonical* :: ‹'*x lbd set* ⇒ ('*x*, '*x tm*) *model*› **where**
‹*canonical S* ≡
gframe.extend (*canonical-gframe S*)
(‖ $\mathcal{N} = \lambda i.\ [\bigcirc i]_S$,
$\mathfrak{N} = \lambda i.\ [\#i]_S$,
$\mathcal{V}$ = *val S o* $\bigcirc$,
$\mathfrak{V}$ = *val S o* #
‖)›

lemma *wf-canonical-frame*: ‹*wf-frame* (*canonical-frame S*)›
⟨*proof*⟩

lemma *val-Pow*: ‹*range* (*val S*) ⊆ *Pow* ($\mathcal{W}$ (*canonical-frame S*))›
⟨*proof*⟩

lemma *wf-cannonical-gframe*: ‹*wf-gframe* (*canonical-gframe S*)›
⟨*proof*⟩

lemma *admits-val*: ‹*admits* (*canonical-frame S*) (*range* (*val S*)) *PI* ⟹ *val S P* ∈ *PI*›
⟨*proof*⟩

lemma *admit-val*: ‹*val S P* ∈ *admit* (*canonical-frame S*) (*range* (*val S*))›
⟨*proof*⟩

lemma *wf-canonical-env*: ‹*wf-env* (*canonical S*)›
⟨*proof*⟩

lemma *wf-gframe-canonical*: ‹*wf-gframe* (*gframe.truncate* (*canonical S*))›
⟨*proof*⟩

lemma *wf-canonical*: ‹*wf-model* (*canonical S*)›
⟨*proof*⟩

lemma *admissible-sqdfs*: ‹*admissible* (*canonical-frame S*) (*sqdfs* (*canonical S*))›
⟨*proof*⟩

lemma *sqdfs-Pow*: ‹*sqdfs* (*canonical S*) ⊆ *Pow* (*𝒲* (*canonical-frame S*))›
⟨*proof*⟩

lemma *val-sqdfs*: ‹*val S P* ∈ *sqdfs* (*canonical S*)›
⟨*proof*⟩

lemma *admits-canonical-sqdfs*: ‹*admits* (*canonical-frame S*) (*range* (*val S*)) (*sqdfs* (*canonical S*))›
⟨*proof*⟩

definition *canonical-ctx* :: ‹'*x lbd set* ⇒ '*x tm* ⇒ ('*x*, '*x tm*) *ctx*› (‹⟦-, -⟧›) **where**
‹⟦*S*, *i*⟧ ≡ (*canonical S*, [*i*]$_S$)›

lemma *sqdfs-canonical*: ‹*sqdfs* (*canonical S*) = Π (*canonical S*)›
⟨*proof*⟩

lemma *canonical-tm-eta* [*simp*]: ‹*case-tm* (λ*i*. [# *i*]$_S$) (λ*n*. [○ *n*]$_S$) *k* = [*k*]$_S$›
⟨*proof*⟩

corollary *canonical-tm-eta'* [*simp*]: ‹*case-tm* (*𝔑* (*canonical S*)) (*𝒩* (*canonical S*)) *k* = [*k*]$_S$›
⟨*proof*⟩

locale *MyHintikka* = *Hintikka map-lbd symbols-lbd* ‹λ-. *True*› *Kinds S*
for *S* :: ‹'*x lbd set*›
begin

lemmas
confl = *sat*$_H$[*of C.kind*] **and**
alpha = *sat*$_H$[*of A.kind*] **and**
beta = *sat*$_H$[*of B.kind*] **and**
gammaI = *sat*$_H$[*of GI.kind*] **and**
gammaP = *sat*$_H$[*of GP.kind*] **and**
δ = *sat*$_H$[*of D.kind*]

lemma *Nom-refl*: ‹(*i*, •*i*) ∈ *S*›
⟨*proof*⟩

lemma *Nom-sym*:
  assumes ‹$(i, \cdot k) \in S$›
  shows ‹$(k, \cdot i) \in S$›
  ⟨*proof*⟩

lemma *Nom-trans*:
  assumes ‹$(i, \cdot j) \in S$› ‹$(j, \cdot k) \in S$›
  shows ‹$(i, \cdot k) \in S$›
  ⟨*proof*⟩

lemma *equiv-nom-ne*: ‹$\{k.\ \textit{equiv-nom}\ S\ i\ k\} \neq \{\}$›
  ⟨*proof*⟩

