

Abstract Consistency Properties

Asta Halkjær Boserup, Anders Schlichtkrull

May 8, 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

1	Abstract Consistency Properties	5
1.1	Utility	5
1.2	Finite Character	8
1.3	Consistency Properties	8
1.3.1	Consistency Kinds	11
1.4	Hintikka Sets	21
1.5	Derivational Consistency	22
1.6	Weak Derivational Consistency	23
1.7	Conflicts	24
1.8	Alpha	27
1.9	Beta	30
1.10	Gamma	34
1.11	Delta	38
1.12	Modal	42
2	Example: First-Order Logic with Restricted Instantiation	47
2.1	Syntax	47
2.2	Semantics	47
2.3	Operations	48
2.3.1	Lemmas	49
2.4	Terms	50
2.4.1	Lemmas	50
2.5	Guard	51
2.6	Model Existence	51
2.7	Compactness	55
2.8	Natural Deduction	60
2.8.1	Soundness	60
2.8.2	Derivational Consistency	61
2.8.3	Strong Completeness	62
2.8.4	Natural Deduction with Lists	63
2.9	Tableau	65
2.9.1	Soundness	66
2.9.2	Derivational Consistency	66

2.9.3	Strong Completeness	68
3	Example: Second-Order Logic	70
3.1	Syntax	70
3.2	Semantics	71
3.3	Operations	71
3.3.1	Shift	71
3.3.2	Parameters	72
3.3.3	Instantiation	73
3.3.4	Size	74
3.4	Model Existence	75
3.5	Propositional Semantics	82
3.6	Calculus	82
3.7	Soundness	83
3.8	Derived Rules	84
3.9	Derivational Consistency	87
3.10	Natural Deduction	90
3.10.1	Soundness	91
3.10.2	Derivational Consistency	92
3.10.3	Strong Completeness	94
4	Example: Prior's Ideal Logic	96
4.1	Syntax	96
4.2	Semantics	97
4.3	Operations	99
4.3.1	Lemmas	102
4.3.2	softqdf	104
4.3.3	Add env	104
4.3.4	Sizes	105
4.4	Propositional Quantification	105
4.5	Model Existence	110
4.6	Natural Deduction	122
4.6.1	Soundness	123
4.6.2	Derived Rules	126
4.6.3	Derivational Consistency	126
4.6.4	Strong Completeness	130
4.7	Natural Deduction with Lists	131
4.8	The Need for SQDFs	134
4.8.1	Finite Unions of Arithmetic Progressions	134
4.8.2	Counterexample	142

Chapter 1

Abstract Consistency Properties

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

1.1 Utility

lemma *Set-Diff-Un*: $\langle X - (Y \cup Z) = X - Y - Z \rangle$
by *blast*

lemma *infinite-diff-finite*: $\langle \text{finite } A \implies \text{infinite } (- B) \implies \text{infinite } (- (A \cup B)) \rangle$
by (*metis Compl-Diff-eq double-complement finite-Diff2 sup-commute*)

lemma *infinite-Diff-fin-Un*: $\langle \text{infinite } (X - Y) \implies \text{finite } Z \implies \text{infinite } (X - (Z \cup Y)) \rangle$
by (*simp add: Set-Diff-Un Un-commute*)

lemma *infinite-Diff-subset*: $\langle \text{infinite } (X - A) \implies B \subseteq A \implies \text{infinite } (X - B) \rangle$
by (*meson Diff-cancel Diff-eq-empty-iff Diff-mono infinite-super*)

lemma *finite-bound*:
fixes $X :: \langle 'a :: \text{size} \rangle \text{ set} \rangle$
assumes $\langle \text{finite } X \rangle \langle X \neq \{\} \rangle$
shows $\langle \exists x \in X. \forall y \in X. \text{size } y \leq \text{size } x \rangle$
using *assms* **by** (*induct X rule: finite-induct*) *force+*

lemma *infinite-UNIV-size*:
fixes $f :: \langle 'a :: \text{size} \rangle \Rightarrow 'a \rangle$
assumes $\langle \bigwedge x. \text{size } x < \text{size } (f x) \rangle$
shows $\langle \text{infinite } (\text{UNIV} :: 'a \text{ set}) \rangle$
proof
assume $\langle \text{finite } (\text{UNIV} :: 'a \text{ set}) \rangle$

then obtain $x :: 'a$ **where** $\langle \forall y :: 'a. \text{size } y \leq \text{size } x \rangle$
using *finite-bound by fastforce*
moreover have $\langle \text{size } x < \text{size } (f\ x) \rangle$
using *assms .*
ultimately show *False*
using *leD by blast*
qed

lemma *infinite-left*: $\langle \text{finite } C \implies \text{infinite } A \implies |A| \leq o \mid - B \implies |A| \leq o \mid - (C \cup B) \rangle$
by (*metis (no-types, opaque-lifting) Compl-Diff-eq card-of-infinite-diff-finite card-of-ordLeq-finite double-complement ordIso-iff-ordLeq ordLeq-transitive sup-commute*)

lemma *card-of-infinite-smaller-Union*:
assumes $\langle \forall x. |f\ x| < o \mid X \rangle \langle \text{infinite } X \rangle$
shows $\langle |\bigcup x \in X. f\ x| \leq o \mid X \rangle$
using *assms by (metis (full-types) Field-card-of card-of-UNION-ordLeq-infinite card-of-well-order-on ordLeq-iff-ordLess-or-ordIso ordLess-or-ordLeq)*

context *wo-rel*
begin

lemma *underS-bound*: $\langle a \in \text{underS } c \implies b \in \text{underS } c \implies a \in \text{under } b \vee b \in \text{under } a \rangle$
by (*meson BNF-Least-Fixpoint.underS-Field REFL Refl-under-in in-mono under-ofilter ofilter-linord*)

lemma *finite-underS-bound*:
assumes $\langle \text{finite } X \rangle \langle X \subseteq \text{underS } a \rangle \langle X \neq \{\} \rangle$
shows $\langle \exists a \in X. \forall b \in X. b \in \text{under } a \rangle$
using *assms*
proof (*induct X rule: finite-induct*)
case (*insert x F*)
then show *?case*
proof (*cases $\langle F = \{\} \rangle$*)
case *True*
then show *?thesis*
using *insert underS-bound by fast*
next
case *False*
then show *?thesis*
using *insert underS-bound by (metis TRANS insert-absorb insert-iff insert-subset under-trans)*
qed
qed *simp*

lemma *finite-bound-under*:
assumes $\langle \text{finite } p \rangle \langle p \subseteq (\bigcup a \in \text{Field } r. f\ a) \rangle$
shows $\langle \exists b. p \subseteq (\bigcup a \in \text{under } b. f\ a) \rangle$

using *assms*
proof (*induct rule: finite-induct*)
case (*insert x p*)
then obtain *b* **where** $\langle p \subseteq (\bigcup a \in \text{under } b. f a) \rangle$
by *fast*
moreover obtain *b'* **where** $\langle x \in f b' \rangle \langle b' \in \text{Field } r \rangle$
using *insert(4)* **by** *blast*
then have $\langle x \in (\bigcup a \in \text{under } b'. f a) \rangle$
using *REFL Refl-under-in* **by** *fast*
ultimately have $\langle \{x\} \cup p \subseteq (\bigcup a \in \text{under } b. f a) \cup (\bigcup a \in \text{under } b'. f a) \rangle$
by *fast*
then show *?case*
by (*metis SUP-union Un-commute insert-is-Un sup.absorb-iff2 ofilter-linord under-ofilter*)
qed *simp*

lemma *underS-trans*: $\langle a \in \text{underS } b \implies b \in \text{underS } c \implies a \in \text{underS } c \rangle$
by (*meson ANTISYM TRANS underS-underS-trans*)

definition *is-chain* :: $\langle 'a \Rightarrow 'a \text{ set} \rangle \Rightarrow \text{bool}$ **where**
 $\langle \text{is-chain } f \equiv \forall a \in \text{Field } r. \forall b \in \text{Field } r. b \in \text{under } a \longrightarrow f b \subseteq f a \rangle$

lemma *is-chainD*: $\langle \text{is-chain } f \implies b \in \text{under } a \implies x \in f b \implies x \in f a \rangle$
unfolding *is-chain-def* **by** (*metis equals0D subsetD under-Field under-empty*)

lemma *chain-index*:

assumes *ch*: $\langle \text{is-chain } f \rangle$ **and** *fin*: $\langle \text{finite } F \rangle$ **and** *ne*: $\langle \text{Field } r \neq \{\} \rangle$
shows $\langle F \subseteq (\bigcup a \in \text{Field } r. f a) \implies \exists a \in \text{Field } r. F \subseteq f a \rangle$
using *fin*

proof (*induct rule: finite-induct*)

case *empty*

then show *?case*

using *ne* **by** *blast*

next

case (*insert x F*)

then have $\langle \exists a \in \text{Field } r. F \subseteq f a \rangle \langle \exists b \in \text{Field } r. x \in f b \rangle \langle F \subseteq (\bigcup x \in \text{Field } r. f x) \rangle$

using *ch* **by** *simp-all*

then obtain *a* **and** *b* **where** *f*: $\langle F \subseteq f a \rangle \langle x \in f b \rangle$ **and** *nm*: $\langle a \in \text{Field } r \rangle \langle b \in \text{Field } r \rangle$

by *blast*

have $\langle b \in \text{under } (\text{max2 } a b) \rangle \langle a \in \text{under } (\text{max2 } a b) \rangle$

using *nm* **by** (*meson REFL Refl-under-in TRANS max2-greater-among subset-iff under-incl-iff*)⁺

have $\langle x \in f (\text{max2 } a b) \rangle$

using *is-chainD* $\langle b \in \text{under } (\text{max2 } a b) \rangle f(2)$ **by** *blast*

moreover have $\langle F \subseteq f (\text{max2 } a b) \rangle$

using *is-chainD* $\langle a \in \text{under } (\text{max2 } a b) \rangle f(1)$ **by** *blast*

ultimately show *?case*

using *nm unfolding max2-def* by *auto*
 qed
 end

1.2 Finite Character

definition *close* :: $\langle 'a \text{ set set} \Rightarrow 'a \text{ set set} \rangle$ **where**
 $\langle \text{close } C \equiv \{S. (\exists S' \in C. S \subseteq S')\} \rangle$

definition *subset-closed* :: $\langle 'a \text{ set set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{subset-closed } C \equiv \forall S' \in C. \forall S \subseteq S'. S \in C \rangle$

lemma *subset-in-close*: $\langle S \subseteq S' \Longrightarrow x \cup S' \in C \Longrightarrow x \cup S \in \text{close } C \rangle$
unfolding *close-def* by *blast*

lemma *close-closed*: $\langle \text{subset-closed } (\text{close } C) \rangle$
unfolding *close-def subset-closed-def* by *blast*

lemma *close-subset*: $\langle C \subseteq \text{close } C \rangle$
unfolding *close-def* by *blast*

definition *finite-char* :: $\langle 'a \text{ set set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{finite-char } C \equiv \forall S. S \in C \longleftrightarrow (\forall S' \subseteq S. \text{finite } S' \longrightarrow S' \in C) \rangle$

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

lemma *finite-char*: $\langle \text{finite-char } (\text{mk-finite-char } C) \rangle$
unfolding *finite-char-def mk-finite-char-def* by *blast*

lemma *finite-char-closed*: $\langle \text{finite-char } C \Longrightarrow \text{subset-closed } C \rangle$
unfolding *finite-char-def subset-closed-def* by (*meson order-trans*)

lemma *finite-char-subset*: $\langle \text{subset-closed } C \Longrightarrow C \subseteq \text{mk-finite-char } C \rangle$
unfolding *mk-finite-char-def subset-closed-def* by *blast*

lemma (*in wo-rel*) *chain-union-closed*:
assumes $\langle \text{finite-char } C \rangle \langle \text{is-chain } f \rangle \langle \forall a \in \text{Field } r. f a \in C \rangle \langle \text{Field } r \neq \{\} \rangle$
shows $\langle (\bigcup a \in \text{Field } r. f a) \in C \rangle$
using *assms chain-index* **unfolding** *finite-char-def* by *metis*

definition *maximal* :: $\langle 'a \text{ set set} \Rightarrow 'a \text{ set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{maximal } C S \longleftrightarrow (\forall S' \in C. S \subseteq S' \longrightarrow S = S') \rangle$

1.3 Consistency Properties

locale *Params* =

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

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

lemma *is-subst-id* [*intro*]: $\langle \text{is-subst id} \rangle$
unfolding *is-subst-def* **by** *simp*

definition *mk-alt-consistency* :: $\langle 'fm \text{ set set} \Rightarrow 'fm \text{ set set} \rangle$ **where**
 $\langle \text{mk-alt-consistency } C \equiv \{S. (\exists f. \text{is-subst } f \wedge \text{map-fm } f ' S \in C)\} \rangle$

lemma *mk-alt-consistency-subset*: $\langle C \subseteq \text{mk-alt-consistency } C \rangle$
unfolding *mk-alt-consistency-def*

proof

fix *x*
assume $\langle x \in C \rangle$
then have $\langle \text{map-fm id } ' x \in C \rangle$
using *map-fm-id* **by** *simp*
then have $\langle \exists f. \text{is-subst } f \wedge \text{map-fm } f ' x \in C \rangle$
by *blast*
then show $\langle x \in \{S. \exists f. \text{is-subst } f \wedge \text{map-fm } f ' S \in C\} \rangle$
by *blast*

qed

lemma *mk-alt-consistency-closed*:

assumes $\langle \text{subset-closed } C \rangle$
shows $\langle \text{subset-closed} (\text{mk-alt-consistency } C) \rangle$
unfolding *subset-closed-def* *mk-alt-consistency-def*

proof *safe*

fix *S S' f*
assume $\langle \text{is-subst } f \rangle \langle \text{map-fm } f ' S' \in C \rangle \langle S \subseteq S' \rangle$
moreover have $\langle \text{map-fm } f ' S \subseteq \text{map-fm } f ' S' \rangle$
using $\langle S \subseteq S' \rangle$ **by** *blast*
moreover have $\langle \forall S' \in C. \forall S \subseteq S'. S \in C \rangle$
using $\langle \text{subset-closed } C \rangle$ **unfolding** *subset-closed-def* **by** *blast*
ultimately show $\langle \exists f. \text{is-subst } f \wedge \text{map-fm } f ' S \in C \rangle$
by *blast*

qed

abbreviation *params* :: $\langle 'fm \text{ set} \Rightarrow 'x \text{ set} \rangle$ **where**
 $\langle \text{params } S \equiv \bigcup p \in S. \text{params-fm } p \rangle$

lemma *infinite-params*: $\langle \text{infinite } (U - \text{params } B) \implies \text{infinite } (U - \text{params } (\text{set } ps \cup B)) \rangle$

using *finite-params-fm* **by** (*metis List.finite-set UN-Un finite-UN-I infinite-Diff-fin-Un*)

lemma *infinite-params-left*:

assumes $\langle \text{infinite } A \rangle \langle |A| \leq o \mid U - \text{params } S \rangle$

shows $\langle |A| \leq o \mid U - \text{params } (\text{set } ps \cup S) \rangle$

proof –

have $\langle \text{infinite } (U - \text{params } S) \rangle$

using *assms card-of-ordLeq-infinite* **by** *blast*

then have $\langle |U - \text{params } S| = o \mid U - \text{params } (\text{set } ps \cup S) \rangle$

by (*simp add: Set-Diff-Un Un-commute card-of-infinite-diff-finite ordIso-symmetric*)

then show *?thesis*

using *assms(2) ordLeq-ordIso-trans* **by** *blast*

qed

definition *enough-new* :: $\langle 'fm \text{ set} \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{enough-new } S \equiv |UNIV :: 'fm \text{ set}| \leq o \mid \text{Collect is-param} - \text{params } S \rangle$

lemma *enough-new-countable*:

assumes $\langle \exists \text{ to-nat} :: 'fm \Rightarrow \text{nat. inj to-nat} \rangle \langle \text{infinite } (\text{Collect is-param} - \text{params } S) \rangle$

shows $\langle \text{enough-new } S \rangle$

unfolding *enough-new-def* **using** *assms*

by (*meson UNIV-I card-of-ordLeqI infinite-iff-card-of-nat ordLeq-transitive*)

lemma *enough-new-all-param*:

assumes $\langle |UNIV :: 'fm \text{ set}| \leq o \mid UNIV - \text{params } S \rangle \langle \bigwedge x. \text{is-param } x \rangle$

shows $\langle \text{enough-new } S \rangle$

unfolding *enough-new-def* **using** *assms* **by** (*simp add: Collect-cong*)

end

datatype $\langle 'x, 'fm \rangle$ *kind*

$= \text{Cond } \langle 'fm \text{ list} \Rightarrow ('fm \text{ set set} \Rightarrow 'fm \text{ set} \Rightarrow \text{bool}) \Rightarrow \text{bool} \rangle \langle 'fm \text{ set} \Rightarrow \text{bool} \rangle$

$| \text{Wits } \langle 'fm \Rightarrow 'x \Rightarrow 'fm \text{ list} \rangle$

inductive (**in** *Params*) *sat_E* :: $\langle ('x, 'fm) \text{ kind} \Rightarrow 'fm \text{ set set} \Rightarrow \text{bool} \rangle$ **where**

sat_E-Cond [*intro!*]: $\langle (\bigwedge S \text{ ps } Q. S \in C \implies \text{set } ps \subseteq S \implies P \text{ ps } Q \implies Q \text{ } C \text{ } S) \implies \text{sat}_E (\text{Cond } P \text{ } H) \text{ } C \rangle$

$| \text{sat}_E\text{-Wits}$ [*intro!*]: $\langle (\bigwedge S \text{ p. } S \in C \implies p \in S \implies (\exists x. \text{is-param } x \wedge \text{set } (W \text{ } p \text{ } x) \cup S \in C) \implies \text{sat}_E (\text{Wits } W) \text{ } C \rangle$

inductive-cases (**in** *Params*) *sat_E-CondE*[*elim!*]: $\langle \text{sat}_E (\text{Cond } P \text{ } H) \text{ } C \rangle$

inductive-cases (**in** *Params*) *sat_E-WitsE*[*elim!*]: $\langle \text{sat}_E (\text{Wits } W) \text{ } C \rangle$

inductive (**in** *Params*) *sat_A* :: $\langle ('x, 'fm) \text{ kind} \Rightarrow 'fm \text{ set set} \Rightarrow \text{bool} \rangle$ **where**

sat_A-Cond [*intro!*]: $\langle (\bigwedge S \text{ ps } Q. S \in C \implies \text{set } ps \subseteq S \implies P \text{ ps } Q \implies Q \text{ } C \text{ } S) \implies \text{sat}_A (\text{Cond } P \text{ } H) \text{ } C \rangle$

| *sat_A-Wits* [*intro!*]: $\langle (\bigwedge S p x. S \in C \implies p \in S \implies x \notin \text{params } S \implies \text{is-param } x \implies \text{set } (W p x) \cup S \in C) \implies \text{sat}_A (\text{Wits } W) C \rangle$

inductive-cases (**in** *Params*) *sat_A-CondE*[*elim!*]: $\langle \text{sat}_A (\text{Cond } P H) C \rangle$

inductive-cases (**in** *Params*) *sat_A-WitsE*[*elim!*]: $\langle \text{sat}_A (\text{Wits } W) C \rangle$

definition (**in** *Params*) *prop_E* :: $\langle ('x, 'fm) \text{ kind list} \implies 'fm \text{ set set} \implies \text{bool} \rangle$ **where**
 $\langle \text{prop}_E Ks C \equiv \forall K \in \text{set } Ks. \text{sat}_E K C \rangle$

definition (**in** *Params*) *prop_A* :: $\langle ('x, 'fm) \text{ kind list} \implies 'fm \text{ set set} \implies \text{bool} \rangle$ **where**
 $\langle \text{prop}_A Ks C \equiv \forall K \in \text{set } Ks. \text{sat}_A K C \rangle$

inductive (**in** *Params*) *sat_H* :: $\langle ('x, 'fm) \text{ kind} \implies 'fm \text{ set} \implies \text{bool} \rangle$ **where**

sat_H-Cond [*intro!*]: $\langle H S \implies \text{sat}_H (\text{Cond } P H) S \rangle$

| *sat_H-Wits* [*intro!*]: $\langle (\bigwedge p. p \in S \implies (\exists x. \text{is-param } x \wedge \text{set } (W p x) \subseteq S)) \implies \text{sat}_H (\text{Wits } W) S \rangle$

inductive-cases (**in** *Params*) *sat_H-CondE*[*elim!*]: $\langle \text{sat}_H (\text{Cond } P H) C \rangle$

inductive-cases (**in** *Params*) *sat_H-WitsE*[*elim!*]: $\langle \text{sat}_H (\text{Wits } W) C \rangle$

definition (**in** *Params*) *prop_H* :: $\langle ('x, 'fm) \text{ kind list} \implies 'fm \text{ set} \implies \text{bool} \rangle$ **where**
 $\langle \text{prop}_H Ks S \equiv \forall K \in \text{set } Ks. \text{sat}_H K S \rangle$

theorem (**in** *Params*) *sat_H-Wits*: $\langle \text{sat}_E (\text{Wits } W) C \implies S \in C \implies \text{maximal } C \implies \text{sat}_H (\text{Wits } W) S \rangle$

unfolding *maximal-def* **by** *fast*

1.3.1 Consistency Kinds

locale *Consistency-Kind* = *Params map-fm params-fm is-param*
for

map-fm :: $\langle ('x \Rightarrow 'x) \Rightarrow 'fm \Rightarrow 'fm \rangle$ **and**

params-fm :: $\langle 'fm \Rightarrow 'x \text{ set} \rangle$ **and**

is-param :: $\langle 'x \Rightarrow \text{bool} \rangle$ +

fixes *K* :: $\langle ('x, 'fm) \text{ kind} \rangle$

assumes *respects-close*: $\langle \bigwedge C. \text{sat}_E K C \implies \text{sat}_E K (\text{close } C) \rangle$

and *respects-alt*: $\langle \bigwedge C. \text{sat}_E K C \implies \text{subset-closed } C \implies \text{sat}_A K (\text{mk-alt-consistency } C) \rangle$

and *respects-fin*: $\langle \bigwedge C. \text{subset-closed } C \implies \text{sat}_A K C \implies \text{sat}_A K (\text{mk-finite-char } C) \rangle$

and *hintikka*: $\langle \bigwedge C S. \text{sat}_E K C \implies S \in C \implies \text{maximal } C S \implies \text{sat}_H K S \rangle$

locale *Consistency-Kinds* = *Params map-fm params-fm is-param*
for

map-fm :: $\langle ('x \Rightarrow 'x) \Rightarrow 'fm \Rightarrow 'fm \rangle$ **and**

params-fm :: $\langle 'fm \Rightarrow 'x \text{ set} \rangle$ **and**

is-param :: $\langle 'x \Rightarrow \text{bool} \rangle$ +

fixes *Ks* :: $\langle ('x, 'fm) \text{ kind list} \rangle$

assumes *all-kinds*: $\langle \bigwedge K. K \in \text{set } Ks \implies \text{Consistency-Kind } \text{map-fm } \text{params-fm} \rangle$

is-param K
begin

lemma *sat_E*: $\langle K \in \text{set } Ks \implies \text{prop}_E Ks C \implies \text{sat}_E K C \rangle$
unfolding *prop_E-def* **by** *blast*

lemma *prop_E-close*: $\langle \text{prop}_E Ks C \implies \text{prop}_E Ks (\text{close } C) \rangle$
unfolding *prop_E-def* **using** *all-kinds Consistency-Kind.respects-close* **by** *fast*

lemma *prop_E-alt*: $\langle \text{prop}_E Ks C \implies \text{subset-closed } C \implies \text{prop}_A Ks (\text{mk-alt-consistency } C) \rangle$
unfolding *prop_E-def prop_A-def* **using** *all-kinds Consistency-Kind.respects-alt* **by** *fast*

lemma *prop_E-fin*: $\langle \text{subset-closed } C \implies \text{prop}_A Ks C \implies \text{prop}_A Ks (\text{mk-finite-char } C) \rangle$
unfolding *prop_A-def* **using** *all-kinds Consistency-Kind.respects-fin* **by** *fast*

definition *mk-alt-fin* :: $\langle 'fm \text{ set } set \Rightarrow 'fm \text{ set } set \rangle$ **where**
 $\langle \text{mk-alt-fin } C \equiv \text{mk-finite-char } (\text{mk-alt-consistency } (\text{close } C)) \rangle$

lemma *mk-alt-fin-subset-closed*: $\langle \text{subset-closed } (\text{mk-alt-fin } C) \rangle$
unfolding *mk-alt-fin-def* **using** *finite-char finite-char-closed* **by** *blast*

lemma *mk-alt-fin-finite-char*: $\langle \text{finite-char } (\text{mk-alt-fin } C) \rangle$
unfolding *mk-alt-fin-def* **using** *finite-char* **by** *blast*

lemma *mk-alt-fin-in*: $\langle S \in C \implies S \in \text{mk-alt-fin } C \rangle$
unfolding *mk-alt-fin-def*
by (*meson close-closed close-subset finite-char-subset in-mono mk-alt-consistency-closed mk-alt-consistency-subset*)

theorem *prop_E*: $\langle \text{prop}_E Ks C \implies \text{prop}_A Ks (\text{mk-alt-fin } C) \rangle$
unfolding *mk-alt-fin-def*
by (*simp add: prop_E-alt prop_E-close prop_E-fin close-closed mk-alt-consistency-closed*)

end

fun (**in** *Params*) *witness-kinds* :: $\langle ('x, 'fm) \text{ kind list} \Rightarrow 'fm \Rightarrow 'fm \text{ set} \Rightarrow 'fm \text{ set} \rangle$
where
 $\langle \text{witness-kinds } [] p S = \{ \} \rangle$
 $| \langle \text{witness-kinds } (\text{Cond } - - \# Ks) p S = \text{witness-kinds } Ks p S \rangle$
 $| \langle \text{witness-kinds } (\text{Wits } W \# Ks) p S =$
(let
 $\text{rest} = \text{witness-kinds } Ks p S;$
 $a = \text{SOME } x. x \in \text{Collect } \text{is-param} - \text{params } (\text{rest} \cup \{p\} \cup S)$
 $\text{in set } (W p a) \cup \text{rest} \rangle$

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))⟩
using assms
proof (induct Ks p S rule: witness-kinds.induct)
  case (1 p S)
  then show ?case
    by (simp add: infinite-Diff-fin-Un)
next
  case (3 W Ks p S)
  then show ?case
    by (metis (no-types, lifting) infinite-params sup-assoc witness-kinds.simps(3))
qed simp-all

```

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⟩
using assms(2–)
proof (induct Ks p S rule: witness-kinds.induct)
  case (3 W' Ks p S)
  moreover have ⟨infinite (Collect is-param – params (witness-kinds Ks p S ∪
{p} ∪ S))⟩
    using inf 3 witness-kinds-new by blast
  then have ⟨∃ x. x ∈ Collect is-param – params (witness-kinds Ks p S ∪ {p} ∪
S)⟩
    by (metis equals0I finite.emptyI)
  then obtain x where x:
    ⟨(SOME x. x ∈ Collect is-param – params (witness-kinds Ks p S ∪ {p} ∪ S))
= x⟩
    ⟨is-param x⟩
    by (metis (mono-tags, lifting) DiffE mem-Collect-eq someI-ex)
  ultimately show ?case
    by (auto simp: Let-def)
qed simp-all

```

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⟩
  by (simp add: cinfinite-def inf-univ)

```

lemma *isLimOrd*: *isLimOrd*

using *Cinfinite-r card-order-infinite-isLimOrd cinfinite-def* **by** *blast*

lemma *aboveS-ne*: $\langle \text{aboveS } a \neq \{\} \rangle$
by (*simp add: isLimOrd isLimOrd-aboveS*)

lemma *params-left*: $\langle \text{enough-new } S \implies \text{enough-new } (\text{set } ps \cup S) \rangle$
unfolding *enough-new-def* **using** *infinite-params-left inf-univ* **by** *blast*

definition *witness* :: $\langle 'fm \Rightarrow 'fm \text{ set} \Rightarrow 'fm \text{ set} \rangle$ **where**
 $\langle \text{witness} \equiv \text{witness-kinds } Ks \rangle$

definition *extendS* :: $\langle 'fm \text{ set set} \Rightarrow 'fm \Rightarrow 'fm \text{ set} \Rightarrow 'fm \text{ set} \rangle$ **where**
 $\langle \text{extendS } C \ a \ \text{prev} \equiv \text{if } (\{a\} \cup \text{prev} \in C) \text{ then } (\text{witness } a \ \text{prev} \cup \{a\} \cup \text{prev}) \text{ else } \text{prev} \rangle$

definition *extendL* :: $\langle 'fm \text{ set set} \Rightarrow ('fm \Rightarrow 'fm \text{ set}) \Rightarrow 'fm \Rightarrow 'fm \text{ set} \rangle$ **where**
 $\langle \text{extendL } C \ \text{rec } a \equiv \bigcup b \in \text{underS } a. \ \text{rec } b \rangle$

definition *extend* :: $\langle 'fm \text{ set set} \Rightarrow 'fm \text{ set} \Rightarrow 'fm \Rightarrow 'fm \text{ set} \rangle$ **where**
 $\langle \text{extend } C \ S \ a \equiv \text{wrecZSL } S \ (\text{extendS } C) \ (\text{extendL } C) \ a \rangle$

lemma *adm-woL-extendL*: $\langle \text{adm-woL } (\text{extendL } C) \rangle$
unfolding *extendL-def adm-woL-def* **by** *blast*

definition *Extend* :: $\langle 'fm \text{ set set} \Rightarrow 'fm \text{ set} \Rightarrow 'fm \text{ set} \rangle$ **where**
 $\langle \text{Extend } C \ S \equiv \bigcup a. \ \text{extend } C \ S \ a \rangle$

lemma *finite-witness-kinds*: $\langle \text{finite } (\text{witness-kinds } Qs \ p \ S) \rangle$
unfolding *witness-def* **by** (*induct Qs p S rule: witness-kinds.induct*) (*simp-all add: Let-def*)

lemma *finite-witness*: $\langle \text{finite } (\text{witness } p \ S) \rangle$
unfolding *witness-def* **using** *finite-witness-kinds* .

lemma *finite-witness-kinds-params*: $\langle \text{finite } (\text{params } (\text{witness-kinds } Qs \ p \ S)) \rangle$
using *finite-witness-kinds* **by** *simp*

lemma *finite-witness-params*: $\langle \text{finite } (\text{params } (\text{witness } p \ S)) \rangle$
using *finite-witness* **by** *simp*

lemma *extend-zero* [*simp*]: $\langle \text{extend } C \ S \ \text{zero} = S \rangle$
unfolding *extend-def wrecZSL-zero[OF adm-woL-extendL]* ..

lemma *extend-succ* [*simp*]: $\langle \text{extend } C \ S \ (\text{succ } a) =$
 $(\text{if } \{a\} \cup \text{extend } C \ S \ a \in C \text{ then } \text{witness } a \ (\text{extend } C \ S \ a) \cup \{a\} \cup \text{extend } C \ S$
 $a \text{ else } \text{extend } C \ S \ a) \rangle$
unfolding *extend-def extendS-def wrecZSL-succ[OF adm-woL-extendL aboveS-ne]*
..

lemma *extend-isLim* [*simp*]:
assumes $\langle \text{isLim } a \rangle \langle a \neq \text{zero} \rangle$
shows $\langle \text{extend } C \ S \ a = (\bigcup b \in \text{underS } a. \text{extend } C \ S \ b) \rangle$
unfolding *extend-def extendL-def wrecZSL-isLim*[*OF adm-woL-extendL assms*]
..

lemma *extend-subset*: $\langle S \subseteq \text{extend } C \ S \ a \rangle$
proof (*induct a rule: well-order-inductZSL*)
case (*Lim i*)
then show *?case*
using *zero-smallest* **by** (*metis Field-card-of SUP-upper2 UNIV-I extend-isLim underS-I*)
qed *auto*

lemma *Extend-subset*: $\langle S \subseteq \text{Extend } C \ S \rangle$
unfolding *Extend-def* **using** *extend-subset* **by** *fast*

lemma *extend-underS*: $\langle b \in \text{underS } a \implies \text{extend } C \ S \ b \subseteq \text{extend } C \ S \ a \rangle$
proof (*induct a rule: well-order-inductZSL*)
case *Zero*
then show *?case*
using *underS-zero* **by** *blast*
next
case (*Suc i*)
moreover from this have $\langle b = i \vee b \in \text{underS } i \rangle$
using *less-succ* **by** (*metis underS-E underS-I*)
ultimately show *?case*
by *auto*
next
case (*Lim i*)
then show *?case*
by *auto*
qed

lemma *extend-under*: $\langle b \in \text{under } a \implies \text{extend } C \ S \ b \subseteq \text{extend } C \ S \ a \rangle$
using *extend-underS supr-greater supr-under*
by (*metis emptyE in-Above-under set-eq-subset underS-I under-empty*)

lemma *params-origin*:
assumes $\langle x \in \text{params } (\text{extend } C \ S \ a) \rangle$
shows $\langle x \in \text{params } S \vee (\exists b \in \text{underS } a. x \in \text{params } (\text{witness } b \ (\text{extend } C \ S \ b) \cup \{b\})) \rangle$
using *assms*
proof (*induct a rule: well-order-inductZSL*)
case *Zero*
then show *?case*
by *simp*
next
case (*Suc i*)

```

then consider
  (here)  $\langle x \in \text{params} (\{i\} \cup \text{witness } i (\text{extend } C S i)) \rangle \mid$ 
  (there)  $\langle x \in \text{params} (\text{extend } C S i) \rangle$ 
  using  $\text{Suc}(\beta)$  by (fastforce split: if-splits)
then show ?case
proof cases
  case here
    then show ?thesis
    using  $\text{Suc}(1)$  succ-diff succ-in by (metis sup-commute underS-I )
  next
    case there
    then show ?thesis
    using  $\text{Suc}$  by (metis in-mono underS-subset-under underS-succ)
  next
qed
next
  case (Lim i)
  then obtain j where  $\langle j \in \text{underS } i \rangle \langle x \in \text{params} (\text{extend } C S j) \rangle$ 
  unfolding extend-def extendL-def worecZSL-isLim[OF adm-woL-extendL Lim(1-2)]
  by blast
  then show ?case
  using Lim underS-trans[of - j i] by meson
qed

```

```

lemma is-chain-extend:  $\langle \text{is-chain} (\text{extend } C S) \rangle$ 
by (simp add: extend-under is-chain-def)

```

```

lemma extend-in-C-step:
  assumes  $\langle \text{prop}_A Ks C \rangle \langle \{a\} \cup \text{extend } C S a \in C \rangle$ 
  and inf:  $\langle \text{infinite} (\text{Collect is-param} - \text{params} (\{a\} \cup \text{extend } C S a)) \rangle$ 
  shows  $\langle \text{extend } C S (\text{succ } a) \in C \rangle$ 
proof -
  have  $\langle \text{set } Qs \subseteq \text{set } Ks \implies \text{witness-kinds } Qs a (\text{extend } C S a) \cup \{a\} \cup \text{extend } C S a \in C \rangle$  for  $Qs$ 
  proof (induct  $Qs$ )
    case Nil
    then show ?case
    using assms(2) by simp
  next
    case (Cons Q  $Qs$ )
    let ?S =  $\langle \text{extend } C S a \rangle$ 
    let ?rest =  $\langle \text{witness-kinds } Qs a ?S \rangle$ 

    have Q:  $\langle Q \in \text{set } Ks \rangle$ 
    using Cons.premis by simp

    have *:  $\langle ?rest \cup \{a\} \cup ?S \in C \rangle$ 
    using Cons by simp

```

```

show ?case
proof (cases Q)
  case (Wits W)

  have ⟨infinite (Collect is-param – params (?rest ∪ {a} ∪ ?S))⟩
    using finite-witness-kinds-params inf by (metis UN-Un Un-assoc infi-
nite-Diff-fin-Un)
  then have **: ⟨∃x. x ∈ Collect is-param – params (?rest ∪ {a} ∪ ?S)⟩
    by (metis ex-in-conv finite.emptyI)
  then obtain x where x:
    ⟨(SOME x. x ∈ Collect is-param – params (?rest ∪ {a} ∪ ?S)) = x⟩
    ⟨x ∉ params (?rest ∪ {a} ∪ ?S)⟩
    ⟨is-param x⟩
    by (metis (mono-tags, lifting) DiffE mem-Collect-eq someI-ex)

  have ⟨a ∈ ?rest ∪ {a} ∪ ?S⟩
    by simp
  then have ⟨∀x. x ∉ params (?rest ∪ {a} ∪ ?S) ⟶ is-param x ⟶ set (W
a x) ∪ ?rest ∪ {a} ∪ ?S ∈ C⟩
    using assms(1) * Q Wits unfolding propA-def Un-assoc by fast
  then have ⟨set (W a x) ∪ ?rest ∪ {a} ∪ ?S ∈ C⟩
    using x by fast

  moreover have ⟨witness-kinds (Q # Qs) a ?S = set (W a x) ∪ ?rest⟩
    using Cons Wits x by (simp add: Let-def)
  ultimately show ?thesis
    by simp
  next
  case (Cond P)
  then show ?thesis
    using * by simp
  qed
qed
then have ⟨witness a (extend C S a) ∪ {a} ∪ extend C S a ∈ C⟩
  unfolding witness-def by blast
then show ?thesis
  unfolding extend-succ using assms(2) by simp
qed

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⟩
  using assms extend-succ by auto

lemma infinite-succ-extend:
  assumes ⟨S ∈ C⟩ ⟨enough-new S⟩ ⟨isSucc p⟩
  shows ⟨infinite (Collect is-param – params (extend C S p))⟩

```

```

using assms
proof (induct p rule: well-order-inductZSL)
  case Zero
  then show ?case
    using not-isSucc-zero by blast
next
  case (Suc i)
  then have *:  $\langle |underS\ i| < o \mid UNIV :: 'fm\ set \rangle$ 
    using card-of-underS by (simp add: Cinfinites-r)

  let ?params =  $\langle \lambda k. params\ (\{k\} \cup witness\ k\ (extend\ C\ S\ k)) \rangle$ 
  let ?X =  $\langle \bigcup k \in underS\ i. ?params\ k \rangle$ 
  have  $\langle |?X| < o \mid UNIV :: 'fm\ set \rangle$ 
  proof (cases  $\langle finite\ (underS\ i) \rangle$ )
    case True
    then have  $\langle finite\ ?X \rangle$ 
      using finite-witness-params by simp
    then show ?thesis
      using Cinfinites-r unfolding cinfinites-def by (simp add: finite-ordLess-infinite)
  next
  case False
  moreover have  $\langle \forall k. finite\ (?params\ k) \rangle$ 
    using finite-witness-params by simp
  then have  $\langle \forall k. |?params\ k| < o \mid underS\ i \rangle$ 
    using False by simp
  ultimately have  $\langle |?X| \leq o \mid underS\ i \rangle$ 
    using card-of-infinite-smaller-Union by fast
  then show ?thesis
    using * ordLeq-ordLess-trans by blast
qed
  then have  $\langle |?X| < o \mid Collect\ is-param - params\ S \rangle$ 
    using Suc(4) ordLess-ordLeq-trans unfolding enough-new-def by blast
  moreover have  $\langle infinite\ (Collect\ is-param - params\ S) \rangle$ 
    using Suc(4) Cinfinites-r unfolding cinfinites-def enough-new-def
    by (metis Field-card-of ordLeq-finite-Field)
  ultimately have  $\langle |Collect\ is-param - params\ S - ?X| = o \mid Collect\ is-param -$ 
params\ S \rangle
    using card-of-Un-diff-infinite by blast
  moreover from this have  $\langle infinite\ (Collect\ is-param - params\ S - ?X) \rangle$ 
    using  $\langle infinite\ (Collect\ is-param - params\ S) \rangle$  card-of-ordIso-finite by blast
  moreover have  $\langle \bigwedge a. a \in params\ (extend\ C\ S\ i) \implies a \in params\ S \vee a \in ?X \rangle$ 
    using params-origin by simp
  then have  $\langle params\ (extend\ C\ S\ i) \subseteq params\ S \cup ?X \rangle$ 
    by fast
  ultimately have  $\langle infinite\ (Collect\ is-param - params\ (extend\ C\ S\ i)) \rangle$ 
    using infinite-Diff-subset by (metis (no-types, lifting) Set-Diff-Un)
  then show ?case
    using Suc extend-succ inf-univ witness-def witness-kinds-new by presburger
next

```

```

case (Lim i)
then show ?case
  using isLim-def by blast
qed

lemma extend-in-C:
  assumes ⟨propA Ks C⟩ ⟨finite-char C⟩ ⟨S ∈ C⟩ ⟨enough-new S⟩
  shows ⟨extend C S a ∈ C⟩
  using assms
proof (induct a rule: well-order-inductZSL)
  case Zero
  then show ?case
    by (simp add: adm-woL-extendL extend-def wrecZSL-zero)
next
  case (Suc i)
  have ⟨infinite (Collect is-param – params (extend C S (succ i)))⟩
    using infinite-succ-extend aboveS-ne assms(3,4) isSucc-succ by blast
  then have ⟨infinite (Collect is-param – params (extend C S i))⟩
    using extend-succ[of C S i] by (metis UN-Un Un-upper2 infinite-Diff-subset)
  then have ⟨infinite (Collect is-param – params ({i} ∪ extend C S i))⟩
    using finite-params-fm by (simp add: Compl-eq-Diff-UNIV infinite-Diff-fin-Un)
  then show ?case
    using Suc extend-in-C-step extend-in-C-stop succ-in[of i] by blast
next
  case (Lim i)
  show ?case
  proof (rule ccontr)
    assume ⟨extend C S i ∉ C⟩
    then obtain S' where S': ⟨S' ⊆ (⋃ a ∈ underS i. extend C S a)⟩ ⟨S' ∉ C⟩
    ⟨finite S'⟩
    using Lim(5) unfolding finite-char-def extend-def extendL-def wrecZSL-isLim[OF
    adm-woL-extendL Lim(1–2)]
    by blast
    then obtain as where as: ⟨S' ⊆ (⋃ a ∈ as. extend C S a)⟩ ⟨as ⊆ underS i⟩
    ⟨finite as⟩
    by (metis finite-subset-Union finite-subset-image)
    moreover from this(1) have ⟨as ≠ {}⟩
    using S' Lim unfolding finite-char-def
    by (metis Union-empty bot.extremum-uniqueI empty-subsetI image-empty)
    ultimately obtain j where ⟨∀ a ∈ as. a ∈ under j⟩ ⟨j ∈ underS i⟩
    using finite-underS-bound by (metis in-mono)
    then have ⟨∀ a ∈ as. extend C S a ⊆ extend C S j⟩
    using extend-under by fast
    then have ⟨S' ⊆ extend C S j⟩
    using S' as(1) by blast
    then show False
    using Lim(3–) S'(2) as(2–3) ⟨∀ a ∈ as. a ∈ under j⟩ ⟨as ≠ {}⟩ ⟨j ∈ underS
    i⟩
    by (meson Order-Relation.underS-Field finite-char-closed subsetD subset-closed-def)
  end

```

qed
qed

lemma *Extend-in-C*:

assumes $\langle \text{prop}_A \text{ Ks } C \rangle \langle \text{finite-char } C \rangle \langle S \in C \rangle \langle \text{enough-new } S \rangle$
shows $\langle \text{Extend } C \ S \in C \rangle$
unfolding *Extend-def* using *assms chain-union-closed is-chain-extend extend-in-C*
by *simp*

theorem *Extend-maximal*:

assumes $\langle \text{subset-closed } C \rangle$
shows $\langle \text{maximal } C \ (\text{Extend } C \ S) \rangle$
unfolding *maximal-def Extend-def*
proof (*intro ballI impI*)
fix S'
assume *: $\langle S' \in C \rangle \langle \bigcup x. \text{extend } C \ S \ x \subseteq S' \rangle$
moreover have $\langle S' \subseteq \bigcup x. \text{extend } C \ S \ x \rangle$
proof (*rule ccontr*)
assume $\langle \neg S' \subseteq \bigcup x. \text{extend } C \ S \ x \rangle$
then have $\langle \exists z. z \in S' \wedge z \notin \bigcup x. \text{extend } C \ S \ x \rangle$
by *blast*
then obtain a where $a: \langle a \in S' \rangle \langle a \notin \bigcup x. \text{extend } C \ S \ x \rangle$
using $\ast(1)$ by *blast*
then have $\langle \{a\} \cup \text{extend } C \ S \ a \subseteq S' \rangle$
using \ast by *blast*
moreover have $\langle \forall S \subseteq S'. S \in C \rangle$
using *assms* $\langle S' \in C \rangle$ unfolding *subset-closed-def* by *blast*
ultimately have $\langle \{a\} \cup \text{extend } C \ S \ a \in C \rangle$
by *blast*
then have $\langle a \in \text{extend } C \ S \ (\text{succ } a) \rangle$
using a by *simp*
then show *False*
using \ast a by *blast*
qed
ultimately show $\langle \bigcup x. \text{extend } C \ S \ x = S' \rangle$
by *simp*
qed

definition *witnessed* :: $\langle 'fm \ \text{set} \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{witnessed } S \equiv \forall p \in S. \exists S'. \text{infinite } (\text{Collect } \text{is-param } - \ \text{params } S') \wedge \text{witness } p \ S' \subseteq S \rangle$

theorem *Extend-witnessed*:

assumes $\langle \text{prop}_A \text{ Ks } C \rangle \langle \text{finite-char } C \rangle \langle S \in C \rangle \langle \text{enough-new } S \rangle$
shows $\langle \text{witnessed } (\text{Extend } C \ S) \rangle$
unfolding *witnessed-def*
proof *safe*
fix p
assume $\langle p \in \text{Extend } C \ S \rangle$

then have $\langle \{p\} \cup \text{extend } C S p \subseteq \text{Extend } C S \rangle$
unfolding *Extend-def* **by** *blast*
moreover have $\langle \text{Extend } C S \in C \rangle$
using *Extend-in-C* **assms** **by** *blast*
ultimately have $\langle \{p\} \cup \text{extend } C S p \in C \rangle$
using $\langle \text{finite-char } C \rangle$ *finite-char-closed* **unfolding** *subset-closed-def* **by** *blast*
moreover have $\langle \text{extend } C S (\text{succ } p) \in C \rangle$
using *assms extend-in-C* **by** *blast*
ultimately have $\langle \text{witness } p (\text{extend } C S p) \cup \{p\} \cup \text{extend } C S p \in C \rangle$
unfolding *extend-succ* **by** *simp*
then have $\langle \text{witness } p (\text{extend } C S p) \cup \{p\} \cup \text{extend } C S p \subseteq \text{Extend } C S \rangle$
unfolding *Extend-def* **using** *extend-succ* $\langle \{p\} \cup \text{extend } C S p \in C \rangle$ **by** *fastforce*
moreover have $\langle \text{infinite } (\text{Collect is-param } - \text{ params } (\text{extend } C S (\text{succ } p))) \rangle$
using *infinite-succ-extend* **by** $(\text{meson aboveS-ne assms}(3,4) \text{ isSucc-def})$
then have $\langle \text{infinite } (\text{Collect is-param } - \text{ params } (\text{extend } C S p)) \rangle$
using *extend-succ* **by** $(\text{metis SUP-union Un-upper2 infinite-Diff-subset})$
ultimately show $\langle \exists S'. \text{infinite } (\text{Collect is-param } - \text{ params } S') \wedge \text{witness } p S' \subseteq \text{Extend } C S \rangle$
by *fast*
qed

abbreviation *mk-mcs* :: $\langle 'fm \text{ set } set \Rightarrow 'fm \text{ set} \Rightarrow 'fm \text{ set} \rangle$ **where**
 $\langle \text{mk-mcs } C S \equiv \text{Extend } (\text{mk-alt-fin } C) S \rangle$

theorem *mk-mcs-rmaximal*: $\langle \text{maximal } C (\text{mk-mcs } C S) \rangle$
using *Extend-maximal maximal-def mk-alt-fin-in mk-alt-fin-subset-closed* **by** *meson*

theorem *mk-mcs-witnessed*:
assumes $\langle \text{prop}_E Ks C \rangle \langle S \in C \rangle \langle \text{enough-new } S \rangle$
shows $\langle \text{witnessed } (\text{mk-mcs } C S) \rangle$
using *assms Extend-witnessed prop_E mk-alt-fin-finite-char mk-alt-fin-in* **by** *blast*

1.4 Hintikka Sets

lemma *mk-mcs-hintikka*:
assumes $\langle \text{prop}_E Ks C \rangle \langle S \in C \rangle \langle \text{enough-new } S \rangle$
shows $\langle \text{prop}_H Ks (\text{mk-mcs } C S) \rangle$
unfolding *prop_H-def*

proof

fix *K*

assume *K*: $\langle K \in \text{set } Ks \rangle$

show $\langle \text{sat}_H K (\text{mk-mcs } C S) \rangle$

proof $(\text{cases } K)$

case $(\text{Cond } P H)$

moreover have $\langle \text{maximal } (\text{mk-alt-fin } C) (\text{mk-mcs } C S) \rangle$

using *Extend-maximal mk-alt-fin-subset-closed* **by** *blast*

moreover have $\langle \text{prop}_A Ks (\text{mk-alt-fin } C) \rangle$

using *assms(1) prop_E* **by** *blast*

```

then have ⟨mk-mcs C S ∈ mk-alt-fin C⟩
  using assms(2-3) Extend-in-C mk-alt-fin-finite-char mk-alt-fin-in by blast
moreover have ⟨satE (Cond P H) (mk-alt-fin C)⟩
  using ⟨propA Ks (mk-alt-fin C)⟩ Cond K unfolding propA-def by fast
ultimately show ?thesis
  using K all-kinds Consistency-Kind.hintikka by meson
next
case (Wits W)
have ⟨witnessed (mk-mcs C S)⟩
  using mk-mcs-witnessed[OF assms(1-3)] .
then have ⟨∀p ∈ mk-mcs C S. ∃x. is-param x ∧ set (W p x) ⊆ mk-mcs C S⟩
  unfolding witnessed-def witness-def using inf-univ Wits witness-kinds K by
fast
  then show ?thesis
    using Wits by fast
qed
qed
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: ⟨propH Ks H⟩
begin

```

