# Abstract Consistency Properties

Asta Halkjær Boserup, Anders Schlichtkrull

April 9, 2026

**Abstract**

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

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

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

# Bibliography


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

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

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

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

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

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

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

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

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

# Contents

# Chapter 1

# Abstract Consistency Properties

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

## 1.1 Utility

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

**lemma** *infinite-diff-finite*: ‹*finite* $A \Longrightarrow$ *infinite* $(- B) \Longrightarrow$ *infinite* $(- (A \cup B))$›
**by** (*metis Compl-Diff-eq double-complement finite-Diff2 sup-commute*)

**lemma** *infinite-Diff-fin-Un*: ‹*infinite* $(X - Y) \Longrightarrow$ *finite* $Z \Longrightarrow$ *infinite* $(X - (Z \cup Y))$›
**by** (*simp add*: *Set-Diff-Un Un-commute*)

**lemma** *infinite-Diff-subset*: ‹*infinite* $(X - A) \Longrightarrow B \subseteq A \Longrightarrow$ *infinite* $(X - B)$›
**by** (*meson Diff-cancel Diff-eq-empty-iff Diff-mono infinite-super*)

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

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

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

**lemma** *infinite-left*: ‹*finite* $C \Longrightarrow$ *infinite* $A \Longrightarrow |A| \leq o\ |-\ B| \Longrightarrow |A| \leq o\ |-\ (C \cup B)|$›
**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** ‹$\forall\, x.\ |f\ x| <o\ |X|$› ‹*infinite* $X$›
**shows** ‹$|\bigcup x \in X.\ f\ x| \leq o\ |X|$›
**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*: ‹$a \in underS\ c \Longrightarrow b \in underS\ c \Longrightarrow a \in under\ b \vee b \in under\ a$›
**by** (*meson BNF-Least-Fixpoint.underS-Field REFL Refl-under-in in-mono under-ofilter ofilter-linord*)

**lemma** *finite-underS-bound*:
**assumes** ‹*finite* $X$› ‹$X \subseteq underS\ a$› ‹$X \neq \{\}$›
**shows** ‹$\exists\, a \in X.\ \forall\, b \in X.\ b \in under\ a$›
**using** *assms*
**proof** (*induct X rule*: *finite-induct*)
**case** (*insert x F*)
**then show** *?case*
**proof** (*cases* ‹$F = \{\}$›)
**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** ‹*finite* $p$› ‹$p \subseteq (\bigcup a \in Field\ r.\ f\ a)$›
**shows** ‹$\exists\, b.\ p \subseteq (\bigcup a \in under\ b.\ f\ a)$›

**using** *assms*
**proof** (*induct rule*: *finite-induct*)
**case** (*insert x p*)
**then obtain** *b* **where** ‹$p \subseteq (\bigcup a \in under\ b.\ f\ a)$›
**by** *fast*
**moreover obtain** $b'$ **where** ‹$x \in f\ b'$› ‹$b' \in Field\ r$›
**using** *insert*(*4*) **by** *blast*
**then have** ‹$x \in (\bigcup a \in under\ b'.\ f\ a)$›
**using** *REFL Refl-under-in* **by** *fast*
**ultimately have** ‹$\{x\} \cup p \subseteq (\bigcup a \in under\ b.\ f\ a) \cup (\bigcup a \in under\ b'.\ f\ a)$›
**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*: ‹$a \in underS\ b \Longrightarrow b \in underS\ c \Longrightarrow a \in underS\ c$›
**by** (*meson ANTISYM TRANS underS-underS-trans*)