lemma *equiv-nom-assign*: ‹*equiv-nom* $S\ i\ ([i]_S)$›
  ⟨*proof*⟩

lemma *equiv-nom-Nom*:
  assumes ‹*equiv-nom* $S\ i\ k$› ‹$(i, p) \in S$›
  shows ‹$(k, p) \in S$›
⟨*proof*⟩

lemma *assign-in-W*: ‹$[i]_S \in \mathcal{W}$ (*canonical* S)›
  ⟨*proof*⟩

theorem *model*: ‹$((i, p) \in S \longrightarrow [\![S, i]\!] \models p) \wedge ((i, \neg p) \in S \longrightarrow \neg\ [\![S, i]\!] \models p)$›
⟨*proof*⟩

end

theorem *model-existence*:
  fixes S :: ‹$'x$ *lbd set*›
  assumes ‹$P.\textit{prop}_E$ *Kinds* C›
    and ‹$S \in C$›
    and ‹$P.\textit{enough-new}\ S$›
    and ‹$(i, p) \in S$›
  shows ‹$[\![\textit{mk-mcs}\ C\ S, i]\!] \models p$›
⟨*proof*⟩

4.6 Natural Deduction

inductive *Calculus-Set* :: ‹$'x$ *lbd set* ⇒ $'x$ *lbd* ⇒ *bool*› (‹- ⊩ -› [50, 50] 50) **where**
  Assm [*dest*]: ‹$(i, p) \in A \Longrightarrow A \Vdash (i, p)$›
| *Ref* [*simp*]: ‹$A \Vdash (i, \cdot i)$›
| *Nom* [*dest*]: ‹$A \Vdash (i, \cdot k) \Longrightarrow A \Vdash (i, p) \Longrightarrow A \Vdash (k, p)$›
| *NotI* [*intro*]: ‹$\{(i, p)\} \cup A \Vdash (i, \bot) \Longrightarrow A \Vdash (i, \neg p)$›
| *NotE* [*elim*]: ‹$A \Vdash (i, \neg p) \Longrightarrow A \Vdash (i, p) \Longrightarrow A \Vdash (k, q)$›
| *AndI* [*intro*]: ‹$A \Vdash (i, p) \Longrightarrow A \Vdash (i, q) \Longrightarrow A \Vdash (i, p \wedge q)$›
| *AndD1* [*dest*]: ‹$A \Vdash (i, p \wedge q) \Longrightarrow A \Vdash (i, p)$›

| *AndD2* [*dest*]: ‹$A \Vdash (i,\ p \wedge q) \Longrightarrow A \Vdash (i,\ q)$›
| *SatI* [*intro*]: ‹$A \Vdash (i,\ p) \Longrightarrow A \Vdash (k,\ @i\ p)$›
| *SatE* [*dest*]: ‹$A \Vdash (i,\ @k\ p) \Longrightarrow A \Vdash (k,\ p)$›
| *BoxI* [*intro*]: ‹$\{(i,\ \Diamond\ (\cdot\ (\bigcirc k)))\} \cup A \Vdash (\bigcirc k,\ p) \Longrightarrow k \notin$ *symbols* $(\{(i,\ p)\} \cup A)$ $\Longrightarrow A \Vdash (i,\ \Box\ p)$›
| *BoxE* [*elim*]: ‹$A \Vdash (i,\ \Box\ p) \Longrightarrow A \Vdash (i,\ \Diamond\ (\cdot k)) \Longrightarrow A \Vdash (k,\ p)$›
| *GloI* [*intro*]: ‹$A \Vdash (\bigcirc k,\ p) \Longrightarrow k \notin$ *symbols* $(\{(i,\ p)\} \cup A) \Longrightarrow A \Vdash (i,\ \mathbf{A}\ p)$›
| *GloE* [*dest*]: ‹$A \Vdash (i,\ \mathbf{A}\ p) \Longrightarrow A \Vdash (k,\ p)$›
| *DwnI* [*intro*]: ‹$A \Vdash (i,\ \langle i \rangle_i\ p) \Longrightarrow A \Vdash (i,\ \downarrow p)$›
| *DwnE* [*dest*]: ‹$A \Vdash (i,\ \downarrow p) \Longrightarrow A \Vdash (i,\ \langle i \rangle_i\ p)$›
| *AllI* [*intro*]: ‹$A \Vdash (i,\ \langle \cdot(\bigcirc P) \rangle_p\ p) \Longrightarrow P \notin$ *symbols* $(\{(i,\ p)\} \cup A) \Longrightarrow A \Vdash (i,$ $\forall\ p)$›
| *AllE* [*dest*]: ‹$A \Vdash (i,\ \forall\ p) \Longrightarrow$ *softqdf* $q \Longrightarrow A \Vdash (i,\ \langle q \rangle_p\ p)$›
| *Clas*: ‹$\{(i,\ \neg\ p)\} \cup A \Vdash (i,\ p) \Longrightarrow A \Vdash (i,\ p)$›