```

lemma satH: ⟨K ∈ set Ks ⟹ satH K H⟩
  using hintikka unfolding propH-def by blast

```

end

```

context Maximal-Consistency
begin

```

```

theorem mk-mcs-Hintikka:
  assumes ⟨propE Ks C⟩ ⟨S ∈ C⟩ ⟨enough-new S⟩
  shows ⟨Hintikka map-fm params-fm is-param Ks (mk-mcs C S)⟩
  using assms mk-mcs-hintikka by unfold-locales

```

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) ⇒ satE 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) ⇒ propE Ks {A. enough-new A ∧ ⊢ A}⟩
begin

```

```

theorem Consistency: ⟨propE Ks {A. enough-new A ∧ ⊢ A}⟩
  using all-consistent inf-univ unfolding propE-def by fast

```

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) ⇒ satE 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) ⇒ propE 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 ⟨ $\rightsquigarrow_{\mathbf{X}}$ ⟩ 50)
assumes confl-map: ⟨ $\bigwedge ps\ qs\ f.\ ps \rightsquigarrow_{\mathbf{X}} qs \implies \text{map } (\text{map-fm } f) ps \rightsquigarrow_{\mathbf{X}} \text{map } (\text{map-fm } f) qs$ ⟩
begin

inductive cond where
  cond [intro!]: ⟨ $ps \rightsquigarrow_{\mathbf{X}} qs \implies \text{cond } ps (\lambda S.\ \text{set } qs \cap S = \{\})$ ⟩

inductive-cases condE[elim!]: ⟨cond ps Q⟩

inductive hint where
  hint [intro!]: ⟨ $(\bigwedge ps\ qs\ q.\ ps \rightsquigarrow_{\mathbf{X}} qs \implies \text{set } ps \subseteq H \implies q \in \text{set } qs \implies q \notin H) \implies \text{hint } H$ ⟩

declare hint.simps[simp]

abbreviation kind :: ⟨('x, 'fm) kind⟩ where
  ⟨kind  $\equiv$  Cond cond hint⟩

end

sublocale Confl  $\subseteq$  Consistency-Kind map-fm params-fm is-param kind
proof
  fix C
  assume conflC: ⟨satE kind C⟩
  then show ⟨satE kind (close C)⟩
  proof safe
    fix S ps qs q
    assume ⟨S  $\in$  close C⟩
    then obtain S' where S': ⟨S'  $\in$  C⟩ and ⟨S  $\subseteq$  S'⟩
    unfolding close-def by blast

    assume ⟨set ps  $\subseteq$  S⟩
    then have *: ⟨set ps  $\subseteq$  S'⟩
    using ⟨S  $\subseteq$  S'⟩ by blast

    assume **: ⟨ps  $\rightsquigarrow_{\mathbf{X}}$  qs⟩
    then have ⟨ $\forall q \in \text{set } qs.\ q \notin S'$ ⟩
    using conflC S' * by blast
    then have ⟨ $\forall q \in \text{set } qs.\ q \notin S$ ⟩
    using ⟨S  $\subseteq$  S'⟩ by blast
    moreover assume ⟨q  $\in$  set qs⟩ ⟨q  $\in$  S⟩
  qed

```

```

ultimately show  $\langle q \in \{\} \rangle$ 
  using ** by auto
qed
next
fix C
assume conflC:  $\langle \text{sat}_E \text{ kind } C \rangle$ 
then show  $\langle \text{sat}_A \text{ kind } (\text{mk-alt-consistency } C) \rangle$ 
proof safe
  fix S ps qs q

  assume  $\langle S \in \text{mk-alt-consistency } C \rangle$ 
  then obtain f where f:  $\langle \text{map-fm } f \text{ ' } S \in C \rangle$ 
    unfolding mk-alt-consistency-def by blast

  let ?C =  $\langle \text{mk-alt-consistency } C \rangle$ 
  let ?S =  $\langle \text{map-fm } f \text{ ' } S \rangle$ 

  assume  $\langle \text{set } ps \subseteq S \rangle$ 
  then have *:  $\langle \text{set } (\text{map } (\text{map-fm } f) ps) \subseteq ?S \rangle$ 
    by auto

  assume  $\langle ps \rightsquigarrow_{\mathbf{X}} qs \rangle$ 
  then have  $\langle \text{map } (\text{map-fm } f) ps \rightsquigarrow_{\mathbf{X}} (\text{map } (\text{map-fm } f) qs) \rangle$ 
    using confl-map by blast
  then have  $\langle \forall q \in \text{set } (\text{map } (\text{map-fm } f) qs). q \notin ?S \rangle$ 
    using conflC f * by blast
  then have  $\langle \forall q \in \text{set } qs. \text{map-fm } f q \notin ?S \rangle$ 
    by simp
  then have  $\langle \forall q \in \text{set } qs. q \notin S \rangle$ 
    by blast
  moreover assume  $\langle q \in \text{set } qs \rangle \langle q \in S \rangle$ 
  ultimately show  $\langle q \in \{\} \rangle$ 
    by auto
qed
next
fix C
assume conflAC:  $\langle \text{sat}_A \text{ kind } C \rangle$  and closedC:  $\langle \text{subset-closed } C \rangle$ 
then show  $\langle \text{sat}_A \text{ kind } (\text{mk-finite-char } C) \rangle$ 
proof safe
  fix S ps qs q
  assume  $\langle S \in \text{mk-finite-char } C \rangle$ 
  then have finc:  $\langle \forall S' \subseteq S. \text{finite } S' \longrightarrow S' \in C \rangle$ 
    unfolding mk-finite-char-def by blast

  have sc:  $\langle \forall S' \in C. \forall S \subseteq S'. S \in C \rangle$ 
    using closedC unfolding subset-closed-def by blast
  then have sc':  $\langle \bigwedge S' x. x \cup S' \in C \implies \forall S \subseteq x \cup S'. S \in C \rangle$ 
    by blast

```

```

    assume *: ⟨set ps ⊆ S⟩ ⟨ps ∼χ qs⟩ ⟨q ∈ set qs⟩ ⟨q ∈ S⟩
    then have ⟨{q} ∪ set ps ∈ C⟩
      using ⟨set ps ⊆ S⟩ fin by simp
    then have ⟨q ∈ {}⟩
      using * conflAC by blast
    then show ⟨q ∈ {}⟩
      by auto
  qed
next
fix C S
assume *: ⟨satE kind C⟩ ⟨S ∈ C⟩ ⟨maximal C S⟩
show ⟨satH kind S⟩
proof safe
  fix ps qs q
  assume **: ⟨set ps ⊆ S⟩ ⟨ps ∼χ qs⟩ ⟨q ∈ set qs⟩ ⟨q ∈ S⟩
  then show False
    using * by blast
  qed
qed

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 ⟨∼χ⟩ 50) +
fixes consistent :: ⟨'fm set ⇒ bool⟩ (⟨⊢ -⟩ [51] 50)
assumes consistent: ⟨∧ S ps qs x. set ps ⊆ S ⇒ ps ∼χ qs ⇒ x ∈ set qs ⇒
x ∈ S ⇒ ¬ ⊢ S⟩

sublocale Derivational-Confl ⊆ Derivational-Kind map-fm params-fm is-param
kind consistent
  using infinite-params-left consistent by unfold-locales blast+

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 ⟨∼χ⟩ 50) +
fixes consistent :: ⟨'fm list ⇒ bool⟩ (⟨⊢ -⟩ [51] 50)
assumes consistent: ⟨∧ A ps qs x. set ps ⊆ set A ⇒ ps ∼χ qs ⇒ x ∈ set qs
⇒ x ∈ set A ⇒ ¬ ⊢ A⟩

sublocale Weak-Derivational-Confl ⊆ Weak-Derivational-Kind map-fm params-fm
is-param kind consistent
  using infinite-params-left consistent by unfold-locales blast+

```

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 ⟨~>α 50)
  assumes alpha-map: ⟨∧ps qs f. ps ~>α qs ⟶ map (map-fm f) ps ~>α map
    (map-fm f) qs⟩
  begin

  inductive cond where
    cond [intro!]: ⟨ps ~>α qs ⟶ cond ps (λC S. set qs ∪ S ∈ C)⟩

  inductive-cases condE[elim!]: ⟨cond ps Q⟩

  inductive hint where
    hint [intro!]: ⟨(∧ps qs q. ps ~>α 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
    fix C
    assume alphaC: ⟨satE kind C⟩
    then show ⟨satE kind (close C)⟩
    proof safe
      fix S ps qs q
      assume ⟨S ∈ close C⟩
      then obtain S' where S': ⟨S' ∈ C⟩ and ⟨S ⊆ S'⟩
      unfolding close-def by blast

      assume ⟨set ps ⊆ S⟩ ⟨ps ~>α qs⟩
      then have *: ⟨set ps ⊆ S'⟩
      using ⟨S ⊆ S'⟩ by blast

      assume **: ⟨ps ~>α qs⟩
      then have ⟨set qs ∪ S' ∈ C⟩
      using alphaC S' * by blast
      then show ⟨set qs ∪ S ∈ close C⟩
      using ⟨S ⊆ S'⟩ subset-in-close by blast
    qed

```

```

next
  fix C
  assume alphaC: ⟨satE kind C⟩
  then show ⟨satA kind (mk-alt-consistency C)⟩
  proof safe
    fix S ps qs

    assume ⟨S ∈ mk-alt-consistency C⟩
    then obtain f where f: ⟨is-subst f⟩ ⟨map-fm f ‘ S ∈ C⟩
      unfolding mk-alt-consistency-def by blast

    let ?C = ⟨mk-alt-consistency C⟩
    let ?S = ⟨map-fm f ‘ S⟩

    assume ⟨set ps ⊆ S⟩
    then have *: ⟨set (map (map-fm f) ps) ⊆ ?S⟩
      by auto

    assume ⟨ps ∼α qs⟩
    then have ⟨map (map-fm f) ps ∼α (map (map-fm f) qs)⟩
      using alpha-map by blast
    then have ⟨set (map (map-fm f) qs) ∪ ?S ∈ C⟩
      using alphaC * f by blast
    then show ⟨set qs ∪ S ∈ ?C⟩
      unfolding mk-alt-consistency-def using f by (auto simp: image-Un)
  qed
next
  fix C
  assume alphaAC: ⟨satA kind C⟩ and closedC: ⟨subset-closed C⟩
  then show ⟨satA kind (mk-finite-char C)⟩
  proof safe
    fix S ps qs q
    assume ⟨S ∈ mk-finite-char C⟩
    then have fnc: ⟨∀ S' ⊆ S. finite S' ⟶ S' ∈ C⟩
      unfolding mk-finite-char-def by blast

    have sc: ⟨∀ S' ∈ C. ∀ S ⊆ S'. S ∈ C⟩
      using closedC unfolding subset-closed-def by blast
    then have sc': ⟨∧ S' x. x ∪ S' ∈ C ⟶ ∀ S ⊆ x ∪ S'. S ∈ C⟩
      by blast

    assume *: ⟨set ps ⊆ S⟩ and **: ⟨ps ∼α qs⟩

    show ⟨set qs ∪ S ∈ mk-finite-char C⟩
      unfolding mk-finite-char-def
    proof safe
      fix S'
      let ?S' = ⟨set ps ∪ (S' - set qs)⟩

```

```

    assume ⟨ $S' \subseteq \text{set } qs \cup S$ ⟩ and ⟨finite  $S'$ ⟩
    then have ⟨ $?S' \subseteq S$ ⟩
      using * by blast
    moreover have ⟨finite  $?S'$ ⟩
      using ⟨finite  $S'$ ⟩ by blast
    ultimately have ⟨ $?S' \in C$ ⟩
      using finc by blast
    then have ⟨ $\text{set } qs \cup ?S' \in C$ ⟩
      using ** alphaAC by fast
    then show ⟨ $S' \in C$ ⟩
      using sc by fast
  qed
qed
next
fix  $C S$ 
assume *: ⟨satE kind  $C$ ⟩ ⟨ $S \in C$ ⟩ ⟨maximal  $C S$ ⟩
show ⟨satH kind  $S$ ⟩
proof safe
  fix  $ps qs q$ 
  assume **: ⟨ $\text{set } ps \subseteq S$ ⟩ ⟨ $ps \rightsquigarrow_\alpha qs$ ⟩
  then have ⟨ $\text{set } qs \cup S \in C$ ⟩
    using * by blast
  moreover assume ⟨ $q \in \text{set } qs$ ⟩
  ultimately show ⟨ $q \in S$ ⟩
    using ⟨maximal  $C S$ ⟩ unfolding maximal-def by fast
qed
qed

locale Derivational-Alpha = Alpha map-fm params-fm is-param classify
for
  map-fm :: ⟨ $'x \Rightarrow 'x$ ⟩  $\Rightarrow$   $'fm \Rightarrow 'fm$ ⟩ and
  params-fm :: ⟨ $'fm \Rightarrow 'x \text{ set}$ ⟩ and
  is-param :: ⟨ $'x \Rightarrow \text{bool}$ ⟩ and
  classify :: ⟨ $'fm \text{ list} \Rightarrow 'fm \text{ list} \Rightarrow \text{bool}$ ⟩ (infix ⟨ $\rightsquigarrow_\alpha$ ⟩ 50) +
fixes consistent :: ⟨ $'fm \text{ set} \Rightarrow \text{bool}$ ⟩ (⟨ $\vdash - \rangle$  [51] 50)
assumes consistent: ⟨ $\bigwedge S ps qs. \text{set } ps \subseteq S \implies ps \rightsquigarrow_\alpha qs \implies \vdash S \implies \vdash \text{set } qs \cup S$ ⟩

sublocale Derivational-Alpha  $\subseteq$  Derivational-Kind map-fm params-fm is-param kind consistent
  using infinite-params-left consistent enough-new-def by unfold-locales blast+

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 :: ⟨ $'fm \Rightarrow 'x \text{ set}$ ⟩ and
  is-param :: ⟨ $'x \Rightarrow \text{bool}$ ⟩ and
  classify :: ⟨ $'fm \text{ list} \Rightarrow 'fm \text{ list} \Rightarrow \text{bool}$ ⟩ (infix ⟨ $\rightsquigarrow_\alpha$ ⟩ 50) +

```

fixes *consistent* :: ⟨'fm list ⇒ bool⟩ (⟨⊢ -⟩ [51] 50)
assumes *consistent*: ⟨ $\bigwedge A$ ps qs. set ps ⊆ set A ⇒ ps ∼_α qs ⇒ ⊢ A ⇒ ⊢ qs
 @ A⟩

sublocale *Weak-Derivational-Alpha* ⊆ *Weak-Derivational-Kind map-fm params-fm is-param kind consistent*

proof

show ⟨*sat_E kind* {S. ∃ A. set A = S ∧ ⊢ A}⟩

proof safe

fix ps qs A

assume ⟨set ps ⊆ set A⟩ ⟨ps ∼_α qs⟩ ⟨⊢ A⟩

then show ⟨∃ B. set B = set qs ∪ set A ∧ ⊢ B⟩

using *consistent*[of ps A qs] **by** (*meson set-append*)

qed

qed

1.9 Beta

locale *Beta* = *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** ⟨∼_β⟩ 50)

assumes *beta-map*: ⟨ \bigwedge ps qs f. ps ∼_β qs ⇒ map (map-fm f) ps ∼_β map (map-fm f) qs⟩

begin

inductive cond where

cond [intro!]: ⟨ps ∼_β qs ⇒ cond ps (λC S. ∃ q ∈ set qs. {q} ∪ S ∈ C)⟩

inductive-cases *condE*[elim!]: ⟨cond ps Q⟩

inductive hint where

hint [intro!]: ⟨(\bigwedge ps qs. ps ∼_β 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 *Beta* ⊆ *Consistency-Kind map-fm params-fm is-param kind*

proof

fix C

assume *betaC*: ⟨*sat_E kind* C⟩

then show ⟨*sat_E kind* (close C)⟩

```

proof safe
  fix  $S$   $ps$   $qs$   $q$ 
  assume  $\langle S \in \text{close } C \rangle$ 
  then obtain  $S'$  where  $S'$ :  $\langle S' \in C \rangle$  and  $\langle S \subseteq S' \rangle$ 
    unfolding close-def by blast

  assume  $\langle \text{set } ps \subseteq S \rangle$   $\langle ps \rightsquigarrow_{\beta} qs \rangle$ 
  then have  $*$ :  $\langle \text{set } ps \subseteq S' \rangle$ 
    using  $\langle S \subseteq S' \rangle$  by blast

  assume  $**$ :  $\langle ps \rightsquigarrow_{\beta} qs \rangle$ 
  then have  $\langle \exists q \in \text{set } qs. \{q\} \cup S' \in C \rangle$ 
    using betaC  $S'$   $*$  by blast
  then show  $\langle \exists q \in \text{set } qs. \text{insert } q S \in \text{close } C \rangle$ 
    using  $\langle S \subseteq S' \rangle$  subset-in-close by (metis insert-is-Un)
qed
next
fix  $C$ 
assume betaC:  $\langle \text{sat}_E \text{ kind } C \rangle$ 
then show  $\langle \text{sat}_A \text{ kind } (\text{mk-alt-consistency } C) \rangle$ 
proof safe
  fix  $S$   $ps$   $qs$ 

  assume  $\langle S \in \text{mk-alt-consistency } C \rangle$ 
  then obtain  $f$  where  $f$ :  $\langle \text{is-subst } f \rangle$   $\langle \text{map-fm } f \text{ ' } S \in C \rangle$ 
    unfolding mk-alt-consistency-def by blast

  let  $?C = \langle \text{mk-alt-consistency } C \rangle$ 
  let  $?S = \langle \text{map-fm } f \text{ ' } S \rangle$ 

  assume  $\langle \text{set } ps \subseteq S \rangle$ 
  then have  $*$ :  $\langle \text{set } (\text{map } (\text{map-fm } f) ps) \subseteq ?S \rangle$ 
    by auto

  assume  $\langle ps \rightsquigarrow_{\beta} qs \rangle$ 
  then have  $\langle \text{map } (\text{map-fm } f) ps \rightsquigarrow_{\beta} (\text{map } (\text{map-fm } f) qs) \rangle$ 
    using beta-map by blast
  then have  $\langle \exists q \in \text{set } (\text{map } (\text{map-fm } f) qs). \{q\} \cup ?S \in C \rangle$ 
    using betaC  $*$   $f$  by blast
  then show  $\langle \exists q \in \text{set } qs. \text{insert } q S \in ?C \rangle$ 
    unfolding mk-alt-consistency-def using  $f$  by (auto simp: image-Un)
qed
next
fix  $C$ 
assume betaAC:  $\langle \text{sat}_A \text{ kind } C \rangle$  and closedC:  $\langle \text{subset-closed } C \rangle$ 
then show  $\langle \text{sat}_A \text{ kind } (\text{mk-finite-char } C) \rangle$ 
proof safe
  fix  $S$   $ps$   $qs$   $q$ 
  assume  $\langle S \in \text{mk-finite-char } C \rangle$ 

```

```

then have finC:  $\langle \forall S' \subseteq S. \text{finite } S' \longrightarrow S' \in C \rangle$ 
  unfolding mk-finite-char-def by blast

have sc:  $\langle \forall S' \in C. \forall S \subseteq S'. S \in C \rangle$ 
  using closedC unfolding subset-closed-def by blast
then have sc':  $\langle \bigwedge S' x. x \cup S' \in C \implies \forall S \subseteq x \cup S'. S \in C \rangle$ 
  by blast

assume *:  $\langle \text{set } ps \subseteq S \rangle$  and **:  $\langle ps \sim_{\beta} qs \rangle$ 

show  $\langle \exists q \in \text{set } qs. \text{insert } q S \in \text{mk-finite-char } C \rangle$ 
proof (rule ccontr)
  assume  $\langle \neg ?thesis \rangle$ 
  then have  $\langle \forall q \in \text{set } qs. \exists Sq. Sq \subseteq \text{insert } q S \wedge \text{finite } Sq \wedge Sq \notin C \rangle$ 
    unfolding mk-finite-char-def by blast
  then obtain f where f:  $\langle \forall q \in \text{set } qs. f q \subseteq \text{insert } q S \wedge \text{finite } (f q) \wedge f q \notin C \rangle$ 
    by metis

  let ?S' =  $\langle \text{set } ps \cup \bigcup \{f q - \{q\} \mid q. q \in \text{set } qs\} \rangle$ 

  have  $\langle ?S' \subseteq S \rangle$ 
    using * by fast
  moreover have  $\langle \text{finite } ?S' \rangle$ 
    using f by auto
  ultimately have  $\langle ?S' \in C \rangle$ 
    using finC by blast
  then have  $\langle \exists q \in \text{set } qs. \{q\} \cup ?S' \in C \rangle$ 
    using ** betaAC by fast
  then have  $\langle \exists q \in \text{set } qs. f q \in C \rangle$ 
    using sc' by blast
  then show False
    using f by blast
qed
qed
next
fix C S
assume *:  $\langle \text{sat}_E \text{ kind } C \rangle$   $\langle S \in C \rangle$   $\langle \text{maximal } C S \rangle$ 
show  $\langle \text{sat}_H \text{ kind } S \rangle$ 
proof safe
  fix ps qs
  assume **:  $\langle \text{set } ps \subseteq S \rangle$   $\langle ps \sim_{\beta} qs \rangle$ 
  then have  $\langle \exists q \in \text{set } qs. \{q\} \cup S \in C \rangle$ 
    using * by blast
  then show  $\langle \exists q \in \text{set } qs. q \in S \rangle$ 
    using  $\langle \text{maximal } C S \rangle$  unfolding maximal-def by fast
qed
qed

```

locale *Derivational-Beta* = *Beta map-fm params-fm is-param classify*
for
map-fm :: $\langle 'x \Rightarrow 'x \rangle \Rightarrow 'fm \Rightarrow 'fm$ **and**
params-fm :: $\langle 'fm \Rightarrow 'x \text{ set} \rangle$ **and**
is-param :: $\langle 'x \Rightarrow \text{bool} \rangle$ **and**
classify :: $\langle 'fm \text{ list} \Rightarrow 'fm \text{ list} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_\beta \rangle$ 50) +
fixes *consistent* :: $\langle 'fm \text{ set} \Rightarrow \text{bool} \rangle$ ($\langle \vdash \rightarrow \rangle$ [51] 50)
assumes *consistent*: $\langle \bigwedge S \text{ ps qs. set ps} \subseteq S \implies \text{ps} \rightsquigarrow_\beta \text{qs} \implies \vdash S \implies \exists q \in \text{set qs. } \vdash \{q\} \cup S \rangle$

sublocale *Derivational-Beta* \subseteq *Derivational-Kind map-fm params-fm is-param kind consistent*
proof
assume *inf*: $\langle \text{infinite } (UNIV :: 'fm \text{ set}) \rangle$
then show $\langle \text{sat}_E \text{ kind } \{A. \text{enough-new } A \wedge \vdash A\} \rangle$
proof safe
fix *S ps qs*
assume *: $\langle \text{set ps} \subseteq S \rangle \langle \text{ps} \rightsquigarrow_\beta \text{qs} \rangle \langle \vdash S \rangle$
then have $\langle \exists q \in \text{set qs. } \vdash \{q\} \cup S \rangle$
using *consistent by blast*
moreover assume $\langle \text{enough-new } S \rangle$
ultimately show $\langle \exists q \in \text{set qs. insert } q \text{ } S \in \{A. \text{enough-new } A \wedge \vdash A\} \rangle$
using *infinite-params-left[OF inf] unfolding enough-new-def*
by (*metis (no-types, lifting) empty-set insert-code(1) insert-is-Un mem-Collect-eq*)
qed
qed

locale *Weak-Derivational-Beta* = *Beta map-fm params-fm is-param classify*
for
map-fm :: $\langle 'x \Rightarrow 'x \rangle \Rightarrow 'fm \Rightarrow 'fm$ **and**
params-fm :: $\langle 'fm \Rightarrow 'x \text{ set} \rangle$ **and**
is-param :: $\langle 'x \Rightarrow \text{bool} \rangle$ **and**
classify :: $\langle 'fm \text{ list} \Rightarrow 'fm \text{ list} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_\beta \rangle$ 50) +
fixes *consistent* :: $\langle 'fm \text{ list} \Rightarrow \text{bool} \rangle$ ($\langle \vdash \rightarrow \rangle$ [51] 50)
assumes *consistent*: $\langle \bigwedge A \text{ ps qs. set ps} \subseteq \text{set } A \implies \text{ps} \rightsquigarrow_\beta \text{qs} \implies \vdash A \implies \exists q \in \text{set qs. } \vdash q \# A \rangle$

sublocale *Weak-Derivational-Beta* \subseteq *Weak-Derivational-Kind map-fm params-fm is-param kind consistent*
proof
show $\langle \text{sat}_E \text{ kind } \{S. \exists A. \text{set } A = S \wedge \vdash A\} \rangle$
proof safe
fix *ps qs A*
assume *: $\langle \text{set ps} \subseteq \text{set } A \rangle \langle \text{ps} \rightsquigarrow_\beta \text{qs} \rangle \langle \vdash A \rangle$
then have $\langle \exists q \in \text{set qs. } \vdash q \# A \rangle$
using *consistent by blast*
then show $\langle \exists q \in \text{set qs. insert } q \text{ } (\text{set } A) \in \{S. \exists A. \text{set } A = S \wedge \vdash A\} \rangle$
by (*metis (mono-tags, lifting) CollectI list.simps(15)*)
qed

qed

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
⟨∼γ⟩ 50)
assumes gamma-map: ⟨∧ps F qs f. ps ∼γ (F, qs) ⇒ (∃ G rs. map (map-fm f)
ps ∼γ (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: ⟨∧ps F qs S S'. ps ∼γ (F, qs) ⇒ S ⊆ S' ⇒ F S ⊆ F
S'⟩
and gamma-fin: ⟨∧ps F qs t A. ps ∼γ (F, qs) ⇒ t ∈ F A ⇒ ∃ B ⊆ A. finite
B ∧ t ∈ F B⟩
begin

inductive cond where
  cond [intro!]: ⟨ps ∼γ (F, qs) ⇒ cond ps (λC S. ∀ t ∈ F S. set (qs t) ∪ S ∈ C)⟩

inductive-cases condE[elim!]: ⟨cond ps Q⟩

inductive hint where
  hint [intro!]: ⟨(∧ps F qs. ps ∼γ (F, qs) ⇒ set ps ⊆ H ⇒ (∀ t ∈ F H. set (qs
t) ⊆ H)) ⇒ 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
  fix C
  assume gammaC: ⟨satE kind C⟩
  then show ⟨satE kind (close C)⟩
  proof safe
    fix S ps qs F t
    assume ⟨S ∈ close C⟩
    then obtain S' where S': ⟨S' ∈ C⟩ and ⟨S ⊆ S'⟩
    unfolding close-def by blast
```

```

assume  $\langle \text{set } ps \subseteq S \rangle$ 
then have *:  $\langle \text{set } ps \subseteq S' \rangle$ 
  using  $\langle S \subseteq S' \rangle$  by blast

assume **:  $\langle ps \rightsquigarrow_\gamma (F, qs) \rangle \langle t \in F S \rangle$ 
then have  $\langle t \in F S' \rangle$ 
  using ** gamma-mono  $\langle S \subseteq S' \rangle$  by blast
then have  $\langle \text{set } (qs t) \cup S' \in C \rangle$ 
  using gammaC  $S' * **$  by blast
then have  $\langle \text{set } (qs t) \cup S \in \text{close } C \rangle$ 
  using  $\langle S \subseteq S' \rangle$  subset-in-close by blast
then show  $\langle \text{set } (qs t) \cup S \in \text{close } C \rangle$ 
  by (meson close-closed equalityE subset-closed-def sup.mono)
qed
next
fix  $C$ 
assume gammaC:  $\langle \text{sat}_E \text{ kind } C \rangle$ 
then show  $\langle \text{sat}_A \text{ kind } (\text{mk-alt-consistency } C) \rangle$ 
proof safe
  fix  $S ps F qs t$ 

  assume  $\langle S \in \text{mk-alt-consistency } C \rangle$ 
then obtain  $f$  where  $f$ :  $\langle \text{is-subst } f \rangle \langle \text{map-fm } f ' S \in C \rangle$ 
  unfolding mk-alt-consistency-def by blast

  let  $?C = \langle \text{mk-alt-consistency } C \rangle$ 
  let  $?S = \langle \text{map-fm } f ' S \rangle$ 

  assume  $\langle \text{set } ps \subseteq S \rangle$ 
then have *:  $\langle \text{set } (\text{map } (\text{map-fm } f) ps) \subseteq ?S \rangle$ 
  by auto

  assume **:  $\langle ps \rightsquigarrow_\gamma (F, qs) \rangle$  and  $t$ :  $\langle t \in F S \rangle$ 
obtain  $rs G$  where  $rs$ :
     $\langle \text{map } (\text{map-fm } f) ps \rightsquigarrow_\gamma (G, rs) \rangle$ 
     $\langle \forall S. \text{map-tm } f ' F S \subseteq G (\text{map-fm } f ' S) \rangle$ 
     $\langle \forall t. \text{map } (\text{map-fm } f) (qs t) = rs (\text{map-tm } f t) \rangle$ 
    using gamma-map ** by meson
then have  $\langle \text{set } (rs (\text{map-tm } f t)) \cup ?S \in C \rangle$ 
  using gammaC  $f * t$  by blast
then have  $\langle \text{set } (\text{map } (\text{map-fm } f) (qs t)) \cup ?S \in C \rangle$ 
  using  $rs$  by simp
then show  $\langle \text{set } (qs t) \cup S \in ?C \rangle$ 
  unfolding mk-alt-consistency-def using  $f$  by (auto simp: image-Un)
qed
next
fix  $C$ 
assume gammaAC:  $\langle \text{sat}_A \text{ kind } C \rangle$  and closedC:  $\langle \text{subset-closed } C \rangle$ 
then show  $\langle \text{sat}_A \text{ kind } (\text{mk-finite-char } C) \rangle$ 

```

```

proof safe
  fix  $S$   $ps$   $F$   $qs$   $t$ 
  assume  $\langle S \in \text{mk-finite-char } C \rangle$ 
  then have  $\text{finc}$ :  $\langle \forall S' \subseteq S. \text{finite } S' \longrightarrow S' \in C \rangle$ 
    unfolding  $\text{mk-finite-char-def}$  by  $\text{blast}$ 

  have  $\text{sc}$ :  $\langle \forall S' \in C. \forall S \subseteq S'. S \in C \rangle$ 
    using  $\text{closedC}$  unfolding  $\text{subset-closed-def}$  by  $\text{blast}$ 
  then have  $\text{sc}'$ :  $\langle \bigwedge S' x. x \cup S' \in C \implies \forall S \subseteq x \cup S'. S \in C \rangle$ 
    by  $\text{blast}$ 

  assume  $*$ :  $\langle \text{set } ps \subseteq S \rangle$  and  $**$ :  $\langle ps \rightsquigarrow_\gamma (F, qs) \rangle$  and  $t$ :  $\langle t \in F S \rangle$ 

  show  $\langle \text{set } (qs \ t) \cup S \in \text{mk-finite-char } C \rangle$ 
    unfolding  $\text{mk-finite-char-def}$ 
  proof safe
    fix  $S'$ 
    assume  $1$ :  $\langle S' \subseteq \text{set } (qs \ t) \cup S \rangle$  and  $2$ :  $\langle \text{finite } S' \rangle$ 

    obtain  $A$  where  $A$ :  $\langle A \subseteq S \rangle$   $\langle \text{finite } A \rangle$   $\langle S' \subseteq \text{set } (qs \ t) \cup A \rangle$ 
      using  $1$   $2$  by  $(\text{meson } \text{Diff-subset-conv } \text{equalityD2 } \text{finite-Diff})$ 

    obtain  $B$  where  $B$ :  $\langle B \subseteq S \rangle$   $\langle \text{finite } B \rangle$   $\langle t \in F B \rangle$ 
      using  $**$   $t$   $\text{gamma-fin}$  by  $\text{meson}$ 

    let  $?S = \langle \text{set } ps \cup A \cup B \rangle$ 

    have  $\langle \text{finite } ?S \rangle$ 
      using  $A(2)$   $B(2)$  by  $\text{blast}$ 
    moreover have  $\langle ?S \subseteq S \rangle$ 
      using  $*$   $A(1)$   $B(1)$  by  $\text{simp}$ 
    ultimately have  $\langle ?S \in C \rangle$ 
      using  $\text{finc}$  by  $\text{simp}$ 

    moreover have  $\langle \text{set } ps \subseteq ?S \rangle$ 
      by  $\text{blast}$ 
    moreover have  $\langle t \in F ?S \rangle$ 
      using  $**$   $B(3)$   $\text{gamma-mono}$  by  $\text{blast}$ 
    ultimately have  $\langle \text{set } (qs \ t) \cup ?S \in C \rangle$ 
      using  $**$   $\text{gammaAC}$  by  $\text{blast}$ 

    moreover have  $\langle S' \subseteq \text{set } (qs \ t) \cup ?S \rangle$ 
      using  $A(3)$  by  $\text{blast}$ 
    ultimately show  $\langle S' \in C \rangle$ 
      using  $\text{sc}$  by  $\text{blast}$ 
  qed
qed
next
  fix  $C$   $S$ 

```

```

assume *: ⟨satE kind C⟩ ⟨S ∈ C⟩ ⟨maximal C S⟩
show ⟨satH kind S⟩
proof safe
  fix ps F qs t
  assume **: ⟨set ps ⊆ S⟩ ⟨ps ∼γ (F, qs)⟩
  then have ⟨∀ t ∈ F S. set (qs t) ∪ S ∈ C⟩
    using * by blast
  then have ⟨∀ t ∈ F S. set (qs t) ⊆ S⟩
    using ⟨maximal C S⟩ unfolding maximal-def by fast
  then show ⟨t ∈ F S ⇒ x ∈ set (qs t) ⇒ x ∈ S⟩ for x
    by blast
qed
qed

```

locale *Derivational-Gamma* = *Gamma map-tm map-fm params-fm is-param classify*

```

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⟩ and
  classify :: ⟨'fm list ⇒ ('fm set ⇒ 'tm set) × ('tm ⇒ 'fm list) ⇒ bool⟩ (infix
  ⟨∼γ⟩ 50) +
  fixes consistent :: ⟨'fm set ⇒ bool⟩ (⟨⊢ -⟩ [51] 50)
  assumes consistent: ⟨∧ S ps F qs t. set ps ⊆ S ⇒ ps ∼γ (F, qs) ⇒ t ∈ F S
  ⇒ ⊢ S ⇒ ⊢ set (qs t) ∪ S⟩

```

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

using *infinite-params-left consistent enough-new-def* **by** *unfold-locales blast+*

locale *Weak-Derivational-Gamma* = *Gamma map-tm map-fm params-fm is-param classify*

```

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⟩ and
  classify :: ⟨'fm list ⇒ ('fm set ⇒ 'tm set) × ('tm ⇒ 'fm list) ⇒ bool⟩ (infix
  ⟨∼γ⟩ 50) +
  fixes consistent :: ⟨'fm list ⇒ bool⟩ (⟨⊢ -⟩ [51] 50)
  assumes consistent: ⟨∧ A ps F qs t. set ps ⊆ set A ⇒ ps ∼γ (F, qs) ⇒ t ∈
  F (set A) ⇒ ⊢ A ⇒ ⊢ qs t @ A⟩

```

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

proof

```

show ⟨satE kind {S. ∃ A. set A = S ∧ ⊢ A}⟩
proof safe

```

```

fix ps qs A F t
assume *: ⟨set ps ⊆ set A⟩ ⟨ps ∼γ (F, qs)⟩ ⟨⊢ A⟩ ⟨t ∈ F (set A)⟩
then show ⟨∃ B. set B = set (qs t) ∪ set A ∧ ⊢ B⟩
using consistent[of ps A F qs t] by (meson set-append)
qed
qed

locale Gamma-UNIV = 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 ⇒ ('tm ⇒ 'fm list) ⇒ bool⟩ (infix ⟨∼γ'⟩ 50)
assumes gamma-map-UNIV: ⟨∧ps qs f. ps ∼γ' qs ⇒ ∃ rs. map (map-fm f) ps
  ∼γ' rs ∧
  (∀ t. map (map-fm f) (qs t) = rs (map-tm f t))⟩
begin

abbreviation (input) classify-UNIV where
  ⟨classify-UNIV ≡ λps (F, qs). (F = (λ-. UNIV)) ∧ ps ∼γ' qs⟩

```

end

```

sublocale Gamma-UNIV ⊆ Gamma map-tm map-fm params-fm is-param clas-
sify-UNIV
proof
show ⟨∧ps F qs f.
  (case (F, qs) of (F, qs) ⇒ F = (λ-. UNIV) ∧ ps ∼γ' qs) ⇒
  ∃ G rs.
  (case (G, rs) of (F, qs) ⇒ F = (λ-. UNIV) ∧ map (map-fm f) ps ∼γ'
qs) ∧
  (∀ S. map-tm f ' F S ⊆ G (map-fm f ' S)) ∧ (∀ t. map (map-fm f) (qs t)
= rs (map-tm f t))⟩
using gamma-map-UNIV image-subset-iff by fastforce
show ⟨∧ps F qs S S'. (case (F, qs) of (F, qs) ⇒ F = (λ-. UNIV) ∧ ps ∼γ' qs)
⇒ S ⊆ S' ⇒ F S ⊆ F S'⟩
by simp
show ⟨∧ps F qs t A. (case (F, qs) of (F, qs) ⇒ F = (λ-. UNIV) ∧ ps ∼γ' qs)
⇒ t ∈ F A ⇒ ∃ B ⊆ A. finite B ∧ t ∈ F B⟩
by blast
qed

```

1.11 Delta

```

locale Delta = 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 δ :: ⟨'fm ⇒ 'x ⇒ 'fm list⟩
  assumes delta-map: ⟨∧p f x. is-param x ⇒ is-subst f ⇒ δ (map-fm f p) (f x)
= map (map-fm f) (δ p x)⟩
begin

```

```

abbreviation ⟨kind ≡ Wits δ⟩
end

```

```

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

```

```

  fix C
  assume deltaC: ⟨satE kind C⟩
  then show ⟨satE kind (close C)⟩
  proof safe
    fix S p qs q
    assume ⟨S ∈ close C⟩
    then obtain S' where S': ⟨S' ∈ C⟩ and ⟨S ⊆ S'⟩
      unfolding close-def by blast

    assume ⟨p ∈ S⟩
    then have *: ⟨p ∈ S'⟩
      using ⟨S ⊆ S'⟩ by blast

    have ⟨∃x. is-param x ∧ set (δ p x) ∪ S' ∈ C⟩
      using deltaC S' * by blast
    then show ⟨∃x. is-param x ∧ set (δ p x) ∪ S ∈ close C⟩
      using ⟨S ⊆ S'⟩ by (metis subset-in-close)
  end

```

```

qed
next
fix C
assume deltaC: ⟨satE kind C⟩
then show ⟨satA kind (mk-alt-consistency C)⟩
proof safe
  fix S p qs x

```

```

  assume ⟨S ∈ mk-alt-consistency C⟩
  then obtain f where f: ⟨is-subst f⟩ ⟨map-fm f ' S ∈ C⟩
    unfolding mk-alt-consistency-def by blast

```

```

  let ?C = ⟨mk-alt-consistency C⟩
  let ?S = ⟨map-fm f ' S⟩

```

```

  assume ⟨p ∈ S⟩
  then have *: ⟨map-fm f p ∈ ?S⟩
    by auto

```

```

  assume x: ⟨x ∉ params S⟩ ⟨is-param x⟩

```

```

obtain  $y$  where  $y$ :  $\langle is-param\ y \rangle \langle set\ (\delta\ (map-fm\ f\ p)\ y) \cup ?S \in C \rangle$ 
  using  $deltaC\ f\ *$  by  $blast$ 

let  $?g = \langle f(x := y) \rangle$ 

have  $g$ :  $\langle is-subst\ ?g \rangle$ 
  using  $f\ x\ y$  unfolding  $is-subst-def$  by  $simp$ 

have  $\langle \forall x \in params\ S.\ f\ x = ?g\ x \rangle$ 
  using  $x$  by  $simp$ 
then have  $S$ :  $\langle \forall q \in S.\ map-fm\ ?g\ q = map-fm\ f\ q \rangle$ 
  using  $map-params-fm$  by  $simp$ 
then have  $\langle map-fm\ ?g\ p = map-fm\ f\ p \rangle$ 
  using  $\langle p \in S \rangle$  by  $auto$ 

moreover have  $\langle set\ (\delta\ (map-fm\ f\ p)\ (?g\ x)) \cup ?S \in C \rangle$ 
  using  $y$  by  $simp$ 
ultimately have  $\langle set\ (map\ (map-fm\ ?g)\ (\delta\ p\ x)) \cup ?S \in C \rangle$ 
  using  $delta-map\ x\ g$  by  $metis$ 
then have  $\langle \exists f.\ is-subst\ f \wedge set\ (map\ (map-fm\ f)\ (\delta\ p\ x)) \cup map-fm\ f\ `S \in$ 
 $C \rangle$ 
  using  $S\ g$  by  $(metis\ image-cong)$ 
then show  $\langle set\ (\delta\ p\ x) \cup S \in ?C \rangle$ 
  unfolding  $mk-alt-consistency-def$ 
  by  $(metis\ (mono-tags,\ lifting)\ image-Un\ image-set\ mem-Collect-eq)$ 
qed
next
fix  $C$ 
assume  $deltaAC$ :  $\langle sat_A\ kind\ C \rangle$  and  $closedC$ :  $\langle subset-closed\ C \rangle$ 
then show  $\langle sat_A\ kind\ (mk-finite-char\ C) \rangle$ 
proof safe
  fix  $S\ p\ qs\ x$ 
  assume  $\langle S \in mk-finite-char\ C \rangle$ 
  then have  $finc$ :  $\langle \forall S' \subseteq S.\ finite\ S' \longrightarrow S' \in C \rangle$ 
    unfolding  $mk-finite-char-def$  by  $blast$ 

  have  $sc$ :  $\langle \forall S' \in C.\ \forall S \subseteq S'.\ S \in C \rangle$ 
    using  $closedC$  unfolding  $subset-closed-def$  by  $blast$ 
  then have  $sc'$ :  $\langle \bigwedge S' x.\ x \cup S' \in C \implies \forall S \subseteq x \cup S'.\ S \in C \rangle$ 
    by  $blast$ 

  assume  $*$ :  $\langle p \in S \rangle$  and  $x$ :  $\langle x \notin params\ S \rangle \langle is-param\ x \rangle$ 
  show  $\langle set\ (\delta\ p\ x) \cup S \in mk-finite-char\ C \rangle$ 
    unfolding  $mk-finite-char-def$ 
  proof safe
    fix  $S'$ 
    let  $?S' = \langle \{p\} \cup (S' - set\ (\delta\ p\ x)) \rangle$ 

    assume  $\langle S' \subseteq set\ (\delta\ p\ x) \cup S \rangle$  and  $\langle finite\ S' \rangle$ 

```

```

then have ⟨ $?S' \subseteq S$ ⟩
  using * by fast
moreover have ⟨finite  $?S'$ ⟩
  using ⟨finite  $S'$ ⟩ by blast
ultimately have ⟨ $?S' \in C$ ⟩
  using fin by blast
moreover have ⟨ $\forall a \in ?S'. x \notin \text{params-fm } a$ ⟩
  using  $x$  ⟨ $?S' \subseteq S$ ⟩ by blast
ultimately have ⟨ $\text{set } (\delta \text{ } p \text{ } x) \cup ?S' \in C$ ⟩
  using  $x$  deltaAC by blast
then show ⟨ $S' \in C$ ⟩
  using sc by fast
qed
qed
next
  show ⟨ $\bigwedge C \ S. \text{sat}_E \text{ kind } C \implies S \in C \implies \text{maximal } C \ S \implies \text{sat}_H \text{ kind } S$ ⟩
  using satH-Wits .
qed

locale Derivational-Delta = Delta map-fm params-fm is-param  $\delta$ 
  for
    map-fm :: ⟨ $'x \Rightarrow 'x \Rightarrow 'fm \Rightarrow 'fm$ ⟩ and
    params-fm :: ⟨ $'fm \Rightarrow 'x \text{ set}$ ⟩ and
    is-param :: ⟨ $'x \Rightarrow \text{bool}$ ⟩ and
     $\delta$  :: ⟨ $'fm \Rightarrow 'x \Rightarrow 'fm \text{ list}$ ⟩ +
  fixes consistent :: ⟨ $'fm \text{ set} \Rightarrow \text{bool}$ ⟩ (⟨ $\vdash \rightarrow$  [51] 50)
  assumes consistent: ⟨ $\bigwedge S \ p \ x. p \in S \implies \text{is-param } x \implies x \notin \text{params } S \implies \vdash S \implies \vdash \text{set } (\delta \text{ } p \text{ } x) \cup S$ ⟩

sublocale Derivational-Delta  $\subseteq$  Derivational-Kind map-fm params-fm is-param
  kind consistent
proof
  assume inf: ⟨infinite (UNIV ::  $'fm \text{ set}$ )⟩
  show ⟨satE kind {A. enough-new  $A \wedge \vdash A$ }⟩
  proof safe
    fix  $S \ p$ 
    assume *: ⟨ $p \in S$ ⟩ ⟨enough-new  $S$ ⟩ ⟨ $\vdash S$ ⟩
    then have ⟨infinite (Collect is-param - (params ( $\{p\} \cup S$ )))⟩
      unfolding enough-new-def using card-of-ordLeq-finite inf by auto
    then obtain  $x$  where ⟨is-param  $x$ ⟩ ⟨ $x \notin \text{params } (\{p\} \cup S)$ ⟩
      using infinite-imp-nonempty by blast
    then have ⟨ $\exists x. \text{is-param } x \wedge \vdash \text{set } (\delta \text{ } p \text{ } x) \cup S$ ⟩
      using *(1,3) consistent ⟨ $\vdash S$ ⟩ by fast
    then show ⟨ $\exists x. \text{is-param } x \wedge \text{set } (\delta \text{ } p \text{ } x) \cup S \in \{A. \text{enough-new } A \wedge \vdash A\}$ ⟩
      using * inf infinite-params-left unfolding enough-new-def by blast
  qed
qed

locale Weak-Derivational-Delta = Delta map-fm params-fm is-param  $\delta$ 

```

```

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

```

assume inf: ⟨infinite (Collect is-param :: 'x set)⟩
show ⟨satE kind {S. ∃ A. set A = S ∧ ⊢ A}⟩
proof safe
  fix p A
  assume *: ⟨p ∈ set A⟩ ⟨⊢ A⟩
  then have ⟨infinite (Collect is-param - (params (set (p # A))))⟩
    using inf finite-compl by fastforce
  then obtain x where ⟨is-param x⟩ ⟨x ∉ params (set (p # A))⟩
    using infinite-imp-nonempty by blast
  then have ⟨∃ x. is-param x ∧ ⊢ δ p x @ A⟩
    using * consistent[of p A x] by auto
  then show ⟨∃ x. is-param x ∧ set (δ p x) ∪ set A ∈ {S. ∃ A. set A = S ∧ ⊢
A}⟩
    by (metis (mono-tags, lifting) CollectI set-append)
  qed
qed

```

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 ⟨∼□⟩
50)
  and hint :: ⟨'fm set ⇒ bool⟩
assumes modal-map: ⟨∧ ps F qs f. ps ∼□ (F, qs) ⇒ ∃ G. map (map-fm f) ps
∼□ (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 ∼□ (F, qs) ⇒ S ⊆ S' ⇒ F S ⊆ F S'⟩
  and modal-fin: ⟨∧ ps F qs S A. ps ∼□ (F, qs) ⇒ finite A ⇒ A ⊆ F S ⇒
∃ S' ⊆ S. finite S' ∧ A ⊆ F S'⟩
begin

```

inductive cond where
cond [intro!]: $\langle ps \rightsquigarrow_{\square} (F, qs) \implies cond\ ps (\lambda C\ S.\ set\ qs \cup F\ S \in C) \rangle$

inductive-cases *condE[elim!]:* $\langle cond\ ps\ Q \rangle$

abbreviation *kind* :: $\langle ('x, 'fm)\ kind \rangle$ **where**
kind \equiv *Cond cond hint*

end

locale *ModalH = Modal map-fm params-fm is-param classify hint*
for
map-fm :: $\langle ('x \Rightarrow 'x) \Rightarrow 'fm \Rightarrow 'fm \rangle$ **and**
params-fm :: $\langle 'fm \Rightarrow 'x\ set \rangle$ **and**
is-param :: $\langle 'x \Rightarrow bool \rangle$ **and**
classify :: $\langle 'fm\ list \Rightarrow ('fm\ set \Rightarrow 'fm\ set) \times 'fm\ list \Rightarrow bool \rangle$ (**infix** $\langle \rightsquigarrow_{\square} \rangle$ 50)
and
hint :: $\langle 'fm\ set \Rightarrow bool \rangle$ +
assumes *modal-hintikka:* $\langle \bigwedge C\ S.\ sat_E\ kind\ C \implies S \in C \implies maximal\ C\ S \implies sat_H\ kind\ S \rangle$

sublocale *ModalH* \subseteq *Consistency-Kind map-fm params-fm is-param kind*
proof
fix *C*
assume *modalC:* $\langle sat_E\ kind\ C \rangle$
then show $\langle sat_E\ kind\ (close\ C) \rangle$
proof safe
fix *S ps F qs*
assume $\langle S \in close\ C \rangle$
then obtain *S'* **where** *S'*: $\langle S' \in C \rangle$ **and** $\langle S \subseteq S' \rangle$
unfolding *close-def* **by** *blast*

assume $\langle set\ ps \subseteq S \rangle$
then have *: $\langle set\ ps \subseteq S' \rangle$
using $\langle S \subseteq S' \rangle$ **by** *blast*

assume **: $\langle ps \rightsquigarrow_{\square} (F, qs) \rangle$
then have $\langle set\ qs \cup F\ S' \in C \rangle$
using *modalC S' * *** **by** *blast*
then show $\langle set\ qs \cup F\ S \in close\ C \rangle$
using $\langle S \subseteq S' \rangle$ *subset-in-close ** modal-mono* **by** *metis*

qed

next
fix *C*
assume *modalC:* $\langle sat_E\ kind\ C \rangle$ **and** *closedC:* $\langle subset-closed\ C \rangle$
then show $\langle sat_A\ kind\ (mk-alt-consistency\ C) \rangle$
proof safe
fix *S ps F qs*