4.6.1 Soundness

theorem *soundness*:
assumes ‹$A \Vdash (i,\ p)$› ‹$\forall (k,\ q) \in A.\ (Model\ W\ R\ PI\ N\ e\ V\ f,\ case\text{-}tm\ e\ N\ k) \models q$›
‹*wf-model* $(Model\ W\ R\ PI\ N\ e\ V\ f)$›
shows ‹$(Model\ W\ R\ PI\ N\ e\ V\ f,\ case\text{-}tm\ e\ N\ i) \models p$›
⟨*proof*⟩

corollary *soundness′*:
assumes ‹$\{\} \Vdash (\bigcirc i,\ p)$› ‹$i \notin$ *symbols-fm* p›
and ‹*wf-model* M› ‹$w \in \mathcal{W}\ M$›
shows ‹$(M,\ w) \models p$›
⟨*proof*⟩

lemma *no-bot*: ‹$\neg\ (M,\ w) \models \bot$›
⟨*proof*⟩

corollary ‹$\neg\ (\{\} \Vdash (\bigcirc i,\ \bot))$›
⟨*proof*⟩

4.6.2 Derived Rules

lemma *Assm-head* [*simp*]: ‹$\{(p,\ i)\} \cup A \Vdash (p,\ i)$›
⟨*proof*⟩

lemma *Boole*: ‹$\{(i,\ \neg\ p)\} \cup A \Vdash (i,\ \bot) \Longrightarrow A \Vdash (i,\ p)$›
⟨*proof*⟩

lemma *FlsE* [*dest*]: ‹$A \Vdash (i,\ \bot) \Longrightarrow A \Vdash (k,\ p)$›
⟨*proof*⟩

4.6.3 Derivational Consistency

lemma *calculus-confl*:

assumes ‹$ps \rightsquigarrow\!\!\!\!\!\!\times\ qs$› ‹$set\ ps \subseteq A$› ‹$q \in set\ qs$› ‹$q \in A$›
shows ‹$A \Vdash (i, \bot)$›
⟨*proof*⟩

lemma *calculus-alpha*:
assumes ‹$ps \rightsquigarrow_{\alpha}\ qs$› ‹$set\ ps \subseteq A$› ‹$set\ qs \cup A \Vdash (i, \bot)$›
shows ‹$A \Vdash (i, \bot)$›
⟨*proof*⟩

lemma *calculus-beta*:
assumes ‹$ps \rightsquigarrow_{\beta}\ qs$› ‹$set\ ps \subseteq A$› ‹$\forall\, q \in set\ qs.\ \{q\} \cup A \Vdash (i, \bot)$›
shows ‹$A \Vdash (i, \bot)$›
⟨*proof*⟩

lemma *calculus-gammaI*:
assumes ‹$ps \rightsquigarrow_{\gamma i}\ qs$› ‹$set\ ps \subseteq A$› ‹$set\ (qs\ k) \cup A \Vdash (i, \bot)$›
shows ‹$A \Vdash (i, \bot)$›
⟨*proof*⟩

lemma *calculus-gammaP*:
assumes ‹$ps \rightsquigarrow_{\gamma p}\ (F,\ qs)$› ‹$set\ ps \subseteq A$› ‹$k \in F\ A$› ‹$set\ (qs\ k) \cup A \Vdash (i, \bot)$›
shows ‹$A \Vdash (i, \bot)$›
⟨*proof*⟩

lemma *calculus-δ*:
assumes ‹$p \in A$› ‹$k \notin symbols\ A$› ‹$set\ (\delta\ p\ k) \cup A \Vdash (a, \bot)$›
shows ‹$A \Vdash (a, \bot)$›
⟨*proof*⟩

interpretation *DC*: *Derivational-Confl map-lbd symbols-lbd* ‹λ-. *True*› *confl-class* ‹$\lambda A.\ \neg\ A \Vdash (a, \bot)$›
⟨*proof*⟩

interpretation *DA*: *Derivational-Alpha map-lbd symbols-lbd* ‹λ-. *True*› *alpha-class* ‹$\lambda A.\ \neg\ A \Vdash (a, \bot)$›
⟨*proof*⟩

interpretation *DB*: *Derivational-Beta map-lbd symbols-lbd* ‹λ-. *True*› *beta-class* ‹$\lambda A.\ \neg\ A \Vdash (a, \bot)$›
⟨*proof*⟩

interpretation *DGI*: *Derivational-Gamma map-tm map-lbd symbols-lbd* ‹λ-. *True*› *GI.classify-UNIV* ‹$\lambda A.\ \neg\ A \Vdash (a, \bot)$›
⟨*proof*⟩

interpretation *DGP*: *Derivational-Gamma map-fm map-lbd symbols-lbd* ‹λ-. *True*› *gamma-class-fm* ‹$\lambda A.\ \neg\ A \Vdash (a, \bot)$›
⟨*proof*⟩

interpretation *DD*: *Derivational-Delta map-lbd symbols-lbd* ‹*λ-. True*› *δ* ‹*λA. ¬ A ⊩ (a, ⊥)*›
⟨*proof*⟩

interpretation *Derivational-Consistency map-lbd symbols-lbd* ‹*λ-. True*› *Kinds* ‹*λA. ¬ A ⊩ (a, ⊥)*›
⟨*proof*⟩

4.6.4 Strong Completeness

theorem *strong-completeness*:
fixes *p* :: ‹*'x fm*›
assumes *mod*: ‹*⋀(M :: ('x, 'x tm) model) w. wf-model M ⟹ w ∈ 𝒲 M ⟹ ∀(k, q) ∈ A. (M, case-tm (𝔑 M) (𝒩 M) k) ⊨ q ⟹ (M, w) ⊨ p*›
and ‹*P.enough-new A*›
shows ‹*A ⊩ (i, p)*›
⟨*proof*⟩

4.7 Natural Deduction with Lists

inductive *Calculus* :: ‹*'x lbd list ⇒ 'x lbd ⇒ bool*› (**infix** ‹*⊢*› *50*) **where**
Assm [*dest*]: ‹*(i, p) ∈ set A ⟹ A ⊢ (i, p)*›
| *Ref* [*simp*]: ‹*A ⊢ (i, ·i)*›
| *Nom* [*dest*]: ‹*A ⊢ (i, ·k) ⟹ A ⊢ (i, p) ⟹ A ⊢ (k, p)*›
| *NotI* [*intro*]: ‹*(i, p) # A ⊢ (i, ⊥) ⟹ A ⊢ (i, ¬ p)*›
| *NotE* [*elim*]: ‹*A ⊢ (i, ¬ p) ⟹ A ⊢ (i, p) ⟹ A ⊢ (k, q)*›
| *AndI* [*intro*]: ‹*A ⊢ (i, p) ⟹ A ⊢ (i, q) ⟹ A ⊢ (i, p ∧ q)*›
| *AndD1* [*dest*]: ‹*A ⊢ (i, p ∧ q) ⟹ A ⊢ (i, p)*›
| *AndD2* [*dest*]: ‹*A ⊢ (i, p ∧ q) ⟹ A ⊢ (i, q)*›
| *SatI* [*intro*]: ‹*A ⊢ (i, p) ⟹ A ⊢ (k, @i p)*›
| *SatE* [*dest*]: ‹*A ⊢ (i, @k p) ⟹ A ⊢ (k, p)*›
| *BoxI* [*intro*]: ‹*(i, ◇ (· (○k))) # A ⊢ (○k, p) ⟹ k ∉ symbols ({(i, p)} ∪ set A) ⟹ A ⊢ (i, □ p)*›
| *BoxE* [*elim*]: ‹*A ⊢ (i, □ p) ⟹ A ⊢ (i, ◇ (·k)) ⟹ A ⊢ (k, p)*›
| *GloI* [*intro*]: ‹*A ⊢ (○k, p) ⟹ k ∉ symbols ({(i, p)} ∪ set A) ⟹ A ⊢ (i, **A** p)*›
| *GloE* [*dest*]: ‹*A ⊢ (i, **A** p) ⟹ A ⊢ (k, p)*›
| *DwnI* [*intro*]: ‹*A ⊢ (i, ⟨i⟩ᵢ p) ⟹ A ⊢ (i, ↓ p)*›
| *DwnE* [*dest*]: ‹*A ⊢ (i, ↓ p) ⟹ A ⊢ (i, ⟨i⟩ᵢ p)*›
| *AllI* [*intro*]: ‹*A ⊢ (i, ⟨·(○P)⟩ₚ p) ⟹ P ∉ symbols ({(i, p)} ∪ set A) ⟹ A ⊢ (i, ∀ p)*›
| *AllE* [*dest*]: ‹*A ⊢ (i, ∀ p) ⟹ softqdf q ⟹ A ⊢ (i, ⟨q⟩ₚ p)*›
| *Clas*: ‹*(i, ¬ p) # A ⊢ (i, p) ⟹ A ⊢ (i, p)*›