```

assume  $\langle S \in \text{mk-alt-consistency } C \rangle$ 
then obtain  $f$  where  $f$ :  $\langle \text{is-subst } f \rangle \langle \text{map-fm } f \text{ ' } S \in C \rangle$ 
  unfolding  $\text{mk-alt-consistency-def}$  by  $\text{blast}$ 

let  $?C = \langle \text{mk-alt-consistency } C \rangle$ 
let  $?S = \langle \text{map-fm } f \text{ ' } S \rangle$ 

assume  $\langle \text{set } ps \subseteq S \rangle$ 
then have  $*$ :  $\langle \text{set } (\text{map } (\text{map-fm } f) ps) \subseteq ?S \rangle$ 
  by  $\text{auto}$ 

assume  $**$ :  $\langle ps \rightsquigarrow_{\square} (F, qs) \rangle$ 
obtain  $G$  where  $G$ :
   $\langle \text{map } (\text{map-fm } f) ps \rightsquigarrow_{\square} (G, \text{map } (\text{map-fm } f) qs) \rangle$ 
   $\langle \forall S. \text{map-fm } f \text{ ' } F S \subseteq G (\text{map-fm } f \text{ ' } S) \rangle$ 
  using  $\text{modal-map } **$  by  $\text{meson}$ 
then have  $\langle \text{set } (\text{map } (\text{map-fm } f) qs) \cup G ?S \in C \rangle$ 
  using  $\text{modalC } f *$  by  $\text{blast}$ 
then have  $\langle \text{set } (\text{map } (\text{map-fm } f) qs) \cup \text{map-fm } f \text{ ' } F S \in C \rangle$ 
  using  $G \text{ closedC}$  unfolding  $\text{subset-closed-def}$  by  $(\text{meson } \text{Un-mono } \text{order-refl})$ 
then show  $\langle \text{set } qs \cup F S \in ?C \rangle$ 
  unfolding  $\text{mk-alt-consistency-def}$  using  $f$ 
  by  $(\text{auto } \text{split: } \text{option.splits } \text{simp: } \text{image-Un})$ 
qed
next
fix  $C$ 
assume  $\text{modalAC}$ :  $\langle \text{sat}_A \text{ kind } C \rangle$  and  $\text{closedC}$ :  $\langle \text{subset-closed } C \rangle$ 
then show  $\langle \text{sat}_A \text{ kind } (\text{mk-finite-char } C) \rangle$ 
proof  $\text{safe}$ 
  fix  $S ps F qs t$ 
  assume  $S$ :  $\langle S \in \text{mk-finite-char } C \rangle$ 
  then have  $\text{finc}$ :  $\langle \forall S' \subseteq S. \text{finite } S' \longrightarrow S' \in C \rangle$ 
    unfolding  $\text{mk-finite-char-def}$  by  $\text{blast}$ 

  have  $\text{sc}$ :  $\langle \forall S' \in C. \forall S \subseteq S'. S \in C \rangle$ 
    using  $\text{closedC}$  unfolding  $\text{subset-closed-def}$  by  $\text{blast}$ 
  then have  $\text{sc'}$ :  $\langle \bigwedge S' x. x \cup S' \in C \implies \forall S \subseteq x \cup S'. S \in C \rangle$ 
    by  $\text{blast}$ 

  assume  $*$ :  $\langle \text{set } ps \subseteq S \rangle$  and  $**$ :  $\langle ps \rightsquigarrow_{\square} (F, qs) \rangle$ 

  show  $\langle \text{set } qs \cup F S \in \text{mk-finite-char } C \rangle$ 
    unfolding  $\text{mk-finite-char-def}$ 
  proof  $\text{safe}$ 
    fix  $S'$ 
    assume  $1$ :  $\langle S' \subseteq \text{set } qs \cup F S \rangle$  and  $2$ :  $\langle \text{finite } S' \rangle$ 

    obtain  $A$  where  $A$ :  $\langle A \subseteq S \rangle \langle \text{finite } A \rangle \langle S' \subseteq \text{set } qs \cup F A \rangle$ 

```

```

using 1 2 ** modal-fin by (meson Diff-subset-conv finite-Diff)

let ?S = ⟨set ps ∪ A⟩

have ⟨finite ?S⟩
  using A(2) by blast
moreover have ⟨?S ⊆ S⟩
  using * A(1) by simp
ultimately have ⟨?S ∈ C⟩
  using fnc by simp

moreover have ⟨set ps ⊆ ?S⟩
  by blast
ultimately have ⟨set qs ∪ F ?S ∈ C⟩
  using ** modalAC by blast

moreover have ⟨S' ⊆ set qs ∪ F ?S⟩
  using A(3) ** modal-mono by (meson Diff-subset-conv inf-sup-ord(4)
subset-trans)
ultimately show ⟨S' ∈ C⟩
  using sc by blast
qed
qed
next
fix C S
assume *: ⟨satE kind C⟩ ⟨S ∈ C⟩ ⟨maximal C S⟩
then show ⟨satH kind S⟩
  using modal-hintikka by simp
qed

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 ⟨~□⟩ 50)
+
  fixes consistent :: ⟨'fm set ⇒ bool⟩ (⟨!- -⟩ [51] 50)
  assumes consistent: ⟨∧ S ps F qs. set ps ⊆ S ⇒ ps ~□ (F, qs) ⇒ ⊢ S ⇒ ⊢
set qs ∪ F S⟩
  and params-subset: ⟨∧ ps F qs S. ps ~□ (F, qs) ⇒ params (F S) ⊆ params
S⟩

sublocale Derivational-Modal ⊆ Derivational-Kind map-fm params-fm is-param
kind consistent
proof
assume inf: ⟨infinite (UNIV :: 'fm set)⟩
then show ⟨satE kind {A. enough-new A ∧ ⊢ A}⟩
proof safe

```

```

fix  $S$   $ps$   $F$   $qs$ 
assume  $\langle ps \rightsquigarrow_{\square} (F, qs) \rangle \langle enough\text{-}new\ S \rangle$ 
then have  $\langle enough\text{-}new (F\ S) \rangle$ 
  unfolding  $enough\text{-}new\text{-}def$  using  $params\text{-}subset[of\ ps\ F\ qs\ S]$   $ordLeq\text{-}transitive$ 
     $card\text{-}of\text{-}mono1[of\ Collect\ is\text{-}param - params\ S\ Collect\ is\text{-}param - params$ 
       $(F\ S)]$ 
    by  $blast$ 
  then show  $\langle enough\text{-}new (set\ qs \cup F\ S) \rangle$ 
    using  $infinite\text{-}params\text{-}left[OF\ inf]$  unfolding  $enough\text{-}new\text{-}def$  by  $blast$ 
qed ( $use\ consistent\ in\ blast$ )
qed

```

```

locale  $Weak\text{-}Derivational\text{-}Modal = ModalH\ map\text{-}fm\ params\text{-}fm\ is\text{-}param\ classify$ 
for
   $map\text{-}fm :: \langle 'x \Rightarrow 'x \rangle \Rightarrow 'fm \Rightarrow 'fm$  and
   $params\text{-}fm :: \langle 'fm \Rightarrow 'x\ set \rangle$  and
   $is\text{-}param :: \langle 'x \Rightarrow bool \rangle$  and
   $classify :: \langle 'fm\ list \Rightarrow ('fm\ set \Rightarrow 'fm\ set) \times 'fm\ list \Rightarrow bool \rangle$  (infix  $\langle \rightsquigarrow_{\square} \rangle 50$ )
+
  fixes  $consistent :: \langle 'fm\ list \Rightarrow bool \rangle$  ( $\langle \vdash - \rangle [51] 50$ )
  assumes  $consistent: \langle \bigwedge S\ ps\ F\ qs\ S'. set\ ps \subseteq set\ S \Longrightarrow ps \rightsquigarrow_{\square} (F, qs) \Longrightarrow \vdash S$ 
     $\Longrightarrow set\ S' = F (set\ S) \Longrightarrow \vdash qs @ S' \rangle$ 
  and  $params\text{-}subset: \langle \bigwedge ps\ F\ qs\ S. ps \rightsquigarrow_{\square} (F, qs) \Longrightarrow params (F\ S) \subseteq params$ 
     $S \rangle$ 
  and  $F\text{-}size: \langle \bigwedge ps\ F\ qs\ S. ps \rightsquigarrow_{\square} (F, qs) \Longrightarrow |F\ S| \leq_o |S| \rangle$ 

```

```

sublocale  $Weak\text{-}Derivational\text{-}Modal \subseteq Weak\text{-}Derivational\text{-}Kind\ map\text{-}fm\ params\text{-}fm$ 
 $is\text{-}param\ kind\ consistent$ 
proof
  assume  $inf: \langle infinite (Collect\ is\text{-}param :: 'x\ set) \rangle$ 
  show  $\langle sat_E\ kind\ \{S. \exists A. set\ A = S \wedge \vdash A\} \rangle$ 
  proof safe
    fix  $ps\ qs\ A\ F$ 
    assume  $\langle set\ ps \subseteq set\ A \rangle \langle ps \rightsquigarrow_{\square} (F, qs) \rangle \langle \vdash A \rangle$ 
    then show  $\langle \exists B. set\ B = set\ qs \cup F (set\ A) \wedge \vdash B \rangle$ 
      using  $consistent[of\ ps\ A\ F\ qs]$   $F\text{-}size$ 
      by ( $metis\ card\text{-}of\text{-}ordLeq\text{-}infinite\ finite\text{-}list\ list.set\text{-}finite\ set\text{-}append$ )
  qed
qed
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* :: $\langle 'a, 'f, 'p \rangle \text{ model} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{wf-model } (\text{Model } U \ E \ F \ G) \longleftrightarrow (\forall n. E \ n \in U) \wedge (\forall f \ ts. F \ f \ ts \in U) \rangle$

fun *semantics-tm* :: $\langle (\text{nat} \Rightarrow 'a) \times ('f \Rightarrow 'a \ \text{list} \Rightarrow 'a) \Rightarrow 'f \ \text{tm} \Rightarrow 'a \rangle$ ($\langle \langle - \rangle \rangle$)
where
 $\langle \langle (E, -) \rangle \ (\#n) = E \ n \rangle$
 $\mid \langle \langle (E, F) \rangle \ (\text{Of } ts) = F \ f \ (\text{map } \langle (E, F) \rangle \ ts) \rangle$

primrec *add-env* :: $\langle 'a \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow \text{nat} \Rightarrow 'a \rangle$ (**infix** $\langle \gg \rangle$ 0) **where**
 $\langle (t \gg s) \ 0 = t \rangle$
 $\mid \langle (t \gg s) \ (\text{Suc } n) = s \ n \rangle$

fun *semantics-fm* :: $\langle 'a, 'f, 'p \rangle \text{ model} \Rightarrow ('f, 'p) \ \text{fm} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \models \rangle$ 50) **where**
 $\langle (- \models \perp) = \text{False} \rangle$
 $\mid \langle (\text{Model } - \ E \ F \ G \models \cdot P \ ts) = G \ P \ (\text{map } \langle (E, F) \rangle \ ts) \rangle$
 $\mid \langle (\text{Model } U \ E \ F \ G \models p \longrightarrow q) = (\text{Model } U \ E \ F \ G \models p \longrightarrow \text{Model } U \ E \ F \ G \models q) \rangle$
 $\mid \langle (\text{Model } U \ E \ F \ G \models \forall p) = (\forall x \in U. \text{Model } U \ (x \gg E) \ F \ G \models p) \rangle$

2.3 Operations

primrec *lift-tm* :: $\langle 'f \ \text{tm} \Rightarrow 'f \ \text{tm} \rangle$ **where**
 $\langle \text{lift-tm } (\#n) = \#(n+1) \rangle$
 $\mid \langle \text{lift-tm } (\text{Of } ts) = \text{Of } (\text{map } \text{lift-tm } ts) \rangle$

primrec *sub-tm* :: $\langle (\text{nat} \Rightarrow 'f \ \text{tm}) \Rightarrow 'f \ \text{tm} \Rightarrow 'f \ \text{tm} \rangle$ **where**
 $\langle \text{sub-tm } s \ (\#n) = s \ n \rangle$
 $\mid \langle \text{sub-tm } s \ (\text{Of } ts) = \text{Of } (\text{map } (\text{sub-tm } s) \ ts) \rangle$

primrec *sub-fm* :: $\langle (\text{nat} \Rightarrow 'f \ \text{tm}) \Rightarrow ('f, 'p) \ \text{fm} \Rightarrow ('f, 'p) \ \text{fm} \rangle$ **where**
 $\langle \text{sub-fm } - \ \perp = \perp \rangle$
 $\mid \langle \text{sub-fm } s \ (\cdot P \ ts) = \cdot P \ (\text{map } (\text{sub-tm } s) \ ts) \rangle$
 $\mid \langle \text{sub-fm } s \ (p \longrightarrow q) = \text{sub-fm } s \ p \longrightarrow \text{sub-fm } s \ q \rangle$
 $\mid \langle \text{sub-fm } s \ (\forall p) = \forall (\text{sub-fm } (\#0 \gg \lambda n. \text{lift-tm } (s \ n)) \ p) \rangle$

abbreviation *inst-single* :: $\langle 'f \ \text{tm} \Rightarrow ('f, 'p) \ \text{fm} \Rightarrow ('f, 'p) \ \text{fm} \rangle$ ($\langle \langle - \rangle \rangle$) **where**
 $\langle \langle t \rangle \equiv \text{sub-fm } (t \gg \#) \rangle$

abbreviation *psub* :: $\langle ('f \Rightarrow 'g) \Rightarrow ('f, 'p) \ \text{fm} \Rightarrow ('g, 'p) \ \text{fm} \rangle$ **where**
 $\langle \text{psub } f \equiv \text{map-fm } f \ \text{id} \rangle$

primrec *size-fm* :: $\langle ('f, 'p) \ \text{fm} \Rightarrow \text{nat} \rangle$ **where**
 $\langle \text{size-fm } \perp = 1 \rangle$
 $\mid \langle \text{size-fm } (\cdot -) = 1 \rangle$
 $\mid \langle \text{size-fm } (p \longrightarrow q) = 1 + \text{size-fm } p + \text{size-fm } q \rangle$
 $\mid \langle \text{size-fm } (\forall p) = 1 + \text{size-fm } p \rangle$

2.3.1 Lemmas

lemma *wf-model-tm* [*simp*]: $\langle \text{wf-model } (Model\ U\ E\ F\ G) \implies \llbracket (E, F) \rrbracket t \in U \rangle$
by (*induct t*) *simp-all*

lemma *size-sub-fm* [*simp*]: $\langle \text{size-fm } (sub\text{-fm } s\ p) = \text{size-fm } p \rangle$
by (*induct p arbitrary: s*) *simp-all*

lemma *upd-params-tm* [*simp*]: $\langle f \notin \text{params-tm } t \implies \llbracket (E, F(f := x)) \rrbracket t = \llbracket (E, F) \rrbracket t \rangle$
by (*induct t*) (*auto cong: map-cong*)

lemma *upd-params-fm* [*simp*]:
assumes $\langle f \notin \text{params-fm } p \rangle$
shows $\langle Model\ U\ E\ (F(f := x))\ G \models p \iff Model\ U\ E\ F\ G \models p \rangle$
using *assms* **by** (*induct p arbitrary: E*) (*auto cong: map-cong*)

lemma *finite-params-tm* [*simp*]: $\langle \text{finite } (\text{params-tm } t) \rangle$
by (*induct t*) *simp-all*

lemma *finite-params-fm'* [*simp*]: $\langle \text{finite } (\text{params-fm } p) \rangle$
by (*induct p*) *simp-all*

lemma *env* [*simp*]: $\langle P ((x \gg E)\ n) = (P\ x \gg \lambda n. P\ (E\ n))\ n \rangle$
by (*induct n*) *simp-all*

lemma *lift-lemma*: $\langle \llbracket (x \gg E, F) \rrbracket (\text{lift-tm } t) = \llbracket (E, F) \rrbracket t \rangle$
by (*induct t*) (*auto cong: map-cong*)

lemma *sub-tm-antics*: $\langle \llbracket (E, F) \rrbracket (\text{sub-tm } s\ t) = \llbracket (\lambda n. \llbracket (E, F) \rrbracket (s\ n), F) \rrbracket t \rangle$
by (*induct t*) (*auto cong: map-cong*)

lemma *sub-fm-antics* [*simp*]:
 $\langle Model\ U\ E\ F\ G \models \text{sub-fm } s\ p \iff Model\ U\ (\lambda n. \llbracket (E, F) \rrbracket (s\ n))\ F\ G \models p \rangle$
by (*induct p arbitrary: E s*) (*auto cong: map-cong simp: sub-tm-antics lift-lemma*)

lemma *map-tm-sub-tm* [*simp*]: $\langle \text{map-tm } f\ (\text{sub-tm } g\ t) = \text{sub-tm } (\text{map-tm } f\ o\ g)\ (\text{map-tm } f\ t) \rangle$
by (*induct t*) *simp-all*

lemma *map-tm-lift-tm* [*simp*]: $\langle \text{map-tm } f\ (\text{lift-tm } t) = \text{lift-tm } (\text{map-tm } f\ t) \rangle$
by (*induct t*) *simp-all*

lemma *psub-sub-fm*: $\langle \text{psub } f\ (\text{sub-fm } g\ p) = \text{sub-fm } (\text{map-tm } f\ o\ g)\ (\text{psub } f\ p) \rangle$
by (*induct p arbitrary: g*) (*simp-all add: comp-def*)

lemma *map-tm-inst-single*: $\langle (\text{map-tm } f\ o\ (u \gg \#))\ t = (\text{map-tm } f\ u \gg \#)\ t \rangle$
by (*induct t*) *auto*

lemma *psub-inst-single* [*simp*]: $\langle \text{psub } f\ (\langle t \rangle p) = \langle \text{map-tm } f\ t \rangle (\text{psub } f\ p) \rangle$

unfolding *psub-sub-fm map-tm-inst-single ..*

2.4 Terms

primrec *terms-tm* :: $\langle 'f\ tm \Rightarrow 'f\ tm\ set \rangle$ **where**
 $\langle terms\text{-}tm\ (\#n) = \{\#n\} \rangle$
 $| \langle terms\text{-}tm\ (\bigcirc f\ ts) = \{\bigcirc f\ ts\} \cup \bigcup (set\ (map\ terms\text{-}tm\ ts)) \rangle$

primrec *terms-fm* :: $\langle ('f, 'p)\ fm \Rightarrow 'f\ tm\ set \rangle$ **where**
 $\langle terms\text{-}fm\ \perp = \{\} \rangle$
 $| \langle terms\text{-}fm\ (\cdot\text{-}\ ts) = \bigcup (set\ (map\ terms\text{-}tm\ ts)) \rangle$
 $| \langle terms\text{-}fm\ (p \longrightarrow q) = terms\text{-}fm\ p \cup terms\text{-}fm\ q \rangle$
 $| \langle terms\text{-}fm\ (\forall p) = terms\text{-}fm\ p \rangle$

definition *terms* :: $\langle ('f, 'p)\ fm\ set \Rightarrow 'f\ tm\ set \rangle$ **where**
 $\langle terms\ S \equiv \bigcup p \in S. terms\text{-}fm\ p \rangle$

2.4.1 Lemmas

lemma *terms-mono*: $\langle S \subseteq S' \Longrightarrow terms\ S \subseteq terms\ S' \rangle$
unfolding *terms-def* **by** *blast*

lemma *terms-tm-refl* [*intro*]: $\langle t \in terms\text{-}tm\ t \rangle$
by (*induct t*) *simp-all*

lemma *terms-tm-trans* [*trans*]: $\langle s \in terms\text{-}tm\ t \Longrightarrow r \in terms\text{-}tm\ s \Longrightarrow r \in terms\text{-}tm\ t \rangle$
by (*induct t*) *auto*

lemma *map-terms-tm* [*simp*]: $\langle terms\text{-}tm\ (map\text{-}tm\ f\ t) = map\text{-}tm\ f\ ' terms\text{-}tm\ t \rangle$
by (*induct t*) *auto*

lemma *map-terms-fm* [*simp*]: $\langle terms\text{-}fm\ (map\text{-}fm\ f\ g\ p) = map\text{-}tm\ f\ ' terms\text{-}fm\ p \rangle$
by (*induct p*) *auto*

lemma *terms-tm-closed* [*dest*]: $\langle t \in terms\text{-}tm\ s \Longrightarrow terms\text{-}tm\ t \subseteq terms\text{-}tm\ s \rangle$
using *terms-tm-trans* **by** *blast*

lemma *terms-fm-closed*: $\langle t \in terms\text{-}fm\ p \Longrightarrow terms\text{-}tm\ t \subseteq terms\text{-}fm\ p \rangle$
by (*induct p*) *auto*

lemma *terms-source*: $\langle t \in terms\ S \Longrightarrow \exists S' \subseteq S. finite\ S' \wedge t \in terms\ S' \rangle$
unfolding *terms-def* **using** *insert-subset* **by** *blast*

lemma *terms-tm-Fun*: $\langle \bigcirc f\ ts \in terms\text{-}tm\ s \Longrightarrow t \in set\ ts \Longrightarrow t \in terms\text{-}tm\ s \rangle$
using *terms-tm-refl* *terms-tm-trans*
by (*metis UN-I UnI2 list.set-map terms-tm.simps(2)*)

lemma *terms-Fun*: $\langle \bigcirc f\ ts \in terms\ S \Longrightarrow set\ ts \subseteq terms\ S \rangle$

using *terms-tm-Fun terms-tm-refl terms-fm-closed*
 unfolding *terms-def* by *fast*

2.5 Guard

definition *guard* :: $\langle 'a \Rightarrow 'a \text{ set} \Rightarrow 'a \rangle$ (**infix** $\langle \in? \rangle$ 50) **where**
 $\langle x \in? S \equiv \text{if } x \in S \text{ then } x \text{ else } \text{SOME } y. y \in S \rangle$

lemma *guard-in*: $\langle S \neq \{\} \implies (x \in? S) \in S \rangle$
 unfolding *guard-def* using *some-in-eq* by *auto*

lemma *guard-refl* [*simp*]: $\langle t \in S \implies t \in? S = t \rangle$
 unfolding *guard-def* by *simp*

2.6 Model Existence

inductive

confl-class :: $\langle ('f, 'p) \text{ fm list} \Rightarrow ('f, 'p) \text{ fm list} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\mathbf{x}} \rangle$ 50) **and**
alpha-class :: $\langle ('f, 'p) \text{ fm list} \Rightarrow ('f, 'p) \text{ fm list} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\alpha} \rangle$ 50) **and**
beta-class :: $\langle ('f, 'p) \text{ fm list} \Rightarrow ('f, 'p) \text{ fm list} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\beta} \rangle$ 50) **and**
gamma-class :: $\langle ('f, 'p) \text{ fm list} \Rightarrow ((f, 'p) \text{ fm set} \Rightarrow -) \times (f \text{ tm} \Rightarrow -) \Rightarrow \text{bool} \rangle$
 (**infix** $\langle \rightsquigarrow_{\gamma} \rangle$ 50)

where

CFIs [*intro*]: $\langle [\perp] \rightsquigarrow_{\mathbf{x}} [\perp] \rangle$
 | *CNeg* [*intro*]: $\langle [\neg (\cdot P \text{ ts})] \rightsquigarrow_{\mathbf{x}} [\cdot P \text{ ts}] \rangle$
 | *CImpN* [*intro*]: $\langle [\neg (p \longrightarrow q)] \rightsquigarrow_{\alpha} [p, \neg q] \rangle$
 | *CImpP* [*intro*]: $\langle [p \longrightarrow q] \rightsquigarrow_{\beta} [\neg p, q] \rangle$
 | *CAllP* [*intro*]: $\langle [\forall p] \rightsquigarrow_{\gamma} (\text{terms}, \lambda t. [\langle t \rangle p]) \rangle$

fun δ :: $\langle ('f, 'p) \text{ fm} \Rightarrow 'f \Rightarrow ('f, 'p) \text{ fm list} \rangle$ **where**
 $\langle \delta (\neg \forall p) x = [\neg \langle \star x \rangle p] \rangle$
 | $\langle \delta _ _ = [] \rangle$

interpretation *P*: *Params* *psub* *params-fm* $\langle \lambda _ . \text{True} \rangle$
 by *unfold-locales* (*auto simp*: *fm.map-id0 cong*: *fm.map-cong0*)

interpretation *C*: *Confl* *psub* *params-fm* $\langle \lambda _ . \text{True} \rangle$ *confl-class*
 by *unfold-locales* (*auto elim!*: *confl-class.cases*)

interpretation *A*: *Alpha* *psub* *params-fm* $\langle \lambda _ . \text{True} \rangle$ *alpha-class*
 by *unfold-locales* (*auto elim!*: *alpha-class.cases*)

interpretation *B*: *Beta* *psub* *params-fm* $\langle \lambda _ . \text{True} \rangle$ *beta-class*
 by *unfold-locales* (*auto elim!*: *beta-class.cases*)

interpretation *G*: *Gamma* *map-tm* *psub* *params-fm* $\langle \lambda _ . \text{True} \rangle$ *gamma-class*

proof

show $\langle \bigwedge ps F qs t A. ps \rightsquigarrow_{\gamma} (F, qs) \implies t \in F A \implies \exists B \subseteq A. \text{finite } B \wedge t \in F$

B
 by (*elim gamma-class.cases*) (*auto simp: terms-source*)
qed (*fastforce simp: terms-def elim: gamma-class.cases*)+

interpretation D : *Delta psub params-fm* $\langle \lambda-. True \rangle \delta$
proof
 show $\langle \bigwedge f. \delta (psub f p) (f x) = map (psub f) (\delta p x) \rangle$ **for** $p :: \langle ('f, 'p) fm \rangle$ **and** x
 by (*induct p x rule: $\delta.induct$*) *simp-all*
qed

abbreviation $Kinds :: \langle ('f, ('f, 'p) fm) kind list \rangle$ **where**
 $\langle Kinds \equiv [C.kind, A.kind, B.kind, G.kind, D.kind] \rangle$

lemma *prop_E-Kinds* [*intro*]:
 assumes $\langle P.sat_E C.kind C \rangle \langle P.sat_E A.kind C \rangle \langle P.sat_E B.kind C \rangle \langle P.sat_E G.kind C \rangle \langle P.sat_E D.kind C \rangle$
 shows $\langle P.prop_E Kinds C \rangle$
 unfolding *P.prop_E-def* **using** *assms* **by** *simp*

interpretation *Consistency-Kinds* *psub params-fm* $\langle \lambda-. True \rangle Kinds$
using *P.Params-axioms C.Consistency-Kind-axioms A.Consistency-Kind-axioms B.Consistency-Kind-axioms G.Consistency-Kind-axioms D.Consistency-Kind-axioms*
by (*auto intro: Consistency-Kinds.intro simp: Consistency-Kinds-axioms-def*)

interpretation *Maximal-Consistency* *psub params-fm* $\langle \lambda-. True \rangle Kinds$
proof
 show $\langle infinite (UNIV :: ('f, 'p) fm set) \rangle$
using *infinite-UNIV-size*[of $\langle \lambda p. p \longrightarrow p \rangle$] **by** *simp*
qed *simp*

abbreviation *canonical* :: $\langle ('f, 'p) fm set \Rightarrow ('f tm, 'f, 'p) model \rangle$ **where**
 $\langle canonical H \equiv Model (terms H) (\lambda n. \#n \in ? terms H) (\lambda f ts. \bigcirc f ts \in ? terms H) (\lambda P ts. \bullet P ts \in H) \rangle$

lemma *wf-canonical*:
 assumes $\langle terms H \neq \{\} \rangle$
 shows $\langle wf-model (canonical H) \rangle$
using *assms guard-in* **by** (*metis (no-types, lifting) wf-model.simps*)

lemma *canonical-tm-id* [*simp*]:
 $\langle t \in terms H \Longrightarrow \langle \langle (\lambda n. \#n \in ? terms H, \lambda P ts. \bigcirc P ts \in ? terms H) \rangle t = t \rangle$
proof (*induct t*)
 case (*Fun f ts*)
moreover from this have $\langle t \in set ts \Longrightarrow t \in terms H \rangle$ **for** t
by (*meson in-mono terms-Fun*)
ultimately show *?case*
by (*simp add: list.map-ident-strong*)
qed *simp*

```

lemma canonical-tm-id-map [simp]:
  ⟨set ts ⊆ terms H ⇒ map ⟨(λn. #n ∈? terms H, λP ts. OP ts ∈? terms H)⟩
  ts = ts⟩
  by (induct ts simp-all)

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

lemmas
  confl = satH[of C.kind] and
  alpha = satH[of A.kind] and
  beta = satH[of B.kind] and
  gamma = satH[of G.kind] and
  delta = satH[of D.kind]

theorem model: ⟨(p ∈ H → canonical H ⊨ p) ∧ (¬ p ∈ H → ¬ canonical H
  ⊨ p)⟩
proof (induct p rule: wf-induct[where r=⟨measure size-fm⟩])
  case 1
  then show ?case
    by simp
next
  case (2 x)
  then show ?case
  proof (cases x)
    case F!s
    then show ?thesis
      using confl by fastforce
  next
  case (Pre P ts)
  then show ?thesis
  proof (safe del: notI)
    assume ⟨x = ·P ts⟩ ⟨·P ts ∈ H⟩
    moreover from this have ⟨set ts ⊆ terms H⟩
      using terms-tm-refl terms-def by fastforce
    ultimately show ⟨canonical H ⊨ ·P ts⟩
      by simp
  next
  assume ⟨x = ·P ts⟩ ⟨¬ ·P ts ∈ H⟩
  then have ⟨·P ts ∉ H⟩
    using confl by force
  moreover have ⟨set ts ⊆ terms H⟩
    using ⟨¬ ·P ts ∈ H⟩ terms-def terms-tm-refl by fastforce
  ultimately show ⟨¬ canonical H ⊨ ·P ts⟩
    by simp
qed

```

```

next
  case (Imp p q)
  then show ?thesis
  proof (safe del: notI)
    assume ⟨x = p ⟶ q⟩ ⟨p ⟶ q ∈ H⟩
    then have ⟨¬ p ∈ H ∨ q ∈ H⟩
      using beta by force
    then show ⟨canonical H ⊨ p ⟶ q⟩
      using 2 Imp by auto
  next
    assume ⟨x = p ⟶ q⟩ ⟨¬ (p ⟶ q) ∈ H⟩
    then have ⟨p ∈ H ∧ ¬ q ∈ H⟩
      using alpha by force
    then show ⟨¬ canonical H ⊨ p ⟶ q⟩
      using 2 Imp by auto
  qed
next
  case (Uni p)
  then show ?thesis
  proof (safe del: notI)
    assume ⟨x = ∀ p⟩ ⟨∀ p ∈ H⟩
    then have ⟨∀ t ∈ terms H. ⟨t⟩p ∈ H⟩
      using gamma by fastforce
    moreover have ⟨∀ t. (⟨t⟩p, ∀ p) ∈ measure size-fm⟩
      by simp
    ultimately have ⟨∀ t ∈ terms H. canonical H ⊨ ⟨t⟩p⟩
      using 2 ⟨x = ∀ p⟩ by blast
    then show ⟨canonical H ⊨ ∀ p⟩
      by simp
  next
    assume ⟨x = ∀ p⟩ ⟨¬ ∀ p ∈ H⟩
    then obtain a where ⟨¬ ⟨★a⟩p ∈ H⟩
      using delta by fastforce
    moreover have ⟨(⟨★a⟩p, ∀ p) ∈ measure size-fm⟩
      by simp
    ultimately have ⟨¬ canonical H ⊨ ⟨★a⟩p⟩
      using 2 ⟨x = ∀ p⟩ by blast
    then show ⟨¬ canonical H ⊨ ∀ p⟩
      using wf-canonical[OF terms-ne] by auto
  qed
qed
qed
end

theorem model-existence:
  fixes S :: ⟨'f, 'p⟩ fm set
  assumes ⟨P.propE Kinds C⟩
  and ⟨S ∈ C⟩

```

```

    and ⟨P.enough-new S⟩
    and ⟨terms S ≠ {}⟩
    and ⟨p ∈ S⟩
  shows ⟨canonical (mk-mcs C S) ⊨ p⟩
proof –
  have *: ⟨MyHintikka (mk-mcs C S)⟩
proof
  show ⟨terms (mk-mcs C S) ≠ {}⟩
    using assms(4) terms-mono Extend-subset by blast
next
  show ⟨P.propH Kinds (mk-mcs C S)⟩
    using mk-mcs-Hintikka[OF assms(1–3)] Hintikka.hintikka by blast
qed
moreover have ⟨p ∈ mk-mcs C S⟩
  using assms(5) Extend-subset by blast
ultimately show ?thesis
  using MyHintikka.model by blast
qed

```

2.7 Compactness

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

lemma *compact-C*: ⟨P.sat_E C.kind {S :: ('f, 'p) fm set. P.enough-new S ∧ (∀ S' ⊆ S. finite S' → (∃ (U :: 'a set) E F G. wf-model (Model U E F G) ∧ Model U E F G ⊨ S'))}⟩

```

proof safe
  fix S ps qs and q :: ⟨('f, 'p) fm⟩
  assume ⟨ps ~χ qs⟩ and *: ⟨set ps ⊆ S⟩
  ⟨∀ S' ⊆ S. finite S' → (∃ (U :: 'a set) E F G. wf-model (Model U E F G) ∧ Model U E F G ⊨ S')⟩
  ⟨q ∈ set qs⟩ ⟨q ∈ S⟩
  then show ⟨q ∈ {}⟩
proof cases
  case CFls
  then show ?thesis
    using *(1–2) by (meson List.finite-set list.set-intros(1) semantics-fm.simps(1))
next
  case (CNeg P ts)
  then have ⟨{·P ts, ¬·P ts} ⊆ S⟩ ⟨finite {·P ts, ¬·P ts}⟩
    using * by simp-all
  then have ⟨∃ (U :: 'a set) E F G. wf-model (Model U E F G) ∧ Model U E F G ⊨ ·P ts ∧ Model U E F G ⊨ ¬·P ts⟩
    using * by (meson insertCI)
  then show ?thesis
    by simp
qed

```

qed

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

proof safe

fix qs **and** $S :: \langle (f, p) \text{ fm set} \rangle$
assume $\langle P.\text{enough-new } S \rangle$
then show $\langle P.\text{enough-new } (\text{set } qs \cup S) \rangle$
using params-left by blast

next

fix ps qs **and** $S S' :: \langle (f, p) \text{ fm set} \rangle$
assume $\langle ps \rightsquigarrow_\alpha qs \rangle$ **and** $*$: $\langle \text{set } ps \subseteq S \rangle$
 $\langle \forall S' \subseteq S. \text{finite } S' \longrightarrow (\exists (U :: 'a \text{ set}) E F G. \text{wf-model } (\text{Model } U E F G) \wedge \text{Model } U E F G \models S') \rangle$
 $\langle S' \subseteq \text{set } qs \cup S \rangle$ $\langle \text{finite } S' \rangle$
then show $\langle \exists (U :: 'a \text{ set}) E F G. \text{wf-model } (\text{Model } U E F G) \wedge \text{Model } U E F G \models S' \rangle$

proof cases

case $(CImpN p q)$
let $?S = \langle \{ \neg (p \longrightarrow q) \} \cup (S' - \{p, \neg q\}) \rangle$
have $\langle ?S \subseteq S \rangle$ $\langle \text{finite } ?S \rangle$
using $CImpN$ *** by fastforce+**
then obtain $U :: \langle 'a \text{ set} \rangle$ **and** $E F G$ **where**
 $M: \langle \text{wf-model } (\text{Model } U E F G) \rangle$ $\langle \text{Model } U E F G \models ?S \rangle$
using $*$ **by meson**
then have $\langle \text{Model } U E F G \models \{p, \neg q\} \cup ?S \rangle$
by auto
then show $?thesis$
using $M(1)$ **by blast**

qed

qed

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

proof safe

fix ps qs **and** $S S' :: \langle (f, p) \text{ fm set} \rangle$
assume $\langle ps \rightsquigarrow_\beta qs \rangle$ **and** $*$: $\langle \text{set } ps \subseteq S \rangle$ $\langle P.\text{enough-new } S \rangle$
 $\langle \forall S' \subseteq S. \text{finite } S' \longrightarrow (\exists (U :: 'a \text{ set}) E F G. \text{wf-model } (\text{Model } U E F G) \wedge \text{Model } U E F G \models S') \rangle$

then show $\langle \exists q \in \text{set } qs. \text{insert } q S \in \{S. P.\text{enough-new } S \wedge (\forall S' \subseteq S. \text{finite } S' \longrightarrow (\exists (U :: 'a \text{ set}) E F G. \text{wf-model } (\text{Model } U E F G) \wedge \text{Model } U E F G \models S')) \rangle$

proof cases

case $(CImpP p q)$
then have $P: \langle \forall q \in \text{set } qs. \{q\} \cup S \in \{S. P.\text{enough-new } S \} \rangle$
using $*$ **params-left by** $(\text{metis List.set-insert insert-is-Un list.set}(1) \text{mem-Collect-eq})$

show $?thesis$

proof (*rule ccontr*)
assume $\langle \neg (\exists q \in \text{set } qs. \text{insert } q \ S \in \{S. P.\text{enough-new } S \wedge (\forall S' \subseteq S. \text{finite } S' \longrightarrow (\exists (U :: 'a \text{ set}) \ E \ F \ G. \text{wf-model } (\text{Model } U \ E \ F \ G) \wedge \text{Model } U \ E \ F \ G \models S'))\}) \rangle$
then have $\langle \forall q \in \text{set } qs. \exists S' \subseteq \{q\} \cup S. \text{finite } S' \wedge \neg (\exists (U :: 'a \text{ set}) \ E \ F \ G. \text{wf-model } (\text{Model } U \ E \ F \ G) \wedge \text{Model } U \ E \ F \ G \models S') \rangle$
using *P* **by** *simp*
then have $\langle \forall q \in \text{set } qs. \exists S' \subseteq \{q\} \cup S. \text{finite } S' \wedge (\forall (U :: 'a \text{ set}) \ E \ F \ G. \text{wf-model } (\text{Model } U \ E \ F \ G) \longrightarrow \neg \text{Model } U \ E \ F \ G \models S') \rangle$
by *meson*
then obtain *Sp Sq* **where**
Sp: $\langle Sp \subseteq \{\neg p\} \cup S \rangle \langle \text{finite } Sp \rangle \langle \forall (U :: 'a \text{ set}) \ E \ F \ G. \text{wf-model } (\text{Model } U \ E \ F \ G) \longrightarrow \neg \text{Model } U \ E \ F \ G \models Sp \rangle$ **and**
Sq: $\langle Sq \subseteq \{q\} \cup S \rangle \langle \text{finite } Sq \rangle \langle \forall (U :: 'a \text{ set}) \ E \ F \ G. \text{wf-model } (\text{Model } U \ E \ F \ G) \longrightarrow \neg \text{Model } U \ E \ F \ G \models Sq \rangle$
using *CImpP* **by** *auto*

let $?S = \langle \{p \longrightarrow q\} \cup (Sp - \{\neg p\}) \cup (Sq - \{q\}) \rangle$
have $\langle \text{finite } ?S \rangle$
using *Sp(2) Sq(2)* **by** *blast*
moreover have $\langle ?S \subseteq S \rangle$
using **(1) CImpP(1) Sp(1) Sq(1)* **by** *auto*
ultimately obtain *U :: 'a set* **and** *E F G* **where**
M: $\langle \text{wf-model } (\text{Model } U \ E \ F \ G) \rangle \langle \text{Model } U \ E \ F \ G \models ?S \rangle$
using *** **by** *meson*
then have $\langle \text{Model } U \ E \ F \ G \models Sp \vee \text{Model } U \ E \ F \ G \models Sq \rangle$
by *auto*
then show *False*
using *M(1) Sp(3) Sq(3)* **by** *blast*

qed
qed
qed

lemma *compact-G*: $\langle P.\text{sat}_E \ G.\text{kind } \{S :: ('f, 'p) \text{ fm set}. P.\text{enough-new } S \wedge (\forall S' \subseteq S. \text{finite } S' \longrightarrow (\exists (U :: 'a \text{ set}) \ E \ F \ G. \text{wf-model } (\text{Model } U \ E \ F \ G) \wedge \text{Model } U \ E \ F \ G \models S'))\} \rangle$
proof *safe*
fix *qs :: 'f tm \Rightarrow ('f, 'p) fm list* **and** *t* **and** *S :: ('f, 'p) fm set*
assume $\langle P.\text{enough-new } S \rangle$
then show $\langle P.\text{enough-new } (\text{set } (qs \ t) \cup S) \rangle$
using *params-left* **by** *fast*

next
fix *ps qs F t* **and** *S S' :: ('f, 'p) fm set*
assume $\langle ps \rightsquigarrow_\gamma (F, qs) \rangle$ **and** ***: $\langle \text{set } ps \subseteq S \rangle \langle P.\text{enough-new } S \rangle$
 $\langle \forall S' \subseteq S. \text{finite } S' \longrightarrow (\exists (U :: 'a \text{ set}) \ E \ F \ G. \text{wf-model } (\text{Model } U \ E \ F \ G) \wedge \text{Model } U \ E \ F \ G \models S') \rangle$
 $\langle t \in F \ S \rangle \langle S' \subseteq \text{set } (qs \ t) \cup S \rangle \langle \text{finite } S' \rangle$
then show $\langle \exists (U :: 'a \text{ set}) \ E \ F \ G. \text{wf-model } (\text{Model } U \ E \ F \ G) \wedge \text{Model } U \ E \ F \ G \models S' \rangle$

```

proof cases
case (CALLP p)
  let  $?S = \langle \{\forall p\} \cup (S' - \text{set } (qs\ t)) \rangle$ 
  have  $\langle ?S \subseteq S \rangle \langle \text{finite } ?S \rangle$ 
  using CALLP * by fastforce+
  then obtain  $U :: \langle 'a\ \text{set} \rangle$  and  $E\ F\ G$  where
     $M: \langle \text{wf-model } (Model\ U\ E\ F\ G) \rangle \langle Model\ U\ E\ F\ G \models ?S \rangle$ 
  using * by meson
  then have  $\langle Model\ U\ E\ F\ G \models \text{set } (qs\ t) \cup ?S \rangle$ 
  using CALLP by auto
  then show ?thesis
  using  $M(1)$  by blast
qed
qed

lemma compact-D:  $\langle P.\text{sat}_E\ D.\text{kind } \{S :: \langle 'f, 'p \rangle\ \text{fm set. } P.\text{enough-new } S \wedge$ 
   $(\forall S' \subseteq S. \text{finite } S' \longrightarrow (\exists (U :: 'a\ \text{set})\ E\ F\ G. \text{wf-model } (Model\ U\ E\ F\ G) \wedge$ 
   $Model\ U\ E\ F\ G \models S')) \rangle$ 
proof safe
  fix p and  $S :: \langle \langle 'f, 'p \rangle\ \text{fm set} \rangle$ 
  assume *:  $\langle p \in S \rangle \langle P.\text{enough-new } S \rangle$ 
   $\langle \forall S' \subseteq S. \text{finite } S' \longrightarrow (\exists (U :: 'a\ \text{set})\ E\ F\ G. \text{wf-model } (Model\ U\ E\ F\ G) \wedge$ 
   $Model\ U\ E\ F\ G \models S') \rangle$ 
  then show  $\langle \exists x. \text{True} \wedge \text{set } (\delta\ p\ x) \cup S \in \{S. P.\text{enough-new } S \wedge$ 
   $(\forall S' \subseteq S. \text{finite } S' \longrightarrow (\exists (U :: 'a\ \text{set})\ E\ F\ G. \text{wf-model } (Model\ U\ E\ F\ G) \wedge$ 
   $Model\ U\ E\ F\ G \models S')) \rangle$ 
  proof (induct p - rule:  $\delta.\text{induct}$ )
    case ( $1\ p\ -$ )
    then have  $P: \langle \forall x. \text{set } (\delta\ (\neg\ \forall\ p)\ x) \cup S \in \{S. P.\text{enough-new } S\} \rangle$ 
    using * params-left by fast

    show ?case
    proof (rule ccontr)
      assume  $\langle \nexists x. \text{True} \wedge \text{set } (\delta\ (\neg\ \forall\ p)\ x) \cup S \in \{S. P.\text{enough-new } S \wedge (\forall S' \subseteq S. \text{finite } S' \longrightarrow (\exists (U :: 'a\ \text{set})\ E\ F\ G. \text{wf-model } (Model\ U\ E\ F\ G) \wedge Model\ U\ E\ F\ G \models S')) \rangle$ 
      then have  $\langle \forall x. \exists S' \subseteq \{\neg \langle \star x \rangle p\} \cup S. \text{finite } S' \wedge \neg (\exists (U :: 'a\ \text{set})\ E\ F\ G. \text{wf-model } (Model\ U\ E\ F\ G) \wedge Model\ U\ E\ F\ G \models S') \rangle$ 
      using P by simp
      moreover obtain  $x$  where  $x: \langle x \notin P.\text{params } S \rangle$ 
      using  $\langle P.\text{enough-new } S \rangle$  unfolding P.enough-new-def
      by (metis Diff-eq-empty-iff card-of-ordLeq-finite finite.emptyI inf-univ subsetI)
      ultimately obtain  $S'$  where
         $S': \langle S' \subseteq \{\neg \langle \star x \rangle p\} \cup S \rangle \langle \text{finite } S' \rangle \langle \forall (U :: 'a\ \text{set})\ E\ F\ G. \text{wf-model } (Model\ U\ E\ F\ G) \longrightarrow \neg Model\ U\ E\ F\ G \models S' \rangle$ 
        using  $1$  by meson

      let  $?S = \langle \{\neg \forall p\} \cup (S' - \{\neg \langle \star x \rangle p\}) \rangle$ 
      have  $\langle \text{finite } ?S \rangle$ 

```

```

    using  $S'(2)$  by blast
  moreover have **:  $\langle ?S \subseteq S \rangle$ 
    using  $*(1)$   $1(1)$   $S'(1)$  by auto
  ultimately obtain  $U :: \langle 'a \text{ set} \rangle$  and  $E F G$  where
     $M: \langle \text{wf-model } (Model U E F G) \rangle \langle Model U E F G \models ?S \rangle$ 
    using * by meson
  then obtain  $z$  where  $z: \langle z \in U \rangle \langle \neg Model U (z \gg E) F G \models p \rangle$ 
    by auto

  let  $?F = \langle F(x := \lambda-. z) \rangle$ 
  have  $\langle \neg Model U (?F x [] \gg E) ?F G \models p \rangle$ 
    using  $M x z **$  by auto
  moreover have  $\langle Model U E ?F G \models ?S \rangle$ 
    using  $M x **$  by (metis (no-types, lifting) UN-iff in-mono upd-params-fm)
  ultimately have  $\langle \exists F. \text{wf-model } (Model U E F G) \wedge Model U E F G \models S' \rangle$ 
    using  $M(1) z$  by (auto intro!: exI[of - ?F])
  then show False
    using  $S'(3)$  by blast
qed
qed simp-all
qed

lemma compact-prop:  $\langle P.\text{prop}_E \text{ Kinds } \{S :: \langle 'f, 'p \rangle \text{ fm set. } P.\text{enough-new } S \wedge$ 
   $(\forall S' \subseteq S. \text{finite } S' \longrightarrow (\exists (U :: 'a \text{ set}) E F G. \text{wf-model } (Model U E F G) \wedge$ 
   $Model U E F G \models S')\} \rangle$ 
  using compact-C compact-A compact-B compact-G compact-D ..

theorem compactness:
  fixes  $S :: \langle 'f, 'p \rangle \text{ fm set} \rangle$ 
  assumes  $\langle P.\text{enough-new } S \rangle$ 
  shows  $\langle (\exists (U :: 'f \text{ tm set}) E F G. \text{wf-model } (Model U E F G) \wedge Model U E F G \models S) \longleftrightarrow$ 
   $(\forall S' \subseteq S. \text{finite } S' \longrightarrow (\exists (U :: 'f \text{ tm set}) E F G. \text{wf-model } (Model U E F G) \wedge Model U E F G \models S')) \rangle$ 
  proof safe
    fix  $U :: \langle 'f \text{ tm set} \rangle$  and  $E F G$  and  $S' :: \langle ('f, 'p) \text{ fm set} \rangle$ 
    assume  $\langle \text{wf-model } (Model U E F G) \rangle \langle Model U E F G \models S \rangle \langle S' \subseteq S \rangle$ 
    then show  $\langle \exists (U :: 'f \text{ tm set}) E F G. \text{wf-model } (Model U E F G) \wedge Model U E F G \models S' \rangle$ 
      by blast
  next
    let  $?S = \langle \{has\text{-subterm}\} \cup S \rangle$ 
    assume *:  $\langle \forall S' \subseteq S. \text{finite } S' \longrightarrow (\exists (U :: 'f \text{ tm set}) E F G. \text{wf-model } (Model U E F G) \wedge Model U E F G \models S') \rangle$ 
    have  $\langle \forall S' \subseteq ?S. \text{finite } S' \longrightarrow (\exists (U :: 'f \text{ tm set}) E F G. \text{wf-model } (Model U E F G) \wedge Model U E F G \models S') \rangle$ 
      proof safe
        fix  $S'$ 
        assume  $\langle S' \subseteq \{has\text{-subterm}\} \cup S \rangle \langle \text{finite } S' \rangle$ 
        then obtain  $U :: \langle 'f \text{ tm set} \rangle$  and  $E F G$  where  $M: \langle \text{wf-model } (Model U E F$ 