definition *bounded* :: ‹*'a list ⇒ 'a set ⇒ ('a list ⇒ bool) ⇒ bool*› **where**
‹*bounded K A P ≡ set K ⊆ A ∧ (∀B. set K ⊆ set B ⟶ set B ⊆ A ⟶ P B)*›

lemma *bounded-one* [*elim*]:
assumes ‹*bounded K A P*› ‹*⋀A. P A ⟹ Q A*›

shows ‹*bounded K A Q*›
⟨*proof*⟩

lemma *bounded-two* [*elim*]:
assumes ‹*bounded K A P*› ‹*bounded K′ A Q*› ‹⋀*A*. *P A* ⟹ *Q A* ⟹ *R A*›
shows ‹*bounded* (*K* @ *K′*) *A R*›
⟨*proof*⟩

lemma *bounded-removeAll* [*dest*]:
assumes ‹*bounded K* ({*p*} ∪ *A*) *P*›
shows ‹*bounded* (*removeAll p K*) *A* (λ*B*. *P* (*p* # *B*))›
⟨*proof*⟩

lemma *bounded-params*:
assumes ‹*a* ∉ *P.params* ({*p*} ∪ *A*)› ‹*bounded K A P*›
shows ‹*bounded K A* (λ*B*. *a* ∉ *P.params* (*set* (*p* # *B*)))›
⟨*proof*⟩

lemma *finite-kernel*: ‹*A* ⊩ (*i*, *p*) ⟹ ∃ *K*. *bounded K A* (λ*B*. *B* ⊢ (*i*, *p*))›
⟨*proof*⟩

corollary *finite-assumptions*: ‹*A* ⊩ (*i*, *p*) ⟹ ∃ *B*. *set B* ⊆ *A* ∧ *B* ⊢ (*i*, *p*)›
⟨*proof*⟩

lemma *calculus-set*: ‹*A* ⊢ (*i*, *p*) ⟹ *set A* ⊩ (*i*, *p*)›
⟨*proof*⟩

corollary *soundness-list*:
assumes ‹*A* ⊢ (*i*, *p*)› ‹∀ (*k*, *q*) ∈ *set A*. (*M*, *case-tm* (𝔑 *M*) (𝒩 *M*) *k*) ⊨ *q*›
and ‹*wf-model M*›
shows ‹(*M*, *case-tm* (𝔑 *M*) (𝒩 *M*) *i*) ⊨ *p*›
⟨*proof*⟩

corollary *soundness-nil*:
‹[] ⊢ (○*i*, *p*) ⟹ *i* ∉ *symbols-fm p* ⟹ *wf-model M* ⟹ *w* ∈ 𝒲 *M* ⟹ (*M*, *w*) ⊨ *p*›
⟨*proof*⟩

corollary ‹¬ ([] ⊢ (*i*, ⊥))›
⟨*proof*⟩

corollary *strong-completeness-list*:
fixes *p* :: ‹′*x fm*›
assumes ‹⋀(*M* :: (′*x*, ′*x tm*) *model*) *w*. *wf-model M* ⟹ *w* ∈ 𝒲 *M* ⟹
∀ (*k*, *q*) ∈ *A*. (*M*, *case-tm* (𝔑 *M*) (𝒩 *M*) *k*) ⊨ *q* ⟹ (*M*, *w*) ⊨ *p*›
and ‹*P.enough-new A*›
shows ‹∃ *B*. *set B* ⊆ *A* ∧ *B* ⊢ (*i*, *p*)›
⟨*proof*⟩

theorem *main*:
fixes $p :: \langle 'x\ fm \rangle$
assumes $\langle i \notin$ *symbols-fm* $p \rangle$ $\langle |UNIV :: 'x\ lbd\ set| \leq o\ |UNIV :: 'x\ set| \rangle$
shows $\langle [] \vdash (\bigcirc i,\ p) \longleftrightarrow (\forall (M :: ('x,\ 'x\ tm)\ model).\ \forall w \in \mathcal{W}\ M.\ wf\text{-}model\ M \longrightarrow (M,\ w) \models p) \rangle$
⟨*proof*⟩