```

```

G)⟩ ⟨Model U E F G ⊨ S' - {has-subterm}⟩
  using * by (meson Diff-subset-conv finite-Diff)
  then have ⟨Model U E F G ⊨ S'⟩
    by auto
  then show ⟨∃(U :: 'f tm set) E F G. wf-model (Model U E F G) ∧ Model U
E F G ⊨ S'⟩
    using M(1) by blast
qed
moreover have P: ⟨P.enough-new ?S⟩
  using assms by (metis List.set-insert empty-set params-left)
ultimately have *: ⟨?S ∈ {S :: ('f, 'p) fm set. P.enough-new S ∧
(∀ S' ⊆ S. finite S' ⟶ (∃(U :: 'f tm set) E F G. wf-model (Model U E F G)
∧ Model U E F G ⊨ S'))}⟩
  by fast

```

```

have **: ⟨terms ?S ≠ {}⟩
  unfolding terms-def by simp
have ⟨∃ C. canonical (mk-mcs C ?S) ⊨ ?S⟩
  using model-existence[OF compact-prop * P **] by blast
moreover have ⟨terms (mk-mcs C ?S) ≠ {}⟩ for C
  using ** by (metis Extend-subset empty-subsetI subset-antisym terms-mono)
ultimately show ⟨∃(U :: 'f tm set) E F G. wf-model (Model U E F G) ∧ Model
U E F G ⊨ S⟩
  using wf-canonical by fast
qed

```

2.8 Natural Deduction

locale *Natural-Deduction*
begin

```

inductive ND-Set :: ⟨('f, 'p) fm set ⇒ ('f, 'p) fm ⇒ bool⟩ (infix «⊢» 50) where
  Assm [dest]: ⟨p ∈ 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} ∪ A) ⟹ A ⊢ ∀ p⟩
| UniE [dest]: ⟨A ⊢ ∀ p ⟹ t ∈ terms ({p} ∪ A) ⟹ A ⊢ ⟨t⟩p⟩
| Clas: ⟨{p ⟶ q} ∪ A ⊢ p ⟹ A ⊢ p⟩

```

2.8.1 Soundness

theorem *soundness-set*:

```

  assumes ⟨A ⊢ p⟩ ⟨wf-model (Model U E F G)⟩
  shows ⟨∀ q ∈ A. Model U E F G ⊨ q ⟹ Model U E F G ⊨ p⟩
  using assms
proof (induct A p arbitrary: F pred: ND-Set)
  case (UniI A a p)
  have ⟨∀ x ∈ U. ∀ q ∈ A. Model U E (F(a := λ-. x)) G ⊨ q⟩

```

```

    using UniI(3-) by simp
  moreover have  $\langle \forall x \in U. \text{wf-model } (\text{Model } U \ E \ (F(a := \lambda-. x)) \ G) \rangle$ 
    using UniI(5) by simp
  ultimately have  $\langle \forall x \in U. \text{Model } U \ E \ (F(a := \lambda-. x)) \ G \models \langle \star a \rangle p \rangle$ 
    using UniI by meson
  then show ?case
    using UniI by simp
qed auto

```

2.8.2 Derivational Consistency

```

lemma Boole:  $\langle \{\neg p\} \cup A \Vdash \perp \implies A \Vdash p \rangle$ 
  unfolding Neg-def using Clas FlsE by fast

```

```

sublocale DC: Derivational-Confl psub params-fm  $\langle \lambda-. \text{True} \rangle$  confl-class  $\langle \lambda A. \neg A \Vdash \perp \rangle$ 

```

proof

```

  fix A ps qs and q ::  $\langle ('f, 'p) \text{fm} \rangle$ 
  assume  $\langle ps \rightsquigarrow_{\chi} qs \rangle \langle \text{set } ps \subseteq A \rangle \langle q \in \text{set } qs \rangle \langle q \in A \rangle$ 
  then show  $\langle \neg \neg A \Vdash \perp \rangle$ 
    by cases auto

```

qed

```

sublocale DA: Derivational-Alpha psub params-fm  $\langle \lambda-. \text{True} \rangle$  alpha-class  $\langle \lambda A. \neg A \Vdash \perp \rangle$ 

```

proof (standard; safe)

```

  fix A and ps qs ::  $\langle ('f, 'p) \text{fm list} \rangle$ 
  assume  $\langle ps \rightsquigarrow_{\alpha} qs \rangle$  and *:  $\langle \text{set } ps \subseteq A \rangle \langle \neg A \Vdash \perp \rangle \langle \text{set } qs \cup A \Vdash \perp \rangle$ 
  then show False
  proof cases
    case (CImpN p q)
    then have  $\langle A \Vdash \neg (p \longrightarrow q) \rangle$ 
      using *(1) by auto
    moreover have  $\langle A \Vdash p \longrightarrow q \rangle$ 
      using CImpN(2) * Boole[of q  $\langle \{p\} \cup A \rangle$ ] by auto
    ultimately show ?thesis
      using * by blast
  qed

```

qed

```

sublocale DB: Derivational-Beta psub params-fm  $\langle \lambda-. \text{True} \rangle$  beta-class  $\langle \lambda A. \neg A \Vdash \perp \rangle$ 

```

proof

```

  fix A and ps qs ::  $\langle ('f, 'p) \text{fm list} \rangle$ 
  assume  $\langle ps \rightsquigarrow_{\beta} qs \rangle$  and *:  $\langle \text{set } ps \subseteq A \rangle \langle \neg A \Vdash \perp \rangle$ 
  then show  $\langle \exists q \in \text{set } qs. \neg \{q\} \cup A \Vdash \perp \rangle$ 
  proof cases
    case (CImpP p q)
    then show ?thesis

```

```

    using * Boole[of p A]
    by (metis Assm ImpE ImpI list.set-intros(1) set-subset-Cons subset-iff)
  qed
qed

sublocale DG: Derivational-Gamma map-tm psub params-fm ⟨λ-. True⟩ gamma-class
⟨λA. ¬ A ⊢ ⊥⟩
proof
  fix A F qs t and ps :: ⟨('f, 'p) fm list⟩
  assume ⟨ps ↘γ (F, qs)⟩ and *: ⟨set ps ⊆ A⟩ ⟨t ∈ F A⟩ ⟨¬ A ⊢ ⊥⟩
  then show ⟨¬ set (qs t) ∪ A ⊢ ⊥⟩
  proof cases
    case (CALLP p)
    then have ⟨t ∈ terms ({p} ∪ A)⟩
    using * terms-mono by blast
    then show ?thesis
    using CALLP * UniE[of A p t] ImpI by auto
  qed
qed

sublocale DD: Derivational-Delta psub params-fm ⟨λ-. True⟩ δ ⟨λA. ¬ A ⊢ ⊥⟩
proof (standard; safe)
  fix A a and p :: ⟨('f, 'p) fm⟩
  assume ⟨p ∈ A⟩ ⟨a ∉ P.params A⟩ ⟨¬ A ⊢ ⊥⟩ ⟨set (δ p a) ∪ A ⊢ ⊥⟩
  then show False
  proof (induct p a rule: δ.induct)
    case (1 p x)
    then have ⟨x ∉ P.params ({p} ∪ A)⟩
    by auto
    moreover have ⟨A ⊢ ⟨★ x⟩ p⟩
    using 1(4) Boole by auto
    ultimately show ?case
    using 1 UniI by blast
  qed simp-all
qed

sublocale Derivational-Consistency psub params-fm ⟨λ-. True⟩ Kinds ⟨λA. ¬ A ⊢ ⊥⟩
  using propE-Kinds[OF DC.kind DA.kind DB.kind DG.kind DD.kind] by unfold-locales

```

2.8.3 Strong Completeness

lemma *with-subterm-elim*: $\langle A \Vdash \text{with-subterm } p \implies A \Vdash p \rangle$
 using Assm ImpE by blast

theorem *strong-completeness*:
 fixes p :: ⟨('f, 'p) fm⟩
 assumes mod: $\langle \bigwedge (U :: 'f \text{ tm set}) E F G. \text{wf-model (Model } U E F G) \implies$

```

  (∀ q ∈ A. Model U E F G ⊨ q) ⇒ Model U E F G ⊨ p
  and ⟨P.enough-new A⟩
  shows ⟨A ⊢ p⟩
proof (rule ccontr)
  assume ⟨¬ A ⊢ p⟩
  then have ⟨¬ A ⊢ with-subterm p⟩
    using with-subterm-elim by blast
  then have *: ⟨¬ {¬ with-subterm p} ∪ A ⊢ ⊥⟩
    using Boole by (metis insert-is-Un)

  let ?S = ⟨set [¬ with-subterm p] ∪ A⟩
  let ?C = ⟨{A. P.enough-new A ∧ ¬ A ⊢ ⊥}⟩
  let ?M = ⟨canonical (mk-mcs ?C ?S)⟩

  have ne: ⟨terms ?S ≠ {}⟩
    unfolding terms-def by simp
  then have ⟨terms (mk-mcs ?C ?S) ≠ {}⟩
    by (metis (no-types, lifting) ext Extend-subset subset-empty terms-mono)
  then have wf: ⟨wf-model ?M⟩
    using wf-canonical by fast

  have ⟨P.prop_E Kinds ?C⟩
    using Consistency by blast
  moreover have ⟨P.enough-new ?S⟩
    using assms(2) params-left by blast
  moreover from this have ⟨?S ∈ ?C⟩
    using * by simp
  ultimately have *: ⟨∀ p ∈ ?S. ?M ⊨ p⟩
    using model-existence ne by blast
  then have ⟨?M ⊨ p⟩
    using mod[OF wf] by fast
  then show False
    using * by simp
qed

```

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 $\langle \text{bounded } K \ A \ P \rangle \langle \bigwedge A. P \ A \implies Q \ A \rangle$
shows $\langle \text{bounded } K \ A \ Q \rangle$
using *assms unfolding bounded-def by simp*

lemma *bounded-two* [elim]:
assumes $\langle \text{bounded } K \ A \ P \rangle \langle \text{bounded } K' \ A \ Q \rangle \langle \bigwedge A. P \ A \implies Q \ A \implies R \ A \rangle$
shows $\langle \text{bounded } (K \ @ \ K') \ A \ R \rangle$
using *assms unfolding bounded-def by simp*

lemma *bounded-removeAll* [dest]:
assumes $\langle \text{bounded } K \ (\{p\} \cup A) \ P \rangle$
shows $\langle \text{bounded } (\text{removeAll } p \ K) \ A \ (\lambda B. P \ (p \ \# \ B)) \rangle$
using *assms unfolding bounded-def*
by (*metis Diff-subset-conv insert-is-Un insert-mono list.simps(15) set-removeAll*)

lemma *bounded-terms*:
assumes $\langle t \in \text{terms } (\{p\} \cup A) \rangle$
shows $\langle t \in \text{terms-fm } p \ \wedge \ \text{bounded } [] \ A \ (\lambda B. t \in \text{terms } (\text{set } (p \ \# \ B))) \vee$
 $\langle \exists q \in A. t \in \text{terms-fm } q \ \wedge \ \text{bounded } [q] \ A \ (\lambda B. t \in \text{terms } (\text{set } (p \ \# \ B))) \rangle$
using *assms unfolding terms-def bounded-def by auto*

lemma *bounded-params*:
assumes $\langle a \notin P.\text{params } (\{p\} \cup A) \rangle \langle \text{bounded } K \ A \ P \rangle$
shows $\langle \text{bounded } K \ A \ (\lambda B. a \notin P.\text{params } (\text{set } (p \ \# \ B))) \rangle$
using *assms unfolding bounded-def by auto*

lemma *finite-kernel*: $\langle A \Vdash p \implies \exists K. \text{bounded } K \ A \ (\lambda B. B \vdash p) \rangle$
proof (*induct A p pred: ND-Set*)
case (*Assm p A*)
then show *?case*
unfolding *bounded-def by (auto intro!: exI[of - <[p]>])*
next
case (*UniI A a p*)
then show *?case*
using *bounded-params by fastforce*
next
case (*UniE A p t*)
then show *?case*
using *bounded-terms by fastforce*
next
case (*Clas p q A*)
then have $\langle \exists K. \text{bounded } K \ A \ (\lambda B. (p \longrightarrow q) \ \# \ B \vdash p) \rangle$
by *fast*
then show *?case*
using *ND-List.Clas unfolding bounded-def by meson*
qed *fast+*

corollary *finite-assumptions*: $\langle A \Vdash p \implies \exists B. \text{set } B \subseteq A \ \wedge \ B \vdash p \rangle$

using *finite-kernel unfolding bounded-def* by *blast*

lemma *to-set*: $\langle A \vdash p \implies \text{set } A \Vdash p \rangle$
 by (*induct* A *p pred*: *ND-List*) (*auto intro*: *ND-Set.Clas*)

corollary *soundness-list*:

assumes $\langle A \vdash p \rangle \langle \text{wf-model } (\text{Model } U \ E \ F \ G) \rangle \langle \forall q \in \text{set } A. \text{Model } U \ E \ F \ G \models q \rangle$
shows $\langle \text{Model } U \ E \ F \ G \models p \rangle$
 using *assms soundness-set to-set* by *fast*

corollary *soundness-nil*: $\langle [] \vdash p \implies \text{wf-model } (\text{Model } U \ E \ F \ G) \implies \text{Model } U \ E \ F \ G \models p \rangle$
 using *soundness-list* by (*metis empty-iff list.set(1)*)

corollary $\langle \neg ([] \vdash \perp) \rangle$
 using *soundness-nil* by *fastforce*

corollary *strong-completeness-list*:

fixes $p :: \langle ('f, 'p) \text{ fm} \rangle$
assumes *mod*: $\langle \bigwedge (U :: 'f \text{ tm set}) \ E \ F \ G. \text{wf-model } (\text{Model } U \ E \ F \ G) \implies (\forall q \in A. \text{Model } U \ E \ F \ G \models q) \implies \text{Model } U \ E \ F \ G \models p \rangle$
and $\langle P.\text{enough-new } A \rangle$
shows $\langle \exists B. \text{set } B \subseteq A \wedge B \vdash p \rangle$
 using *assms strong-completeness finite-assumptions* by *blast*

theorem *main*:

fixes $p :: \langle ('f, 'p) \text{ fm} \rangle$
assumes $\langle |UNIV :: ('f, 'p) \text{ fm set}| \leq o \ |UNIV :: 'f \text{ set}| \rangle$
shows $\langle [] \vdash p \iff (\forall (U :: 'f \text{ tm set}) \ E \ F \ G. \text{wf-model } (\text{Model } U \ E \ F \ G) \longrightarrow \text{Model } U \ E \ F \ G \models p) \rangle$
 using *assms strong-completeness-list*[of $\langle \{ \} \rangle$ p] *soundness-nil*[of p] **unfolding** *P.enough-new-def*
 by *simp blast*

end

2.9 Tableau

locale *Tableau*
begin

inductive *TC* :: $\langle ('f, 'p) \text{ fm set} \implies \text{bool} \rangle (\langle \vdash \rightarrow [51] \ 50 \rangle)$ **where**

Axiom [*simp*]: $\langle \neg \cdot P \text{ ts} \in A \implies \cdot P \text{ ts} \in A \implies \vdash A \rangle$
 | *FlsP* [*simp*]: $\langle \perp \in A \implies \vdash A \rangle$
 | *FlsN* [*intro*]: $\langle \vdash A \implies \vdash \{ \neg \perp \} \cup A \rangle$
 | *ImpP* [*intro*]: $\langle \vdash \{ \neg p \} \cup A \implies \vdash \{ q \} \cup A \implies \vdash \{ p \longrightarrow q \} \cup A \rangle$
 | *ImpN* [*intro*]: $\langle \vdash \{ p, \neg q \} \cup A \implies \vdash \{ \neg (p \longrightarrow q) \} \cup A \rangle$
 | *UniP* [*intro*]: $\langle \vdash \{ \langle t \rangle p \} \cup A \implies t \in \text{terms } (\{ p \} \cup A) \implies \vdash \{ \forall p \} \cup A \rangle$

| *UniN* [*intro*]: $\langle \vdash \{\neg \langle \star a \rangle p\} \cup A \implies a \notin P.params (\{p\} \cup A) \implies \vdash \{\neg \forall p\} \cup A \rangle$

2.9.1 Soundness

theorem *soundness*:

assumes $\langle \vdash A \rangle \langle wf-model (Model U E F G) \rangle$

shows $\langle \exists q \in A. \neg Model U E F G \models q \rangle$

using *assms*

proof (*induct A arbitrary: F pred: TC*)

case (*Axiom P ts A*)

then show *?case*

by (*meson semantics-fm.simps(1) semantics-fm.simps(3)*)

next

case (*FlsP A*)

then show *?case*

by *force*

next

case (*UniP t p A*)

then have $\langle \exists q \in \{\langle t \rangle p\} \cup A. \neg Model U E F G \models q \rangle$

by *blast*

moreover have $\langle \langle (E, F) \rangle t \in U \rangle$

using *UniP.prem* **by** *auto*

then have $\langle \neg Model U E F G \models \langle t \rangle p \implies \neg Model U E F G \models \forall p \rangle$

by *auto*

ultimately show *?case*

by *blast*

next

case (*UniN a p A*)

then have $\langle \forall x \in U. wf-model (Model U E (F(a := \lambda-. x)) G) \rangle$

by *simp*

then have $\langle \forall x \in U. \exists q \in \{\neg \langle \star a \rangle p\} \cup A. \neg Model U E (F(a := \lambda-. x)) G \models q \rangle$

using *UniN(2)* **by** *fast*

then show *?case*

using *UniN* **by** *simp*

qed *auto*

2.9.2 Derivational Consistency

sublocale *DC*: *Derivational-Confl* *psub params-fm* $\langle \lambda-. True \rangle$ *confl-class* $\langle \lambda A. \neg \vdash A \rangle$

proof

fix *A ps qs and q* :: $\langle ('f, 'p) fm \rangle$

assume $\langle ps \rightsquigarrow_{\mathbf{x}} qs \rangle \langle set ps \subseteq A \rangle \langle q \in set qs \rangle \langle q \in A \rangle$

then show $\langle \neg \neg \vdash A \rangle$

by *cases auto*

qed

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

proof

fix *A* **and** *ps qs* :: $\langle ('f, 'p) \text{ fm list} \rangle$
assume $\langle ps \rightsquigarrow_{\alpha} qs \rangle$ **and** *: $\langle \text{set } ps \subseteq A \rangle \langle \neg \vdash A \rangle$
then show $\langle \neg \vdash \text{set } qs \cup A \rangle$

proof cases

case (*CImpN* *p q*)
then show *?thesis*
using * *ImpN*[*of p q A*]
by (*auto simp: sup.order-iff*)

qed

qed

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

proof

fix *A* **and** *ps qs* :: $\langle ('f, 'p) \text{ fm list} \rangle$
assume $\langle ps \rightsquigarrow_{\beta} qs \rangle$ **and** *: $\langle \text{set } ps \subseteq A \rangle \langle \neg \vdash A \rangle$
then show $\langle \exists q \in \text{set } qs. \neg \vdash \{q\} \cup A \rangle$

proof cases

case (*CImpP* *p q*)
then show *?thesis*
using * *ImpP*[*of p A q*]
by (*auto simp: sup.order-iff*)

qed

qed

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

proof

fix *A F qs t* **and** *ps* :: $\langle ('f, 'p) \text{ fm list} \rangle$
assume $\langle ps \rightsquigarrow_{\gamma} (F, qs) \rangle$ **and** *: $\langle \text{set } ps \subseteq A \rangle \langle t \in F A \rangle \langle \neg \vdash A \rangle$
then show $\langle \neg \vdash \text{set } (qs t) \cup A \rangle$

proof cases

case (*CALLP* *p*)
then have $\langle t \in \text{terms } (\{p\} \cup A) \rangle$
using * *terms-mono* **by** *blast*
then show *?thesis*
using *CALLP* * *UniP*[*of t p A*]
by (*auto simp: sup.order-iff*)

qed

qed

sublocale *DD*: Derivational-Delta psub params-fm $\langle \lambda\cdot. \text{True} \rangle$ δ $\langle \lambda A. \neg \vdash A \rangle$

proof

fix *A a* **and** *p* :: $\langle ('f, 'p) \text{ fm} \rangle$
assume $\langle p \in A \rangle \langle a \notin P.\text{params } A \rangle \langle \neg \vdash A \rangle$
then show $\langle \neg \vdash \text{set } (\delta p a) \cup A \rangle$

```

proof (induct p a rule:  $\delta$ .induct)
  case (1 p x)
  then have  $\langle x \notin P.params (\{p\} \cup A) \rangle$ 
  by auto
  then show ?case
  using 1 UniN[of x p A] by (auto simp: sup.order-iff insert-absorb)
qed simp-all
qed

```

```

sublocale Derivational-Consistency psub params-fm  $\langle \lambda-. True \rangle$  Kinds  $\langle \lambda A. \neg \vdash A \rangle$ 
using propE-Kinds[OF DC.kind DA.kind DB.kind DG.kind DD.kind] by unfold-locales

```

2.9.3 Strong Completeness

theorem strong-completeness:

```

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

```

```

let ?S =  $\langle set [\neg \text{with-subterm } p] \cup A \rangle$ 
let ?C =  $\langle \{ A. P.enough-new A \wedge \neg \vdash A \} \rangle$ 
let ?M =  $\langle canonical (mk-mcs ?C ?S) \rangle$ 

```

```

have ne:  $\langle terms ?S \neq \{ \} \rangle$ 
unfolding terms-def by simp
then have  $\langle terms (mk-mcs ?C ?S) \neq \{ \} \rangle$ 
by (metis (no-types, lifting) ext Extend-subset subset-empty terms-mono)
then have wf:  $\langle wf-model ?M \rangle$ 
using wf-canonical by fast

```

```

have  $\langle P.prop_E Kinds ?C \rangle$ 
using Consistency by blast
moreover have  $\langle P.enough-new ?S \rangle$ 
using assms(2) params-left by blast
moreover from this have  $\langle ?S \in ?C \rangle$ 
using * by simp
ultimately have *:  $\langle \forall p \in ?S. ?M \models p \rangle$ 
using model-existence ne by blast
then have  $\langle ?M \models p \rangle$ 
using mod[OF wf] by fast
then show False
using * by simp
qed

```

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 (<O>2)
```

```
datatype (params-tm: 'f) tm  
  = Var nat (<#>1)  
  | Fun <'f sym> <'f tm list> (<O>1)  
  | 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 $ExiF$ ($\langle \exists_F \rangle$) **where** $\langle \exists_F p \equiv \neg(\forall_F(\neg p)) \rangle$

abbreviation $ExiP$ ($\langle \exists_P \rangle$) **where** $\langle \exists_P p \equiv \neg(\forall_P(\neg p)) \rangle$

term $\langle \forall(\perp \longrightarrow (\cdot(\circ_2 ''P'') [\circ(\circ_2 ''f'') [\#0]])) \rangle$

3.2 Semantics

definition *shift* ($\langle \cdot \langle - \rangle \rangle$) **where**

$\langle E \langle n : x \rangle m \equiv \text{if } m < n \text{ then } E m \text{ else if } m = n \text{ then } x \text{ else } E (m-1) \rangle$

primrec *semantics-fn* ($\langle \langle \cdot, - \rangle_2 \rangle$) **where**

$\langle \langle E_F, F \rangle_2 (\#_2 n) = E_F n \rangle$
 $| \langle \langle E_F, F \rangle_2 (\circ_2 f) = F f \rangle$

primrec *semantics-tm* ($\langle \langle \cdot, -, -, - \rangle \rangle$) **where**

$\langle \langle E, E_F, C, F \rangle (\#n) = E n \rangle$
 $| \langle \langle E, E_F, C, F \rangle (\circ f \ ts) = (\langle E_F, F \rangle_2 f) (\text{map } \langle E, E_F, C, F \rangle \ ts) \rangle$
 $| \langle \langle E, E_F, C, F \rangle (\star c) = C c \rangle$

fun *semantics-fm* (**infix** $\langle \models \rangle$ 50) **where**

$\langle \langle \langle \cdot, -, -, -, -, - \rangle \models \perp \rangle = \text{False} \rangle$
 $| \langle \langle \langle E, E_F, E_P, C, F, G, PS, FS \rangle \models \cdot P \ ts \rangle = \langle E_P, G \rangle_2 P (\text{map } \langle E, E_F, C, F \rangle \ ts) \rangle$
 $| \langle \langle \langle E, E_F, E_P, C, F, G, PS, FS \rangle \models p \longrightarrow q \rangle = (\langle E, E_F, E_P, C, F, G, PS, FS \rangle \models p \longrightarrow (\langle E, E_F, E_P, C, F, G, PS, FS \rangle \models q)) \rangle$
 $| \langle \langle \langle E, E_F, E_P, C, F, G, PS, FS \rangle \models \forall p \rangle = (\forall x. (\langle E \langle 0 : x \rangle, E_F, E_P, C, F, G, PS, FS \rangle \models p)) \rangle$
 $| \langle \langle \langle E, E_F, E_P, C, F, G, PS, FS \rangle \models \forall_P p \rangle = (\forall x \in PS. (\langle E, E_F, E_P \langle 0 : x \rangle, C, F, G, PS, FS \rangle \models p)) \rangle$
 $| \langle \langle \langle E, E_F, E_P, C, F, G, PS, FS \rangle \models \forall_{FP} p \rangle = (\forall x \in FS. (\langle E, E_F \langle 0 : x \rangle, E_P, C, F, G, PS, FS \rangle \models p)) \rangle$

proposition $\langle \langle \langle E, E_F, E_P, C, F, G, PS, FS \rangle \models (\forall (\cdot P [\# 0]) \longrightarrow \cdot P [\star a]) \rangle$

by (*simp add: shift-def*)

3.3 Operations

3.3.1 Shift

context *fixes* $n \ m :: \text{nat}$ **begin**

lemma *shift-eq* [*simp*]: $\langle n = m \implies E \langle n : x \rangle m = x \rangle$

by (*simp add: shift-def*)

lemma *shift-gt* [*simp*]: $\langle m < n \implies E \langle n : x \rangle m = E m \rangle$

by (*simp add: shift-def*)

lemma *shift-lt* [*simp*]: $\langle n < m \implies E\langle n:x \rangle m = E(m-1) \rangle$
by (*simp add: shift-def*)

lemma *shift-commute* [*simp*]: $\langle (E\langle n:y \rangle\langle 0:x \rangle) = (E\langle 0:x \rangle\langle n+1:y \rangle) \rangle$

proof

fix *m*

show $\langle (E\langle n:y \rangle\langle 0:x \rangle) m = (E\langle 0:x \rangle\langle n+1:y \rangle) m \rangle$

unfolding *shift-def* by (*cases m*) *simp-all*

qed

end

3.3.2 Parameters

abbreviation $\langle \text{params } S \equiv \bigcup p \in S. \text{params-fm } p \rangle$

lemma *upd-params-sym* [*simp*]: $\langle f \notin \text{params-sym } fn \implies \langle E_F, F(f := x) \rangle_2 fn = \langle E_F, F \rangle_2 fn \rangle$
by (*induct fn*) (*auto cong: map-cong*)

lemma *upd-params-tm* [*simp*]: $\langle f \notin \text{params-tm } t \implies \langle E, E_F, C, F(f := x) \rangle t = \langle E, E_F, C, F \rangle t \rangle$
by (*induct t*) (*auto cong: map-cong*)

lemma *upd-params-tm-c* [*simp*]: $\langle c \notin \text{params-tm } t \implies \langle E, E_F, C(c := x), F \rangle t = \langle E, E_F, C, F \rangle t \rangle$
by (*induct t*) (*auto cong: map-cong*)

lemma *upd-params-fm* [*simp*]: $\langle f \notin \text{params-fm } p \implies (E, E_F, E_P, C, F(f := x), G, PS, FS) \models p \longleftrightarrow (E, E_F, E_P, C, F, G, PS, FS) \models p \rangle$
by (*induct p arbitrary: E E_P E_F*) (*auto cong: map-cong*)

lemma *upd-params-fm-c* [*simp*]: $\langle c \notin \text{params-fm } p \implies (E, E_F, E_P, C(c := x), F, G, PS, FS) \models p \longleftrightarrow (E, E_F, E_P, C, F, G, PS, FS) \models p \rangle$
by (*induct p arbitrary: E E_P E_F*) (*auto cong: map-cong*)

lemma *upd-params-fm-G* [*simp*]: $\langle P \notin \text{params-fm } p \implies (E, E_F, E_P, C, F, G(P := x), PS, FS) \models p \longleftrightarrow (E, E_F, E_P, C, F, G, PS, FS) \models p \rangle$
by (*induct p arbitrary: E E_F E_P*) (*auto cong: map-cong*)

lemma *finite-params-sym* [*simp*]: $\langle \text{finite } (\text{params-sym } fn) \rangle$
by (*induct fn*) *simp-all*

lemma *finite-params-tm* [*simp*]: $\langle \text{finite } (\text{params-tm } t) \rangle$
by (*induct t*) *simp-all*

lemma *finite-params-fm* [*simp*]: $\langle \text{finite } (\text{params-fm } p) \rangle$

by (induct p) simp-all

3.3.3 Instantiation

primrec lift-tm ($\langle \uparrow \rangle$) where

$$\begin{aligned} & \langle \uparrow (\#n) = \#(n+1) \rangle \\ | & \langle \uparrow (\text{Of } ts) = \text{Of } (\text{map } \uparrow \text{ } ts) \rangle \\ | & \langle \uparrow (\star c) = \star c \rangle \end{aligned}$$

primrec lift-sym ($\langle \uparrow_2 \rangle$) where

$$\begin{aligned} & \langle \uparrow_2 (\#_2 n) = \#_2 (n+1) \rangle \\ | & \langle \uparrow_2 (\text{O}_2 p) = \text{O}_2 p \rangle \end{aligned}$$

primrec lift-fn ($\langle \uparrow_F \rangle$) where

$$\begin{aligned} & \langle \uparrow_F (\#n) = \#n \rangle \\ | & \langle \uparrow_F (\text{Of } ts) = \text{O}(\uparrow_2 f) (\text{map } \uparrow_F \text{ } ts) \rangle \\ | & \langle \uparrow_F (\star c) = \star c \rangle \end{aligned}$$

primrec inst-tm ($\langle \langle -' / - \rangle \rangle$) where

$$\begin{aligned} & \langle \langle s/m \rangle (\#n) = (\text{if } n < m \text{ then } \#n \text{ else if } n = m \text{ then } s \text{ else } \#(n-1)) \rangle \\ | & \langle \langle s/m \rangle (\text{Of } ts) = \text{Of } (\text{map } \langle s/m \rangle \text{ } ts) \rangle \\ | & \langle \langle s/m \rangle (\star c) = \star c \rangle \end{aligned}$$

primrec inst-sym ($\langle \langle -' / - \rangle_2 \rangle$) where

$$\begin{aligned} & \langle \langle s/m \rangle_2 (\#_2 n) = (\text{if } n < m \text{ then } \#_2 n \text{ else if } n = m \text{ then } s \text{ else } \#_2 (n-1)) \rangle \\ | & \langle \langle s/m \rangle_2 (\text{O}_2 p) = \text{O}_2 p \rangle \end{aligned}$$

primrec inst-fn ($\langle \langle -' / - \rangle_F \rangle$) where

$$\begin{aligned} & \langle \langle s/m \rangle_F (\#n) = (\#n) \rangle \\ | & \langle \langle s/m \rangle_F (\text{Of } ts) = \text{O}(\langle s/m \rangle_2 f) (\text{map } \langle s/m \rangle_F \text{ } ts) \rangle \\ | & \langle \langle s/m \rangle_F (\star c) = (\star c) \rangle \end{aligned}$$

primrec inst-fm ($\langle \langle -' / - \rangle \rangle$) where

$$\begin{aligned} & \langle \langle -' / - \rangle \perp = \perp \rangle \\ | & \langle \langle s/m \rangle (\cdot P \text{ } ts) = \cdot P (\text{map } \langle s/m \rangle \text{ } ts) \rangle \\ | & \langle \langle s/m \rangle (p \longrightarrow q) = \langle s/m \rangle p \longrightarrow \langle s/m \rangle q \rangle \\ | & \langle \langle s/m \rangle (\forall p) = \forall (\langle \uparrow s/m+1 \rangle p) \rangle \\ | & \langle \langle s/m \rangle (\forall_P p) = \forall_P (\langle s/m \rangle p) \rangle \\ | & \langle \langle s/m \rangle (\forall_F p) = \forall_F (\langle \uparrow_F s/m \rangle p) \rangle \end{aligned}$$

primrec inst-fm-P ($\langle \langle -' / - \rangle_P \rangle$) where

$$\begin{aligned} & \langle \langle -' / - \rangle_P \perp = \perp \rangle \\ | & \langle \langle s/m \rangle_P (\cdot P \text{ } ts) = \cdot (\langle s/m \rangle_2 P) \text{ } ts \rangle \\ | & \langle \langle s/m \rangle_P (p \longrightarrow q) = \langle s/m \rangle_P p \longrightarrow \langle s/m \rangle_P q \rangle \\ | & \langle \langle s/m \rangle_P (\forall p) = \forall (\langle s/m \rangle_P p) \rangle \\ | & \langle \langle s/m \rangle_P (\forall_P p) = \forall_P (\langle \uparrow_2 s/m+1 \rangle_P p) \rangle \\ | & \langle \langle s/m \rangle_P (\forall_F p) = \forall_F (\langle s/m \rangle_P p) \rangle \end{aligned}$$

primrec inst-fm-F ($\langle \langle -' / - \rangle_F \rangle$) where

$$\begin{aligned}
& \langle \langle - / - \rangle_F \perp = \perp \rangle \\
& | \langle \langle s/m \rangle_F (\cdot P \text{ ts}) = \cdot P (\text{map } \langle \langle s/m \rangle_F \text{ ts} \rangle) \rangle \\
& | \langle \langle s/m \rangle_F (p \longrightarrow q) = \langle s/m \rangle_F p \longrightarrow \langle s/m \rangle_F q \rangle \\
& | \langle \langle s/m \rangle_F (\forall p) = \forall (\langle s/m \rangle_F p) \rangle \\
& | \langle \langle s/m \rangle_F (\forall_P p) = \forall_P (\langle s/m \rangle_F p) \rangle \\
& | \langle \langle s/m \rangle_F (\forall_F p) = \forall_F (\langle \uparrow_2 s/m+1 \rangle_F p) \rangle
\end{aligned}$$

lemma *lift-lemma* [simp]: $\langle \langle E \langle 0:x \rangle, E_F, C, F \rangle (\uparrow t) = \langle E, E_F, C, F \rangle t \rangle$
by (*induct t*) (*auto cong: map-cong*)

lemma *lift-lemma-P* [simp]: $\langle \langle E_P \langle 0:x \rangle, G \rangle_2 (\uparrow_2 P) = \langle E_P, G \rangle_2 P \rangle$
by (*induct P*) (*auto cong: map-cong*)

lemma *lift-lemma-F* [simp]: $\langle \langle E, E_F \langle 0:x \rangle, C, F \rangle (\uparrow_F tm) = \langle E, E_F, C, F \rangle tm \rangle$
by (*induct tm*) (*auto cong: map-cong*)

lemma *inst-tm-semantic* [simp]: $\langle \langle E, E_F, C, F \rangle (\langle \langle s/m \rangle t) = \langle E \langle m: \langle E, E_F, C, F \rangle s \rangle, E_F, C, F \rangle t \rangle$
by (*induct t*) (*auto cong: map-cong*)

lemma *inst-sym-semantic* [simp]: $\langle \langle E_F, G \rangle_2 (\langle \langle s/m \rangle_2 fn) = \langle E_F \langle m: \langle E_F, G \rangle_2 s \rangle, G \rangle_2 fn \rangle$
by (*induct fn*) (*auto cong: map-cong*)

lemma *inst-tm-semantic-F* [simp]: $\langle \langle E, E_F, C, F \rangle (\langle \langle s/m \rangle_F t) = \langle E, E_F \langle m: \langle E_F, F \rangle_2 s \rangle, C, F \rangle t \rangle$
by (*induct t*) (*auto cong: map-cong*)

lemma *inst-fm-semantic-F* [simp]:
 $\langle (E, E_F, E_P, C, F, G, PS, FS) \models (\langle t/m \rangle_F p) \longleftrightarrow (E, E_F \langle m: \langle E_F, F \rangle_2 t \rangle, E_P, C, F, G, PS, FS) \models p \rangle$
by (*induct p arbitrary: E E_P E_F m t*) (*auto cong: map-cong*)

lemma *inst-fm-semantic* [simp]:
 $\langle (E, E_F, E_P, C, F, G, PS, FS) \models (\langle t/m \rangle p) \longleftrightarrow (E \langle m: \langle E, E_F, C, F \rangle t \rangle, E_F, E_P, C, F, G, PS, FS) \models p \rangle$
by (*induct p arbitrary: E E_P E_F m t*) (*auto cong: map-cong*)

lemma *inst-fm-semantic-P* [simp]: $\langle (E, E_F, E_P, C, F, G, PS, FS) \models (\langle P/m \rangle_P p) \longleftrightarrow (E, E_F, E_P \langle m: \langle E_P, G \rangle_2 P \rangle, C, F, G, PS, FS) \models p \rangle$
by (*induct p arbitrary: E E_F E_P m P*) (*auto cong: map-cong*)

3.3.4 Size

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

primrec *size-fm* **where**

$$\begin{aligned}
& \langle \text{size-fm } \perp = 1 \rangle \\
& | \langle \text{size-fm } (\cdot -) = 1 \rangle \\
& | \langle \text{size-fm } (p \longrightarrow q) = 1 + \text{size-fm } p + \text{size-fm } q \rangle
\end{aligned}$$

| $\langle \text{size-fm } (\forall p) = 1 + \text{size-fm } p \rangle$
| $\langle \text{size-fm } (\forall_P p) = 1 + \text{size-fm } p \rangle$
| $\langle \text{size-fm } (\forall_F p) = 1 + \text{size-fm } p \rangle$

lemma *size-inst-fm* [*simp*]: $\langle \text{size-fm } (\langle t/m \rangle p) = \text{size-fm } p \rangle$
by (*induct p arbitrary: m t simp-all*)

lemma *size-inst-fm-P* [*simp*]: $\langle \text{size-fm } (\langle t/m \rangle_P p) = \text{size-fm } p \rangle$
by (*induct p arbitrary: m t simp-all*)

lemma *size-inst-fm-F* [*simp*]: $\langle \text{size-fm } (\langle t/m \rangle_F p) = \text{size-fm } p \rangle$
by (*induct p arbitrary: m t simp-all*)

3.4 Model Existence

inductive *confl-class* :: $\langle 'f \text{ fm list} \Rightarrow 'f \text{ fm list} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\mathbf{x}} \rangle$ 50) **where**
CFIs: $\langle [\perp] \rightsquigarrow_{\mathbf{x}} [\perp] \rangle$
| *CNeg*: $\langle [\neg (\cdot P \text{ ts})] \rightsquigarrow_{\mathbf{x}} [\cdot P \text{ ts}] \rangle$

inductive *alpha-class* :: $\langle 'f \text{ fm list} \Rightarrow 'f \text{ fm list} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\alpha} \rangle$ 50) **where**
CImpN: $\langle [\neg (p \longrightarrow q)] \rightsquigarrow_{\alpha} [p, \neg q] \rangle$

inductive *beta-class* :: $\langle 'f \text{ fm list} \Rightarrow 'f \text{ fm list} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\beta} \rangle$ 50) **where**
CImpP: $\langle [p \longrightarrow q] \rightsquigarrow_{\beta} [\neg p, q] \rangle$

inductive *gamma-class* :: $\langle 'f \text{ fm list} \Rightarrow ('f \text{ tm} \Rightarrow 'f \text{ fm list}) \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\gamma} \rangle$ 50) **where**
CALLP: $\langle [\forall p] \rightsquigarrow_{\gamma} (\lambda t. [\langle t/0 \rangle p]) \rangle$

inductive *gamma-class-P* :: $\langle 'f \text{ fm list} \Rightarrow ('f \text{ sym} \Rightarrow 'f \text{ fm list}) \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\gamma_P} \rangle$ 50) **where**
CALLPP: $\langle [\forall_P p] \rightsquigarrow_{\gamma_P} (\lambda s. [\langle s/0 \rangle_P p]) \rangle$

inductive *gamma-class-F* :: $\langle 'f \text{ fm list} \Rightarrow ('f \text{ sym} \Rightarrow 'f \text{ fm list}) \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\gamma_F} \rangle$ 50) **where**
CALLFP: $\langle [\forall_F p] \rightsquigarrow_{\gamma_F} (\lambda s. [\langle s/0 \rangle_F p]) \rangle$

fun δ :: $\langle 'f \text{ fm} \Rightarrow 'f \Rightarrow 'f \text{ fm list} \rangle$ **where**
CALLN: $\langle \delta (\neg \forall p) x = [\neg \langle \star x/0 \rangle p] \rangle$
| *CALL2PN*: $\langle \delta (\neg \forall_P p) x = [\neg \langle \bigcirc_2 x/0 \rangle_P p] \rangle$
| *CALL2FN*: $\langle \delta (\neg \forall_F p) x = [\neg \langle \bigcirc_2 x/0 \rangle_F p] \rangle$
| *NOMATCH*: $\langle \delta - - = [] \rangle$

interpretation *P*: *Params map-fm params-fm* $\langle \lambda -. \text{True} \rangle$
by *unfold-locales* (*auto simp: tm.map-id0 fm.map-id0 cong: tm.map-cong0 fm.map-cong0*)

interpretation *C*: *Confl map-fm params-fm* $\langle \lambda -. \text{True} \rangle$ *confl-class*
by *unfold-locales* (*auto elim!: confl-class.cases intro: confl-class.intros*)

interpretation *A*: *Alpha map-fm params-fm* $\langle \lambda-. True \rangle$ *alpha-class*
 by *unfold-locales* (*auto simp*: *fm.map-id0 cong*: *fm.map-cong0 elim*!: *alpha-class.cases*
intro: *alpha-class.intros*)

interpretation *B*: *Beta map-fm params-fm* $\langle \lambda-. True \rangle$ *beta-class*
 by *unfold-locales* (*auto simp*: *fm.map-id0 cong*: *fm.map-cong0 elim*!: *beta-class.cases*
intro: *beta-class.intros*)

lemma *map-tm-inst-tm* [*simp*]:
 $map\text{-}tm\ f\ (\langle\langle t/n \rangle\rangle\ x) = \langle\langle map\text{-}tm\ f\ t/n \rangle\rangle\ (map\text{-}tm\ f\ x)$
 by (*induct* *x*) *simp-all*

lemma *map-tm-lift-tm* [*simp*]: $map\text{-}tm\ f\ (\uparrow t) = \uparrow (map\text{-}tm\ f\ t)$
 by (*induct* *t*) *simp-all*

lemma *map-sym-lift-fn* [*simp*]: $\langle map\text{-}sym\ f\ (\uparrow_2 t) = \uparrow_2 (map\text{-}sym\ f\ t) \rangle$
 by (*induct* *t*) *auto*

lemma *map-tm-lift-fn* [*simp*]: $map\text{-}tm\ f\ (\uparrow_F t) = \uparrow_F (map\text{-}tm\ f\ t)$
 by (*induct* *t*) *simp-all*

lemma *map-fm-inst-single* [*simp*]: $\langle map\text{-}fm\ f\ (\langle t/m \rangle p) = \langle map\text{-}tm\ f\ t/m \rangle (map\text{-}fm\ f\ p) \rangle$
 by (*induct* *p* *arbitrary*: *t m*) *simp-all*

lemma *map-sym-inst-sym* [*simp*]: $\langle map\text{-}sym\ f\ (\langle\langle t/m \rangle\rangle_2 p) = \langle\langle map\text{-}sym\ f\ t/m \rangle\rangle_2 (map\text{-}sym\ f\ p) \rangle$
 by (*induct* *p* *arbitrary*: *t m*) *simp-all*

lemma *psub-inst-single'* [*simp*]: $\langle map\text{-}fm\ f\ (\langle t/m \rangle_P p) = \langle map\text{-}sym\ f\ t/m \rangle_P (map\text{-}fm\ f\ p) \rangle$
 by (*induct* *p* *arbitrary*: *t m*) *simp-all*

lemma *map-tm-inst-fn* [*simp*]: $\langle map\text{-}tm\ f\ (\langle\langle t/m \rangle\rangle_F s) = \langle\langle map\text{-}sym\ f\ t/m \rangle\rangle_F (map\text{-}tm\ f\ s) \rangle$
 by (*induct* *s*) *auto*

lemma *psub-inst-single''* [*simp*]: $\langle map\text{-}fm\ f\ (\langle t/m \rangle_F p) = \langle map\text{-}sym\ f\ t/m \rangle_F (map\text{-}fm\ f\ p) \rangle$
 by (*induct* *p* *arbitrary*: *t m*) *simp-all*

interpretation *G*: *Gamma-UNIV map-tm map-fm params-fm* $\langle \lambda-. True \rangle$ *gamma-class*
 by *unfold-locales* (*fastforce elim*: *gamma-class.cases* *intro*: *gamma-class.intros*)+

interpretation *G_P*: *Gamma-UNIV map-sym map-fm params-fm* $\langle \lambda-. True \rangle$ *gamma-class-P*
 by *unfold-locales* (*fastforce elim*: *gamma-class-P.cases* *intro*: *gamma-class-P.intros*)+

interpretation *G_F*: *Gamma-UNIV map-sym map-fm params-fm* $\langle \lambda-. True \rangle$ *gamma-class-F*
 by *unfold-locales* (*fastforce elim*: *gamma-class-F.cases* *intro*: *gamma-class-F.intros*)+

interpretation D : *Delta map-fm params-fm* $\langle \lambda-. True \rangle \delta$

proof

show $\langle \wedge f. \delta (map\text{-}fm\ f\ p) (f\ x) = map (map\text{-}fm\ f) (\delta\ p\ x) \rangle$ **for** $p :: \langle 'x\ fm \rangle$ **and**
 x

by (*induct p x rule: $\delta.induct$*) *simp-all*

qed

abbreviation $Kinds :: \langle ('x, 'x\ fm)\ kind\ list \rangle$ **where**

$\langle Kinds \equiv [C.kind, A.kind, B.kind, G.kind, G_P.kind, G_F.kind, D.kind] \rangle$

lemma *prop_E-Kinds*:

assumes $\langle P.sat_E\ C.kind\ C \rangle \langle P.sat_E\ A.kind\ C \rangle \langle P.sat_E\ B.kind\ C \rangle \langle P.sat_E\ G.kind\ C \rangle \langle P.sat_E\ G_P.kind\ C \rangle$

$\langle P.sat_E\ G_F.kind\ C \rangle \langle P.sat_E\ D.kind\ C \rangle$

shows $\langle P.prop_E\ Kinds\ C \rangle$

unfolding $P.prop_E\text{-}def$ **using** *assms* **by** *simp*

interpretation *Consistency-Kinds map-fm params-fm* $\langle \lambda-. True \rangle Kinds$

using $P.Params\text{-}axioms\ C.Consistency\text{-}Kind\text{-}axioms\ A.Consistency\text{-}Kind\text{-}axioms$
 $B.Consistency\text{-}Kind\text{-}axioms$

$G.Consistency\text{-}Kind\text{-}axioms\ G_P.Consistency\text{-}Kind\text{-}axioms\ G_F.Consistency\text{-}Kind\text{-}axioms$

$D.Consistency\text{-}Kind\text{-}axioms$

by (*auto intro: Consistency-Kinds.intro simp: Consistency-Kinds-axioms-def*)

interpretation *Maximal-Consistency map-fm params-fm* $\langle \lambda-. True \rangle Kinds$

proof

show $\langle infinite\ (UNIV :: 'x\ fm\ set) \rangle$

using *infinite-UNIV-size*[of $\langle \lambda p. p \longrightarrow p \rangle$] **by** *simp*

qed *simp*

abbreviation $henv_P$ **where** $henv_P\ H == \lambda n\ ts. \cdot(\#_2\ n)\ ts \in H$

abbreviation $hpred$ **where** $hpred\ H == \lambda P\ ts. \cdot(\circ_2\ P)\ ts \in H$

abbreviation $hdom_P$ **where** $hdom_P\ H == range\ (henv_P\ H) \cup range\ (hpred\ H)$

abbreviation $henv_F$ **where** $henv_F == \lambda f. \circ(\#_2\ f)$

abbreviation $hfun$ **where** $hfun == \lambda f. \circ(\circ_2\ f)$

definition $hdom_F$ **where** $hdom_F == range\ henv_F \cup range\ hfun$

abbreviation (*input*) $hmodel\ (\langle \llbracket - \rrbracket \rangle)$ **where** $\langle \llbracket H \rrbracket \equiv (\#, henv_F, henv_P\ H, \star, hfun,$
 $hpred\ H, hdom_P\ H, hdom_F) \rangle$

lemma *semantics-tm-id* [*simp*]: $\langle \llbracket \#, henv_F, \star, \lambda f. \circ(\circ_2\ f) \rrbracket \vdash t = t \rangle$

proof (*induct t*)

case ($Var\ x$)

```

then show ?case
  by (auto cong: map-cong)
next
case (Fun x1a x2)
then show ?case
  by (cases x1a) (auto cong: map-cong)
next
case (Cst c)
then show ?case
  by auto
qed

```

lemma *semantics-tm-id-map* [simp]: $\langle \text{map } \langle \#, \lambda f. \circ (\#_2 f) \rangle, \star, \lambda f. \circ (\circ_2 f) \rangle \Downarrow$
 $ts = ts \rangle$
 by (auto cong: map-cong)

lemma *semantics-fn-h* [simp]: $\langle \langle \text{henv}_P S, \text{hpred } S \rangle_2 P ts \longleftrightarrow \cdot P ts \in S \rangle$
 by (cases P) simp-all

lemma *canonical-henv_P*:
 $\langle \forall t. (\#, \text{henv}_F, (\text{henv}_P S) \langle 0: \langle \text{henv}_P S, \text{hpred } S \rangle_2 t \rangle, \star, \text{hfun}, \text{hpred } S, \text{hdom}_P S, \text{hdom}_F) \models p \implies$
 $(\#, \text{henv}_F, (\text{henv}_P S) \langle 0: \text{henv}_P S n \rangle, \star, \text{hfun}, \text{hpred } S, \text{hdom}_P S, \text{hdom}_F) \models p \rangle$
 by (metis semantics-fn.simps(1))

lemma *canonical-hpred*:
 $\langle \forall t. (\#, \text{henv}_F, (\text{henv}_P S) \langle 0: \langle \text{henv}_P S, \text{hpred } S \rangle_2 t \rangle, \star, \text{hfun}, \text{hpred } S, \text{hdom}_P S, \text{hdom}_F) \models p \implies$
 $(\#, \text{henv}_F, (\text{henv}_P S) \langle 0: \text{hpred } S P \rangle, \star, \text{hfun}, \text{hpred } S, \text{hdom}_P S, \text{hdom}_F) \models p \rangle$
 by (metis semantics-fn.simps(2))

lemma *canonical-henv_F*:
 $\langle \forall t. (\#, \text{henv}_F \langle 0: \langle \text{henv}_F, \text{hfun} \rangle_2 t \rangle, \text{henv}_P S, \star, \text{hfun}, \text{hpred } S, \text{hdom}_P S, \text{hdom}_F) \models p \implies$
 $(\#, \text{henv}_F \langle 0: \text{henv}_F f \rangle, \text{henv}_P S, \star, \text{hfun}, \text{hpred } S, \text{hdom}_P S, \text{hdom}_F) \models p \rangle$
 by (metis semantics-fn.simps(1))

lemma *canonical-hfun*:
 $\langle \forall t. (\#, \text{henv}_F \langle 0: \langle \text{henv}_F, \text{hfun} \rangle_2 t \rangle, \text{henv}_P S, \star, \text{hfun}, \text{hpred } S, \text{hdom}_P S, \text{hdom}_F) \models p \implies$
 $(\#, \text{henv}_F \langle 0: \text{hfun } f \rangle, \text{henv}_P S, \star, \text{hfun}, \text{hpred } S, \text{hdom}_P S, \text{hdom}_F) \models p \rangle$
 by (metis semantics-fn.simps(2))

locale *MyHintikka* = *Hintikka* map-fm params-fm $\langle \lambda \cdot. \text{True} \rangle$ *Kinds* S **for** S :: $\langle 'x$
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*: $\langle (p \in S \longrightarrow \llbracket S \rrbracket \models p) \wedge (\neg p \in S \longrightarrow \neg \llbracket S \rrbracket \models p) \rangle$

proof (*induct p rule: wf-induct*[where $r = \langle \text{measure size-fm} \rangle$])

case 1

then show *?case*

by *simp*

next

case (2 *x*)

then show *?case*

proof (*cases x*)

case *Falsity*

then show *?thesis*

using *confl* **by** (*force intro: CFLs*)

next

case (*Pre P ts*)

then show *?thesis*

proof (*safe del: notI*)

assume $\langle x = \cdot P \text{ ts} \rangle \langle \cdot P \text{ ts} \in S \rangle$

then show $\langle \llbracket S \rrbracket \models (\cdot P \text{ ts}) \rangle$

by *simp*

next

assume $\langle x = \cdot P \text{ ts} \rangle \langle \neg \cdot P \text{ ts} \in S \rangle$

then have $\langle \cdot P \text{ ts} \notin S \rangle$

using *confl* **by** (*force intro: CNeg*)

then show $\langle \neg \llbracket S \rrbracket \models (\cdot P \text{ ts}) \rangle$

by *simp*

qed

next

case (*Imp p q*)

then show *?thesis*

proof (*safe del: notI*)

assume $\langle x = p \longrightarrow q \rangle \langle p \longrightarrow q \in S \rangle$

then have $\langle \neg p \in S \vee q \in S \rangle$

using *beta* **by** (*force intro: CImpP*)

then show $\langle \llbracket S \rrbracket \models (p \longrightarrow q) \rangle$

using 2 *Imp* **by** *auto*

next

assume $\langle x = p \longrightarrow q \rangle \langle \neg (p \longrightarrow q) \in S \rangle$

then have $\langle p \in S \wedge \neg q \in S \rangle$

using *alpha* **by** (*force intro: CImpN*)

then show $\langle \neg \llbracket S \rrbracket \models (p \longrightarrow q) \rangle$

```

    using 2 Imp by auto
qed
next
case (Uni p)
then show ?thesis
proof (safe del: notI)
  assume  $\langle x = \forall p \rangle \langle \forall p \in S \rangle$ 
  then have  $\langle \forall t. \langle t/0 \rangle p \in S \rangle$ 
    using gammaFO by (fastforce intro: CALLP)
  moreover have  $\langle \forall t. (\langle t/0 \rangle p, \forall p) \in \text{measure size-fm} \rangle$ 
    by simp
  ultimately have  $\langle \forall t. \llbracket S \rrbracket \models (\langle t/0 \rangle p) \rangle$ 
    using 2  $\langle x = \forall p \rangle$  by blast
  then show  $\langle \llbracket S \rrbracket \models (\forall p) \rangle$ 
    by simp
next
assume  $\langle x = \forall p \rangle \langle \neg \forall p \in S \rangle$ 
then obtain a where  $\langle \neg \langle \star a/0 \rangle p \in S \rangle$ 
  using delta by auto
moreover have  $\langle (\langle \star a/0 \rangle p, \forall p) \in \text{measure size-fm} \rangle$ 
  by simp
ultimately have  $\langle \neg \llbracket S \rrbracket \models (\langle \star a/0 \rangle p) \rangle$ 
  using 2  $\langle x = \forall p \rangle$  by blast
then show  $\langle \neg \llbracket S \rrbracket \models (\forall p) \rangle$ 
  by auto
qed
next
case (UniP p)
then show ?thesis
proof (safe del: notI)
  assume  $\langle x = \forall_P p \rangle \langle \forall_P p \in S \rangle$ 
  then have  $\langle \forall t. \langle t/0 \rangle_P p \in S \rangle$ 
    using gamma2P by (fastforce intro: CALLPP)
  moreover have *:  $\langle \forall t. (\langle t/0 \rangle_P p, \forall_P p) \in \text{measure size-fm} \rangle$ 
    by simp
  ultimately have  $\langle \forall t. \llbracket S \rrbracket \models (\langle t/0 \rangle_P p) \rangle$ 
    using 2  $\langle x = \forall_P p \rangle$  by blast
  then show  $\langle \llbracket S \rrbracket \models \forall_P p \rangle$ 
    using canonical-henvP canonical-hpred by auto
next
assume  $\langle x = \forall_P p \rangle \langle \neg \forall_P p \in S \rangle$ 
then obtain a where  $\langle \neg \langle \circ_2 a/0 \rangle_P p \in S \rangle$ 
  using delta by auto
moreover have  $\langle (\langle \circ_2 a/0 \rangle_P p, \forall_P p) \in \text{measure size-fm} \rangle$ 
  by simp
ultimately have  $\langle \neg \llbracket S \rrbracket \models (\langle \circ_2 a/0 \rangle_P p) \rangle$ 
  using 2  $\langle x = \forall_P p \rangle$  by blast
then show  $\langle \neg \llbracket S \rrbracket \models (\forall_P p) \rangle$ 
  by auto

```

```

qed
next
case (UniF p)
then show ?thesis
proof (safe del: notI)
  assume ⟨x = ∀F p⟩ ⟨∀F p ∈ S⟩
  then have ⟨∀ t. ⟨t/0⟩F p ∈ S⟩
    using gamma2F by (fastforce intro: CALLFP)
  moreover have ⟨∀ t. (⟨t/0⟩F p, ∀F p) ∈ measure size-fm⟩
    by simp
  ultimately have ⟨∀ t. ⟦S⟧ ⊨ (⟨t/0⟩F p)⟩
    using 2 ⟨x = ∀F p⟩ by blast
  then show ⟨⟦S⟧ ⊨ (∀F p)⟩
    using canonical-henvF canonical-hfun unfolding hdomF-def by auto
next
  assume ⟨x = ∀F p⟩ ⟨¬ ∀F p ∈ S⟩
  then obtain a where ⟨¬ ⟨O2 a/0⟩F p ∈ S⟩
    using delta by auto
  moreover have ⟨(⟨O2 a/0⟩F p, ∀F p) ∈ measure size-fm⟩
    by simp
  ultimately have ⟨¬ ⟦S⟧ ⊨ (⟨O2a/0⟩F p)⟩
    using 2 ⟨x = ∀F p⟩ by blast
  then show ⟨¬ ⟦S⟧ ⊨ (∀F p)⟩
    by (auto simp: hdomF-def)
qed
qed
qed

end

theorem model-existence:
  fixes S :: ⟨'x fm set⟩
  assumes ⟨P.propE Kinds C⟩
    and ⟨S ∈ C⟩
    and ⟨P.enough-new S⟩
    and ⟨p ∈ S⟩
  shows ⟨⟦mk-mcs C S⟧ ⊨ p⟩
proof -
  have *: ⟨MyHintikka (mk-mcs C S)⟩
  proof
    show ⟨P.propH Kinds (mk-mcs C S)⟩
      using mk-mcs-Hintikka[OF assms(1-3)] Hintikka.hintikka by blast
  qed
  moreover have ⟨p ∈ mk-mcs C S⟩
    using assms(4) Extend-subset by blast
  ultimately show ?thesis
    using MyHintikka.model by blast
qed

```

3.5 Propositional Semantics

primrec *boolean where*

$\langle \text{boolean } - \cdot \perp = \text{False} \rangle$
 $| \langle \text{boolean } G \cdot (\cdot P \text{ ts}) = G P \text{ ts} \rangle$
 $| \langle \text{boolean } G A (p \longrightarrow q) = (\text{boolean } G A p \longrightarrow \text{boolean } G A q) \rangle$
 $| \langle \text{boolean } - A (\forall p) = A (\forall p) \rangle$
 $| \langle \text{boolean } - A (\forall_P p) = A (\forall_P p) \rangle$
 $| \langle \text{boolean } - A (\forall_F p) = A (\forall_F p) \rangle$

abbreviation $\langle \text{tautology } p \equiv \forall G A. \text{boolean } G A p \rangle$

proposition $\langle \text{tautology } (\forall (\cdot P [\#0]) \longrightarrow \forall (\cdot P [\#0])) \rangle$
by *simp*

lemma *boolean-semantics*: $\langle \text{boolean } (\lambda a. \langle E_P, G \rangle_2 a \circ \text{map } \langle E, E_F, C, F \rangle) (\lambda p. (E, E_F, E_P, C, F, G, PS, FS) \models p) = (\lambda p. (E, E_F, E_P, C, F, G, PS, FS) \models p) \rangle$

proof

fix p
show $\langle \text{boolean } (\lambda a. \langle E_P, G \rangle_2 a \circ \text{map } \langle E, E_F, C, F \rangle) ((\models) (E, E_F, E_P, C, F, G, PS, FS)) p = ((E, E_F, E_P, C, F, G, PS, FS) \models p) \rangle$
by (*induct p simp-all*)

qed

lemma *tautology[simp]*: $\langle \text{tautology } p \implies (E, E_F, E_P, C, F, G, PS, FS) \models p \rangle$
using *boolean-semantics by metis*

proposition $\langle \exists p. (\forall E E_F E_P C F G PS FS. (E, E_F, E_P, C, F, G, PS, FS) \models p) \wedge \neg \text{tautology } p \rangle$

by (*metis boolean.simps(4) fun-upd-same semantics-fm.simps(3) semantics-fm.simps(4) tautology*)

3.6 Calculus

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

inductive *Axiomatic* ($\langle \vdash \rightarrow [50] 50 \rangle$) **where**

$TA: \langle \text{tautology } p \implies \vdash p \rangle$
 $| IA: \langle \vdash \forall p \longrightarrow \langle t/0 \rangle p \rangle$
 $| IA_P: \langle \vdash \forall_P p \longrightarrow \langle s/0 \rangle_P p \rangle$
 $| IA_F: \langle \vdash \forall_F p \longrightarrow \langle s/0 \rangle_F p \rangle$
 $| MP: \langle \vdash p \longrightarrow q \implies \vdash p \implies \vdash q \rangle$
 $| GR: \langle \vdash q \longrightarrow \langle \star a/0 \rangle p \implies a \notin \text{params } \{p, q\} \implies \vdash q \longrightarrow \forall p \rangle$
 $| GR_P: \langle \vdash q \longrightarrow \langle \circ_2 a/0 \rangle_P p \implies a \notin \text{params } \{p, q\} \implies \vdash q \longrightarrow \forall_P p \rangle$
 $| GR_F: \langle \vdash q \longrightarrow \langle \circ_2 a/0 \rangle_F p \implies a \notin \text{params } \{p, q\} \implies \vdash q \longrightarrow \forall_F p \rangle$

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

primrec *imply* (**infixr** $\langle \rightsquigarrow \rangle$ 56) **where**

$\langle \langle \square \rightsquigarrow q \rangle = q \rangle$
 $| \langle (p \# ps \rightsquigarrow q) = (p \longrightarrow ps \rightsquigarrow q) \rangle$

abbreviation *Axiomatic-assms* ($\langle - \vdash - \rangle$ [50, 50] 50) **where**

$\langle ps \vdash q \equiv \vdash ps \rightsquigarrow q \rangle$

3.7 Soundness

fun *wf-model* **where**

$\langle wf\text{-model } (E, E_F, E_P, C, F, G, PS, FS) \longleftrightarrow range\ G \subseteq PS \wedge range\ E_P \subseteq PS$
 $\wedge range\ F \subseteq FS \wedge range\ E_F \subseteq FS \rangle$

theorem *soundness*:

shows $\langle \vdash p \implies wf\text{-model } (E, E_F, E_P, C, F, G, PS, FS) \implies (E, E_F, E_P, C,$
 $F, G, PS, FS) \models p \rangle$

proof (*induct p arbitrary: C F G PS FS rule: Axiomatic.induct*)

case (*TA p*)

then show *?case*

by *simp*

next

case (*IA p t*)

then show *?case*

by *auto*

next

case (*IA_P p s*)

then show *?case*

by (*cases s*) (*fastforce intro!: rangeI*)+

next

case (*IA_F p s*)

then show *?case*

by (*cases s*) (*fastforce intro!: rangeI*)+

next

case (*MP p q*)

then show *?case*

by *auto*

next

case (*GR q a p*)

moreover from *this have* $\langle (E, E_F, E_P, C(a := x), F, G, PS, FS) \models (q \longrightarrow$
 $\langle \star a / 0 \rangle p) \rangle$ **for** *x*

unfolding *wf-model.simps* **by** *meson*

ultimately show *?case*

by *fastforce*

next

case (*GR_P q a p*)

moreover from *this have* $\langle \forall x. x \in PS \longrightarrow (E, E_F, E_P, C, F, G(a := x),$
 $PS, FS) \models (q \longrightarrow \langle \circ_2 a / 0 \rangle_P p) \rangle$

unfolding *wf-model.simps*

by (*metis (no-types, opaque-lifting) fun-upd-other fun-upd-same image-subset-iff*)

ultimately show $?case$
 by *fastforce*
next
 case $(GR_F q a p)$
moreover from this have $\langle \forall x. x \in FS \longrightarrow (E, E_F, E_P, C, F(a := x), G, PS, FS) \models (q \longrightarrow \langle \bigcirc_2 a / 0 \rangle_{FP}) \rangle$
unfolding *wf-model.simps*
by $(metis (no-types, opaque-lifting) fun-upd-other fun-upd-same image-subset-iff)$
ultimately show $?case$
 by *fastforce*
qed

corollary $\langle \neg (\vdash \perp) \rangle$
using *soundness* [where $p = \perp$ and $G = \lambda p cs. True$ and $PS = \{\lambda cs. True\}$ and $E_P = \lambda n cs. True$ and $F = \lambda p cs. ()$] **by** *fastforce*

3.8 Derived Rules

lemma *Imp1*: $\langle \vdash q \longrightarrow p \longrightarrow q \rangle$
and *Imp2*: $\langle \vdash (p \longrightarrow q \longrightarrow r) \longrightarrow (p \longrightarrow q) \longrightarrow p \longrightarrow r \rangle$
and *Neg*: $\langle \vdash \neg \neg p \longrightarrow p \rangle$
by $(auto\ intro: TA)$

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

lemma *Tran'*: $\langle \vdash (q \longrightarrow r) \longrightarrow (p \longrightarrow q) \longrightarrow p \longrightarrow r \rangle$
by $(meson\ Imp1\ Imp2\ MP)$

lemma *Swap*: $\langle \vdash (p \longrightarrow q \longrightarrow r) \longrightarrow q \longrightarrow p \longrightarrow r \rangle$
by $(meson\ Imp1\ Imp2\ Tran'\ MP)$

lemma *Tran*: $\langle \vdash (p \longrightarrow q) \longrightarrow (q \longrightarrow r) \longrightarrow p \longrightarrow r \rangle$
by $(meson\ Swap\ Tran'\ MP)$

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

lemma *contraposition*: $\langle \vdash (\neg q \longrightarrow \neg p) \longrightarrow p \longrightarrow q \rangle$
by $(meson\ Neg\ Swap\ Tran\ MP)$

lemma *GR'*: $\langle \vdash \neg \langle \star a / 0 \rangle p \longrightarrow q \implies a \notin params\ \{p, q\} \implies \vdash \neg (\forall p) \longrightarrow q \rangle$
proof –

assume $*$: $\langle \vdash \neg \langle \star a / 0 \rangle p \longrightarrow q \rangle$ **and** a : $\langle a \notin params\ \{p, q\} \rangle$
have $\langle \vdash \neg q \longrightarrow \neg \neg \langle \star a / 0 \rangle p \rangle$
using $*$ *Tran MP* **by** *metis*
then have $\langle \vdash \neg q \longrightarrow \langle \star a / 0 \rangle p \rangle$
using *Neg Tran MP* **by** *metis*
then have $\langle \vdash \neg q \longrightarrow \forall p \rangle$
using a **by** $(auto\ intro: GR)$
then have $\langle \vdash \neg (\forall p) \longrightarrow \neg \neg q \rangle$

using *Tran MP by metis*
 then show *?thesis*
 using *Neg Tran MP by metis*
 qed

lemma GR_P' : $\langle \vdash \neg \langle \bigcirc_2 P / 0 \rangle_{Pp} \longrightarrow q \implies P \notin \text{params } \{p, q\} \implies \vdash \neg (\forall_{Pp}) \longrightarrow q \rangle$

proof –

assume *: $\langle \vdash \neg \langle \bigcirc_2 P / 0 \rangle_{Pp} \longrightarrow q \rangle$ and a : $\langle P \notin \text{params } \{p, q\} \rangle$
 have $\langle \vdash \neg q \longrightarrow \neg \neg \langle \bigcirc_2 P / 0 \rangle_{Pp} \rangle$
 using * *Tran MP by metis*
 then have $\langle \vdash \neg q \longrightarrow \langle \bigcirc_2 P / 0 \rangle_{Pp} \rangle$
 using *Neg Tran MP by metis*
 then have $\langle \vdash \neg q \longrightarrow \forall_{Pp} \rangle$
 using a by (*auto intro: GR_P*)
 then have $\langle \vdash \neg (\forall_{Pp}) \longrightarrow \neg \neg q \rangle$
 using *Tran MP by metis*
 then show *?thesis*
 using *Neg Tran MP by metis*
 qed

lemma GR_F' : $\langle \vdash \neg \langle \bigcirc_2 F / 0 \rangle_{Fp} \longrightarrow q \implies F \notin \text{params } \{p, q\} \implies \vdash \neg (\forall_{Fp}) \longrightarrow q \rangle$

proof –

assume *: $\langle \vdash \neg \langle \bigcirc_2 F / 0 \rangle_{Fp} \longrightarrow q \rangle$ and a : $\langle F \notin \text{params } \{p, q\} \rangle$
 have $\langle \vdash \neg q \longrightarrow \neg \neg \langle \bigcirc_2 F / 0 \rangle_{Fp} \rangle$
 using * *Tran MP by metis*
 then have $\langle \vdash \neg q \longrightarrow \langle \bigcirc_2 F / 0 \rangle_{Fp} \rangle$
 using *Neg Tran MP by metis*
 then have $\langle \vdash \neg q \longrightarrow \forall_{Fp} \rangle$
 using a by (*auto intro: GR_F*)
 then have $\langle \vdash \neg (\forall_{Fp}) \longrightarrow \neg \neg q \rangle$
 using *Tran MP by metis*
 then show *?thesis*
 using *Neg Tran MP by metis*
 qed

lemma *imply-ImpE*: $\langle \vdash ps \rightsquigarrow p \longrightarrow ps \rightsquigarrow (p \longrightarrow q) \longrightarrow ps \rightsquigarrow q \rangle$

proof (*induct ps*)

case *Nil*

then show *?case*

by (*metis Imp1 Swap MP imply.simps(1)*)

next

case (*Cons r ps*)

have $\langle \vdash (r \longrightarrow ps \rightsquigarrow p) \longrightarrow r \longrightarrow ps \rightsquigarrow (p \longrightarrow q) \longrightarrow ps \rightsquigarrow q \rangle$

by (*meson Cons.hyps Imp1 Imp2 MP*)

then have $\langle \vdash (r \longrightarrow ps \rightsquigarrow p) \longrightarrow (r \longrightarrow ps \rightsquigarrow (p \longrightarrow q)) \longrightarrow r \longrightarrow ps \rightsquigarrow q \rangle$

by (*meson Imp1 Imp2 MP*)

```

then show ?case
  by simp
qed

lemma MP':  $\langle ps \vdash p \longrightarrow q \implies ps \vdash p \implies ps \vdash q \rangle$ 
  using imply-ImpE MP by metis

lemma imply-Cons [intro]:  $\langle ps \vdash q \implies p \# ps \vdash q \rangle$ 
  by (auto intro: MP Imp1)

lemma imply-head [intro]:  $\langle p \# ps \vdash p \rangle$ 
  by (induct ps) (metis Imp1 Imp2 MP imply.simps, metis Imp1 Imp2 MP imply.simps(2))

lemma add-imply [simp]:  $\langle \vdash q \implies ps \vdash q \rangle$ 
  using imply-head by (metis MP imply.simps(2))

lemma imply-mem [simp]:  $\langle p \in set\ ps \implies ps \vdash p \rangle$ 
  using imply-head imply-Cons by (induct ps) fastforce+

lemma deduct1:  $\langle ps \vdash p \longrightarrow q \implies p \# ps \vdash q \rangle$ 
  by (meson MP' imply-Cons imply-head)

lemma imply-append [iff]:  $\langle (ps @ qs \rightsquigarrow r) = (ps \rightsquigarrow qs \rightsquigarrow r) \rangle$ 
  by (induct ps) simp-all

lemma imply-swap-append:  $\langle ps @ qs \vdash r \implies qs @ ps \vdash r \rangle$ 
proof (induct qs arbitrary: ps)
  case Cons
  then show ?case
    by (metis deduct1 imply.simps(2) imply-append)
qed simp

lemma deduct2:  $\langle p \# ps \vdash q \implies ps \vdash p \longrightarrow q \rangle$ 
  by (metis imply.simps imply-append imply-swap-append)

lemmas deduct [iff] = deduct1 deduct2

lemma cut:  $\langle p \# ps \vdash r \implies q \# ps \vdash p \implies q \# ps \vdash r \rangle$ 
  by (meson MP' deduct(2) imply-Cons)

lemma Boole:  $\langle (\neg p) \# ps \vdash \perp \implies ps \vdash p \rangle$ 
  by (meson MP' Neg add-imply deduct(2))

lemma imply-weaken:  $\langle ps \vdash q \implies set\ ps \subseteq set\ ps' \implies ps' \vdash q \rangle$ 
  by (induct ps arbitrary: q) (simp, metis MP' deduct(2) imply-mem insert-subset list.simps(15))

```

3.9 Derivational Consistency

interpretation *DC*: *Weak-Derivational-Confl map-fm params-fm* $\langle \lambda-. True \rangle$ *confl-class* $\langle \lambda A. \neg A \vdash \perp \rangle$

proof

fix A **ps** qs **and** $q :: \langle 'x \text{ fm} \rangle$
assume $\langle ps \rightsquigarrow_{\chi} qs \rangle$ $\langle set \ ps \subseteq set \ A \rangle$ $\langle q \in set \ qs \rangle$ $\langle q \in set \ A \rangle$
then show $\langle \neg \neg A \vdash \perp \rangle$
by cases (*simp*, *metis MP' empty-set equals0D imply-head imply-mem imply-weaken set-ConsD*)
qed

interpretation *DA*: *Weak-Derivational-Alpha map-fm params-fm* $\langle \lambda-. True \rangle$ *alpha-class* $\langle \lambda A. \neg A \vdash \perp \rangle$

proof (*standard*; *safe*)

fix A **and** ps $qs :: \langle 'x \text{ fm list} \rangle$
assume $\langle ps \rightsquigarrow_{\alpha} qs \rangle$ **and** $*$: $\langle set \ ps \subseteq set \ A \rangle$ $\langle \neg A \vdash \perp \rangle$ $\langle qs @ A \vdash \perp \rangle$
then show *False*
proof cases
case (*CImpN p q*)
then have $\langle A \vdash \neg (p \longrightarrow q) \rangle$
using $*(1)$ **by** *simp*
moreover have $\langle A \vdash p \longrightarrow q \rangle$
using *CImpN(2) * Boole[of q]*
by (*metis deduct2 imply.simps(1) imply.simps(2) imply-append*)
ultimately show *?thesis*
using *MP' ** **by** *blast*
qed
qed

interpretation *DB*: *Weak-Derivational-Beta map-fm params-fm* $\langle \lambda-. True \rangle$ *beta-class* $\langle \lambda A. \neg A \vdash \perp \rangle$

proof

fix A **and** ps $qs :: \langle 'x \text{ fm list} \rangle$
assume $\langle ps \rightsquigarrow_{\beta} qs \rangle$ **and** $*$: $\langle set \ ps \subseteq set \ A \rangle$ $\langle \neg A \vdash \perp \rangle$
then show $\langle \exists q \in set \ qs. \neg q \# A \vdash \perp \rangle$
proof cases
case (*CImpP p q*)
then show *?thesis*
using $* Boole[of p A]$
by (*metis MP' deduct2 imply-head imply-weaken insertI2 list.set-intros(1) list.simps(15)*)
qed
qed

interpretation *DG*: *Weak-Derivational-Gamma map-tm map-fm params-fm* $\langle \lambda-. True \rangle$ *G.classify-UNIV* $\langle \lambda A. \neg A \vdash \perp \rangle$

proof (*unfold-locales*; *safe*)

fix A qs t **and** $ps :: \langle 'x \text{ fm list} \rangle$

assume $\langle ps \rightsquigarrow_{\gamma} qs \rangle$ **and** $*$: $\langle set\ ps \subseteq set\ A \rangle \langle \neg A \vdash \perp \rangle \langle qs\ t @ A \vdash \perp \rangle$
then show *False*
proof *cases*
 case (*CallP* p)
 then show *?thesis*
 using $*$ *IA*[*of* $p\ t$] *MP'*
 by (*metis* (*mono-tags*, *lifting*) *imply.simps*(1–2) *imply-append* *imply-swap-append* *imply-weaken*)
 qed
qed

interpretation *DG_P*: *Weak-Derivational-Gamma map-sym map-fm params-fm* $\langle \lambda-. True \rangle$ *G_P.classify-UNIV* $\langle \lambda A. \neg A \vdash \perp \rangle$

proof (*unfold-locales*; *safe*)
fix $A\ qs\ t$ **and** $ps :: \langle 'x\ fm\ list \rangle$
assume $\langle ps \rightsquigarrow_{\gamma_P} qs \rangle$ **and** $*$: $\langle set\ ps \subseteq set\ A \rangle \langle \neg A \vdash \perp \rangle \langle qs\ t @ A \vdash \perp \rangle$
then show *False*
proof *cases*
 case (*CallPP* p)
 then show *?thesis*
 using $*$ *IA_P*[*of* $p\ t$] *MP'*
 by (*metis* (*mono-tags*, *lifting*) *imply.simps*(1–2) *imply-append* *imply-swap-append* *imply-weaken*)
 qed
qed

interpretation *DG_F*: *Weak-Derivational-Gamma map-sym map-fm params-fm* $\langle \lambda-. True \rangle$ *G_F.classify-UNIV* $\langle \lambda A. \neg A \vdash \perp \rangle$

proof (*unfold-locales*; *safe*)
fix $A\ qs\ t$ **and** $ps :: \langle 'x\ fm\ list \rangle$
assume $\langle ps \rightsquigarrow_{\gamma_F} qs \rangle$ **and** $*$: $\langle set\ ps \subseteq set\ A \rangle \langle \neg A \vdash \perp \rangle \langle qs\ t @ A \vdash \perp \rangle$
then show *False*
proof *cases*
 case (*CallFP* p)
 then show *?thesis*
 using $*$ *IA_F*[*of* $p\ t$] *MP'*
 by (*metis* (*mono-tags*, *lifting*) *imply.simps*(1–2) *imply-append* *imply-swap-append* *imply-weaken*)
 qed
qed

lemma *imply-params-fm*: $\langle params-fm\ (ps \rightsquigarrow q) = params-fm\ q \cup (\bigcup p \in set\ ps. params-fm\ p) \rangle$
by (*induct* ps) *auto*

interpretation *DD*: *Weak-Derivational-Delta map-fm params-fm* $\langle \lambda-. True \rangle$ δ $\langle \lambda A. \neg A \vdash \perp \rangle$

proof (*standard*; *safe*)
fix $A\ a$ **and** $p :: \langle 'x\ fm \rangle$

```

assume ⟨ $p \in \text{set } A$ ⟩ ⟨ $a \notin P.\text{params } (\text{set } A)$ ⟩ ⟨ $\neg A \vdash \perp$ ⟩ ⟨ $\delta p a @ A \vdash \perp$ ⟩
then show False
proof (induct p a rule:  $\delta$ .induct)
  case (1  $p x$ )
  have ⟨ $x \notin P.\text{params } \{p, A \rightsquigarrow \perp\}$ ⟩
    using 1(1-2) imply-params-fm[ $of A \langle \perp \rangle$ ] by auto
  moreover have ⟨ $\neg \langle \star x / 0 \rangle p \# A \vdash \perp$ ⟩
    using 1(4) by simp
  ultimately have ⟨ $\neg \forall p \# A \vdash \perp$ ⟩
    using  $GR'$ [of x p] by simp
  then show ?thesis
    using 1 by (meson Boole MP' imply-mem)
next
  case (2  $p x$ )
  have ⟨ $x \notin P.\text{params } \{p, A \rightsquigarrow \perp\}$ ⟩
    using 2(1-2) imply-params-fm[ $of A \langle \perp \rangle$ ] by auto
  moreover have ⟨ $\neg \langle \bigcirc_2 x / 0 \rangle_P p \# A \vdash \perp$ ⟩
    using 2(4) by simp
  ultimately have ⟨ $\neg \forall_P p \# A \vdash \perp$ ⟩
    using  $GR_P'$ [of x p] by simp
  then show ?thesis
    using 2 by (meson Boole MP' imply-mem)
next
  case (3  $p x$ )
  have ⟨ $x \notin P.\text{params } \{p, A \rightsquigarrow \perp\}$ ⟩
    using 3(1-2) imply-params-fm[ $of A \langle \perp \rangle$ ] by auto
  moreover have ⟨ $\neg \langle \bigcirc_2 x / 0 \rangle_F p \# A \vdash \perp$ ⟩
    using 3(4) by simp
  ultimately have ⟨ $\neg \forall_F p \# A \vdash \perp$ ⟩
    using  $GR_F'$ [of x p] by simp
  then show ?thesis
    using 3 by (meson Boole MP' imply-mem)
qed simp-all
qed

term  $P.\text{params}$ 

interpretation Weak-Derivational-Consistency map-fm params-fm ⟨ $\lambda-. \text{True}$ ⟩ Kinds
  ⟨ $\lambda A. \neg A \vdash \perp$ ⟩
proof
  assume ⟨infinite ( $UNIV :: 'x \text{ set}$ )⟩
  then have inf: ⟨infinite ( $\text{Collect } (\lambda-. \text{True}) :: 'x \text{ set}$ )⟩
    by simp
  then show ⟨ $P.\text{prop}_E \text{ Kinds } \{S :: 'x \text{ fm set}. \exists A. \text{set } A = S \wedge \neg A \vdash \perp\}$ ⟩
    using  $\text{prop}_E\text{-Kinds}$ [OF DC.kind[OF inf] DA.kind DB.kind DG.kind DG_P.kind
   $DG_F.\text{kind DD.kind}$ ]
    by blast
qed

```

theorem *weak-completeness*:

```

fixes  $p :: \langle 'x \text{ fm} \rangle$ 
assumes  $\text{mod}: \langle \forall (E :: - \Rightarrow 'x \text{ tm}) E_F E_P C F G PS FS. \text{wf-model } (E, E_F, E_P,$ 
 $C, F, G, PS, FS) \longrightarrow (\forall q \in \text{set ps. } (E, E_F, E_P, C, F, G, PS, FS) \models q) \longrightarrow (E,$ 
 $E_F, E_P, C, F, G, PS, FS) \models p \rangle$ 
and  $\langle P.\text{enough-new } (\text{set ps}) \rangle$ 
shows  $\langle ps \vdash p \rangle$ 
proof (rule ccontr)
assume  $\langle \neg ps \vdash p \rangle$ 
then have  $*$ :  $\langle \neg (\neg p) \# ps \vdash \perp \rangle$ 
using Boole by blast

let  $?S = \langle \text{set } (\neg p \# ps) \rangle$ 
let  $?C = \langle \{S :: 'x \text{ fm set. } \exists A. \text{set } A = S \wedge \neg A \vdash \perp\} \rangle$ 
let  $?M = \langle \llbracket \text{mk-mcs } ?C ?S \rrbracket \rangle$ 

have  $\langle \text{infinite } (UNIV :: 'x \text{ set}) \rangle$ 
using assms(2) card-of-ordLeq-infinite finite-subset inf-univ subset-UNIV
unfolding P.enough-new-def by blast
then have  $\langle P.\text{prop}_E \text{ Kinds } ?C \rangle$ 
using Consistency by blast
moreover have  $\langle ?S \in ?C \rangle$ 
using  $*$  by blast
moreover have  $\langle P.\text{enough-new } ?S \rangle$ 
using assms(2) P.infinite-params-left unfolding P.enough-new-def
by (metis Set-Diff-Un UN-Un card-of-diff inf-univ ordLeq-transitive)
ultimately have  $*$ :  $\langle \forall p \in ?S. ?M \models p \rangle$ 
using model-existence by blast
then have  $\langle ?M \models p \rangle$ 
using mod unfolding hdomF-def by auto
then show False
using  $*$  by simp
qed

```

theorem *completeness*:

```

fixes  $p :: \langle 'x \text{ fm} \rangle$ 
assumes  $\langle \forall (E :: \text{nat} \Rightarrow 'x \text{ tm}) E_P E_F C F G PS FS. \text{wf-model } (E, E_F, E_P,$ 
 $C, F, G, PS, FS) \longrightarrow (E, E_F, E_P, C, F, G, PS, FS) \models p \rangle$ 
and  $\langle |UNIV :: 'x \text{ fm set}| \leq o \ |UNIV :: 'x \text{ set}| \rangle$ 
shows  $\langle \vdash p \rangle$ 
using assms weak-completeness [where  $ps = \langle [] \rangle$ , of p] unfolding P.enough-new-def
by simp

```

3.10 Natural Deduction

locale *Natural-Deduction*

begin

inductive *ND-Set* :: $\langle 'x \text{ fm set} \Rightarrow 'x \text{ fm} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \Vdash \rangle$ 50) **where**

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

3.10.1 Soundness

theorem *soundness-set*:

assumes $\langle A \Vdash p \rangle \langle wf\text{-model}(E, E_F, E_P, C, F, G, PS, FS) \rangle$

shows $\langle \forall q \in A. (E, E_F, E_P, C, F, G, PS, FS) \models q \implies (E, E_F, E_P, C, F, G, PS, FS) \models p \rangle$

using *assms*

proof (*induct A p arbitrary: C F G pred: ND-Set*)

case (*UniI A a p*)

have $\langle \forall q \in A. (E, E_F, E_P, C(a := x), F, G, PS, FS) \models q \rangle$ **for** x

using *UniI(3-)* **by** *simp*

moreover have $\langle wf\text{-model}(E, E_F, E_P, C(a := x), F, G, PS, FS) \rangle$ **for** x

using *UniI(5)* **by** *simp*

ultimately have $\langle (E, E_F, E_P, C(a := x), F, G, PS, FS) \models \langle \star a/0 \rangle p \rangle$ **for** x

using *UniI* **by** *meson*

then show *?case*

using *UniI* **by** *simp*

next

case (*UniI_P A a p*)

have $\langle \forall x \in PS. \forall q \in A. (E, E_F, E_P, C, F, G(a := x), PS, FS) \models q \rangle$

using *UniI_P(3-)* **by** *simp*

moreover have $\langle \forall x \in PS. wf\text{-model}(E, E_F, E_P, C, F, G(a := x), PS, FS) \rangle$

using *UniI_P(5)* **by** *auto*

ultimately have $\langle \forall x \in PS. (E, E_F, E_P, C, F, G(a := x), PS, FS) \models \langle \bigcirc_2 a/0 \rangle_P p \rangle$

using *UniI_P* **by** *meson*

then show *?case*

using *UniI_P* **by** *simp*

next

case (*UniE_P A p s*)

then show *?case*

by (*cases s*) (*fastforce intro!: rangeI*)+

next

case (*UniI_F A a p*)

have $\langle \forall x \in FS. \forall q \in A. (E, E_F, E_P, C, F(a := x), G, PS, FS) \models q \rangle$

using *UniI_F(3-)* **by** *simp*

moreover have $\langle \forall x \in FS. wf\text{-model}(E, E_F, E_P, C, F(a := x), G, PS, FS) \rangle$

```

    using UniIF(5) by auto
    ultimately have ⟨∀ x ∈ FS. (E, EF, EP, C, F(a := x), G, PS, FS) ⊨ ⟨O2
a/0⟩F p⟩
    using UniIF by meson
    then show ?case
    using UniIF by simp
next
  case (UniEF A p s)
  then show ?case
    by (cases s) (fastforce intro!: rangeI)+
qed auto

```

3.10.2 Derivational Consistency

```

lemma Boole: ⟨{¬ p} ∪ A ⊨ ⊥ ⟹ A ⊨ p⟩
  using Clas FlsE by blast

```

```

sublocale DC: Derivational-Confl map-fm params-fm ⟨λ-. True⟩ confl-class ⟨λA.
¬ A ⊨ ⊥⟩

```

```

proof
  fix A ps qs and q :: ⟨'x fm⟩
  assume ⟨ps ∼χ qs⟩ ⟨set ps ⊆ A⟩ ⟨q ∈ set qs⟩ ⟨q ∈ A⟩
  then show ⟨¬ ¬ A ⊨ ⊥⟩
    by cases auto
qed

```

```

sublocale DA: Derivational-Alpha map-fm params-fm ⟨λ-. True⟩ alpha-class ⟨λA.
¬ A ⊨ ⊥⟩

```

```

proof (standard; safe)
  fix A and ps qs :: ⟨'x fm list⟩
  assume ⟨ps ∼α qs⟩ and *: ⟨set ps ⊆ A⟩ ⟨¬ A ⊨ ⊥⟩ ⟨set qs ∪ A ⊨ ⊥⟩
  then show False
  proof cases
    case (CImpN p q)
    then have ⟨A ⊨ ¬ (p ⟶ q)⟩
      using *(1) by auto
    moreover have ⟨A ⊨ p ⟶ q⟩
      using CImpN(2) * Boole[of q ⟨{p} ∪ A⟩] by auto
    ultimately show ?thesis
      using * by blast
  qed
qed

```

```

sublocale DB: Derivational-Beta map-fm params-fm ⟨λ-. True⟩ beta-class ⟨λA.
¬ A ⊨ ⊥⟩

```

```

proof
  fix A and ps qs :: ⟨'x fm list⟩
  assume ⟨ps ∼β qs⟩ and *: ⟨set ps ⊆ A⟩ ⟨¬ A ⊨ ⊥⟩
  then show ⟨∃ q ∈ set qs. ¬ {q} ∪ A ⊨ ⊥⟩

```

```

proof cases
  case (CImpP p q)
  then show ?thesis
    using * Boole[of p A]
    by (metis Assm ImpE ImpI list.set-intros(1) set-subset-Cons subset-iff)
qed
qed

```

```

sublocale DG: Derivational-Gamma map-tm map-fm params-fm  $\langle \lambda-. True \rangle$  G.classify-UNIV
 $\langle \lambda A. \neg A \Vdash \perp \rangle$ 
proof (unfold-locales; safe)
  fix A qs t and ps ::  $\langle 'x \text{ fm list} \rangle$ 
  assume  $\langle ps \rightsquigarrow_{\gamma} qs \rangle$  and *:  $\langle set\ ps \subseteq A \rangle \langle \neg A \Vdash \perp \rangle \langle set\ (qs\ t) \cup A \Vdash \perp \rangle$ 
  then show False
  proof cases
    case (CALLP p)
    then show ?thesis
      using * UniE[of A p t] ImpI by auto
  qed
qed

```

```

sublocale DGP: Derivational-Gamma map-sym map-fm params-fm  $\langle \lambda-. True \rangle$ 
GP.classify-UNIV  $\langle \lambda A. \neg A \Vdash \perp \rangle$ 
proof (unfold-locales; safe)
  fix A qs t and ps ::  $\langle 'x \text{ fm list} \rangle$ 
  assume  $\langle ps \rightsquigarrow_{\gamma_P} qs \rangle$  and *:  $\langle set\ ps \subseteq A \rangle \langle \neg A \Vdash \perp \rangle \langle set\ (qs\ t) \cup A \Vdash \perp \rangle$ 
  then show False
  proof cases
    case (CALLPP p)
    then show ?thesis
      using * UniEP[of A p t] ImpI by auto
  qed
qed

```

```

sublocale DGF: Derivational-Gamma map-sym map-fm params-fm  $\langle \lambda-. True \rangle$ 
GF.classify-UNIV  $\langle \lambda A. \neg A \Vdash \perp \rangle$ 
proof (unfold-locales; safe)
  fix A qs t and ps ::  $\langle 'x \text{ fm list} \rangle$ 
  assume  $\langle ps \rightsquigarrow_{\gamma_F} qs \rangle$  and *:  $\langle set\ ps \subseteq A \rangle \langle \neg A \Vdash \perp \rangle \langle set\ (qs\ t) \cup A \Vdash \perp \rangle$ 
  then show False
  proof cases
    case (CALLFP p)
    then show ?thesis
      using * UniEF[of A p t] ImpI by auto
  qed
qed

```

```

sublocale DD: Derivational-Delta map-fm params-fm  $\langle \lambda-. True \rangle$   $\delta \langle \lambda A. \neg A \Vdash \perp \rangle$ 
proof (standard; safe)

```

```

fix  $A$   $a$  and  $p :: \langle 'x \text{ fm} \rangle$ 
assume  $\langle p \in A \rangle \langle a \notin P.\text{params } A \rangle \langle \neg A \Vdash \perp \rangle \langle \text{set } (\delta \text{ } p \text{ } a) \cup A \Vdash \perp \rangle$ 
then show  $\text{False}$ 
proof (induct  $p$   $a$  rule:  $\delta.\text{induct}$ )
  case ( $1 \text{ } p \text{ } x$ )
    then have  $\langle x \notin P.\text{params } (\{p\} \cup A) \rangle$ 
    by auto
    moreover have  $\langle A \Vdash \langle \star x / 0 \rangle p \rangle$ 
    using  $1(4)$  Boole by auto
    ultimately show ?case
    using  $1 \text{ } \text{UniI}$  by blast
  next
    case ( $2 \text{ } p \text{ } x$ )
    then have  $\langle x \notin P.\text{params } (\{p\} \cup A) \rangle$ 
    by auto
    moreover have  $\langle A \Vdash \langle \textcircled{2} x / 0 \rangle_P p \rangle$ 
    using  $2(4)$  Boole by auto
    ultimately show ?case
    using  $2 \text{ } \text{UniI}_P$  by blast
  next
    case ( $3 \text{ } p \text{ } x$ )
    then have  $\langle x \notin P.\text{params } (\{p\} \cup A) \rangle$ 
    by auto
    moreover have  $\langle A \Vdash \langle \textcircled{2} x / 0 \rangle_F p \rangle$ 
    using  $3(4)$  Boole by auto
    ultimately show ?case
    using  $3 \text{ } \text{UniI}_F$  by blast
  qed simp-all
qed

sublocale Derivational-Consistency map-fm params-fm  $\langle \lambda-. \text{True} \rangle$  Kinds  $\langle \lambda A. \neg A \Vdash \perp \rangle$ 
using propE-Kinds[OF DC.kind DA.kind DB.kind DG.kind DGP.kind DGF.kind DD.kind] by unfold-locales

```

3.10.3 Strong Completeness

theorem *strong-completeness*:

```

fixes  $p :: \langle 'x \text{ fm} \rangle$ 
assumes  $\text{mod} : \langle \bigwedge (E :: \text{nat} \Rightarrow 'x \text{ tm}) E_F E_P C F G PS FS. \text{wf-model } (E, E_F, E_P, C, F, G, PS, FS) \implies$ 
   $(\forall q \in A. (E, E_F, E_P, C, F, G, PS, FS) \models q) \implies (E, E_F, E_P, C, F, G, PS, FS) \models p \rangle$ 
and  $\langle P.\text{enough-new } A \rangle$ 
shows  $\langle A \Vdash p \rangle$ 
proof (rule ccontr)
assume  $\langle \neg A \Vdash p \rangle$ 
then have  $\star : \langle \neg \{ \neg p \} \cup A \Vdash \perp \rangle$ 
using Boole by (metis insert-is-Un)

```

```

let ?S = ⟨set [¬ p] ∪ A⟩
let ?C = ⟨{A. P.enough-new A ∧ ¬ A ⊢ ⊥}⟩
let ?M = ⟨[[mk-mcs ?C ?S]]⟩

have wf: ⟨wf-model ?M⟩
  unfolding hdomF-def by simp

have ⟨P.propE Kinds ?C⟩
  using Consistency by blast
moreover have ⟨P.enough-new ?S⟩
  using assms(2) params-left by blast
moreover from this have ⟨?S ∈ ?C⟩
  using * by simp
ultimately have *: ⟨∀ p ∈ ?S. ?M ⊨ p⟩
  using model-existence by blast
then have ⟨?M ⊨ p⟩
  using mod[OF wf] by fast
then show False
  using * by simp
qed

proposition ⟨∃ M. wf-model M⟩
  by (meson sup.cobounded2 sup-ge1 wf-model.simps)

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  $\Rightarrow$  'a set  $\Rightarrow$  'b set  $\Rightarrow$  ('a  $\Rightarrow$  'b) set
  ( $\langle$  ( $\langle$  indent=3 notation= $\langle$  binder  $\Pi \in \rangle \Pi$  - $\in$  ./ - $\rangle$  10)
```

4.1 Syntax

```
datatype (symbols-tm: 'x) tm
  = Var nat ( $\langle$  # $\rangle$ )
  | Sym 'x ( $\langle$   $\circ$  $\rangle$ )
```

```
datatype (symbols-fm: 'x) fm
  = TmI  $\langle$  'x tm $\rangle$  ( $\langle$   $\cdot$  $\rangle$ )
  | TmP  $\langle$  'x tm $\rangle$  ( $\langle$   $\cdot$  $\rangle$ )
  | Neg  $\langle$  'x fm $\rangle$  ( $\langle$   $\neg$  - $\rangle$  [70] 70)
  | Con  $\langle$  'x fm $\rangle$   $\langle$  'x fm $\rangle$  (infixr  $\langle$   $\wedge$  $\rangle$  35)
  | Box  $\langle$  'x fm $\rangle$  ( $\langle$   $\square$  $\rangle$ )
  | Sat  $\langle$  'x tm $\rangle$   $\langle$  'x fm $\rangle$  ( $\langle$   $\@$  $\rangle$ )
  | Glo  $\langle$  'x fm $\rangle$  ( $\langle$  A $\rangle$ )
  | Dwn  $\langle$  'x fm $\rangle$  ( $\langle$   $\downarrow$  $\rangle$ )
  | All  $\langle$  'x fm $\rangle$  ( $\langle$   $\forall$  $\rangle$ )
```

```
abbreviation Fls ::  $\langle$  'x fm $\rangle$  ( $\langle$   $\perp$  $\rangle$ ) where
   $\langle$   $\perp$   $\equiv$  undefined  $\wedge$   $\neg$  undefined $\rangle$ 
```

```
abbreviation Imp ::  $\langle$  'x fm  $\Rightarrow$  'x fm  $\Rightarrow$  'x fm $\rangle$  (infixr  $\langle$   $\longrightarrow$  $\rangle$  55) where
   $\langle$   $p \longrightarrow q \equiv \neg (p \wedge \neg q)$  $\rangle$ 
```

```
abbreviation Dia ::  $\langle$  'x fm  $\Rightarrow$  'x fm $\rangle$  ( $\langle$   $\diamond$  $\rangle$ ) where
   $\langle$   $\diamond p \equiv \neg (\square (\neg p))$  $\rangle$ 
```

type-synonym $\langle 'x \text{ lbd} = \langle 'x \text{ tm} \times 'x \text{ fm} \rangle \rangle$