4.8 The Need for SQDFs

4.8.1 Finite Unions of Arithmetic Progressions

From Manuel Eberl's Furstenberg-Topology AFP entry

definition *arith-prog* :: *int* ⇒ *nat* ⇒ *int set* **where**
arith-prog a b = {*x*. [*x* = *a*] (*mod int b*)}

lemma *arith-prog-0-right* [*simp*]: *arith-prog a 0* = {*a*}
⟨*proof*⟩

lemma *arith-prog-Suc-0-right* [*simp*]: *arith-prog a* (*Suc 0*) = *UNIV*
⟨*proof*⟩

lemma *in-arith-progI* [*intro*]: [*x* = *a*] (*mod b*) ⟹ *x* ∈ *arith-prog a b*
⟨*proof*⟩

lemma *arith-prog-disjoint*:
assumes [*a* ≠ *a'*] (*mod int b*) **and** *b* > *0*
shows *arith-prog a b* ∩ *arith-prog a' b* = {}
⟨*proof*⟩

lemma *arith-prog-dvd-mono*: *b dvd b'* ⟹ *arith-prog a b'* ⊆ *arith-prog a b*
⟨*proof*⟩

lemma *bij-betw-arith-prog*:
assumes *b* > *0*
shows *bij-betw* (λ*n*. *a* + *int b* ∗ *n*) *UNIV* (*arith-prog a b*)
⟨*proof*⟩

lemma *arith-prog-altdef*: *arith-prog a b* = *range* (λ*n*. *a* + *int b* ∗ *n*)
⟨*proof*⟩

lemma *infinite-arith-prog*: *b* > *0* ⟹ *infinite* (*arith-prog a b*)
⟨*proof*⟩

lemma *arith-prog-complement*:
assumes ⟨*b* > *0*⟩
shows −*arith-prog a b* = (⋃*i*∈{*1*..<*b*}. *arith-prog* (*a* + *int i*) *b*)
⟨*proof*⟩

lemma *arith-prog-distinguish*:
assumes $x \neq y$
shows $\exists\, a\ c\ b.\ b > 0 \wedge x \in \textit{arith-prog}\ a\ b \wedge y \in \textit{arith-prog}\ c\ b \wedge \textit{arith-prog}\ a\ b \cap \textit{arith-prog}\ c\ b = \{\}$
⟨*proof*⟩

Unions of Arithmetic Progressions

lemma *arith-prog-offset-in*: ‹$k \in \textit{arith-prog}\ a\ b \Longrightarrow \textit{arith-prog}\ k\ b = \textit{arith-prog}\ a\ b$›
⟨*proof*⟩

lemma *arith-prog-mod*: ‹$\textit{arith-prog}\ (a\ \textit{mod int}\ b)\ b = \textit{arith-prog}\ a\ b$›
⟨*proof*⟩

lemma *mod-bounds*: ‹$b > 0 \Longrightarrow a\ \textit{mod int}\ b \geq 0 \wedge a\ \textit{mod int}\ b < b$›
⟨*proof*⟩

lemma *mod-range*: ‹$\{a\ \textit{mod int}\ b\ |a.\ b > 0\} \subseteq \{0..<\textit{int}\ b\}$›
⟨*proof*⟩

lemma *finite-mod-range*: ‹$\textit{finite}\ \{a\ \textit{mod int}\ b\ |a.\ b > 0\}$›
⟨*proof*⟩

definition *arith* :: ‹*nat set* ⇒ *int set* ⇒ *bool*› **where**
‹$\textit{arith}\ B\ U \equiv \forall\, a \in U.\ \exists\, b \in B.\ b > 0 \wedge \textit{arith-prog}\ a\ b \subseteq U$›

lemma *arith-mono*: ‹$\textit{arith}\ B\ U \Longrightarrow B \subseteq C \Longrightarrow \textit{arith}\ C\ U$›
⟨*proof*⟩

lemma *arith-empty-steps*: ‹$\textit{arith}\ B\ U \Longrightarrow B = \{\} \Longrightarrow U = \{\}$›
⟨*proof*⟩

lemma *arith-ne-steps*: ‹$\textit{arith}\ B\ U \Longrightarrow U \neq \{\} \Longrightarrow B \neq \{\}$›
⟨*proof*⟩