4.2 Semantics

record $\langle 'w \text{ frame} =$
 $\mathcal{W} :: \langle 'w \text{ set} \rangle$
 $\mathcal{R} :: \langle 'w \Rightarrow 'w \text{ set} \rangle$

record $\langle 'w \text{ gframe} = \langle 'w \text{ frame} \rangle +$
 $\Pi :: \langle 'w \text{ set set} \rangle$

record $\langle ('x, 'w) \text{ model} = \langle 'w \text{ gframe} \rangle +$
 $\mathcal{N} :: \langle 'x \Rightarrow 'w \rangle$
 $\mathfrak{N} :: \langle \text{nat} \Rightarrow 'w \rangle$
 $\mathcal{V} :: \langle 'x \Rightarrow 'w \text{ set} \rangle$
 $\mathfrak{V} :: \langle \text{nat} \Rightarrow 'w \text{ set} \rangle$

abbreviation $\langle \text{Model } W R PI N e V f \equiv (\mathcal{W} = W, \mathcal{R} = R, \Pi = PI, \mathcal{N} = N, \mathfrak{N} = e, \mathcal{V} = V, \mathfrak{V} = f) \rangle$

definition $\langle \text{admissible} :: \langle 'w \text{ frame} \Rightarrow 'w \text{ set set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{admissible } M PI \equiv$
 $(\forall X \in PI. \mathcal{W} M - X \in PI) \wedge$
 $(\forall X \in PI. \forall Y \in PI. X \cap Y \in PI) \wedge$
 $(\forall X \in PI. \{w \in \mathcal{W} M. \forall v \in \mathcal{R} M w. v \in X\} \in PI) \rangle$

definition $\langle \text{wf-frame} :: \langle 'w \text{ frame} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{wf-frame } M \equiv \mathcal{W} M \neq \{\} \wedge \mathcal{R} M \subseteq \text{Pow } (\mathcal{W} M) \rangle$

definition $\langle \text{wf-gframe} :: \langle 'w \text{ gframe} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{wf-gframe } M \equiv \text{wf-frame } (\text{frame.truncate } M) \wedge \Pi M \neq \{\} \wedge \Pi M \subseteq \text{Pow } (\mathcal{W} M) \wedge \text{admissible } (\text{frame.truncate } M) (\Pi M) \rangle$

definition $\langle \text{wf-env} :: \langle ('x, 'w) \text{ model} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{wf-env } M \equiv \text{range } (\mathcal{N} M) \subseteq \mathcal{W} M \wedge \text{range } (\mathfrak{N} M) \subseteq \mathcal{W} M \wedge \text{range } (\mathcal{V} M) \subseteq \Pi M \wedge \text{range } (\mathfrak{V} M) \subseteq \Pi M \rangle$

definition $\langle \text{wf-model} :: \langle ('x, 'w) \text{ model} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{wf-model } M \equiv \text{wf-gframe } (\text{gframe.truncate } M) \wedge \text{wf-env } M \rangle$

lemmas $\text{unfolds} =$
 $\text{model.defs gframe.defs frame.defs}$
 $\text{model.select-convs gframe.select-convs frame.select-convs}$

context
fixes $M :: \langle ('x, 'w) \text{ model} \rangle$
assumes $\text{wf}: \langle \text{wf-model } M \rangle$
begin

lemma *wf-compl*: $\langle X \in \Pi M \implies \mathcal{W} M - X \in \Pi M \rangle$
using *wf unfolding wf-model-def wf-gframe-def admissible-def unfolds by blast*

lemma *wf-inter*: $\langle X \in \Pi M \implies Y \in \Pi M \implies X \cap Y \in \Pi M \rangle$
using *wf unfolding wf-model-def wf-gframe-def admissible-def unfolds by blast*

lemma *wf-modal*: $\langle X \in \Pi M \implies \{w \in \mathcal{W} M. \forall v \in \mathcal{R} M w. v \in X\} \in \Pi M \rangle$
using *wf unfolding wf-model-def wf-gframe-def admissible-def unfolds by blast*

lemma *wf-empty*: $\langle \{\} \in \Pi M \rangle$
using *wf wf-compl wf-inter unfolding wf-model-def wf-gframe-def unfolds by force*

lemma *wf-univ*: $\langle \mathcal{W} M \in \Pi M \rangle$
using *wf wf-empty wf-compl by fastforce*

lemma *wf- Π* : $\langle P \in \Pi M \implies P \subseteq \mathcal{W} M \rangle$
using *wf unfolding wf-gframe-def wf-model-def unfolds by fast*

lemma *wf- \mathcal{N}* : $\langle \mathcal{N} M i \in \mathcal{W} M \rangle$
using *wf unfolding wf-model-def wf-env-def by fast*

lemma *wf- \mathfrak{N}* : $\langle \mathfrak{N} M n \in \mathcal{W} M \rangle$
using *wf unfolding wf-model-def wf-env-def by blast*

lemma *wf- \mathcal{V}* : $\langle \mathcal{V} M P \in \Pi M \rangle$
using *wf unfolding wf-model-def wf-env-def by blast*

lemma *wf- \mathcal{V}'* : $\langle \mathcal{V} M n \subseteq \mathcal{W} M \rangle$
using *wf wf- \mathcal{V} wf- Π by blast*

lemma *wf- \mathfrak{V}* : $\langle \mathfrak{V} M n \in \Pi M \rangle$
using *wf unfolding wf-model-def wf-env-def by blast*

lemma *wf- \mathfrak{V}'* : $\langle \mathfrak{V} M n \subseteq \mathcal{W} M \rangle$
using *wf wf- \mathfrak{V} wf- Π by blast*

end

type-synonym $\langle 'x, 'w \rangle \text{ ctx} = \langle ('x, 'w) \text{ model} \times 'w \rangle$

primrec *add-env* :: $\langle 'a \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow \text{nat} \Rightarrow 'a \rangle$ (**infix** $\langle \gg \rangle$ 0) **where**
 $\langle (t \gg e) \ 0 = t \rangle$
 $| \langle (- \gg e) \ (\text{Suc } n) = e \ n \rangle$

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

$\langle (M, w) \models (\neg p) = (\neg (M, w) \models p) \rangle$
 $\langle (M, w) \models (p \wedge q) = ((M, w) \models p \wedge (M, w) \models q) \rangle$
 $\langle (M, w) \models \Box p = (\forall v \in \mathcal{R} M w. (M, v) \models p) \rangle$
 $\langle (M, -) \models @i p = ((M, \text{case-tm } (\mathfrak{N} M) (\mathcal{N} M) i) \models p) \rangle$
 $\langle (M, -) \models \mathbf{A} p = (\forall v \in \mathcal{W} M. (M, v) \models p) \rangle$
 $\langle (M, w) \models \Downarrow p = ((M(\mathfrak{N} := (w \gg \mathfrak{N} M)), w) \models p) \rangle$
 $\langle (M, w) \models \forall p = (\forall P \in \Pi M. (M(\mathfrak{V} := (P \gg \mathfrak{V} M)), w) \models p) \rangle$

lemma $\langle (M, w) \models p \longrightarrow q \longleftrightarrow (M, w) \models p \longrightarrow (M, w) \models q \rangle$
by *simp*

lemma $\langle (M, w) \models \Downarrow(\#0) \rangle$
by (*cases M*) *simp*

lemma $\langle (M, w) \models \forall(@(\circ k) (\cdot(\#0))) \longrightarrow @(\circ k) (\forall(\cdot(\#0))) \rangle$
by (*cases M*) *auto*

lemma $\langle (M, w) \models @(\circ k) (\forall(\cdot(\#0))) \longrightarrow \forall(@(\circ k) (\cdot(\#0))) \rangle$
by (*cases M*) *auto*

lemma $\langle \text{wf-model } M \implies (M, w) \models \forall(\cdot(\#0)) \longrightarrow \cdot P \rangle$
unfolding *wf-model-def wf-env-def* **by** (*fastforce split: tm.splits*)

lemma $\langle \text{wf-model } M \implies (M, w) \models \forall(@(\circ k) (\cdot(\#0))) \longrightarrow \Downarrow(\circ k) \longrightarrow @(\#0) (\cdot P) \rangle$
unfolding *wf-model-def wf-env-def* **by** (*fastforce split: tm.splits*)

4.3 Operations

abbreviation *map-lbd* :: $\langle 'x \Rightarrow 'k \Rightarrow 'x \text{ lbd} \Rightarrow 'k \text{ lbd} \rangle$ **where**
 $\langle \text{map-lbd } f \equiv \text{map-prod } (\text{map-tm } f) (\text{map-fm } f) \rangle$

primrec *symbols-lbd* :: $\langle 'x \text{ lbd} \Rightarrow 'x \text{ set} \rangle$ **where**
 $\langle \text{symbols-lbd } (i, p) = \text{symbols-tm } i \cup \text{symbols-fm } p \rangle$

abbreviation *symbols* :: $\langle 'x \text{ lbd set} \Rightarrow 'x \text{ set} \rangle$ **where**
 $\langle \text{symbols } S \equiv \bigcup ip \in S. \text{symbols-lbd } ip \rangle$

abbreviation *lift-tm* :: $\langle \text{nat} \Rightarrow 'x \text{ tm} \Rightarrow 'x \text{ tm} \rangle$ **where**
 $\langle \text{lift-tm } m \equiv \text{case-tm } (\lambda n. \text{if } n < m \text{ then } \#n \text{ else } \#(n + m + 1)) \circ \rangle$

primrec *vars-tm* :: $\langle \text{nat} \Rightarrow 'x \text{ tm} \Rightarrow \text{nat set} \rangle$ **where**
 $\langle \text{vars-tm } m (\#n) = (\text{if } n \leq m \text{ then } \{\} \text{ else } \{n\}) \rangle$
 $\langle \text{vars-tm } (\circ -) = \{\} \rangle$

primrec *vars-fm* :: $\langle \text{nat} \Rightarrow 'x \text{ fm} \Rightarrow \text{nat set} \rangle$ **where**
 $\langle \text{vars-fm } m (\cdot t) = \text{vars-tm } m t \rangle$
 $\langle \text{vars-fm } m (\cdot P) = \text{vars-tm } m P \rangle$
 $\langle \text{vars-fm } m (\neg p) = \text{vars-fm } m p \rangle$

$\langle \text{vars-fm } m (p \wedge q) = \text{vars-fm } m p \cup \text{vars-fm } m q \rangle$
 $\langle \text{vars-fm } m (\Box p) = \text{vars-fm } m p \rangle$
 $\langle \text{vars-fm } m (@i p) = \text{vars-tm } m i \cup \text{vars-fm } m p \rangle$
 $\langle \text{vars-fm } m (\mathbf{A} p) = \text{vars-fm } m p \rangle$
 $\langle \text{vars-fm } m (\Downarrow p) = \text{vars-fm } (m + 1) p \rangle$
 $\langle \text{vars-fm } m (\forall p) = \text{vars-fm } (m + 1) p \rangle$

Nominals

primrec *sub-tm* :: $\langle (\text{nat} \Rightarrow 'x \text{ tm}) \Rightarrow 'x \text{ tm} \Rightarrow 'x \text{ tm} \rangle$ **where**
 $\langle \text{sub-tm } s (\#n) = s n \rangle$
 $\langle \text{sub-tm } - (\circ k) = \circ k \rangle$

primrec *sub-nom* :: $\langle (\text{nat} \Rightarrow 'x \text{ tm}) \Rightarrow 'x \text{ fm} \Rightarrow 'x \text{ fm} \rangle$ **where**
 $\langle \text{sub-nom } s (\cdot t) = \cdot (\text{sub-tm } s t) \rangle$
 $\langle \text{sub-nom } - (\cdot P) = \cdot P \rangle$
 $\langle \text{sub-nom } s (\neg p) = \neg \text{sub-nom } s p \rangle$
 $\langle \text{sub-nom } s (p \wedge q) = (\text{sub-nom } s p \wedge \text{sub-nom } s q) \rangle$
 $\langle \text{sub-nom } s (\Box p) = \Box (\text{sub-nom } s p) \rangle$
 $\langle \text{sub-nom } s (@i p) = @(\text{sub-tm } s i) (\text{sub-nom } s p) \rangle$
 $\langle \text{sub-nom } s (\mathbf{A} p) = \mathbf{A} (\text{sub-nom } s p) \rangle$
 $\langle \text{sub-nom } s (\Downarrow p) = \Downarrow (\text{sub-nom } (\#0 \gg \lambda n. \text{lift-tm } 0 (s n)) p) \rangle$
 $\langle \text{sub-nom } s (\forall p) = \forall (\text{sub-nom } s p) \rangle$

abbreviation *inst-single-nom* :: $\langle 'x \text{ tm} \Rightarrow 'x \text{ fm} \Rightarrow 'x \text{ fm} \rangle (\langle \langle - \rangle_i \rangle)$ **where**
 $\langle \langle t \rangle_i \equiv \text{sub-nom } (t \gg \#) \rangle$

Propositions

fun *softqdf* :: $\langle 'x \text{ fm} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{softqdf } (\cdot) = \text{False} \rangle$
 $\langle \text{softqdf } (\cdot P) = \text{True} \rangle$
 $\langle \text{softqdf } (\neg p) = \text{softqdf } p \rangle$
 $\langle \text{softqdf } (p \wedge q) = (\text{softqdf } p \wedge \text{softqdf } q) \rangle$
 $\langle \text{softqdf } (\Box p) = \text{softqdf } p \rangle$
 $\langle \text{softqdf } (@i p) = \text{softqdf } p \rangle$
 $\langle \text{softqdf } (\mathbf{A} p) = \text{softqdf } p \rangle$
 $\langle \text{softqdf } (\Downarrow p) = \text{False} \rangle$
 $\langle \text{softqdf } (\forall p) = \text{False} \rangle$

abbreviation *softqdf-sub* :: $\langle (\text{nat} \Rightarrow 'x \text{ fm}) \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{softqdf-sub } s \equiv \forall n. \text{softqdf } (s n) \rangle$

primrec *lift-fm-nom* :: $\langle \text{nat} \Rightarrow 'x \text{ fm} \Rightarrow 'x \text{ fm} \rangle$ **where**
 $\langle \text{lift-fm-nom } m (\cdot t) = \cdot (\text{lift-tm } m t) \rangle$
 $\langle \text{lift-fm-nom } - (\cdot P) = \cdot P \rangle$
 $\langle \text{lift-fm-nom } m (\neg p) = \neg \text{lift-fm-nom } m p \rangle$
 $\langle \text{lift-fm-nom } m (p \wedge q) = (\text{lift-fm-nom } m p \wedge \text{lift-fm-nom } m q) \rangle$
 $\langle \text{lift-fm-nom } m (\Box p) = \Box (\text{lift-fm-nom } m p) \rangle$
 $\langle \text{lift-fm-nom } m (@i p) = @(\text{lift-tm } m i) (\text{lift-fm-nom } m p) \rangle$

$\langle \text{lift-fm-nom } m (\mathbf{A} \ p) = \mathbf{A} (\text{lift-fm-nom } m \ p) \rangle$
 $\langle \text{lift-fm-nom } m (\downarrow \ p) = \downarrow (\text{lift-fm-nom } (m + 1) \ p) \rangle$
 $\langle \text{lift-fm-nom } m (\forall \ p) = \forall (\text{lift-fm-nom } m \ p) \rangle$

primrec *lift-fm-pro* :: $\langle \text{nat} \Rightarrow 'x \text{ fm} \Rightarrow 'x \text{ fm} \rangle$ **where**

$\langle \text{lift-fm-pro } - (\cdot t) = \cdot t \rangle$
 $\langle \text{lift-fm-pro } m (\cdot P) = \cdot (\text{lift-tm } m \ P) \rangle$
 $\langle \text{lift-fm-pro } m (\neg \ p) = \neg \ \text{lift-fm-pro } m \ p \rangle$
 $\langle \text{lift-fm-pro } m (p \wedge \ q) = (\text{lift-fm-pro } m \ p \wedge \ \text{lift-fm-pro } m \ q) \rangle$
 $\langle \text{lift-fm-pro } m (\Box \ p) = \Box (\text{lift-fm-pro } m \ p) \rangle$
 $\langle \text{lift-fm-pro } m (@i \ p) = @i (\text{lift-fm-pro } m \ p) \rangle$
 $\langle \text{lift-fm-pro } m (\mathbf{A} \ p) = \mathbf{A} (\text{lift-fm-pro } m \ p) \rangle$
 $\langle \text{lift-fm-pro } m (\downarrow \ p) = \downarrow (\text{lift-fm-pro } m \ p) \rangle$
 $\langle \text{lift-fm-pro } m (\forall \ p) = \forall (\text{lift-fm-pro } (m + 1) \ p) \rangle$

primrec *sub-pro* :: $\langle (\text{nat} \Rightarrow 'x \text{ fm}) \Rightarrow 'x \text{ fm} \Rightarrow 'x \text{ fm} \rangle$ **where**

$\langle \text{sub-pro } - (\cdot t) = \cdot t \rangle$
 $\langle \text{sub-pro } s (\cdot P) = \text{case-tm } s (\cdot \ o \ \circ) \ P \rangle$
 $\langle \text{sub-pro } s (\neg \ p) = \neg \ \text{sub-pro } s \ p \rangle$
 $\langle \text{sub-pro } s (p \wedge \ q) = (\text{sub-pro } s \ p \wedge \ \text{sub-pro } s \ q) \rangle$
 $\langle \text{sub-pro } s (\Box \ p) = \Box (\text{sub-pro } s \ p) \rangle$
 $\langle \text{sub-pro } s (@i \ p) = @i (\text{sub-pro } s \ p) \rangle$
 $\langle \text{sub-pro } s (\mathbf{A} \ p) = \mathbf{A} (\text{sub-pro } s \ p) \rangle$
 $\langle \text{sub-pro } s (\downarrow \ p) = \downarrow (\text{sub-pro } (\text{lift-fm-nom } 0 \ o \ s) \ p) \rangle$
 $\langle \text{sub-pro } s (\forall \ p) = \forall (\text{sub-pro } (\cdot (\#0) \gg \ \lambda n. \ \text{lift-fm-pro } 0 \ (s \ n)) \ p) \rangle$

abbreviation *inst-single-pro* :: $\langle 'x \text{ fm} \Rightarrow 'x \text{ fm} \Rightarrow 'x \text{ fm} \rangle (\langle _ \rangle_p)$ **where**

$\langle \langle q \rangle_p \equiv \text{sub-pro } (q \gg \cdot \ o \ \#) \rangle$

primrec *sz-fm* :: $\langle 'x \text{ fm} \Rightarrow \text{nat} \rangle$ **where**

$\langle \text{sz-fm } (\cdot t) = 1 \rangle$
 $\langle \text{sz-fm } (\cdot P) = 1 \rangle$
 $\langle \text{sz-fm } (\neg \ p) = \text{sz-fm } p + 1 \rangle$
 $\langle \text{sz-fm } (p \wedge \ q) = \text{sz-fm } p + \text{sz-fm } q + 1 \rangle$
 $\langle \text{sz-fm } (\Box \ p) = \text{sz-fm } p + 1 \rangle$
 $\langle \text{sz-fm } (@i \ p) = \text{sz-fm } p + 1 \rangle$
 $\langle \text{sz-fm } (\mathbf{A} \ p) = \text{sz-fm } p + 1 \rangle$
 $\langle \text{sz-fm } (\downarrow \ p) = \text{sz-fm } p + 1 \rangle$
 $\langle \text{sz-fm } (\forall \ p) = \text{sz-fm } p + 1 \rangle$

primrec *qs-fm* :: $\langle 'x \text{ fm} \Rightarrow \text{nat} \rangle$ **where**

$\langle \text{qs-fm } (\cdot t) = 0 \rangle$
 $\langle \text{qs-fm } (\cdot P) = 0 \rangle$
 $\langle \text{qs-fm } (\neg \ p) = \text{qs-fm } p \rangle$
 $\langle \text{qs-fm } (p \wedge \ q) = \max (\text{qs-fm } p) (\text{qs-fm } q) \rangle$
 $\langle \text{qs-fm } (\Box \ p) = \text{qs-fm } p \rangle$
 $\langle \text{qs-fm } (@i \ p) = \text{qs-fm } p \rangle$
 $\langle \text{qs-fm } (\mathbf{A} \ p) = \text{qs-fm } p \rangle$
 $\langle \text{qs-fm } (\downarrow \ p) = \text{qs-fm } p \rangle$

| $\langle qs\text{-fm } (\forall p) = qs\text{-fm } p + 1 \rangle$

4.3.1 Lemmas

Finite

lemma *finite-symbols-tm* [*simp*]: $\langle \text{finite } (\text{symbols-tm } t) \rangle$
by (*induct t*) *simp-all*

lemma *finite-symbols-fm* [*simp*]: $\langle \text{finite } (\text{symbols-fm } p) \rangle$
by (*induct p*) *simp-all*

lemma *finite-symbols-lbd*: $\langle \text{finite } (\text{symbols-lbd } p) \rangle$
by (*cases p*) *simp*

Terms

lemma *env* [*simp*]: $\langle P ((x \gg E) n) = (P x \gg \lambda n. P (E n)) n \rangle$
by (*induct n*) *simp-all*

lemma *lift-lemma* [*simp*]: $\langle \text{case-tm } (x \gg e) s (\text{lift-tm } 0 t) = \text{case-tm } e s t \rangle$
by (*induct t*) *auto*

lemma *sub-tm-semantics* [*simp*]: $\langle \text{case-tm } e g (\text{sub-tm } s t) = \text{case-tm } (\lambda n. \text{case-tm } e g (s n)) g t \rangle$
by (*induct t*) (*auto split: tm.splits*)

lemma *semantics-tm-id* [*simp*]: $\langle \text{case-tm } \# \circ t = t \rangle$
by (*induct t*) *auto*

lemma *semantics-tm-id-map* [*simp*]: $\langle \text{map } (\text{case-tm } \# \circ) ts = ts \rangle$
by (*auto cong: map-cong*)

lemma *map-tm-sub-tm* [*simp*]: $\langle \text{map-tm } f (\text{sub-tm } g t) = \text{sub-tm } (\text{map-tm } f o g) (\text{map-tm } f t) \rangle$
by (*induct t*) *simp-all*

lemma *map-tm-lift-tm* [*simp*]: $\langle \text{map-tm } f (\text{lift-tm } m t) = \text{lift-tm } m (\text{map-tm } f t) \rangle$
by (*induct t*) *simp-all*

lemma *semantics-tm-fresh* [*simp*]: $\langle x \notin \text{symbols-tm } t \implies \text{case-tm } e (g(x := a)) t = \text{case-tm } e g t \rangle$
by (*induct t*) *auto*

lemma *map-tm-inst-single*: $\langle (\text{map-tm } f o (u \gg \#)) t = (\text{map-tm } f u \gg \#) t \rangle$
by (*induct t*) *auto*

Nominals

lemma *size-sub-nom* [*simp*]: $\langle \text{sz-fm } (\text{sub-nom } s p) = \text{sz-fm } p \rangle$

by (induct p arbitrary: s) simp-all

lemma semantics-symbols-cong-nom:

assumes $\langle \forall i \in \text{symbols-fm } p. N i = N' i \rangle$

shows $\langle (\text{Model } W R P I N e V f, w) \models p \longleftrightarrow (\text{Model } W R P I N' e V f, w) \models p \rangle$

using assms by (induct p arbitrary: e f w) (simp-all split: tm.splits)

corollary semantics-symbols-other-nom [simp]:

assumes $\langle i \notin \text{symbols-fm } p \rangle$

shows $\langle (\text{Model } W R P I (N(i := x)) e V f, w) \models p \longleftrightarrow (\text{Model } W R P I N e V f, w) \models p \rangle$

using assms by (simp add: semantics-symbols-cong-nom)

lemma sub-nom-semantics [simp]:

$\langle (\text{Model } W R P I N e V f, w) \models \text{sub-nom } s p \longleftrightarrow (\text{Model } W R P I N (\lambda n. \text{case-tm } e N (s n)) V f, w) \models p \rangle$

by (induct p arbitrary: e f s w) (simp-all split: tm.splits)

lemma map-fm-sub-nom: $\langle \text{map-fm } f (\text{sub-nom } s p) = \text{sub-nom } (\text{map-tm } f o s) (\text{map-fm } f p) \rangle$

by (induct p arbitrary: s) (simp-all add: comp-def)

lemma map-fm-inst-single-nom [simp]: $\langle \text{map-fm } f (\langle t \rangle_i p) = \langle \text{map-tm } f t \rangle_i (\text{map-fm } f p) \rangle$

unfolding map-fm-sub-nom map-tm-inst-single ..

Propositions

lemma semantics-symbols-cong-pro:

$\langle \forall P \in \text{symbols-fm } p. V P = V' P \implies$

$(\text{Model } W R P I N e V f, w) \models p \longleftrightarrow (\text{Model } W R P I N e V' f, w) \models p \rangle$

by (induct p arbitrary: e f w) (simp-all split: tm.splits)

corollary semantics-symbols-other-pro [simp]:

assumes $\langle P \notin \text{symbols-fm } p \rangle$

shows $\langle (\text{Model } W R P I N e (V(P := x)) f, w) \models p \longleftrightarrow (\text{Model } W R P I N e V f, w) \models p \rangle$

using assms by (simp add: semantics-symbols-cong-pro)

lemma semantics-symbols-lbd-cong-pro:

$\langle \forall P \in \text{symbols-lbd } (i, p). V P = V' P \implies$

$(\text{Model } W R P I N e V f, w) \models p \longleftrightarrow (\text{Model } W R P I N e V' f, w) \models p \rangle$

by (simp add: semantics-symbols-cong-pro)

lemma map-lift-fm-pro [simp]: $\langle \text{map-fm } f (\text{lift-fm-pro } m p) = \text{lift-fm-pro } m (\text{map-fm } f p) \rangle$

by (induct p arbitrary: m) simp-all

lemma map-lift-fm-nom [simp]: $\langle \text{map-fm } f (\text{lift-fm-nom } m p) = \text{lift-fm-nom } m$

$\langle \text{map-fm } f \ p \rangle$
by $(\text{induct } p \text{ arbitrary: } m) \text{ simp-all}$

lemma map-fm-sub-pro : $\langle \text{map-fm } f \ (\text{sub-pro } s \ p) = \text{sub-pro } (\text{map-fm } f \ o \ s) \ (\text{map-fm } f \ p) \rangle$
by $(\text{induct } p \text{ arbitrary: } s) \ (\text{simp-all add: comp-def split: tm.splits})$

lemma $\text{map-fm-inst-single}$: $\langle (\text{map-fm } f \ o \ (q \gg \cdot \ o \ \#)) \ p = (\text{map-fm } f \ q \gg \cdot \ o \ \#) \ p \rangle$
by $(\text{induct } p) \text{ auto}$

lemma $\text{map-fm-inst-single-pro}$ $[\text{simp}]$: $\langle \text{map-fm } f \ (\langle q \rangle_p \ p) = \langle \text{map-fm } f \ q \rangle_p \ (\text{map-fm } f \ p) \rangle$
unfolding $\text{map-fm-sub-pro map-fm-inst-single ..}$

4.3.2 softqdf

lemma softqdf-map-fm $[\text{simp}]$: $\langle \text{softqdf } (\text{map-fm } f \ p) \longleftrightarrow \text{softqdf } p \rangle$
by $(\text{induct } p) \text{ auto}$

lemma $\text{softqdf-lift-fm-nom}$ $[\text{simp}]$: $\langle \text{softqdf } (\text{lift-fm-nom } m \ q) \longleftrightarrow \text{softqdf } q \rangle$
by $(\text{induct } q \text{ arbitrary: } m) \text{ auto}$

lemma $\text{softqdf-lift-fm-pro}$ $[\text{simp}]$: $\langle \text{softqdf } (\text{lift-fm-pro } m \ q) \longleftrightarrow \text{softqdf } q \rangle$
by $(\text{induct } q) \text{ auto}$

4.3.3 Add env

lemma range-add-env :
assumes $\langle \text{range } f \subseteq A \rangle \langle a \in A \rangle$
shows $\langle \text{range } (a \gg f) \subseteq A \rangle$
proof safe
fix $x \ n$
show $\langle (a \gg f) \ n \in A \rangle$
using $\text{assms by } (\text{induct } n) \text{ auto}$
qed

lemma softqdf-add-env : $\langle \text{softqdf } q \Longrightarrow \text{softqdf-sub } (q \gg \cdot \ o \ \#) \rangle$
by $(\text{metis add-env.simps comp-apply not0-implies-Suc softqdf.simps}(2))$

lemma wf-env-add-nom : $\langle \text{wf-env } (\text{Model } W \ R \ PI \ N \ e \ V \ f) \Longrightarrow w \in W \Longrightarrow \text{wf-env } (\langle \mathcal{W} = W, \mathcal{R} = R, \Pi = PI, \mathcal{N} = N, \mathfrak{N} = w \gg e, \mathcal{V} = V, \mathfrak{V} = f \rangle) \rangle$
unfolding $\text{wf-env-def unfolds using range-add-env by meson}$

lemma wf-model-add-nom : $\langle \text{wf-model } (\text{Model } W \ R \ PI \ N \ e \ V \ f) \Longrightarrow w \in W \Longrightarrow \text{wf-model } (\langle \mathcal{W} = W, \mathcal{R} = R, \Pi = PI, \mathcal{N} = N, \mathfrak{N} = w \gg e, \mathcal{V} = V, \mathfrak{V} = f \rangle) \rangle$
using $\text{wf-env-add-nom unfolding wf-model-def wf-env-def wf-frame-def wf-gframe-def admissible-def unfolds by meson}$

lemma wf-env-add-pro : $\langle \text{wf-env } (\text{Model } W \ R \ PI \ N \ e \ V \ f) \Longrightarrow P \in PI \Longrightarrow$

$\langle wf\text{-env } (\mathcal{W} = W, \mathcal{R} = R, \Pi = PI, \mathcal{N} = N, \mathfrak{N} = e, \mathcal{V} = V, \mathfrak{B} = P \gg f) \rangle$
unfolding $wf\text{-env-def}$ unfolds using $range\text{-add-env}$ by $meson$

lemma $wf\text{-model-add-pro}$:

$\langle wf\text{-model } (Model\ W\ R\ PI\ N\ e\ V\ f) \implies P \in PI \implies wf\text{-model } (Model\ W\ R\ PI\ N\ e\ V\ (P \gg f)) \rangle$

using $wf\text{-env-add-pro}$ **unfolding** $wf\text{-model-def}$ $wf\text{-env-def}$ $wf\text{-frame-def}$ $wf\text{-gframe-def}$ $admissible\text{-def}$ unfolds by $meson$

lemma $softqdf\text{-lift-fm-nom-add-env}$ [simp]:

$\langle softqdf\ p \implies (Model\ W\ R\ PI\ N\ (v \gg e)\ V\ f, w) \models lift\text{-fm-nom}\ 0\ p \iff (Model\ W\ R\ PI\ N\ e\ V\ f, w) \models p \rangle$

by ($induct\ p\ arbitrary: e\ w$) ($simp\text{-all}\ split: tm.\ splits$)

lemma $softqdf\text{-lift-fm-pro-add-env}$ [simp]:

$\langle softqdf\ p \implies (Model\ W\ R\ PI\ N\ e\ V\ (P \gg f), w) \models lift\text{-fm-pro}\ 0\ p \iff (Model\ W\ R\ PI\ N\ e\ V\ f, w) \models p \rangle$

by ($induct\ p\ arbitrary: e\ w$) ($simp\text{-all}\ split: tm.\ splits$)

4.3.4 Sizes

lemma $qs\text{-fm-sub-nom}$ [simp]: $\langle qs\text{-fm } (sub\text{-nom}\ s\ p) = qs\text{-fm}\ p \rangle$

by ($induct\ p\ arbitrary: s$) $auto$

lemma $softqdf\text{-qs-fm}$ [simp]: $\langle softqdf\ q \implies qs\text{-fm}\ q = 0 \rangle$

by ($induct\ q$) $simp\text{-all}$

lemma $softqdf\text{-sub-pro}$: $\langle softqdf\text{-sub}\ s \implies softqdf\text{-sub}\ (\bullet(\#0) \gg \lambda n. lift\text{-fm-pro}\ 0\ (s\ n)) \rangle$

by ($metis\ add\text{-env}.\ simps\ not0\ implies\ Suc\ softqdf.\ simps(2)\ softqdf\text{-lift-fm-pro}$)

lemma $qs\text{-fm-sub-pro}$ [simp]: $\langle softqdf\text{-sub}\ s \implies qs\text{-fm } (sub\text{-pro}\ s\ p) = qs\text{-fm}\ p \rangle$

proof ($induct\ p\ arbitrary: s$)

case ($All\ p$)

then show $?case$

using $softqdf\text{-sub-pro}$ **by force**

qed ($auto\ split: tm.\ splits$)

4.4 Propositional Quantification

definition $worlds$:: $\langle ('x, 'w)\ model \Rightarrow 'x\ fm \Rightarrow 'w\ set \rangle$ **where**

$\langle worlds\ M\ p \equiv \{ w \in \mathcal{W}\ M. (M, w) \models p \} \rangle$

lemma $worlds\text{-op}$ [simp]:

assumes $\langle wf\text{-model}\ M \rangle$

shows

$\langle worlds\ M\ (\neg p) = \mathcal{W}\ M - worlds\ M\ p \rangle$

$\langle worlds\ M\ (p \wedge q) = worlds\ M\ p \cap worlds\ M\ q \rangle$

$\langle worlds\ M\ (\Box p) = \{ w \in \mathcal{W}\ M. \forall v \in \mathcal{R}\ M\ w. v \in worlds\ M\ p \} \rangle$

using *assms* **unfolding** *worlds-def wf-model-def wf-frame-def wf-gframe-def unf-
folds* **by** *auto*

lemma *softqdf-worlds*:

assumes $\langle wf\text{-model } M \rangle \langle softqdf\ p \rangle$

shows $\langle worlds\ M\ p \in \Pi\ M \rangle$

using *assms*

proof (*induct p*)

case (*TmI x*)

then show *?case*

by *simp*

next

case (*TmP x*)

then have $\langle \mathcal{V}\ M\ P \in \Pi\ M \rangle \langle \mathfrak{W}\ M\ n \in \Pi\ M \rangle$ **for** $P\ n$

using *wf-V wf-W* **by** *fastforce+*

moreover have $\langle \mathcal{W}\ M \in \Pi\ M \rangle$

using *TmP wf-univ* **by** *fastforce*

ultimately have $\langle \mathcal{V}\ M\ P \cap \mathcal{W}\ M \in \Pi\ M \rangle \langle \mathfrak{W}\ M\ n \cap \mathcal{W}\ M \in \Pi\ M \rangle$ **for** $P\ n$

using *TmP wf-inter* **by** *fastforce+*

then have $\langle \{w \in \mathcal{W}\ M. w \in \mathcal{V}\ M\ P\} \in \Pi\ M \rangle \langle \{w \in \mathcal{W}\ M. w \in \mathfrak{W}\ M\ n\} \in \Pi\ M \rangle$ **for** $P\ n$

by (*metis Int-def inf-commute*)+

then show *?case*

unfolding *worlds-def*

by (*auto split: tm.splits*)

next

case (*Neg p*)

then show *?case*

by (*simp add: wf-compl*)

next

case (*Con p q*)

then show *?case*

by (*simp add: wf-inter*)

next

case (*Box p*)

then show *?case*

by (*simp add: wf-modal*)

next

case (*Sat k p*)

then have $\langle (\forall w. (M, w) \models @k\ p) \vee (\nexists w. (M, w) \models @k\ p) \rangle$

by (*auto split: tm.splits*)

then have $\langle worlds\ M\ (@k\ p) = \{\} \vee worlds\ M\ (@k\ p) = \mathcal{W}\ M \rangle$

unfolding *worlds-def* **by** *blast*

then show *?case*

using *Sat(2) wf-univ wf-empty* **by** *fastforce*

next

case (*Glo p*)

then have $\langle (\forall w. (M, w) \models \mathbf{A}\ p) \vee (\nexists w. (M, w) \models \mathbf{A}\ p) \rangle$

by *auto*

```

then have  $\langle \text{worlds } M \ (\mathbf{A} \ p) = \{\} \vee \text{worlds } M \ (\mathbf{A} \ p) = \mathcal{W} \ M \rangle$ 
  unfolding worlds-def by blast
then show ?case
  using Glo(2) wf-univ wf-empty by fastforce
next
  case (Dwn p)
  then show ?case
    by simp
next
  case (All p)
  then show ?case
    by simp
qed

```

```

definition sqdfs ::  $\langle 'x, 'w \rangle \text{ model} \Rightarrow 'w \text{ set set} \rangle$  where
   $\langle \text{sqdfs } M \equiv \{ \text{worlds } M \ p \mid p. \text{softqdf } p \} \rangle$ 

```

```

lemma sqdfs:
  assumes  $\langle \text{wf-model } M \rangle$ 
  shows  $\langle \text{sqdfs } M \subseteq \Pi \ M \rangle$ 
  unfolding sqdfs-def using assms softqdf-worlds by blast

```

```

lemma sqdfs-admissible:
  assumes  $\langle \text{wf-model } M \rangle$ 
  shows  $\langle \text{admissible } (\text{frame.truncate } M) \ (\text{sqdfs } M) \rangle$ 
  unfolding admissible-def

```

```

proof safe
  fix X
  assume  $\langle X \in \text{sqdfs } M \rangle$ 
  then obtain p where  $\langle X = \text{worlds } M \ p \rangle \langle \text{softqdf } p \rangle$ 
    unfolding sqdfs-def by fast
  moreover from this have  $\langle \text{worlds } M \ (\neg \ p) \in \text{sqdfs } M \rangle$ 
    unfolding sqdfs-def by auto
  ultimately show  $\langle \mathcal{W} \ (\text{frame.truncate } M) - X \in \text{sqdfs } M \rangle$ 
    unfolding sqdfs-def unfolds using assms by simp
next
  fix X Y
  assume  $\langle X \in \text{sqdfs } M \rangle \langle Y \in \text{sqdfs } M \rangle$ 
  then obtain p q where  $\langle X = \text{worlds } M \ p \rangle \langle \text{softqdf } p \rangle \langle Y = \text{worlds } M \ q \rangle \langle \text{softqdf } q \rangle$ 
    unfolding sqdfs-def by fast
  moreover from this have  $\langle \text{worlds } M \ (p \wedge \ q) \in \text{sqdfs } M \rangle$ 
    unfolding sqdfs-def by auto
  ultimately show  $\langle X \cap Y \in \text{sqdfs } M \rangle$ 
    unfolding sqdfs-def using assms by simp
next
  fix X
  assume  $\langle X \in \text{sqdfs } M \rangle$ 
  then obtain p where  $\langle X = \text{worlds } M \ p \rangle \langle \text{softqdf } p \rangle$ 

```

unfolding *sqdfs-def* **by** *fast*
moreover from this have $\langle \text{worlds } M (\Box p) \in \text{sqdfs } M \rangle$
unfolding *sqdfs-def* **by** *auto*
ultimately show $\langle \{w \in \mathcal{W} (\text{frame.truncate } M). \forall v \in \mathcal{R} (\text{frame.truncate } M) w. v \in X\} \in \text{sqdfs } M \rangle$
unfolding *sqdfs-def* **unfolds using** *assms* **by** *simp*
qed

definition *with-worlds* $:: \langle ('x, 'w) \text{ model} \Rightarrow (\text{nat} \Rightarrow 'x \text{ fm}) \Rightarrow ('x, 'w) \text{ model} \rangle$
where
 $\langle \text{with-worlds } M s \equiv M(\mathfrak{W} := \text{worlds } M \circ s) \rangle$

lemma *sub-pro-with-worlds*:

assumes $\langle \text{wf-model } (\text{Model } W R P I N e V f) \rangle \langle w \in W \rangle \langle \text{softqdf-sub } s \rangle$
shows $\langle (\text{Model } W R P I N e V f, w) \models \text{sub-pro } s p \iff (\text{with-worlds } (\text{Model } W R P I N e V f) s, w) \models p \rangle$

using *assms*

proof (*induct p arbitrary: s e f w*)

case (*Box p*)

moreover from this have $\langle v \in R w \implies v \in W \rangle$ **for** *v*

unfolding *wf-model-def wf-gframe-def wf-frame-def* **unfolds by** *auto*

ultimately show *?case*

unfolding *with-worlds-def* **by** *simp*

next

case (*Sat x p*)

moreover from this have $\langle \text{case-tm } e N x \in W \rangle$

unfolding *wf-model-def wf-env-def*

by (*auto split: tm.splits*)

ultimately show *?case*

unfolding *with-worlds-def*

by (*simp split: tm.splits*)

next

case (*Dwn p*)

let $?s = \langle \text{lift-fm-nom } 0 \circ s \rangle$

have $\langle \text{wf-model } (\text{Model } W R P I N (w \gg e) V f) \rangle$

using *Dwn* **by** (*simp add: wf-model-add-nom*)

moreover have $\langle \text{softqdf-sub } ?s \rangle$

using *Dwn* **by** *simp*

ultimately have $\langle (\text{Model } W R P I N (w \gg e) V f, w) \models \text{sub-pro } ?s p \iff$

$(\text{with-worlds } (\text{Model } W R P I N (w \gg e) V f) ?s, w) \models p \rangle$

using *Dwn* **by** *fastforce*

moreover have $\langle \text{worlds } (\text{Model } W R P I N (w \gg e) V f) \circ ?s = \text{worlds } (\text{Model } W R P I N e V f) \circ s \rangle$

unfolding *worlds-def comp-def* **using** *Dwn(4)* **by** *simp*

ultimately have $*$: $\langle (\text{Model } W R P I N (w \gg e) V f, w) \models \text{sub-pro } ?s p \iff$

$(\text{Model } W R P I N (w \gg e) V (\text{worlds } (\text{Model } W R P I N e V f) \circ s), w) \models p \rangle$

unfolding *with-worlds-def* **by** *simp*

moreover have $\langle (with\text{-}worlds (Model\ W\ R\ PI\ N\ e\ V\ f)\ ?s, w) \models \downarrow p \longleftrightarrow (Model\ W\ R\ PI\ N\ (w \gg e)\ V\ (worlds (Model\ W\ R\ PI\ N\ e\ V\ f)\ o\ ?s), w) \models p \rangle$
unfolding *with-worlds-def* **by** *simp*
then have $\langle (with\text{-}worlds (Model\ W\ R\ PI\ N\ e\ V\ f)\ ?s, w) \models \downarrow p \longleftrightarrow (Model\ W\ R\ PI\ N\ (w \gg e)\ V\ (worlds (Model\ W\ R\ PI\ N\ e\ V\ f)\ o\ ?s), w) \models p \rangle$
by *metis*

ultimately show *?case*
unfolding *with-worlds-def* **by** *simp*
next
case *(All p)*
let $?s = \langle \cdot (\# 0) \gg \lambda n. lift\text{-}fm\text{-}pro\ 0\ (s\ n) \rangle$

have $\langle softqdf\ (?s\ n) \rangle$ **for** n
using *All(4)* **by** *(induct n) auto*
then have $\langle softqdf\text{-}sub\ ?s \rangle$
by *blast*
moreover have $\langle \forall P \in PI. wf\text{-}model (Model\ W\ R\ PI\ N\ e\ V\ (P \gg f)) \rangle$
using *All(2)* *wf-model-add-pro* **by** *blast*
ultimately have $\langle (\forall P \in PI. (Model\ W\ R\ PI\ N\ e\ V\ (P \gg f), w) \models sub\text{-}pro\ ?s\ p) \longleftrightarrow (\forall P \in PI. (with\text{-}worlds (Model\ W\ R\ PI\ N\ e\ V\ (P \gg f))\ ?s, w) \models p) \rangle$
using *All* **by** *blast*

moreover have $\langle \forall P \in PI. P \subseteq W \rangle$
using *All(2)* **unfolding** *wf-model-def* *wf-gframe-def* **unfolds** **by** *blast*
then have $\langle P \in PI \implies (worlds (Model\ W\ R\ PI\ N\ e\ V\ (P \gg f))\ o\ ?s)\ n = (P \gg worlds (Model\ W\ R\ PI\ N\ e\ V\ f)\ o\ s)\ n \rangle$ **for** $P\ n$
unfolding *worlds-def* **using** *All(4)* **by** *(induct n) auto*
then have $\langle P \in PI \implies (worlds (Model\ W\ R\ PI\ N\ e\ V\ (P \gg f))\ o\ ?s) = (P \gg worlds (Model\ W\ R\ PI\ N\ e\ V\ f)\ o\ s) \rangle$ **for** P
by *blast*

ultimately show *?case*
unfolding *with-worlds-def* **by** *simp*
qed *(simp-all add: worlds-def with-worlds-def split: tm.splits)*

lemma *worlds-id-sub:*
assumes $\langle wf\text{-}model (Model\ W\ R\ PI\ N\ e\ V\ f) \rangle$
shows $\langle worlds (Model\ W\ R\ PI\ N\ e\ V\ f)\ (\cdot (\# n)) = f\ n \rangle$
using *wf- \mathfrak{N} [OF assms]* **unfolding** *worlds-def* **unfolds** **by** *auto*

lemma *worlds-inst-single-pro:*
assumes $\langle wf\text{-}model (Model\ W\ R\ PI\ N\ e\ V\ f) \rangle$
shows $\langle worlds (Model\ W\ R\ PI\ N\ e\ V\ f)\ o\ (q \gg \cdot o \#) = (worlds (Model\ W\ R$

$PI N e V f) q \gg f)$
unfolding *comp-def env worlds-id-sub[OF assms]* ..

corollary *inst-single-worlds*:

assumes $\langle wf\text{-model} (Model W R PI N e V f) \rangle \langle w \in W \rangle \langle softqdf q \rangle$

shows

$\langle (Model W R PI N e V f, w) \models \langle q \rangle_p p \longleftrightarrow$
 $(Model W R PI N e V (worlds (Model W R PI N e V f) q \gg f), w) \models p \rangle$

proof –

have

$\langle (Model W R PI N e V f, w) \models \langle q \rangle_p p \longleftrightarrow$
 $(Model W R PI N e V (worlds (Model W R PI N e V f) o (q \gg \cdot o \#)), w)$

$\models p \rangle$

using *sub-pro-with-worlds[OF assms(1–2)] assms(3) unfolding with-worlds-def*

by (*metis model.update-convs(4) softqdf-add-env*)

then show *?thesis*

using *assms worlds-inst-single-pro by metis*

qed

4.5 Model Existence

inductive *conft-class* :: $\langle 'x \text{ lbd list} \Rightarrow 'x \text{ lbd list} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\mathbf{X}} \rangle$ 50) **where**

CNegP: $\langle [(i, \neg \cdot P)] \rightsquigarrow_{\mathbf{X}} [(i, \cdot P)] \rangle$

| *CNegI*: $\langle [(i, \neg \cdot k)] \rightsquigarrow_{\mathbf{X}} [(i, \cdot k)] \rangle$

inductive *alpha-class* :: $\langle 'x \text{ lbd list} \Rightarrow 'x \text{ lbd list} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\alpha} \rangle$ 50) **where**

CSym: $\langle [(i, \cdot k)] \rightsquigarrow_{\alpha} [(k, \cdot i)] \rangle$

| *CNom*: $\langle [(i, \cdot k), (i, p)] \rightsquigarrow_{\alpha} [(k, p)] \rangle$

| *CNegN*: $\langle [(i, \neg \neg p)] \rightsquigarrow_{\alpha} [(i, p)] \rangle$

| *CConP*: $\langle [(i, p \wedge q)] \rightsquigarrow_{\alpha} [(i, p), (i, q)] \rangle$

| *CSatP*: $\langle [(i, @k p)] \rightsquigarrow_{\alpha} [(k, p)] \rangle$

| *CSatN*: $\langle [(i, \neg @k p)] \rightsquigarrow_{\alpha} [(k, \neg p)] \rangle$

| *CBoxP*: $\langle [(i, \Box p), (i, \Diamond \cdot k)] \rightsquigarrow_{\alpha} [(k, p)] \rangle$

| *CDwnP*: $\langle [(i, \Downarrow p)] \rightsquigarrow_{\alpha} [(i, \langle i \rangle_i p)] \rangle$

| *CDwnN*: $\langle [(i, \neg \Downarrow p)] \rightsquigarrow_{\alpha} [(i, \neg \langle i \rangle_i p)] \rangle$

inductive *beta-class* :: $\langle 'x \text{ lbd list} \Rightarrow 'x \text{ lbd list} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\beta} \rangle$ 50) **where**

CConN: $\langle [(i, \neg (p \wedge q))] \rightsquigarrow_{\beta} [(i, \neg p), (i, \neg q)] \rangle$

inductive *gamma-class-nom* :: $\langle 'x \text{ lbd list} \Rightarrow ('x \text{ tm} \Rightarrow -) \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\gamma i} \rangle$ 50) **where**

CRefI: $\langle [] \rightsquigarrow_{\gamma i} (\lambda i. [(i, \cdot i)]) \rangle$

| *CGloP*: $\langle [(i, \mathbf{A} p)] \rightsquigarrow_{\gamma i} (\lambda k. [(k, p)]) \rangle$

inductive *gamma-class-fm* :: $\langle 'x \text{ lbd list} \Rightarrow ('x \text{ lbd set} \Rightarrow 'x \text{ fm set}) \times ('x \text{ fm} \Rightarrow -) \Rightarrow \text{bool} \rangle$ (**infix** $\langle \rightsquigarrow_{\gamma p} \rangle$ 50) **where**

CALLP: $\langle [(i, \forall p)] \rightsquigarrow_{\gamma p} (\lambda \cdot. \{q. softqdf q\}, \lambda q. [(i, \langle q \rangle_p p)]) \rangle$

fun δ :: $\langle 'x \text{ lbd} \Rightarrow 'x \Rightarrow 'x \text{ lbd list} \rangle$ **where**