lemma *arith-decomp*:
assumes ‹$\textit{arith}\ B\ U$›
obtains *abs* **where**
‹$\textit{finite}\ B \Longrightarrow \textit{finite}\ \textit{abs}$›
‹$\bigwedge a\ b.\ (a, b) \in \textit{abs} \Longrightarrow b > 0 \wedge b \in B$›
‹$U = (\bigcup (a, b) \in \textit{abs}.\ \textit{arith-prog}\ a\ b)$›
⟨*proof*⟩

lemma *arith-UNIV*: ‹$\textit{arith}\ \{1\}\ \textit{UNIV}$›
⟨*proof*⟩

lemma *arith-empty*: *arith B* {}
⟨*proof*⟩

lemma *finite-case-prod-lcm*: ‹*finite B* ⟹ *finite C* ⟹ *finite* (*case-prod lcm* ‘ (*B* × *C*))›
⟨*proof*⟩

lemma *arith-inter*:
assumes *U*: ‹*arith B U*› **and** *V*: ‹*arith C V*›
shows ‹*arith* (*case-prod lcm* ‘ (*B* × *C*)) (*U* ∩ *V*)›
⟨*proof*⟩

lemma *arith-Inter*:
assumes ‹*finite X*› **and** *X*: ‹∀ *U* ∈ *X*. *arith B U*› ‹*X* ≠ {}›
shows ‹∃ *B'*. (*finite B* ⟶ *finite B'*) ∧ *arith B'* (⋂ *X*)›
⟨*proof*⟩

lemma *arith-union*:
assumes ‹*arith B U*› ‹*arith C V*›
shows ‹*arith* (*B* ∪ *C*) (*U* ∪ *V*)›
⟨*proof*⟩

lemma *arith-ne-infinite*:
assumes ‹*arith B U*› ‹*U* ≠ {}›
shows ‹*infinite U*›
⟨*proof*⟩

lemma *arith-prog-arith* [*intro*]:
assumes ‹*b* > *0*›
shows ‹*arith* {*b*} (*arith-prog a b*)›
⟨*proof*⟩

lemma *arith-prog-complement-arith* [*intro*]:
assumes ‹*b* > *0*›
shows ‹*arith* {*b*} (− *arith-prog a b*)›
⟨*proof*⟩

lemma *arith-complement-arith* [*intro*]:
assumes ‹*arith B U*› ‹*finite B*›
shows ‹∃ *B'*. *finite B'* ∧ *arith B'* (− *U*)›
⟨*proof*⟩

lemma *arith-distinguish*:
assumes ‹*x* ≠ *y*›
shows ‹∃ *B U V*. *finite B* ∧ *arith B U* ∧ *arith B V* ∧ *x* ∈ *U* ∧ *y* ∈ *V* ∧ *U* ∩ *V* = {}›
⟨*proof*⟩

Finite Unions of Arithmetic Progressions

definition *fin-arith* :: ‹*int set* ⇒ *bool*› **where**
‹*fin-arith U* ≡ ∃ *B*. *finite B* ∧ *arith B U*›

lemma *fin-arith-UNIV* [*intro*]: ‹*fin-arith UNIV*›
⟨*proof*⟩

lemma *fin-arith-empty* [*intro*]: ‹*fin-arith* {}›
⟨*proof*⟩

lemma *fin-arith-inter* [*intro*]: ‹*fin-arith U* ⟹ *fin-arith V* ⟹ *fin-arith* (*U* ∩ *V*)›
⟨*proof*⟩

lemma *fin-arith-union* [*intro*]: ‹*fin-arith U* ⟹ *fin-arith V* ⟹ *fin-arith* (*U* ∪ *V*)›
⟨*proof*⟩

lemma *fin-arith-compl* [*intro*]: ‹*fin-arith U* ⟹ *fin-arith* (− *U*)›
⟨*proof*⟩

lemma *fin-arith-distinguish*:
assumes ‹*x* ≠ *y*›
shows ‹∃ *U V*. *fin-arith U* ∧ *fin-arith V* ∧ *x* ∈ *U* ∧ *y* ∈ *V* ∧ *U* ∩ *V* = {}›
⟨*proof*⟩