$CBoxN: \langle \delta (i, \neg \Box p) k = [(\Box k, \neg p), (i, \Diamond (\cdot (\Box k)))] \rangle$
 $| CGloN: \langle \delta (i, \neg \mathbf{A} p) k = [(\Box k, \neg p)] \rangle$
 $| CAllN: \langle \delta (i, \neg \forall p) P = [(i, \neg \langle \cdot (\Box P) \rangle_p p)] \rangle$
 $| \langle \delta - - = [] \rangle$

interpretation P : *Params map-lbd symbols-lbd* $\langle \lambda-. True \rangle$
by *unfold-locales (auto simp: tm.map-id0 fm.map-id0 cong: tm.map-cong0 fm.map-cong0)*

interpretation C : *Confl map-lbd symbols-lbd* $\langle \lambda-. True \rangle$ *confl-class*
by *unfold-locales (auto elim!: confl-class.cases intro: confl-class.intros)*

interpretation A : *Alpha map-lbd symbols-lbd* $\langle \lambda-. True \rangle$ *alpha-class*
by *unfold-locales (auto simp: fm.map-id0 cong: fm.map-cong0 elim!: alpha-class.cases intro: alpha-class.intros)*

interpretation B : *Beta map-lbd symbols-lbd* $\langle \lambda-. True \rangle$ *beta-class*
by *unfold-locales (auto simp: fm.map-id0 cong: fm.map-cong0 elim!: beta-class.cases intro: beta-class.intros)*

interpretation GI : *Gamma-UNIV map-tm map-lbd symbols-lbd* $\langle \lambda-. True \rangle$ *gamma-class-nom*
by *unfold-locales (fastforce elim: gamma-class-nom.cases intro: gamma-class-nom.intros)+*

interpretation GP : *Gamma map-fm map-lbd symbols-lbd* $\langle \lambda-. True \rangle$ *gamma-class-fm*
by *unfold-locales (fastforce elim: gamma-class-fm.cases intro: gamma-class-fm.intros)+*

interpretation D : *Delta map-lbd symbols-lbd* $\langle \lambda-. True \rangle$ δ
proof

show $\langle \wedge f. \delta (map\text{-}lbd\ f\ p) (f\ x) = map (map\text{-}lbd\ f) (\delta\ p\ x) \rangle$ **for** $p :: \langle 'x\ lbd \rangle$ **and**
 x
by *(induct p x rule: δ .induct) simp-all*
qed

abbreviation $Kinds :: \langle 'x, 'x\ lbd \rangle$ *kind list* **where**
 $\langle Kinds \equiv [C.kind, A.kind, B.kind, GI.kind, GP.kind, D.kind] \rangle$

lemma *prop_E-Kinds*:

assumes $\langle P.sat_E\ C.kind\ C \rangle \langle P.sat_E\ A.kind\ C \rangle \langle P.sat_E\ B.kind\ C \rangle \langle P.sat_E\ GI.kind\ C \rangle \langle P.sat_E\ GP.kind\ C \rangle \langle P.sat_E\ D.kind\ C \rangle$
shows $\langle P.prop_E\ Kinds\ C \rangle$
unfolding $P.prop_E\text{-}def$ **using** *assms* **by** *simp*

interpretation *Consistency-Kinds map-lbd symbols-lbd* $\langle \lambda-. True \rangle$ $Kinds$
using $P.Params\text{-}axioms\ C.Consistency\text{-}Kind\text{-}axioms\ A.Consistency\text{-}Kind\text{-}axioms\ B.Consistency\text{-}Kind\text{-}axioms$

$GI.Consistency\text{-}Kind\text{-}axioms\ GP.Consistency\text{-}Kind\text{-}axioms\ D.Consistency\text{-}Kind\text{-}axioms$
by *(auto intro: Consistency-Kinds.intro simp: Consistency-Kinds-axioms-def)*

interpretation *Maximal-Consistency map-lbd symbols-lbd* $\langle \lambda-. True \rangle$ $Kinds$
proof

have $\langle \text{infinite } (UNIV :: 'x \text{ fm set}) \rangle$
using $\text{infinite-UNIV-size}$ [of $\langle \lambda p. p \longrightarrow p \rangle$] **by** simp
then show $\langle \text{infinite } (UNIV :: 'x \text{ lbd set}) \rangle$
using finite-prod **by** blast
qed simp

context begin

lemma $\langle P.\text{prop}_E \text{ Kinds } C \Longrightarrow S \in C \Longrightarrow (i, \cdot P) \notin S \vee (i, \neg \cdot P) \notin S \rangle$
using sat_E [of $C.\text{kind}$] **by** $(\text{force intro: } C\text{Neg}P)$

lemma $\langle P.\text{prop}_E \text{ Kinds } C \Longrightarrow S \in C \Longrightarrow (i, p \wedge q) \in S \Longrightarrow \{(i, p), (i, q)\} \cup S \in C \rangle$
using sat_E [of $A.\text{kind}$] **by** $(\text{force intro: } C\text{Con}P)$

lemma $\langle P.\text{prop}_E \text{ Kinds } C \Longrightarrow S \in C \Longrightarrow (i, \neg (p \wedge q)) \in S \Longrightarrow \{(i, \neg p)\} \cup S \in C \vee \{(i, \neg q)\} \cup S \in C \rangle$
using sat_E [of $B.\text{kind}$] **by** $(\text{force intro: } C\text{Con}N)$

lemma $\langle P.\text{prop}_E \text{ Kinds } C \Longrightarrow S \in C \Longrightarrow (i, \Box p) \in S \Longrightarrow (i, \Diamond(\cdot k)) \in S \Longrightarrow \{(k, p)\} \cup S \in C \rangle$
using sat_E [of $A.\text{kind}$] **by** $(\text{force intro: } C\text{Box}P)$

lemma $\langle P.\text{prop}_E \text{ Kinds } C \Longrightarrow S \in C \Longrightarrow (i, \neg \Box p) \in S \Longrightarrow \exists k. \{(k, \neg p), (i, \Diamond(\cdot k))\} \cup S \in C \rangle$
using sat_E [of $D.\text{kind}$] **by** fastforce

lemma $\langle P.\text{prop}_E \text{ Kinds } C \Longrightarrow S \in C \Longrightarrow \{(i, \cdot i)\} \cup S \in C \rangle$
using sat_E [of $GI.\text{kind}$] **by** $(\text{force intro: } C\text{Refl})$

lemma $\langle P.\text{prop}_E \text{ Kinds } C \Longrightarrow S \in C \Longrightarrow (i, \mathbf{A} p) \in S \Longrightarrow \{(k, p)\} \cup S \in C \rangle$
using sat_E [of $GI.\text{kind}$] **by** $(\text{force intro: } C\text{Glo}P)$

lemma $\langle P.\text{prop}_E \text{ Kinds } C \Longrightarrow S \in C \Longrightarrow (i, \neg \mathbf{A} p) \in S \Longrightarrow \exists k. \{(k, \neg p)\} \cup S \in C \rangle$
using sat_E [of $D.\text{kind}$] **by** fastforce

lemma $\langle P.\text{prop}_E \text{ Kinds } C \Longrightarrow S \in C \Longrightarrow (i, \forall p) \in S \Longrightarrow \text{softqdf } q \Longrightarrow \{(i, \langle q \rangle_p p)\} \cup S \in C \rangle$
using sat_E [of $GP.\text{kind}$] **by** $(\text{force intro: } C\text{All}P)$

lemma $\langle P.\text{prop}_E \text{ Kinds } C \Longrightarrow S \in C \Longrightarrow (i, \neg \forall p) \in S \Longrightarrow \exists P. \{(i, \neg \langle \cdot(\circ P) \rangle_p p)\} \cup S \in C \rangle$
using sat_E [of $D.\text{kind}$] **by** fastforce

end

definition $\text{equiv-nom} :: \langle 'x \text{ lbd set} \Rightarrow 'x \text{ tm} \Rightarrow 'x \text{ tm} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{equiv-nom } S \ i \ k \equiv (i, \cdot k) \in S \rangle$

definition *assign* :: $\langle 'x \text{ tm} \Rightarrow 'x \text{ lbd set} \Rightarrow 'x \text{ tm} \rangle (\langle [-] \rangle [0, 100] 100)$ **where**
 $\langle [i]_S \equiv \text{wo-rel.minim} (|UNIV|) \{k. \text{equiv-nom } S \ i \ k\} \rangle$

definition *reach* :: $\langle 'x \text{ lbd set} \Rightarrow 'x \text{ tm} \Rightarrow 'x \text{ tm set} \rangle$ **where**
 $\langle \text{reach } S \ i \equiv \{[k]_S \mid k. (i, \diamond (\cdot k)) \in S\} \rangle$

definition *val* :: $\langle 'x \text{ lbd set} \Rightarrow 'x \text{ tm} \Rightarrow 'x \text{ tm set} \rangle$ **where**
 $\langle \text{val } S \ P \equiv \{[i]_S \mid i. (i, \cdot P) \in S\} \rangle$

lemma *range-val-ne*: $\langle \text{range} (\text{val } S) \neq \{\} \rangle$
unfolding *val-def* **by** *blast*

lemma *admissible-Pow*: $\langle \text{admissible } F \ (\text{Pow } (\mathcal{W} \ F)) \rangle$
unfolding *admissible-def* **by** *blast*

definition *admits* :: $\langle 'w \text{ frame} \Rightarrow 'w \text{ set set} \Rightarrow 'w \text{ set set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{admits } F \ B \ PI \equiv PI \neq \{\} \wedge PI \subseteq \text{Pow } (\mathcal{W} \ F) \wedge B \subseteq PI \wedge \text{admissible } F \ PI \rangle$

definition *adm- δ* :: $\langle 'w \text{ frame} \Rightarrow 'w \text{ set set} \Rightarrow 'w \text{ set set} \rangle$ **where**
 $\langle \text{adm-}\delta \ M \ PI \equiv$
 $\{ \mathcal{W} \ M - X \mid X. X \in PI \} \cup$
 $\{ X \cap Y \mid X \ Y. X \in PI \wedge Y \in PI \} \cup$
 $\{ \{w \in \mathcal{W} \ M. \forall v \in \mathcal{R} \ M \ w. v \in X\} \mid X. X \in PI \} \rangle$

abbreviation *grow* $F \ B \equiv \lambda PI. B \cup PI \cup \text{adm-}\delta \ F \ (B \cup PI)$

definition *admit* :: $\langle 'w \text{ frame} \Rightarrow 'w \text{ set set} \Rightarrow 'w \text{ set set} \rangle$ **where**
 $\langle \text{admit } F \ B \equiv \text{lfp} (\text{grow } F \ B) \rangle$

lemma *mono-grow*: $\langle \text{mono} (\text{grow } F \ B) \rangle$
unfolding *adm- δ -def* *mono-def* **by** *(auto simp: subset-eq)*

lemma *admissible- δ* : $\langle \text{admissible } F \ B \longleftrightarrow \text{adm-}\delta \ F \ B \subseteq B \rangle$
unfolding *admissible-def* *adm- δ -def* **by** *auto*

lemma *admit-B*: $\langle B \subseteq \text{admit } F \ B \rangle$
unfolding *admit-def* **by** *(meson le-sup-iff lfp-greatest)*

lemma *admit-Pow*: $\langle B \subseteq \text{Pow } (\mathcal{W} \ F) \implies \text{admit } F \ B \subseteq \text{Pow } (\mathcal{W} \ F) \rangle$
unfolding *admit-def* **using** *admissible-Pow* *admissible- δ* *lfp-lowerbound*
by *(metis (no-types, lifting) order-class.order-eq-iff subset-Un-eq sup.absorb-iff1)*

lemma *admissible-grow*: $\langle \text{admissible } F \ B \longleftrightarrow \text{grow } F \ B \ B = B \rangle$
using *admissible- δ* **by** *auto*

lemma *lfp-grow*: $\langle \text{grow } F \ B \ (\text{lfp} (\text{grow } F \ B)) = \text{lfp} (\text{grow } F \ B) \rangle$
using *lfp-unfold* *mono-grow* **by** *blast*

lemma *admit-admissible*: $\langle \text{admissible } F \text{ (admit } F \text{ } B) \rangle$
unfolding *admit-def*
proof –
have $B \cup \text{lfp (grow } F \text{ } B) = \text{lfp (grow } F \text{ } B)$
using *lfp-grow* **by** *blast*
then show *admissible* $F \text{ (lfp (grow } F \text{ } B))$
using *lfp-grow[of B F]* **by** (*simp add: admissible- δ sup.order-iff*)
qed

lemma *admits-admit*: $\langle B \neq \{\} \implies B \subseteq \text{Pow } (\mathcal{W} \text{ } F) \implies \text{admits } F \text{ } B \text{ (admit } F \text{ } B) \rangle$
by (*metis admit-B admit-Pow admit-admissible admits-def subset-empty*)

lemma *admissible-admit*:
assumes $\langle B \neq \{\} \rangle \langle B \subseteq \text{Pow } (\mathcal{W} \text{ } F) \rangle$
shows
 $\langle \text{admit } F \text{ } B \neq \{\} \rangle$
 $\langle \text{admit } F \text{ } B \subseteq \text{Pow } (\mathcal{W} \text{ } F) \rangle$
 $\langle \text{admissible } F \text{ (admit } F \text{ } B) \rangle$
using *assms admits-admit* **unfolding** *admits-def* **by** *metis+*

abbreviation *canonical-frame* :: $\langle 'x \text{ lbd set} \Rightarrow ('x \text{ tm}) \text{ frame} \rangle$ **where**
 $\langle \text{canonical-frame } S \equiv \langle \mathcal{W} = \{[k]_S \mid k. \text{True}\}, \mathcal{R} = \text{reach } S \rangle \rangle$

abbreviation *canonical-gframe* :: $\langle 'x \text{ lbd set} \Rightarrow ('x \text{ tm}) \text{ gframe} \rangle$ **where**
 $\langle \text{canonical-gframe } S \equiv \text{frame.extend (canonical-frame } S)$
 $\langle \Pi = \text{admit (canonical-frame } S) \text{ (range (val } S)) \rangle \rangle$

definition *canonical* :: $\langle 'x \text{ lbd set} \Rightarrow ('x, 'x \text{ tm}) \text{ model} \rangle$ **where**
 $\langle \text{canonical } S \equiv$
 $\text{gframe.extend (canonical-gframe } S)$
 $\langle \mathcal{N} = \lambda i. [\circ i]_S,$
 $\mathfrak{N} = \lambda i. [\# i]_S,$
 $\mathcal{V} = \text{val } S \text{ } o \text{ } \circ,$
 $\mathfrak{V} = \text{val } S \text{ } o \text{ } \#$
 $\rangle \rangle$

lemma *wf-canonical-frame*: $\langle \text{wf-frame (canonical-frame } S) \rangle$
unfolding *wf-frame-def* **unfolds** *reach-def* **by** *auto*

lemma *val-Pow*: $\langle \text{range (val } S) \subseteq \text{Pow } (\mathcal{W} \text{ (canonical-frame } S)) \rangle$
unfolding *val-def* **by** *auto*

lemma *wf-cannonical-gframe*: $\langle \text{wf-gframe (canonical-gframe } S) \rangle$
unfolding *wf-gframe-def* **unfolds** **using** *wf-canonical-frame* *admissible-admit(2)*
admit-B *admit-admissible*
by (*metis (mono-tags, lifting) frame.select-convs(1) range-val-ne subset-empty*
val-Pow)

lemma *admits-val*: $\langle \text{admits } (\text{canonical-frame } S) (\text{range } (\text{val } S)) \text{ } PI \implies \text{val } S \text{ } P \in PI \rangle$
unfolding *admits-def* **by** *blast*

lemma *admit-val*: $\langle \text{val } S \text{ } P \in \text{admit } (\text{canonical-frame } S) (\text{range } (\text{val } S)) \rangle$
using *admits-val* *admits-admit* *val-Pow* **by** (*simp* *add: admit-B* *range-subsetD*)

lemma *wf-canonical-env*: $\langle \text{wf-env } (\text{canonical } S) \rangle$
unfolding *wf-env-def* *canonical-def* *unfolds* **using** *admit-val* **by** *auto*

lemma *wf-gframe-canonical*: $\langle \text{wf-gframe } (\text{gframe.truncate } (\text{canonical } S)) \rangle$
using *wf-cannonical-gframe* **unfolding** *canonical-def* *unfolds* .

lemma *wf-canonical*: $\langle \text{wf-model } (\text{canonical } S) \rangle$
unfolding *wf-model-def* **using** *wf-gframe-canonical* *wf-canonical-env* **by** *blast*

lemma *admissible-sqdfs*: $\langle \text{admissible } (\text{canonical-frame } S) (\text{sqdfs } (\text{canonical } S)) \rangle$
using *sqdfs-admissible*[*OF* *wf-canonical*[*of* *S*]]
unfolding *canonical-def* *unfolds* .

lemma *sqdfs-Pow*: $\langle \text{sqdfs } (\text{canonical } S) \subseteq \text{Pow } (\mathcal{W} (\text{canonical-frame } S)) \rangle$
unfolding *sqdfs-def* *canonical-def* *unfolds* *worlds-def* **by** *blast*

lemma *val-sqdfs*: $\langle \text{val } S \text{ } P \in \text{sqdfs } (\text{canonical } S) \rangle$
unfolding *val-def* *sqdfs-def* *canonical-def* *unfolds* *worlds-def*
by (*auto* *split: tm.splits* *intro!*: *exI*[*of* - $\langle P \rangle$])

lemma *admits-canonical-sqdfs*: $\langle \text{admits } (\text{canonical-frame } S) (\text{range } (\text{val } S)) (\text{sqdfs } (\text{canonical } S)) \rangle$
unfolding *admits-def* **using** *admissible-sqdfs* *sqdfs-Pow* *val-sqdfs* **by** *blast*

definition *canonical-ctx* :: $\langle 'x \text{ lbd set} \Rightarrow 'x \text{ tm} \Rightarrow ('x, 'x \text{ tm}) \text{ ctx} \rangle (\langle \llbracket -, _ \rrbracket \rangle)$ **where**
 $\langle \llbracket S, i \rrbracket \equiv (\text{canonical } S, [i]_S) \rangle$

lemma *sqdfs-canonical*: $\langle \text{sqdfs } (\text{canonical } S) = \Pi (\text{canonical } S) \rangle$
proof
show $\langle \text{sqdfs } (\text{canonical } S) \subseteq \Pi (\text{canonical } S) \rangle$
using *sqdfs* *wf-canonical* **by** *blast*
next
have $\langle \text{grow } (\text{canonical-frame } S) (\text{range } (\text{val } S)) (\text{sqdfs } (\text{canonical } S)) \subseteq \text{sqdfs } (\text{canonical } S) \rangle$
using *admissible-grow*[*of* $\langle \text{canonical-frame } S \rangle \langle \text{sqdfs } - \rangle$] *admits-canonical-sqdfs*[*of* *S*]
unfolding *admits-def*
by (*metis* (*no-types*, *lifting*) *equalityE* *le-iff-sup*)
then have $\langle \text{admit } (\text{canonical-frame } S) (\text{range } (\text{val } S)) \subseteq \text{sqdfs } (\text{canonical } S) \rangle$
by (*simp* *add: admit-def* *lfp-lowerbound*)
then show $\langle \Pi (\text{canonical } S) \subseteq \text{sqdfs } (\text{canonical } S) \rangle$
unfolding *canonical-def* *unfolds* .
qed

lemma *canonical-tm-eta* [*simp*]: $\langle \text{case-tm } (\lambda i. [\# i]_S) (\lambda n. [\circ n]_S) k = [k]_S \rangle$
by (*cases k*) *simp-all*

corollary *canonical-tm-eta'* [*simp*]: $\langle \text{case-tm } (\mathfrak{N} (\text{canonical } S)) (\mathcal{N} (\text{canonical } S))$
 $k = [k]_S \rangle$
unfolding *canonical-def* **unfolds by** *simp*

locale *MyHintikka* = *Hintikka map-lbd symbols-lbd* $\langle \lambda-. \text{True} \rangle$ *Kinds S*
for *S* :: $\langle 'x \text{ lbd set} \rangle$
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**
 $\delta = \text{sat}_H[\text{of } D.\text{kind}]$

lemma *Nom-refl*: $\langle (i, \cdot i) \in S \rangle$
using *gammaI* **by** (*fastforce intro: CRefl*)

lemma *Nom-sym*:
assumes $\langle (i, \cdot k) \in S \rangle$
shows $\langle (k, \cdot i) \in S \rangle$
using *assms alpha* **by** (*fastforce intro: CSym*)

lemma *Nom-trans*:
assumes $\langle (i, \cdot j) \in S \rangle \langle (j, \cdot k) \in S \rangle$
shows $\langle (i, \cdot k) \in S \rangle$
using *assms*

proof –
have $\langle (j, \cdot i) \in S \rangle$
using *assms Nom-sym* **by** *blast*
then show *?thesis*
using *assms(2) alpha* **by** (*fastforce intro: CNom*)
qed

lemma *equiv-nom-ne*: $\langle \{k. \text{equiv-nom } S \ i \ k\} \neq \{\} \rangle$
unfolding *equiv-nom-def* **using** *Nom-refl* **by** *blast*

lemma *equiv-nom-assign*: $\langle \text{equiv-nom } S \ i \ ([i]_S) \rangle$
unfolding *assign-def* **using** *equiv-nom-ne*
by (*metis Field-card-of card-of-well-order-on mem-Collect-eq top.extremum wo-rel-def wo-rel.minim-in*)

lemma *equiv-nom-Nom*:
assumes $\langle \text{equiv-nom } S \ i \ k \rangle \langle (i, p) \in S \rangle$

```

shows  $\langle (k, p) \in S \rangle$ 
proof -
  have  $\langle (i, \cdot k) \in S \rangle$ 
    using assms unfolding equiv-nom-def by blast
  then show ?thesis
    using assms alpha by (fastforce intro: CNom)
qed

lemma assign-in-W:  $\langle [i]_S \in \mathcal{W} \text{ (canonical } S) \rangle$ 
  unfolding canonical-def gframe.defs frame.defs unfolds by blast

theorem model:  $\langle ((i, p) \in S \longrightarrow \llbracket S, i \rrbracket \models p) \wedge ((i, \neg p) \in S \longrightarrow \neg \llbracket S, i \rrbracket \models p) \rangle$ 
proof (induct p arbitrary: i rule: wf-induct[where r= $\langle \text{measures } [qs\text{-}fm, sz\text{-}fm] \rangle$ ])
  case 1
  then show ?case
    by simp
next
  case (2 x)
  then show ?case
  proof (cases x)
    case (TmI k)
    then show ?thesis
  proof (safe del: notI)
    assume  $\langle x = \cdot k \rangle \langle (i, \cdot k) \in S \rangle$ 
    moreover from this have  $\langle (k, \cdot i) \in S \rangle$ 
      using Nom-sym by fast
    ultimately have  $\langle [i]_S = [k]_S \rangle$ 
      using Nom-trans unfolding assign-def equiv-nom-def by metis
    then show  $\langle \llbracket S, i \rrbracket \models \cdot k \rangle$ 
      unfolding canonical-ctx-def canonical-def gframe.defs
      by (simp split: tm.splits)
  next
    assume  $\langle x = \cdot k \rangle \langle (i, \neg \cdot k) \in S \rangle$ 
    then have  $\langle (i, \cdot k) \notin S \rangle$ 
      using confl by (fastforce intro: CNegI)
    then have  $\langle \neg \text{equiv-nom } S \ i \ k \rangle$ 
      unfolding equiv-nom-def by blast
    moreover have  $\langle (i, \cdot k) \notin S \rangle$ 
      using  $\langle (i, \neg \cdot k) \in S \rangle$  using confl by (fastforce intro: CNegI)
    ultimately have  $\langle [i]_S \neq [k]_S \rangle$ 
      using Nom-sym Nom-trans equiv-nom-assign unfolding equiv-nom-def by
metis
    then show  $\langle \neg \llbracket S, i \rrbracket \models \cdot k \rangle$ 
      unfolding canonical-ctx-def canonical-def gframe.defs
      by (auto split: tm.splits)
  qed
next
  case (TmP P)
  then show ?thesis

```

```

proof (safe del: notI)
  assume  $\langle x = \cdot P \rangle \langle (i, \cdot P) \in S \rangle$ 
  then show  $\langle \llbracket S, i \rrbracket \models \cdot P \rangle$ 
    unfolding canonical-ctx-def canonical-def val-def gframe.defs
    by (auto split: tm.splits)
next
  assume  $\langle x = \cdot P \rangle \langle (i, \neg \cdot P) \in S \rangle$ 
  then have  $\langle (i, \cdot P) \notin S \rangle$ 
    using confl by (fastforce intro: CNegP)
  then have  $\langle \bigwedge k. [k]_S = [i]_S \implies (k, \cdot P) \notin S \rangle$ 
    by (metis Nom-refl equiv-nom-Nom equiv-nom-assign equiv-nom-def)
  then show  $\langle \neg \llbracket S, i \rrbracket \models \cdot P \rangle$ 
    unfolding canonical-ctx-def canonical-def val-def gframe.defs
    by (auto split: tm.splits)
qed
next
  case (Neg p)
  then show ?thesis
  proof (safe del: notI)
    assume  $\langle x = \neg p \rangle \langle (i, \neg p) \in S \rangle$ 
    then show  $\langle \llbracket S, i \rrbracket \models \neg p \rangle$ 
      using 2 unfolding canonical-ctx-def by simp
  next
    assume  $\langle x = \neg p \rangle \langle (i, \neg \neg p) \in S \rangle$ 
    then have  $\langle (i, p) \in S \rangle$ 
      using alpha by (fastforce intro: CNegN)
    then show  $\langle \neg \llbracket S, i \rrbracket \models \neg p \rangle$ 
      using 2 Neg unfolding canonical-ctx-def by auto
  qed
next
  case (Con p q)
  then show ?thesis
  proof (safe del: notI)
    assume  $\langle x = (p \wedge q) \rangle \langle (i, p \wedge q) \in S \rangle$ 
    then have  $\langle (i, p) \in S \wedge (i, q) \in S \rangle$ 
      using alpha by (fastforce intro: CConP)
    moreover have
       $\langle (p, x) \in \text{measures } [qs\text{-}fm, sz\text{-}fm] \rangle$ 
       $\langle (q, x) \in \text{measures } [qs\text{-}fm, sz\text{-}fm] \rangle$ 
      using Con by auto
    ultimately show  $\langle \llbracket S, i \rrbracket \models (p \wedge q) \rangle$ 
      using 2 unfolding canonical-ctx-def by auto
  next
    assume  $\langle x = (p \wedge q) \rangle \langle (i, \neg (p \wedge q)) \in S \rangle$ 
    then have  $\langle (i, \neg p) \in S \vee (i, \neg q) \in S \rangle$ 
      using beta by (fastforce intro: CConN)
    moreover have
       $\langle (\neg p, \neg x) \in \text{measures } [qs\text{-}fm, sz\text{-}fm] \rangle$ 
       $\langle (\neg q, \neg x) \in \text{measures } [qs\text{-}fm, sz\text{-}fm] \rangle$ 

```

```

    using Con by auto
  ultimately show  $\langle \neg \llbracket S, i \rrbracket \models (p \wedge q) \rangle$ 
    using 2 unfolding canonical-ctx-def by auto
qed
next
case (Box  $p$ )
then show ?thesis
proof (safe del: notI)
  assume  $\langle x = \Box p \rangle \langle (i, \Box p) \in S \rangle$ 
  {
    fix  $k$ 
    assume  $\langle [k]_S \in \text{reach } S \ ([i]_S) \rangle$ 
    then obtain  $k'$  where  $\langle ([i]_S, \Diamond (\cdot k')) \in S \rangle \langle [k']_S = [k]_S \rangle$ 
      unfolding reach-def by auto
    then have  $\langle (i, \Diamond (\cdot k')) \in S \rangle$ 
      using Nom-sym equiv-nom-Nom equiv-nom-assign equiv-nom-def by blast
    then have  $\langle (k', p) \in S \rangle$ 
      using  $\langle (i, \Box p) \in S \rangle$  alpha by (fastforce intro: CBoxP)
    then have  $\langle \llbracket S, k' \rrbracket \models p \rangle$ 
      using 2 Box by simp
    then have  $\langle \llbracket S, k \rrbracket \models p \rangle$ 
      unfolding canonical-ctx-def canonical-def using  $\langle [k']_S = [k]_S \rangle$  by simp
  }
  then show  $\langle \llbracket S, i \rrbracket \models \Box p \rangle$ 
    unfolding canonical-ctx-def canonical-def reach-def gframe.defs frame.defs
    by auto
next
assume  $\langle x = \Box p \rangle \langle (i, \neg (\Box p)) \in S \rangle$ 
then obtain  $k$  where  $k: \langle (k, \neg p) \in S \rangle \langle (i, \Diamond (\cdot k)) \in S \rangle$ 
  using  $\delta$  by force
then have  $\langle \neg \llbracket S, k \rrbracket \models p \rangle$ 
  using 2 Box by simp
moreover have  $\langle ([i]_S, \Diamond (\cdot k)) \in S \rangle$ 
  using  $k(2)$  equiv-nom-Nom equiv-nom-assign by fastforce
then have  $\langle [k]_S \in \text{reach } S \ ([i]_S) \rangle$ 
  unfolding reach-def by blast
ultimately show  $\langle \neg \llbracket S, i \rrbracket \models \Box p \rangle$ 
  unfolding canonical-ctx-def canonical-def gframe.defs frame.defs
  by auto
qed
next
case (Sat  $k$   $p$ )
then show ?thesis
proof (safe del: notI)
  assume  $\langle x = @k p \rangle \langle (i, @k p) \in S \rangle$ 
  then have  $\langle (k, p) \in S \rangle$ 
    using alpha by (fastforce intro: CSatP)
  then have  $\langle \llbracket S, k \rrbracket \models p \rangle$ 
    using 2 Sat by simp

```

```

then show  $\langle \llbracket S, i \rrbracket \models @k p \rangle$ 
  unfolding canonical-ctx-def canonical-def gframe.defs
  by (auto split: tm.splits)
next
assume  $\langle x = @k p \rangle \langle (i, \neg @k p) \in S \rangle$ 
then have  $\langle (k, \neg p) \in S \rangle$ 
  using alpha by (fastforce intro: CSatN)
then have  $\langle \neg \llbracket S, k \rrbracket \models p \rangle$ 
  using 2 Sat by simp
then show  $\langle \neg \llbracket S, i \rrbracket \models @k p \rangle$ 
  unfolding canonical-ctx-def canonical-def gframe.defs
  by (auto split: tm.splits)
qed
next
case (Glo p)
then show ?thesis
proof (safe del: notI)
  assume  $\langle x = \mathbf{A} p \rangle \langle (i, \mathbf{A} p) \in S \rangle$ 
  then have  $\langle (k, p) \in S \rangle$  for  $k$ 
    using gammaI by (fastforce intro: CGloP)
  then have  $\langle \llbracket S, k \rrbracket \models p \rangle$  for  $k$ 
    using 2 Glo by simp
  then show  $\langle \llbracket S, i \rrbracket \models \mathbf{A} p \rangle$ 
    unfolding canonical-ctx-def canonical-def gframe.defs frame.defs
    by auto
next
assume  $\langle x = \mathbf{A} p \rangle \langle (i, \neg \mathbf{A} p) \in S \rangle$ 
then have  $\langle \exists k. (k, \neg p) \in S \rangle$ 
  using  $\delta$  by fastforce
then have  $\langle \exists k. \neg \llbracket S, k \rrbracket \models p \rangle$ 
  using 2 Glo by auto
then show  $\langle \neg \llbracket S, i \rrbracket \models \mathbf{A} p \rangle$ 
  unfolding canonical-ctx-def canonical-def gframe.defs frame.defs
  by auto
qed
next
case (Dwn p)
then show ?thesis
proof (safe del: notI)
  assume  $\langle x = \downarrow p \rangle \langle (i, \downarrow p) \in S \rangle$ 
  then have  $\langle (i, \langle i \rangle_i p) \in S \rangle$ 
    using alpha by (fastforce intro: CDwnP)
  moreover have  $\langle (\langle i \rangle_i p, x) \in \text{measures } [qs\text{-}fm, sz\text{-}fm] \rangle$ 
    using Dwn by simp
  ultimately have  $\langle \llbracket S, i \rrbracket \models \langle i \rangle_i p \rangle$ 
    using 2 by (meson in-measure)
  then show  $\langle \llbracket S, i \rrbracket \models \downarrow p \rangle$ 
    unfolding canonical-ctx-def canonical-def gframe.defs
    by simp

```

```

next
  assume  $\langle x = \downarrow p \rangle \langle (i, \neg \downarrow p) \in S \rangle$ 
  then have  $\langle (i, \neg \langle i \rangle_i p) \in S \rangle$ 
    using alpha by (fastforce intro: CDwnN)
  moreover have  $\langle \langle i \rangle_i p, x \rangle \in \text{measures } [qs\text{-}fm, sz\text{-}fm]$ 
    using Dwn by simp
  ultimately have  $\langle \neg \llbracket S, i \rrbracket \models \langle i \rangle_i p \rangle$ 
    using 2 by (meson in-measure)
  then show  $\langle \neg \llbracket S, i \rrbracket \models \downarrow p \rangle$ 
    unfolding canonical-ctx-def canonical-def gframe.defs
    by simp
qed
next
case (All p)
then show ?thesis
proof (safe del: notI)
  assume  $\langle x = \forall p \rangle \langle (i, \forall p) \in S \rangle$ 
  then have  $\langle \text{softqdf } q \implies (i, \langle q \rangle_p p) \in S \rangle$  for  $q$ 
    using gammaP by (fastforce intro: CALLP)
  moreover have  $\langle \text{softqdf } q \implies (\langle q \rangle_p p, x) \in \text{measures } [qs\text{-}fm, sz\text{-}fm] \rangle$  for  $q$ 
    using All by (metis less-add-one measures-less qs-fm.simps(9) qs-fm-sub-pro
softqdf-add-env)
  ultimately have  $\ast: \langle \text{softqdf } q \implies \llbracket S, i \rrbracket \models \langle q \rangle_p p \rangle$  for  $q$ 
    using 2 by (meson in-measure)

  moreover note wf-canonical[of S] assign-in-W[of i]
  ultimately have  $\langle \text{softqdf } q \implies$ 
     $((\text{canonical } S) \langle \mathfrak{W} := (\text{worlds } (\text{canonical } S) q \gg \mathfrak{W} (\text{canonical } S))) \rangle, [i]_S \models$ 
 $p \rangle$  for  $q$ 
    using inst-single-worlds[where W= $\langle \mathcal{W} (\text{canonical } S) \rangle$  and  $q=q$ ] unfolding
canonical-ctx-def canonical-def unfolds by fastforce
  then have  $\langle (\forall P \in \text{sqdfs } (\text{canonical } S). ((\text{canonical } S) \langle \mathfrak{W} := (P \gg \mathfrak{W} (\text{canonical } S))) \rangle, [i]_S \models p) \rangle$ 
    unfolding sqdfs-def by blast
  then have  $\langle (\forall P \in \Pi (\text{canonical } S). ((\text{canonical } S) \langle \mathfrak{W} := (P \gg \mathfrak{W} (\text{canonical } S))) \rangle, [i]_S \models p) \rangle$ 
    using sqdfs-canonical by blast
  then show  $\langle \llbracket S, i \rrbracket \models \forall p \rangle$ 
    unfolding canonical-ctx-def by simp
next
  assume  $\langle x = \forall p \rangle \langle (i, \neg \forall p) \in S \rangle$ 
  then obtain  $P$  where  $\langle (i, \neg \langle \cdot (\circ P) \rangle_p p) \in S \rangle$ 
    using  $\delta$  by auto
  moreover have  $\langle (\neg \langle \cdot (\circ P) \rangle_p p, x) \in \text{measures } [qs\text{-}fm, sz\text{-}fm] \rangle$ 
    using All
  by (metis less-add-one measures-less qs-fm.simps(3) qs-fm.simps(9) qs-fm-sub-pro
softqdf.simps(2) softqdf-add-env)
  ultimately have  $\langle \neg \llbracket S, i \rrbracket \models \langle \cdot (\circ P) \rangle_p p \rangle$ 
    using 2 unfolding canonical-ctx-def by auto

```

moreover have $\langle \text{softqdf } (\cdot \ (\circ \ P)) \rangle$
by *simp*
moreover note *wf-canonical[of S] assign-in-W[of i]*
ultimately have $\langle \text{softqdf } (\cdot \ (\circ \ P)) \wedge$
 $\neg ((\text{canonical } S) \langle \mathfrak{V} := (\text{worlds } (\text{canonical } S) (\cdot \ (\circ \ P)) \gg \mathfrak{V} (\text{canonical } S)) \rangle), [i]_S \models p \rangle$
using *inst-single-worlds[where W= $\langle \mathcal{W} (\text{canonical } S) \rangle$] unfolding canonical-ctx-def canonical-def unfolds*
by *fastforce*
then have $\langle (\exists P \in \text{sqdfs } (\text{canonical } S). \neg ((\text{canonical } S) \langle \mathfrak{V} := (P \gg \mathfrak{V} (\text{canonical } S)) \rangle), [i]_S \models p) \rangle$
unfolding *sqdfs-def by blast*
then have $\langle (\exists P \in \Pi (\text{canonical } S). \neg ((\text{canonical } S) \langle \mathfrak{V} := (P \gg \mathfrak{V} (\text{canonical } S)) \rangle), [i]_S \models p) \rangle$
using *sqdfs-canonical by blast*
then show $\langle \neg \llbracket S, i \rrbracket \models \forall p \rangle$
unfolding *canonical-ctx-def by simp*
qed
qed
qed
end

theorem *model-existence:*

fixes $S :: \langle 'x \text{ lbd set} \rangle$
assumes $\langle P.\text{prop}_E \text{ Kinds } C \rangle$
and $\langle S \in C \rangle$
and $\langle P.\text{enough-new } S \rangle$
and $\langle (i, p) \in S \rangle$
shows $\langle \llbracket \text{mk-mcs } C \ S, i \rrbracket \models p \rangle$
proof –
have $*$: $\langle \text{MyHintikka } (\text{mk-mcs } C \ S) \rangle$
proof
show $\langle P.\text{prop}_H \text{ Kinds } (\text{mk-mcs } C \ S) \rangle$
using *mk-mcs-Hintikka[OF assms(1–3)] Hintikka.hintikka by blast*
qed
moreover have $\langle (i, p) \in \text{mk-mcs } C \ S \rangle$
using *assms(4) Extend-subset by blast*
ultimately show *?thesis*
using *MyHintikka.model by blast*
qed

4.6 Natural Deduction

inductive *Calculus-Set* :: $\langle 'x \text{ lbd set} \Rightarrow 'x \text{ lbd} \Rightarrow \text{bool} \rangle (\langle \vdash \rightarrow [50, 50] \ 50 \rangle)$ **where**

Assm [*dest*]: $\langle (i, p) \in A \Longrightarrow A \Vdash (i, p) \rangle$
| *Ref* [*simp*]: $\langle A \Vdash (i, \cdot i) \rangle$
| *Nom* [*dest*]: $\langle A \Vdash (i, \cdot k) \Longrightarrow A \Vdash (i, p) \Longrightarrow A \Vdash (k, p) \rangle$
| *NotI* [*intro*]: $\langle \{(i, p)\} \cup A \Vdash (i, \perp) \Longrightarrow A \Vdash (i, \neg p) \rangle$

| *NotE* [*elim*]: $\langle A \Vdash (i, \neg p) \implies A \Vdash (i, p) \implies A \Vdash (k, q) \rangle$
 | *AndI* [*intro*]: $\langle A \Vdash (i, p) \implies A \Vdash (i, q) \implies A \Vdash (i, p \wedge q) \rangle$
 | *AndD1* [*dest*]: $\langle A \Vdash (i, p \wedge q) \implies A \Vdash (i, p) \rangle$
 | *AndD2* [*dest*]: $\langle A \Vdash (i, p \wedge q) \implies A \Vdash (i, q) \rangle$
 | *SatI* [*intro*]: $\langle A \Vdash (i, p) \implies A \Vdash (k, @i p) \rangle$
 | *SatE* [*dest*]: $\langle A \Vdash (i, @k p) \implies A \Vdash (k, p) \rangle$
 | *BoxI* [*intro*]: $\langle \{(i, \diamond (\cdot \circ k))\} \cup A \Vdash (\circ k, p) \implies k \notin \text{symbols} (\{(i, p)\} \cup A) \implies A \Vdash (i, \square p) \rangle$
 | *BoxE* [*elim*]: $\langle A \Vdash (i, \square p) \implies A \Vdash (i, \diamond (\cdot k)) \implies A \Vdash (k, p) \rangle$
 | *GloI* [*intro*]: $\langle A \Vdash (\circ k, p) \implies k \notin \text{symbols} (\{(i, p)\} \cup A) \implies A \Vdash (i, \mathbf{A} p) \rangle$
 | *GloE* [*dest*]: $\langle A \Vdash (i, \mathbf{A} p) \implies A \Vdash (k, p) \rangle$
 | *DwnI* [*intro*]: $\langle A \Vdash (i, \langle i \rangle_i p) \implies A \Vdash (i, \downarrow p) \rangle$
 | *DwnE* [*dest*]: $\langle A \Vdash (i, \downarrow p) \implies A \Vdash (i, \langle i \rangle_i p) \rangle$
 | *AllI* [*intro*]: $\langle A \Vdash (i, \langle \cdot (\circ P) \rangle_p p) \implies P \notin \text{symbols} (\{(i, p)\} \cup A) \implies A \Vdash (i, \forall p) \rangle$
 | *AllE* [*dest*]: $\langle A \Vdash (i, \forall p) \implies \text{softqdf } q \implies A \Vdash (i, \langle q \rangle_p p) \rangle$
 | *Clas*: $\langle \{(i, \neg p)\} \cup A \Vdash (i, p) \implies A \Vdash (i, p) \rangle$

4.6.1 Soundness

theorem *soundness*:

assumes $\langle A \Vdash (i, p) \rangle \langle \forall (k, q) \in A. (\text{Model } W R P I N e V f, \text{case-tm } e N k) \models q \rangle$

$\langle \text{wf-model } (\text{Model } W R P I N e V f) \rangle$

shows $\langle (\text{Model } W R P I N e V f, \text{case-tm } e N i) \models p \rangle$

using *assms*

proof (*induct A* $\langle (i, p) \rangle$ *arbitrary: i p N V pred: Calculus-Set*)

case (*Ref A i*)

then show *?case*

by (*auto split: tm.splits*)

next

case (*Nom A i k p*)

then show *?case*

by (*auto split: tm.splits*)

next

case (*SatI A i p k*)

then show *?case*

by (*auto split: tm.splits*)

next

case (*SatE A i k p*)

then show *?case*

by (*auto split: tm.splits*)

next

case (*BoxI i k A p*)

{

fix *v*

assume *v*: $\langle v \in R (\text{case-tm } e N i) \rangle$

moreover have $\langle \text{case-tm } e N i \in W \rangle$

using *BoxI(5) unfolding wf-env-def wf-model-def unfolds by (auto split:*

```

tm.splits)
  ultimately have ⟨v ∈ W⟩
    using BoxI(5) unfolding wf-model-def wf-gframe-def wf-frame-def unfolds
by blast

  let ?N = ⟨N(k := v)⟩
  have ⟨∀(i, p) ∈ A. (Model W R PI ?N e Vf, case-tm e ?N i) ⊨ p⟩
    using BoxI by fastforce
  moreover have ⟨(Model W R PI ?N e Vf, case-tm e ?N i) ⊨ ◇(·(○k))⟩
    using v BoxI.hyps(3) by (simp split: tm.splits)
  moreover have ⟨wf-model (Model W R PI ?N e Vf)⟩
    using BoxI.prem(2) ⟨v ∈ W⟩ unfolding wf-env-def wf-model-def unfolds by
auto
  ultimately have ⟨(Model W R PI ?N e Vf, ?N k) ⊨ p⟩
    using BoxI.hyps(2) by fastforce
}
then have ⟨∀v ∈ R (case-tm e N i). (Model W R PI (N(k := v)) e Vf, v) ⊨
p⟩
  by simp
then have ⟨∀v ∈ R (case-tm e N i). (Model W R PI N e Vf, v) ⊨ p⟩
  using BoxI.hyps(3) by simp
then show ?case
  by simp
next
case (BoxE A i p k)
then show ?case
  by (auto split: tm.splits)
next
case (GloI A k p i)
{
  fix v
  assume ⟨v ∈ W⟩
  let ?N = ⟨N(k := v)⟩
  have ⟨∀v. ∀(i, p) ∈ A. (Model W R PI ?N e Vf, case-tm e ?N i) ⊨ p⟩
    using GloI.prem(1) GloI.hyps(3) by fastforce
  moreover have ⟨wf-model (Model W R PI ?N e Vf)⟩
    using GloI.prem(2) ⟨v ∈ W⟩ unfolding wf-env-def wf-model-def unfolds
by auto
  ultimately have ⟨(Model W R PI ?N e Vf, ?N k) ⊨ p⟩
    using GloI.hyps(2) by fastforce
}
then have ⟨∀v ∈ W. (Model W R PI (N(k := v)) e Vf, v) ⊨ p⟩
  by simp
then have ⟨∀v ∈ W. (Model W R PI N e Vf, v) ⊨ p⟩
  using GloI.hyps(3) by simp
then show ?case
  by simp
next
case (GloE A i p k)

```

```

then show ?case
  using wf- $\mathfrak{N}$  wf- $\mathcal{N}$ 
  by (cases k) fastforce+
next
case (All A i P p)
  {
    fix X
    assume  $\langle X \in PI \rangle$ 
    let ?V =  $\langle V(P := X) \rangle$ 
    have  $\langle \forall v. \forall (i, p) \in A. (Model\ W\ R\ PI\ N\ e\ ?V\ f, case\ tm\ e\ N\ i) \models p \rangle$ 
      using All.prem(1) All.hyps(3) by fastforce
    moreover have *:  $\langle wf\text{-model}\ (Model\ W\ R\ PI\ N\ e\ ?V\ f) \rangle$ 
      using All.prem(2)  $\langle X \in PI \rangle$  unfolding wf-env-def wf-model-def unfolds by
      auto
    ultimately have  $\langle (Model\ W\ R\ PI\ N\ e\ ?V\ f, case\ tm\ e\ N\ i) \models \langle \cdot (\circ P) \rangle_p p \rangle$ 
      using All.hyps(2) by fast
    moreover have  $\langle case\ tm\ e\ N\ i \in W \rangle$ 
      using All.prem(2) unfolding wf-model-def wf-env-def unfolds by (auto
      split: tm.splits)
    ultimately have  $\langle (Model\ W\ R\ PI\ N\ e\ ?V\ (worlds\ (Model\ W\ R\ PI\ N\ e\ ?V\ f)\ (\cdot (\circ P)) \gg f), case\ tm\ e\ N\ i) \models p \rangle$ 
      using inst-single-worlds * by fastforce
    then have  $\langle (Model\ W\ R\ PI\ N\ e\ ?V\ (\{ w \in W. w \in X \} \gg f), case\ tm\ e\ N\ i) \models p \rangle$ 
      unfolding worlds-def by simp
    moreover have  $\langle X \in Pow\ W \rangle$ 
      using All.prem(2)  $\langle X \in PI \rangle$  unfolding wf-model-def wf-gframe-def unfolds
by auto
    then have  $\langle \{ w \in W. w \in X \} = X \rangle$ 
      by blast
    ultimately have  $\langle (Model\ W\ R\ PI\ N\ e\ ?V\ (X \gg f), case\ tm\ e\ N\ i) \models p \rangle$ 
      by simp
  }
  then have  $\langle \forall X \in PI. (Model\ W\ R\ PI\ N\ e\ (V(P := X)) (X \gg f), case\ tm\ e\ N\ i) \models p \rangle$ 
    by simp
  then have  $\langle \forall X \in PI. (Model\ W\ R\ PI\ N\ e\ V\ (X \gg f), case\ tm\ e\ N\ i) \models p \rangle$ 
    using All.hyps(3) by simp
  then show ?case
    by simp
next
case (All A i p q)
then show ?case
proof (cases i)
  case (Var x1)
  then show ?thesis
    using All(2-) inst-single-worlds softqdf-worlds wf- $\mathfrak{N}$  by fastforce
next
case (Sym x2)

```

then show *?thesis*
using *Alle(2-)* *inst-single-worlds softqdf-worlds wf-N* **by** *fastforce*
qed
qed *auto*

corollary *soundness'*:
assumes $\langle \{\} \Vdash (\bigcirc i, p) \rangle \langle i \notin \text{symbols-fm } p \rangle$
and $\langle \text{wf-model } M \rangle \langle w \in \mathcal{W } M \rangle$
shows $\langle (M, w) \models p \rangle$
proof *(cases M)*
case *(fields W R PI N e V f)*
let $?N = \langle N(i := w) \rangle$
have $\langle \text{wf-model } (Model W R PI ?N e V f) \rangle$
using *fields assms(3-4)* **unfolding** *wf-env-def wf-model-def unfolds* **by** *auto*
then have $\langle (Model W R PI ?N e V f, \text{case-tm } e ?N (\bigcirc i)) \models p \rangle$
using *assms(1)* *soundness* **by** *blast*
then have $\langle (Model W R PI ?N e V f, w) \models p \rangle$
by *simp*
then show *?thesis*
using *assms(2)* *fields* **by** *simp*
qed

lemma *no-bot*: $\langle \neg (M, w) \models \perp \rangle$
by *simp*

corollary $\langle \neg (\{\} \Vdash (\bigcirc i, \perp)) \rangle$
using *soundness no-bot wf-canonical* **unfolding** *canonical-def unfolds*
by *(metis (no-types, lifting) emptyE)*

4.6.2 Derived Rules

lemma *Assm-head* [*simp*]: $\langle \{(p, i)\} \cup A \Vdash (p, i) \rangle$
using *Assm* **by** *blast*

lemma *Boole*: $\langle \{(i, \neg p)\} \cup A \Vdash (i, \perp) \implies A \Vdash (i, p) \rangle$
using *Clas AndD1 AndD2 NotE* **by** *meson*

lemma *FlsE* [*dest*]: $\langle A \Vdash (i, \perp) \implies A \Vdash (k, p) \rangle$
by *(meson Assm-head NotE NotI)*

4.6.3 Derivational Consistency

lemma *calculus-conf*:
assumes $\langle ps \rightsquigarrow_{\mathbf{x}} qs \rangle \langle \text{set } ps \subseteq A \rangle \langle q \in \text{set } qs \rangle \langle q \in A \rangle$
shows $\langle A \Vdash (i, \perp) \rangle$
using *assms*
proof *cases*
case *(CNegP i p)*
then show *?thesis*
using *assms(2-)*

```

    by (metis Assm NotE empty-set equals0D list.set-intros(1) set-ConsD subset-eq)
next
  case (CNegI i k)
  then show ?thesis
    using assms(2-) by fastforce
qed

lemma calculus-alpha:
  assumes ⟨ps  $\rightsquigarrow_\alpha$  qs⟩ ⟨set ps  $\subseteq$  A⟩ ⟨set qs  $\cup$  A  $\Vdash$  (i,  $\perp$ )⟩
  shows ⟨A  $\Vdash$  (i,  $\perp$ )⟩
  using assms
proof cases
  case (CSym i k)
  then show ?thesis
    using assms(2-)
    by (metis Assm-head Diff-partition Nom NotE NotI Ref list.set(1) list.simps(15))
next
  case (CNom i k p)
  then show ?thesis
    using assms(2-)
    by (metis AndD1 AndD2 Assm Nom NotE NotI empty-set insert-subset list.simps(15))
next
  case (CNegN i p)
  then show ?thesis
    using assms(2-)
    by (metis Assm-head Diff-partition NotE NotI empty-set list.simps(15))
next
  case (CConP k p q)
  then have ⟨A  $\Vdash$  (k, p)⟩ ⟨A  $\Vdash$  (k, q)⟩
    using assms(2-) by (meson AndD1 AndD2 Assm list.set-intros(1) subset-eq)+
  moreover have ⟨{(k, p), (k, q)}  $\cup$  A  $\Vdash$  (k,  $\perp$ )⟩
    using CConP assms(2-) by auto
  then have ⟨{(k, q)}  $\cup$  A  $\Vdash$  (k,  $\neg$  p)⟩
    using NotI by auto
  ultimately have ⟨A  $\Vdash$  (k,  $\perp$ )⟩
    using CConP(1) assms(2)
  by (metis AndD1 Assm NotE NotI list.set-intros(1) subset-code(1) sup.coboundedI2)
  then show ?thesis
    by blast
next
  case (CSatP i k p)
  then show ?thesis
    using assms(2-) SatE[of A i k p]
    by (metis Assm-head NotE NotI empty-set le-iff-sup list.simps(15))
next
  case (CSatN i k p)
  then show ?thesis
    using assms(2-) SatI[of A k p]
    by (metis Assm-head Boole FlsE NotE empty-set le-iff-sup list.simps(15))

```

```

next
  case (CBoxP i p k)
  then show ?thesis
    using assms(2-) BoxE[of A i p]
    by (metis AndD1 AndD2 Assm NotE NotI empty-set insert-subset list.simps(15))
next
  case (CDwnP i p)
  then show ?thesis
    using assms(2-) DwnE[of A i p]
    by (metis Assm-head Diff-partition NotE NotI empty-set list.simps(15))
next
  case (CDwnN i p)
  then show ?thesis
    using assms(2-) DwnI[of A i p]
    by (metis AndD1 AndD2 Assm-head Clas NotE empty-set le-iff-sup list.simps(15))
qed

```

lemma *calculus-beta*:

```

assumes ⟨ps  $\rightsquigarrow_{\beta}$  qs⟩ ⟨set ps  $\subseteq$  A⟩ ⟨ $\forall q \in$  set qs.  $\{q\} \cup A \Vdash (i, \perp)$ ⟩
shows ⟨A  $\Vdash (i, \perp)$ ⟩
using assms
proof cases
  case (CConN i p q)
  then show ?thesis
    using assms(2-) AndI[of A i p q]
    by (metis Assm Clas FlsE NotE UnI2 insert-is-Un list.set-intros(1) list.simps(15)
subset-iff)
qed

```

lemma *calculus-gammaI*:

```

assumes ⟨ps  $\rightsquigarrow_{\gamma_i}$  qs⟩ ⟨set ps  $\subseteq$  A⟩ ⟨set (qs k)  $\cup$  A  $\Vdash (i, \perp)$ ⟩
shows ⟨A  $\Vdash (i, \perp)$ ⟩
using assms
proof cases
  case CRefl
  then show ?thesis
    using CRefl assms(2-) Ref[of A k]
    by (metis (mono-tags, lifting) AndD1 AndD2 NotE NotI empty-set list.simps(15))
next
  case (CGloP i p)
  then show ?thesis
    using CGloP assms(2-) GloE[of A i p k]
    by (metis (no-types, lifting) Assm-head NotE NotI empty-set le-iff-sup list.simps(15))
qed

```

lemma *calculus-gammaP*:

```

assumes ⟨ps  $\rightsquigarrow_{\gamma_p}$  (F, qs)⟩ ⟨set ps  $\subseteq$  A⟩ ⟨k  $\in$  F A⟩ ⟨set (qs k)  $\cup$  A  $\Vdash (i, \perp)$ ⟩
shows ⟨A  $\Vdash (i, \perp)$ ⟩
using assms

```

proof *cases*
 case (*CALLP* i p)
 then show *?thesis*
 using *assms*($2-$) *AlLE*[of A i p]
 by (*metis* (*no-types*, *lifting*) *Assm-head NotE NotI le-iff-sup list.set(1) list.simps(15)*
mem-Collect-eq)
qed

lemma *calculus- δ* :

assumes $\langle p \in A \rangle \langle k \notin \text{symbols } A \rangle \langle \text{set } (\delta \ p \ k) \cup A \Vdash (a, \perp) \rangle$
 shows $\langle A \Vdash (a, \perp) \rangle$
 using *assms*
proof (*induct* p k *rule*: δ .*induct*)
 case (1 i p k)
 then have $\langle \{(\circ k, \neg p), (i, \diamond (\cdot(\circ k)))\} \cup A \Vdash (a, \perp) \rangle \langle (i, \neg \square p) \in A \rangle$
 by *simp-all*
 then have $\langle \{i, \diamond (\cdot(\circ k))\} \cup A \Vdash (\circ k, p) \rangle$
 using *Boole* **by** *auto*
 moreover have $\langle k \notin \text{symbols } (\{i, p\} \cup A) \rangle$
 using 1 **by** *auto*
 ultimately have $\langle A \Vdash (i, \square p) \rangle$
 using *BoxI* **by** *blast*
 then show *?thesis*
 using $\langle (i, \neg \square p) \in A \rangle$ **by** (*meson* *Assm NotE*)
next
 case (2 i p k)
 then have $\langle \{(\circ k, \neg p)\} \cup A \Vdash (a, \perp) \rangle \langle (i, \neg \mathbf{A} \ p) \in A \rangle$
 by *simp-all*
 then have $\langle \{(\circ k, \neg p)\} \cup A \Vdash (\circ k, \perp) \rangle$
 by *fast*
 then have $\langle A \Vdash (\circ k, p) \rangle$
 by (*meson* *Boole*)
 moreover have $\langle k \notin \text{symbols } (\{i, p\} \cup A) \rangle$
 using 2 *CGloN* **by** *auto*
 ultimately have $\langle A \Vdash (i, \mathbf{A} \ p) \rangle$
 by *blast*
 then show *?thesis*
 using $\langle (i, \neg \mathbf{A} \ p) \in A \rangle$ *Assm NotE* **by** *meson*
next
 case (3 i p P)
 then have $\langle \{i, \neg \langle \cdot (\circ P) \rangle_p \ p\} \cup A \Vdash (a, \perp) \rangle \langle (i, \neg \forall \ p) \in A \rangle$
 by *simp-all*
 then have $\langle A \Vdash (i, \langle \cdot (\circ P) \rangle_p \ p) \rangle$
 using *Clas* **by** *blast*
 moreover have $\langle P \notin \text{symbols } (\{i, p\} \cup A) \rangle$
 using 3 **by** *auto*
 ultimately have $\langle A \Vdash (i, \forall \ p) \rangle$
 by *blast*
 then show *?thesis*

using $\langle (i, \neg \forall p) \in A \rangle$ *Assm NotE* **by** *meson*
qed *simp-all*

interpretation *DC: Derivational-Confl map-lbd symbols-lbd* $\langle \lambda-. True \rangle$ *confl-class*
 $\langle \lambda A. \neg A \Vdash (a, \perp) \rangle$
using *calculus-confl* **by** *unfold-locales blast*

interpretation *DA: Derivational-Alpha map-lbd symbols-lbd* $\langle \lambda-. True \rangle$ *alpha-class*
 $\langle \lambda A. \neg A \Vdash (a, \perp) \rangle$
using *calculus-alpha* **by** *unfold-locales blast*

interpretation *DB: Derivational-Beta map-lbd symbols-lbd* $\langle \lambda-. True \rangle$ *beta-class*
 $\langle \lambda A. \neg A \Vdash (a, \perp) \rangle$
using *calculus-beta* **by** *unfold-locales blast*

interpretation *DGI: Derivational-Gamma map-tm map-lbd symbols-lbd* $\langle \lambda-. True \rangle$
GI.classify-UNIV $\langle \lambda A. \neg A \Vdash (a, \perp) \rangle$
using *calculus-gammaI* **by** *unfold-locales blast*

interpretation *DGP: Derivational-Gamma map-fm map-lbd symbols-lbd* $\langle \lambda-. True \rangle$
gamma-class-fm $\langle \lambda A. \neg A \Vdash (a, \perp) \rangle$
using *calculus-gammaP* **by** *unfold-locales blast*

interpretation *DD: Derivational-Delta map-lbd symbols-lbd* $\langle \lambda-. True \rangle$ δ $\langle \lambda A. \neg$
 $A \Vdash (a, \perp) \rangle$
by *unfold-locales (meson calculus- δ)*

interpretation *Derivational-Consistency map-lbd symbols-lbd* $\langle \lambda-. True \rangle$ *Kinds*
 $\langle \lambda A. \neg A \Vdash (a, \perp) \rangle$
using *prop_E-Kinds*[*OF DC.kind DA.kind DB.kind DGI.kind DGP.kind DD.kind*]
by *unfold-locales*

4.6.4 Strong Completeness

theorem *strong-completeness:*

fixes $p :: \langle 'x \text{ fm} \rangle$

assumes *mod*: $\langle \bigwedge (M :: \langle 'x, 'x \text{ tm} \rangle \text{ model}) w. \text{wf-model } M \implies w \in \mathcal{W} M \implies$

$\forall (k, q) \in A. (M, \text{case-tm } (\mathfrak{N} M) (\mathcal{N} M) k) \models q \implies$

$(M, w) \models p \rangle$

and $\langle P.\text{enough-new } A \rangle$

shows $\langle A \Vdash (i, p) \rangle$

proof (*rule ccontr*)

assume $\langle \neg A \Vdash (i, p) \rangle$

then have $*$: $\langle \neg \{(i, \neg p)\} \cup A \Vdash (i, \perp) \rangle$

using *Boole* **by** *blast*

let $?S = \langle \{(i, \neg p)\} \cup A \rangle$

let $?C = \langle \{A. P.\text{enough-new } A \wedge \neg A \Vdash (\text{undefined}, \perp)\} \rangle$

let $?H = \langle \text{mk-mcs } ?C ?S \rangle$

let $?M = \langle \text{canonical } ?H \rangle$

have $\langle P.\text{prop}_E \text{ Kinds } ?C \rangle$
using *Consistency by fast*
moreover have $\langle ?S \in ?C \rangle$
using $* \text{ FlsE params-left assms}(?)$
by (*metis (no-types, lifting) ext List.set-insert empty-set mem-Collect-eq*)
moreover from this have $\langle P.\text{enough-new } ?S \rangle$
by *blast*
ultimately have $** : \langle \forall (k, q) \in ?S. \llbracket ?H, k \rrbracket \models q \rangle$
using *model-existence[of ?C ?S] by blast*
then have $\langle \forall (k, q) \in ?S. (?M, \text{case-tm } (\mathfrak{N} ?M) (\mathcal{N} ?M) k) \models q \rangle$
unfolding *canonical-tm-eta' canonical-ctx-def by blast*

moreover have $\langle \text{wf-model } ?M \rangle$
using *wf-canonical by blast*
moreover have $\langle \text{assign } i ?H \in \mathcal{W} ?M \rangle$
unfolding *canonical-def unfolds by auto*

ultimately have $\langle \llbracket ?H, i \rrbracket \models p \rangle$
using *mod unfolding canonical-ctx-def by auto*

moreover have $\langle \neg \llbracket ?H, i \rrbracket \models p \rangle$
using $** \text{ unfolding canonical-ctx-def by simp}$
ultimately show *False*
by *simp*
qed

4.7 Natural Deduction with Lists

inductive Calculus :: $\langle 'x \text{ lbd list} \Rightarrow 'x \text{ lbd} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \vdash \rangle$ 50) **where**
Assm [dest]: $\langle (i, p) \in \text{set } A \Longrightarrow A \vdash (i, p) \rangle$
Ref [simp]: $\langle A \vdash (i, \cdot i) \rangle$
Nom [dest]: $\langle A \vdash (i, \cdot k) \Longrightarrow A \vdash (i, p) \Longrightarrow A \vdash (k, p) \rangle$
NotI [intro]: $\langle (i, p) \# A \vdash (i, \perp) \Longrightarrow A \vdash (i, \neg p) \rangle$
NotE [elim]: $\langle A \vdash (i, \neg p) \Longrightarrow A \vdash (i, p) \Longrightarrow A \vdash (k, q) \rangle$
AndI [intro]: $\langle A \vdash (i, p) \Longrightarrow A \vdash (i, q) \Longrightarrow A \vdash (i, p \wedge q) \rangle$
AndD1 [dest]: $\langle A \vdash (i, p \wedge q) \Longrightarrow A \vdash (i, p) \rangle$
AndD2 [dest]: $\langle A \vdash (i, p \wedge q) \Longrightarrow A \vdash (i, q) \rangle$
SatI [intro]: $\langle A \vdash (i, p) \Longrightarrow A \vdash (k, \textcircled{i} p) \rangle$
SatE [dest]: $\langle A \vdash (i, \textcircled{k} p) \Longrightarrow A \vdash (k, p) \rangle$
BoxI [intro]: $\langle (i, \diamond (\cdot (\textcircled{k}))) \# A \vdash (\textcircled{k}, p) \Longrightarrow k \notin \text{symbols } (\{(i, p)\} \cup \text{set } A) \Longrightarrow A \vdash (i, \square p) \rangle$
BoxE [elim]: $\langle A \vdash (i, \square p) \Longrightarrow A \vdash (i, \diamond (\cdot k)) \Longrightarrow A \vdash (k, p) \rangle$
GloI [intro]: $\langle A \vdash (\textcircled{k}, p) \Longrightarrow k \notin \text{symbols } (\{(i, p)\} \cup \text{set } A) \Longrightarrow A \vdash (i, \mathbf{A} p) \rangle$
GloE [dest]: $\langle A \vdash (i, \mathbf{A} p) \Longrightarrow A \vdash (k, p) \rangle$
DwnI [intro]: $\langle A \vdash (i, \langle i \rangle_i p) \Longrightarrow A \vdash (i, \downarrow p) \rangle$
DwnE [dest]: $\langle A \vdash (i, \downarrow p) \Longrightarrow A \vdash (i, \langle i \rangle_i p) \rangle$
AllI [intro]: $\langle A \vdash (i, \langle \cdot (\textcircled{P}) \rangle_p p) \Longrightarrow P \notin \text{symbols } (\{(i, p)\} \cup \text{set } A) \Longrightarrow A \vdash$

$\langle i, \forall p \rangle$
| *Alle* [*dest*]: $\langle A \vdash (i, \forall p) \implies \text{softqdf } q \implies A \vdash (i, \langle q \rangle_p p) \rangle$
| *Clas*: $\langle (i, \neg p) \# A \vdash (i, p) \implies A \vdash (i, p) \rangle$

definition *bounded* :: $\langle 'a \text{ list} \Rightarrow 'a \text{ set} \Rightarrow ('a \text{ list} \Rightarrow \text{bool}) \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{bounded } K A P \equiv \text{set } K \subseteq A \wedge (\forall B. \text{set } K \subseteq \text{set } B \longrightarrow \text{set } B \subseteq A \longrightarrow P B) \rangle$

lemma *bounded-one* [*elim*]:
assumes $\langle \text{bounded } K A P \rangle \langle \bigwedge A. P A \implies Q A \rangle$
shows $\langle \text{bounded } K A Q \rangle$
using *assms unfolding bounded-def by simp*

lemma *bounded-two* [*elim*]:
assumes $\langle \text{bounded } K A P \rangle \langle \text{bounded } K' A Q \rangle \langle \bigwedge A. P A \implies Q A \implies R A \rangle$
shows $\langle \text{bounded } (K @ K') A R \rangle$
using *assms unfolding bounded-def by simp*

lemma *bounded-removeAll* [*dest*]:
assumes $\langle \text{bounded } K (\{p\} \cup A) P \rangle$
shows $\langle \text{bounded } (\text{removeAll } p K) A (\lambda B. P (p \# B)) \rangle$
using *assms unfolding bounded-def*
by (*metis Diff-subset-conv insert-is-Un insert-mono list.simps(15) set-removeAll*)

lemma *bounded-params*:
assumes $\langle a \notin P.\text{params } (\{p\} \cup A) \rangle \langle \text{bounded } K A P \rangle$
shows $\langle \text{bounded } K A (\lambda B. a \notin P.\text{params } (\text{set } (p \# B))) \rangle$
using *assms unfolding bounded-def by auto*

lemma *finite-kernel*: $\langle A \Vdash (i, p) \implies \exists K. \text{bounded } K A (\lambda B. B \vdash (i, p)) \rangle$

proof (*induct A* $\langle (i, p) \rangle$ *arbitrary: i p pred: Calculus-Set*)

case (*Assm i p A*)

then show *?case*

unfolding *bounded-def using Calculus.Assm*

by (*metis empty-subsetI insert-subset set-replicate-Suc*)

next

case (*Ref A i*)

then show *?case*

unfolding *bounded-def using Calculus.Ref*

by (*metis empty-set empty-subsetI*)

next

case (*Nom A i k p*)

then show *?case*

by *force*

next

case (*NotI i p A*)

then have $\langle \exists K. \text{bounded } K A (\lambda B. (i, p) \# B \vdash (i, \perp)) \rangle$

by *blast*

then show *?case*

by *fast*

next
 case (*BoxI* i k A p)
 then have $\langle \exists K. \text{bounded } K A (\lambda B. (i, \diamond (\cdot (\circ k))) \# B \vdash (\circ k, p)) \rangle$
 by *blast*
 moreover from *this* have $\langle \exists K. \text{bounded } K A (\lambda B. k \notin P.\text{params } (\{(i, p)\} \cup \text{set } B)) \rangle$
 using *BoxI(2-3) bounded-params* by *fastforce*
 ultimately show *?case*
 by *fastforce*
next
 case (*GloI* A k p i)
 then have $\langle \exists K. \text{bounded } K A (\lambda B. B \vdash (\circ k, p)) \rangle$
 by *blast*
 moreover from *this* have $\langle \exists K. \text{bounded } K A (\lambda B. k \notin P.\text{params } (\{(i, p)\} \cup \text{set } B)) \rangle$
 using *GloI(3) bounded-params* by *fastforce*
 ultimately show *?case*
 by *fastforce*
next
 case (*AllI* A i P p)
 then have $\langle \exists K. \text{bounded } K A (\lambda B. B \vdash (i, \langle \cdot (\circ P) \rangle_p p)) \rangle$
 by *blast*
 moreover from *this* have $\langle \exists K. \text{bounded } K A (\lambda B. P \notin P.\text{params } (\{(i, p)\} \cup \text{set } B)) \rangle$
 using *AllI(3) bounded-params* by *fastforce*
 ultimately show *?case*
 by *fastforce*
next
 case (*Clas* i p A)
 then have $\langle \exists K. \text{bounded } K A (\lambda B. (i, \neg p) \# B \vdash (i, p)) \rangle$
 by *fast*
 then show *?case*
 using *Calculus.Clas* by *force*
qed *fast+*

corollary *finite-assumptions*: $\langle A \Vdash (i, p) \implies \exists B. \text{set } B \subseteq A \wedge B \vdash (i, p) \rangle$
 using *finite-kernel unfolding bounded-def* by *blast*

lemma *calculus-set*: $\langle A \vdash (i, p) \implies \text{set } A \Vdash (i, p) \rangle$
proof (*induct* A $\langle (i, p) \rangle$ *arbitrary*: i p *pred*: *Calculus*)
 case (*Clas* p q A)
 then show *?case*
 using *Calculus-Set.Clas* by *auto*
qed *auto*

corollary *soundness-list*:
 assumes $\langle A \vdash (i, p) \rangle$ $\langle \forall (k, q) \in \text{set } A. (M, \text{case-tm } (\mathfrak{N} M) (\mathcal{N} M) k) \models q \rangle$
 and $\langle \text{wf-model } M \rangle$
 shows $\langle (M, \text{case-tm } (\mathfrak{N} M) (\mathcal{N} M) i) \models p \rangle$

using *assms soundness calculus-set*
by (*metis (mono-tags, lifting) model.surjective unit.exhaust split-cong*)

corollary *soundness-nil*:

$\langle [] \vdash (\circ i, p) \implies i \notin \text{symbols-fm } p \implies \text{wf-model } M \implies w \in \mathcal{W} M \implies (M, w) \models p \rangle$
by (*metis calculus-set empty-set soundness'*)

corollary $\langle \neg ([] \vdash (i, \perp)) \rangle$

by (*metis equals0D no-bot set-empty2 soundness-list wf-canonical*)

corollary *strong-completeness-list*:

fixes $p :: \langle 'x \text{ fm} \rangle$

assumes $\langle \bigwedge (M :: ('x, 'x \text{ tm}) \text{ model}) w. \text{wf-model } M \implies w \in \mathcal{W} M \implies$

$\forall (k, q) \in A. (M, \text{case-tm } (\mathfrak{N} M) (\mathcal{N} M) k) \models q \implies (M, w) \models p \rangle$

and $\langle P.\text{enough-new } A \rangle$

shows $\langle \exists B. \text{set } B \subseteq A \wedge B \vdash (i, p) \rangle$

using *assms strong-completeness finite-assumptions* **by** *blast*

theorem *main*:

fixes $p :: \langle 'x \text{ fm} \rangle$

assumes $\langle i \notin \text{symbols-fm } p \rangle \langle |UNIV :: 'x \text{ lbd set}| \leq_o |UNIV :: 'x \text{ set}| \rangle$

shows $\langle [] \vdash (\circ i, p) \longleftrightarrow (\forall (M :: ('x, 'x \text{ tm}) \text{ model}). \forall w \in \mathcal{W} M. \text{wf-model } M \rightarrow (M, w) \models p) \rangle$

using *assms strong-completeness-list* **where** $A = \langle \{ \} \rangle$ *soundness-nil unfolding P.enough-new-def* **by** *fastforce*

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* $:: \text{int} \Rightarrow \text{nat} \Rightarrow \text{int set}$ **where**

arith-prog $a \ b = \{x. [x = a] \ (\text{mod } \text{int } b)\}$

lemma *arith-prog-0-right* [*simp*]: *arith-prog* $a \ 0 = \{a\}$

by (*simp add: arith-prog-def*)

lemma *arith-prog-Suc-0-right* [*simp*]: *arith-prog* $a \ (\text{Suc } 0) = UNIV$

by (*auto simp: arith-prog-def*)

lemma *in-arith-progI* [*intro*]: $[x = a] \ (\text{mod } b) \implies x \in \text{arith-prog } a \ b$

by (*auto simp: arith-prog-def*)

lemma *arith-prog-disjoint*:

assumes $[a \neq a'] \ (\text{mod } \text{int } b)$ **and** $b > 0$

shows $\text{arith-prog } a \ b \cap \text{arith-prog } a' \ b = \{ \}$

using *assms* **by** (*auto simp: arith-prog-def cong-def*)

lemma *arith-prog-dvd-mono*: $b \text{ dvd } b' \implies \text{arith-prog } a \ b' \subseteq \text{arith-prog } a \ b$
by (*auto simp: arith-prog-def cong-dvd-modulus*)

lemma *bij-betw-arith-prog*:
assumes $b > 0$
shows *bij-betw* $(\lambda n. a + \text{int } b * n)$ *UNIV* (*arith-prog* $a \ b$)
proof (*rule bij-betwI[of - - - \lambda x. (x - a) div int b]*, *goal-cases*)
case 1
thus ?*case*
by (*auto simp: arith-prog-def cong-add-lcancel-0 cong-mult-self-right mult-of-nat-commute*)
next
case 4
thus ?*case*
by (*auto simp: arith-prog-def cong-iff-lin*)
qed (*use 0> in <auto simp: arith-prog-def>*)

lemma *arith-prog-altdef*: $\text{arith-prog } a \ b = \text{range } (\lambda n. a + \text{int } b * n)$
proof (*cases b = 0*)
case *False*
thus ?*thesis*
using *bij-betw-arith-prog[of b]* **by** (*auto simp: bij-betw-def*)
qed *auto*

lemma *infinite-arith-prog*: $b > 0 \implies \text{infinite } (\text{arith-prog } a \ b)$
using *bij-betw-finite[OF bij-betw-arith-prog[of b]]* **by** *simp*

lemma *arith-prog-complement*:
assumes $b > 0$
shows $\neg \text{arith-prog } a \ b = (\bigcup_{i \in \{1..<b\}}. \text{arith-prog } (a + \text{int } i) \ b)$
proof –
have *disjoint*: $x \notin \text{arith-prog } a \ b$ **if** $x \in \text{arith-prog } (a + \text{int } i) \ b$ $i \in \{1..<b\}$ **for**
 $x \ i$
proof –
have $[a \neq a + \text{int } i] \ (\text{mod } \text{int } b)$
proof
assume $[a = a + \text{int } i] \ (\text{mod } \text{int } b)$
hence $[a + 0 = a + \text{int } i] \ (\text{mod } \text{int } b)$ **by** *simp*
hence $[0 = \text{int } i] \ (\text{mod } \text{int } b)$ **by** (*subst (asm) cong-add-lcancel*) *auto*
with that **show** *False* **by** (*auto simp: cong-def*)
qed
thus ?*thesis* **using** *arith-prog-disjoint[of a a + int i b] 0>* **that** **by** *auto*
qed

have *covering*: $x \in \text{arith-prog } a \ b \vee x \in (\bigcup_{i \in \{1..<b\}}. \text{arith-prog } (a + \text{int } i) \ b)$
for x
proof –
define i **where** $i = \text{nat } ((x - a) \text{ mod } (\text{int } b))$

```

have [a + int i = a + (x - a) mod int b] (mod int b)
  unfolding i-def using ⟨b > 0⟩ by simp
also have [a + (x - a) mod int b = a + (x - a)] (mod int b)
  by (intro cong-add) auto
finally have [x = a + int i] (mod int b)
  by (simp add: cong-sym-eq)
hence x ∈ arith-prog (a + int i) b
  using ⟨b > 0⟩ by (auto simp: arith-prog-def)
moreover have i < b using ⟨b > 0⟩
  by (auto simp: i-def nat-less-iff)
ultimately show ?thesis using ⟨b > 0⟩
  by (cases i = 0) auto
qed

from disjoint and covering show ?thesis
  by blast
qed

lemma arith-prog-distinguish:
  assumes x ≠ y
  shows ∃ a c b. b > 0 ∧ x ∈ arith-prog a b ∧ y ∈ arith-prog c b ∧ arith-prog a b
    ∩ arith-prog c b = {}
proof -
  define d where d = nat |x - y| + 1
  from ⟨x ≠ y⟩ have d > 0
    unfolding d-def by auto
  define U where U = arith-prog x d
  define V where V = arith-prog y d

  have U ∩ V = {} unfolding U-def V-def d-def
  proof (use ⟨x ≠ y⟩ in transfer, rule arith-prog-disjoint)
    fix x y :: int
    assume x ≠ y
    show [x ≠ y] (mod int (nat |x - y| + 1))
    proof
      assume [x = y] (mod int (nat |x - y| + 1))
      hence |x - y| + 1 dvd |x - y|
        by (auto simp: cong-iff-dvd-diff algebra-simps)
      hence |x - y| + 1 ≤ |x - y|
        by (rule zdvd-imp-le) (use ⟨x ≠ y⟩ in auto)
      thus False by simp
    qed
  qed auto
  moreover have x ∈ U y ∈ V
  unfolding U-def V-def by (use ⟨d > 0⟩ in transfer, fastforce)+
  ultimately show ?thesis
  using U-def V-def ⟨0 < d⟩ by blast
qed

```

Unions of Arithmetic Progressions

lemma *arith-prog-offset-in*: $\langle k \in \text{arith-prog } a \ b \implies \text{arith-prog } k \ b = \text{arith-prog } a \ b \rangle$

unfolding *arith-prog-def* **by** (*simp add: cong-def*)

lemma *arith-prog-mod*: $\langle \text{arith-prog } (a \ \text{mod } \text{int } b) \ b = \text{arith-prog } a \ b \rangle$

unfolding *arith-prog-def* **by** *auto*

lemma *mod-bounds*: $\langle b > 0 \implies a \ \text{mod } \text{int } b \geq 0 \wedge a \ \text{mod } \text{int } b < b \rangle$

by *simp*

lemma *mod-range*: $\langle \{a \ \text{mod } \text{int } b \mid a. \ b > 0\} \subseteq \{0..<\text{int } b\} \rangle$

using *mod-bounds* **by** *auto*

lemma *finite-mod-range*: $\langle \text{finite } \{a \ \text{mod } \text{int } b \mid a. \ b > 0\} \rangle$

using *mod-range* **by** (*meson finite-atLeastLessThan-int finite-subset*)

definition *arith* :: $\langle \text{nat set} \implies \text{int set} \implies \text{bool} \rangle$ **where**

$\langle \text{arith } B \ U \equiv \forall a \in U. \exists b \in B. \ b > 0 \wedge \text{arith-prog } a \ b \subseteq U \rangle$

lemma *arith-mono*: $\langle \text{arith } B \ U \implies B \subseteq C \implies \text{arith } C \ U \rangle$

unfolding *arith-def* **by** *blast*

lemma *arith-empty-steps*: $\langle \text{arith } B \ U \implies B = \{\} \implies U = \{\} \rangle$

unfolding *arith-def* **by** *blast*

lemma *arith-ne-steps*: $\langle \text{arith } B \ U \implies U \neq \{\} \implies B \neq \{\} \rangle$

using *arith-empty-steps* **by** *blast*

lemma *arith-decomp*:

assumes $\langle \text{arith } B \ U \rangle$

obtains *abs* **where**

$\langle \text{finite } B \implies \text{finite } \text{abs} \rangle$

$\langle \bigwedge a \ b. (a, b) \in \text{abs} \implies b > 0 \wedge b \in B \rangle$

$\langle U = (\bigcup (a, b) \in \text{abs}. \text{arith-prog } a \ b) \rangle$

proof –

have $\langle \forall a \in U. \exists b \in B. \ b > 0 \wedge \text{arith-prog } (a \ \text{mod } \text{int } b) \ b \subseteq U \rangle$

using $\langle \text{arith } B \ U \rangle$ **unfolding** *arith-def* **using** *arith-prog-mod* **by** *simp*

then obtain *f* **where** *f*: $\langle \forall a \in U. \exists b \in B. \ f \ a = b \wedge b > 0 \wedge \text{arith-prog } (a \ \text{mod } \text{int } b) \ b \subseteq U \rangle$

by *metis*

let *?abs* = $\langle \{(a \ \text{mod } \text{int } (f \ a), f \ a) \mid a. \ a \in U\} \rangle$

have *abs*: $\langle U = (\bigcup (a, b) \in \text{?abs}. \text{arith-prog } a \ b) \rangle$

using *f* **by** *fastforce*

have $\langle \{a \ \text{mod } \text{int } (f \ a) \mid a. \ a \in U\} \subseteq (\bigcup b \in B. \ \{a \ \text{mod } \text{int } b \mid a. \ b > 0\}) \rangle$

using *f* **by** *blast*

moreover have $\langle \text{finite } (\bigcup b \in B. \{a \text{ mod int } b \mid a. b > 0\}) \rangle$ **if** $\langle \text{finite } B \rangle$
using *that finite-mod-range by fast*
ultimately have $\langle \text{finite } \{a \text{ mod int } (f a) \mid a. a \in U\} \rangle$ **if** $\langle \text{finite } B \rangle$
using that by (*meson finite-subset*)

moreover have $\langle ?abs \subseteq \{a \text{ mod int } (f a) \mid a. a \in U\} \times f ' U \rangle$
by *blast*
moreover have *fin-f*: $\langle \text{finite } (f ' U) \rangle$ **if** $\langle \text{finite } B \rangle$
using that f by (*meson finite-subset image-subsetI*)
ultimately have *fin-abs*: $\langle \text{finite } ?abs \rangle$ **if** $\langle \text{finite } B \rangle$
using that fin-f by (*meson finite-SigmaI finite-subset*)

have *pos*: $\langle \bigwedge a b. (a, b) \in ?abs \implies b > 0 \wedge b \in B \rangle$
using f by *blast*

show *?thesis*
using *fin-abs pos abs that by meson*
qed

lemma *arith-UNIV*: $\langle \text{arith } \{1\} \text{ UNIV} \rangle$
unfolding *arith-def* **by** *blast*

lemma *arith-empty*: $\langle \text{arith } B \ \{\} \rangle$
unfolding *arith-def* **by** *blast*

lemma *finite-case-prod-lcm*: $\langle \text{finite } B \implies \text{finite } C \implies \text{finite } (\text{case-prod lcm } ' (B \times C)) \rangle$
by *blast*

lemma *arith-inter*:
assumes *U*: $\langle \text{arith } B \ U \rangle$ **and** *V*: $\langle \text{arith } C \ V \rangle$
shows $\langle \text{arith } (\text{case-prod lcm } ' (B \times C)) \ (U \cap V) \rangle$
unfolding *arith-def*
proof *safe*
fix *a*
assume *a*: $\langle a \in U \rangle \langle a \in V \rangle$

from *a U* **obtain** *b* **where** *b*: $\langle b \in B \rangle \langle b > 0 \rangle \langle \text{arith-prog } a \ b \subseteq U \rangle$
unfolding *arith-def* **by** *auto*
from *a V* **obtain** *c* **where** *c*: $\langle c \in C \rangle \langle c > 0 \rangle \langle \text{arith-prog } a \ c \subseteq V \rangle$
unfolding *arith-def* **by** *auto*

with *a b c U V* **have** $\langle \text{arith-prog } a \ (\text{lcm } b \ c) \subseteq U \cap V \rangle$
using *arith-prog-dvd-mono*[*of b <lcm b c> a*] *arith-prog-dvd-mono*[*of c <lcm b c> a*] **by** *blast*
moreover from *b c* **have** $\langle \text{lcm } b \ c > 0 \rangle$
using *lcm-pos-nat* **by** *blast*
ultimately show $\langle \exists b \in (\lambda(x, y). \text{lcm } x \ y) ' (B \times C). 0 < b \wedge \text{arith-prog } a \ b \subseteq U \cap V \rangle$

using $b\ c$ by *blast*
qed

lemma *arith-Inter*:
assumes $\langle \text{finite } X \rangle$ **and** $X: \langle \forall U \in X. \text{arith } B\ U \rangle \langle X \neq \{\} \rangle$
shows $\langle \exists B'. (\text{finite } B \longrightarrow \text{finite } B') \wedge \text{arith } B' (\bigcap X) \rangle$
using *assms*
proof (*induct* X *rule*: *finite-induct*)
 case *empty*
 then show *?case*
 by *simp*
next
 case (*insert* $U\ X$)
 then show *?case*
 using *arith-inter finite-case-prod-lcm*
 by (*metis Inf-insert cInf-singleton insertCI*)
qed

lemma *arith-union*:
assumes $\langle \text{arith } B\ U \rangle \langle \text{arith } C\ V \rangle$
shows $\langle \text{arith } (B \cup C) (U \cup V) \rangle$
using *assms* **unfolding** *arith-def* **by** (*metis Un-iff subset-trans sup-ge1 sup-ge2*)

lemma *arith-ne-infinite*:
assumes $\langle \text{arith } B\ U \rangle \langle U \neq \{\} \rangle$
shows $\langle \text{infinite } U \rangle$
using *assms* **unfolding** *arith-def*
by (*meson equals0I infinite-arith-prog rev-finite-subset*)

lemma *arith-prog-arith* [*intro*]:
assumes $\langle b > 0 \rangle$
shows $\langle \text{arith } \{b\} (\text{arith-prog } a\ b) \rangle$
unfolding *arith-def* **using** *assms* *arith-prog-offset-in* **by** *blast*

lemma *arith-prog-complement-arith* [*intro*]:
assumes $\langle b > 0 \rangle$
shows $\langle \text{arith } \{b\} (- \text{arith-prog } a\ b) \rangle$
proof –
 have $-\text{arith-prog } a\ b = (\bigcup i \in \{1..<b\}. \text{arith-prog } (a + \text{int } i)\ b)$
 using *assms* *arith-prog-complement* **by** *blast*
 also from *assms* **have** *arith* $\{b\}$...
 unfolding *arith-def* **using** *arith-prog-offset-in* **by** *blast*
 finally show *?thesis* .
qed

lemma *arith-complement-arith* [*intro*]:
assumes $\langle \text{arith } B\ U \rangle \langle \text{finite } B \rangle$
shows $\langle \exists B'. \text{finite } B' \wedge \text{arith } B' (- U) \rangle$
proof –

obtain *abs* **where**
abs: $\langle \bigwedge a b. (a, b) \in \text{abs} \implies b > 0 \wedge b \in B \rangle$ *finite abs* **and**
U: $\langle U = (\bigcup (a, b) \in \text{abs}. \text{arith-prog } a \ b) \rangle$
using *assms arith-decomp by metis*
then have *: $\langle \bigwedge a b. (a, b) \in \text{abs} \implies - \text{arith-prog } a \ b = (\bigcup_{i \in \{1..<b\}}. \text{arith-prog } (a + \text{int } i) \ b) \rangle$
using *arith-prog-complement by simp*

have $\langle - U = (\bigcap (a, b) \in \text{abs}. - \text{arith-prog } a \ b) \rangle$
using *U by blast*
also have $\langle \dots = (\bigcap (a, b) \in \text{abs}. (\bigcup_{i \in \{1..<b\}}. \text{arith-prog } (a + \text{int } i) \ b)) \rangle$
using * **by** *blast*
finally have **: $\langle - U = (\bigcap (a, b) \in \text{abs}. (\bigcup_{i \in \{1..<b\}}. \text{arith-prog } (a + \text{int } i) \ b)) \rangle$.

have $\langle \bigwedge a b. (a, b) \in \text{abs} \implies \text{arith } \{b\} (\bigcup_{i \in \{1..<b\}}. \text{arith-prog } (a + \text{int } i) \ b) \rangle$
using *abs * by (metis arith-prog-complement-arith)*
then have ***: $\langle \bigwedge a b. (a, b) \in \text{abs} \implies \text{arith } B (\bigcup_{i \in \{1..<b\}}. \text{arith-prog } (a + \text{int } i) \ b) \rangle$
using *arith-mono abs by fast*

define *X* **where** *X*: $\langle X \equiv (\lambda(a, b). \bigcup_{i \in \{1..<b\}}. \text{arith-prog } (a + \text{int } i) \ b) \text{ 'abs}' \rangle$

from *X* **have** *finite X*
using *abs(2) by simp*
moreover have $\langle \forall U \in X. \text{arith } B \ U \rangle$
using *X *** by fast*
ultimately have $\langle \exists B'. \text{finite } B' \wedge \text{arith } B' (\bigcap X) \rangle$
using *arith-Inter[of X] finite B arith-UNIV by fastforce*

moreover from *X* **have** $\langle - U = \bigcap X \rangle$
using ** **by** *simp*
ultimately show *?thesis*
by *simp*

qed

lemma *arith-distinguish*:
assumes $\langle x \neq y \rangle$
shows $\langle \exists B \ U \ V. \text{finite } B \wedge \text{arith } B \ U \wedge \text{arith } B \ V \wedge x \in U \wedge y \in V \wedge U \cap V = \{\} \rangle$
proof –

obtain *a b c* **where**
b: $\langle 0 < b \rangle$ **and** *x*: $\langle x \in \text{arith-prog } a \ b \rangle$ **and** *y*: $\langle y \in \text{arith-prog } c \ b \rangle$ **and**
*: $\langle \text{arith-prog } a \ b \cap \text{arith-prog } c \ b = \{\} \rangle$
using *assms arith-prog-distinguish by meson*

let *?B* = $\langle \{b\} \rangle$
let *?U* = $\langle \text{arith-prog } a \ b \rangle$
let *?V* = $\langle \text{arith-prog } c \ b \rangle$

have $\langle \text{finite } ?B \wedge \text{arith } ?B \text{ ?}U \wedge \text{arith } ?B \text{ ?}V \wedge x \in ?U \wedge y \in ?V \wedge ?U \cap ?V = \{\} \rangle$
using $b \ x \ y \ *$
by *blast*
then show $?thesis$
by *blast*
qed

Finite Unions of Arithmetic Progressions

definition $\text{fin-arith} :: \langle \text{int set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{fin-arith } U \equiv \exists B. \text{finite } B \wedge \text{arith } B \ U \rangle$

lemma fin-arith-UNIV [*intro*]: $\langle \text{fin-arith } UNIV \rangle$
unfolding fin-arith-def **using** arith-UNIV **by** *force*

lemma fin-arith-empty [*intro*]: $\langle \text{fin-arith } \{\} \rangle$
unfolding fin-arith-def **using** arith-empty **by** *blast*

lemma fin-arith-inter [*intro*]: $\langle \text{fin-arith } U \Longrightarrow \text{fin-arith } V \Longrightarrow \text{fin-arith } (U \cap V) \rangle$
unfolding fin-arith-def **using** arith-inter $\text{finite-case-prod-lcm}$ **by** *metis*

lemma fin-arith-union [*intro*]: $\langle \text{fin-arith } U \Longrightarrow \text{fin-arith } V \Longrightarrow \text{fin-arith } (U \cup V) \rangle$
unfolding fin-arith-def **using** arith-union **by** *blast*

lemma fin-arith-compl [*intro*]: $\langle \text{fin-arith } U \Longrightarrow \text{fin-arith } (- \ U) \rangle$
unfolding fin-arith-def **by** *blast*

lemma $\text{fin-arith-distinguish}$:
assumes $\langle x \neq y \rangle$
shows $\langle \exists U \ V. \text{fin-arith } U \wedge \text{fin-arith } V \wedge x \in U \wedge y \in V \wedge U \cap V = \{\} \rangle$
using assms arith-distinguish **unfolding** fin-arith-def **by** *meson*

Singletons

lemma $\text{arith-prog-singleton}$: $\langle \bigcap \{ \text{arith-prog } a \ b \mid a \ b. \ b > 0 \wedge x \in \text{arith-prog } a \ b \} = \{x\} \rangle$

proof

show $\langle \bigcap \{ \text{arith-prog } a \ b \mid a \ b. \ 0 < b \wedge x \in \text{arith-prog } a \ b \} \subseteq \{x\} \rangle$
using $\text{arith-prog-distinguish}$ **by** *fast*

qed *fast*

lemma $\text{fin-arith-Inter-singleton}$: $\langle \bigcap \{ U \mid U. \text{fin-arith } U \wedge x \in U \} = \{x\} \rangle$

proof –

have $\langle \{ \text{arith-prog } a \ b \mid a \ b. \ b > 0 \wedge x \in \text{arith-prog } a \ b \} \subseteq \{ U \mid U \ b. \ \text{arith } \{b\} \ U \wedge x \in U \} \rangle$

using arith-prog-arith **by** *blast*

also have $\langle \dots \subseteq \{ U \mid U \ B. \ \text{finite } B \wedge \text{arith } B \ U \wedge x \in U \} \rangle$

by *force*

finally have $\langle \{arith\text{-}prog\ a\ b \mid a\ b.\ b > 0 \wedge x \in arith\text{-}prog\ a\ b\} \subseteq \{U \mid U.\ fin\text{-}arith\ U \wedge x \in U\} \rangle$
unfolding *fin-arith-def* **by** *auto*
then have $\langle \bigcap \{U \mid U.\ fin\text{-}arith\ U \wedge x \in U\} \subseteq \bigcap \{arith\text{-}prog\ a\ b \mid a\ b.\ b > 0 \wedge x \in arith\text{-}prog\ a\ b\} \rangle$
by *blast*
then show *?thesis*
using *arith-prog-singleton* **by** *auto*
qed

lemma *singleton-not-finarith*: $\langle \neg fin\text{-}arith\ \{x\} \rangle$
unfolding *fin-arith-def* **using** *arith-ne-infinite* **by** *blast*

4.8.2 Counterexample

definition *Pss* :: $\langle int\ set\ set \rangle$ **where**
 $\langle Pss \equiv \{U.\ fin\text{-}arith\ U\} \rangle$

lemma *Pss-empty*: $\langle \{\} \in Pss \rangle$
unfolding *Pss-def* **by** *blast*

lemma *Pss-UNIV*: $\langle UNIV \in Pss \rangle$
unfolding *Pss-def* **by** *blast*

lemma *Pss-union*: $\langle X \in Pss \implies Y \in Pss \implies X \cup Y \in Pss \rangle$
unfolding *Pss-def* **by** *blast*

lemma *Pss-inter*: $\langle X \in Pss \implies Y \in Pss \implies X \cap Y \in Pss \rangle$
unfolding *Pss-def* **by** *blast*

lemma *Pss-compl*: $\langle X \in Pss \implies \neg X \in Pss \rangle$
unfolding *Pss-def* **by** *blast*

definition *my-gframe* :: $\langle int\ gframe \rangle$ **where**
 $\langle my\text{-}gframe \equiv (\lambda W = UNIV, \mathcal{R} = \lambda x.\ UNIV, \Pi = Pss) \rangle$

lemma *wf-frame-mygframe*: $\langle wf\text{-}frame\ (frame.\ truncate\ my\text{-}gframe) \rangle$
unfolding *wf-frame-def* **unfolds** *my-gframe-def* **by** *blast*

lemma *admissible-mygframe*: $\langle admissible\ (frame.\ truncate\ my\text{-}gframe)\ (\Pi\ my\text{-}gframe) \rangle$
unfolding *admissible-def* **unfolds** *my-gframe-def* *Pss-def*
by $(auto\ simp:\ Compl\text{-}eq\text{-}Diff\text{-}UNIV[symmetric])$

lemma *wf-mygframe*: $\langle wf\text{-}gframe\ my\text{-}gframe \rangle$
using *wf-frame-mygframe* *admissible-mygframe*
unfolding *wf-gframe-def* **unfolds** *my-gframe-def* *Pss-def* **by** *fast*

definition *my-model* :: $\langle (int,\ int)\ model \rangle$ **where**
 $\langle my\text{-}model \equiv gframe.\ extend\ my\text{-}gframe\ (\mathcal{N} = \lambda i.\ i,\ \mathfrak{N} = \lambda i.\ int\ i,\ \mathcal{V} = \lambda n.\ \{\}) \rangle$

$\mathfrak{V} = \lambda n. \{\}\rangle$

lemma *wf-my-model*: $\langle wf\text{-model } my\text{-model} \rangle$

using *wf-my-gframe* **unfolding** *wf-model-def wf-env-def unfolds my-model-def my-gframe-def Pss-def*
by *blast*

abbreviation *GloE* :: $\langle 'x\ fm \Rightarrow 'x\ fm \rangle (\langle \mathbf{E} \rangle)$ **where**

$\langle \mathbf{E} \ p \equiv \neg (\mathbf{A} (\neg p)) \rangle$

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* $p \equiv$

$(\diamond p) \longrightarrow$
 $(\diamond (\downarrow (\diamond (\downarrow ($
 $(@ (\#1) p) \wedge$
 $(@ (\#0) p) \wedge$
 $\neg (@ (\#0) (\cdot (\#1))))))))) \rangle$

Finite unions of arithmetic progressions are either empty or infinite.

lemma *fin-arith-nowhere-or-twice*:

assumes $\langle fin\text{-arith } U \rangle$
shows $\langle U = \{\} \vee (\exists x\ y. x \in U \wedge y \in U \wedge x \neq y) \rangle$
using *assms arith-ne-infinite* **unfolding** *fin-arith-def*
by (*metis infinite-int-iff-unbounded less-le-not-le*)

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

lemma *nowhere-or-twice-admissible*: $\langle (my\text{-model}, x) \models \forall (nowhere\text{-or-twice } (\cdot (\#0))) \rangle$

unfolding *my-model-def my-gframe-def unfolds Pss-def*
using *fin-arith-nowhere-or-twice* **by** *simp*

However, propositional quantification lets us form a singleton.

abbreviation *singleton* $x \equiv \forall (@(\circ x) (\cdot (\#0)) \longrightarrow \cdot (\#0)) \rangle$

lemma *singleton*: $\langle ((my\text{-model}, x) \models singleton\ y) \longleftrightarrow x = y \rangle$

unfolding *my-model-def my-gframe-def unfolds Pss-def*
using *fin-arith-Inter-singleton* **by** *auto*

lemma *fin-arith-distinguish'*:

$\langle \forall P. fin\text{-arith } P \longrightarrow y \in P \longrightarrow v \in P \Longrightarrow v \neq w \Longrightarrow \exists P. y \in P \wedge fin\text{-arith } P$
 $\wedge w \notin P \rangle$
by (*metis disjoint-iff fin-arith-distinguish*)

The singleton does not hold nowhere-or-twice.

lemma *not-nowhere-or-twice-singleton*: $\langle \neg (my\text{-model}, x) \models nowhere\text{-or-twice } (singleton\ y) \rangle$

unfolding *my-model-def my-gframe-def unfolds Pss-def*
using *fin-arith-distinguish'* **by** *auto*

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

theorem *counter:*

shows $\langle \neg (my-model, x) \models \forall (nowhere-or-twice (\cdot(\#0))) \longrightarrow nowhere-or-twice (singleton y) \rangle$

using *nowhere-or-twice-admissible not-nowhere-or-twice-singleton*

by *(meson semantics.simps(3,4))*

end