Singletons

lemma *arith-prog-singleton*: ‹⋂{*arith-prog a b* |*a b*. *b* > *0* ∧ *x* ∈ *arith-prog a b*}
= {*x*}›
⟨*proof*⟩

lemma *fin-arith-Inter-singleton*: ‹⋂{*U* |*U*. *fin-arith U* ∧ *x* ∈ *U*} = {*x*}›
⟨*proof*⟩

lemma *singleton-not-finarith*: ‹¬ *fin-arith* {*x*}›
⟨*proof*⟩

4.8.2 Counterexample

definition *Pss* :: ‹*int set set*› **where**
‹*Pss* ≡ {*U*. *fin-arith U*}›

lemma *Pss-empty*: ‹{} ∈ *Pss*›
⟨*proof*⟩

lemma *Pss-UNIV*: ‹*UNIV* ∈ *Pss*›
⟨*proof*⟩

lemma *Pss-union*: ‹*X* ∈ *Pss* ⟹ *Y* ∈ *Pss* ⟹ *X* ∪ *Y* ∈ *Pss*›
⟨*proof*⟩

lemma *Pss-inter*: ‹$X \in Pss \Longrightarrow Y \in Pss \Longrightarrow X \cap Y \in Pss$›
⟨*proof*⟩

lemma *Pss-compl*: ‹$X \in Pss \Longrightarrow - X \in Pss$›
⟨*proof*⟩

definition *my-gframe* :: ‹*int gframe*› **where**
‹*my-gframe* ≡ (| $\mathcal{W} = UNIV$, $\mathcal{R} = \lambda x.\ UNIV$, $\Pi = Pss$ |)›

lemma *wf-frame-mygframe*: ‹*wf-frame* (*frame.truncate my-gframe*)›
⟨*proof*⟩

lemma *admissible-mygframe*: ‹*admissible* (*frame.truncate my-gframe*) (Π *my-gframe*)›
⟨*proof*⟩

lemma *wf-mygframe*: ‹*wf-gframe my-gframe*›
⟨*proof*⟩

definition *my-model* :: ‹(*int*, *int*) *model*› **where**
‹*my-model* ≡ *gframe.extend my-gframe* (|$\mathcal{N} = \lambda i.\ i$, $\mathfrak{N} = \lambda i.\ int\ i$, $\mathcal{V} = \lambda n.\ \{\}$, $\mathfrak{V} = \lambda n.\ \{\}$|)›

lemma *wf-mymodel*: ‹*wf-model my-model*›
⟨*proof*⟩

abbreviation *GloE* :: ‹'*x fm* ⇒ '*x fm*› (‹**E**›) **where**
‹**E** *p* ≡ ¬ (**A** (¬ *p*))›

Nowhere-or-twice says that if formula p holds somewhere, then it holds in at least two distinct worlds.

(We ignore de Bruijn complications and only instantiate with closed formulas.)

abbreviation ‹*nowhere-or-twice p* ≡
(◇ *p*) ⟶
(◇ (↓ (◇ (↓ (
(@ (#*1*) *p*) ∧
(@ (#*0*) *p*) ∧
¬ (@ (#*0*) (•(#*1*)))))))))›

Finite unions of arithmetic progressions are either empty or infinite.

lemma *fin-arith-nowhere-or-twice*:
assumes ‹*fin-arith U*›
shows ‹$U = \{\} \vee (\exists x\ y.\ x \in U \wedge y \in U \wedge x \neq y)$›
⟨*proof*⟩

So nowhere-or-twice holds for all admissible propositions.

lemma *nowhere-or-twice-admissible*: ‹(*my-model*, *x*) ⊨ ∀ (*nowhere-or-twice* (•(#*0*)))›

⟨*proof*⟩

However, propositional quantification lets us form a singleton.

abbreviation ‹*singleton x* ≡ ∀ (@(◯*x*) (•(#*0*)) ⟶ •(#*0*))›

lemma *singleton*: ‹((*my-model*, *x*) ⊨ *singleton y*) ⟷ *x* = *y*›
⟨*proof*⟩

lemma *fin-arith-distinguish'*:
‹∀ *P*. *fin-arith P* ⟶ *y* ∈ *P* ⟶ *v* ∈ *P* ⟹ *v* ≠ *w* ⟹ ∃ *P*. *y* ∈ *P* ∧ *fin-arith P* ∧ *w* ∉ *P*›
⟨*proof*⟩

The singleton does not hold nowhere-or-twice.

lemma *not-nowhere-or-twice-singleton*: ‹¬ (*my-model*, *x*) ⊨ *nowhere-or-twice* (*singleton y*)›
⟨*proof*⟩

So we cannot always eliminate a quantifier with a non-quantifier-free formula.

theorem *counter*:
shows ‹¬ (*my-model*, *x*) ⊨ ∀ (*nowhere-or-twice* (•(#*0*))) ⟶ *nowhere-or-twice* (*singleton y*)›
⟨*proof*⟩

end