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

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

**lemma** *chain-index*:
**assumes** *ch*: ‹*is-chain f*› **and** *fin*: ‹*finite F*› **and** *ne*: ‹$Field\ r \neq \{\}$›
**shows** ‹$F \subseteq (\bigcup a \in Field\ r.\ f\ a) \Longrightarrow \exists a \in Field\ r.\ F \subseteq f\ a$›
**using** *fin*
**proof** (*induct rule*: *finite-induct*)
**case** *empty*
**then show** *?case*
**using** *ne* **by** *blast*
**next**
**case** (*insert x F*)
**then have** ‹$\exists a \in Field\ r.\ F \subseteq f\ a$› ‹$\exists b \in Field\ r.\ x \in f\ b$› ‹$F \subseteq (\bigcup x \in Field\ r.\ f\ x)$›
**using** *ch* **by** *simp-all*
**then obtain** *a* **and** *b* **where** *f*: ‹$F \subseteq f\ a$› ‹$x \in f\ b$› **and** *nm*: ‹$a \in Field\ r$› ‹$b \in Field\ r$›
**by** *blast*
**have** ‹$b \in under\ (max2\ a\ b)$› ‹$a \in under\ (max2\ a\ b)$›
**using** *nm* **by** (*meson REFL Refl-under-in TRANS max2-greater-among subset-iff under-incl-iff*)+
**have** ‹$x \in f\ (max2\ a\ b)$›
**using** *is-chainD ch* ‹$b \in under\ (max2\ a\ b)$› *f*(*2*) **by** *blast*
**moreover have** ‹$F \subseteq f\ (max2\ a\ b)$›
**using** *is-chainD ch* ‹$a \in local.under\ (max2\ a\ b)$› *f*(*1*) **by** *blast*
**ultimately show** *?case*

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

**end**

## 1.2 Finite Character

**definition** *close* :: ‹'*a set set* ⇒ '*a set set*› **where**
‹*close C* ≡ {*S*. (∃ *S'* ∈ *C*. *S* ⊆ *S'*)}›

**definition** *subset-closed* :: ‹'*a set set* ⇒ *bool*› **where**
‹*subset-closed C* ≡ ∀ *S'* ∈ *C*. ∀ *S* ⊆ *S'*. *S* ∈ *C*›

**lemma** *subset-in-close*: ‹*S* ⊆ *S'* ⟹ *x* ∪ *S'* ∈ *C* ⟹ *x* ∪ *S* ∈ *close C*›
**unfolding** *close-def* **by** *blast*

**lemma** *close-closed*: ‹*subset-closed* (*close C*)›
**unfolding** *close-def subset-closed-def* **by** *blast*

**lemma** *close-subset*: ‹*C* ⊆ *close C*›
**unfolding** *close-def* **by** *blast*

**definition** *finite-char* :: ‹'*a set set* ⇒ *bool*› **where**
‹*finite-char C* ≡ ∀ *S*. *S* ∈ *C* ⟷ (∀ *S'* ⊆ *S*. *finite S'* ⟶ *S'* ∈ *C*)›

**definition** *mk-finite-char* :: ‹'*a set set* ⇒ '*a set set*› **where**
‹*mk-finite-char C* ≡ {*S*. (∀ *S'* ⊆ *S*. *finite S'* ⟶ *S'* ∈ *C*)}›

**lemma** *finite-char*: ‹*finite-char* (*mk-finite-char C*)›
**unfolding** *finite-char-def mk-finite-char-def* **by** *blast*

**lemma** *finite-char-closed*: ‹*finite-char C* ⟹ *subset-closed C*›
**unfolding** *finite-char-def subset-closed-def* **by** (*meson order-trans*)

**lemma** *finite-char-subset*: ‹*subset-closed C* ⟹ *C* ⊆ *mk-finite-char C*›
**unfolding** *mk-finite-char-def subset-closed-def* **by** *blast*

**lemma** (**in** *wo-rel*) *chain-union-closed*:
**assumes** ‹*finite-char C*› ‹*is-chain f*› ‹∀ *a* ∈ *Field r*. *f a* ∈ *C*› ‹*Field r* ≠ {}›
**shows** ‹(⋃ *a* ∈ *Field r*. *f a*) ∈ *C*›
**using** *assms chain-index* **unfolding** *finite-char-def* **by** *metis*

**definition** *maximal* :: ‹'*a set set* ⇒ '*a set* ⇒ *bool*› **where**
‹*maximal C S* ⟷ (∀ *S'* ∈ *C*. *S* ⊆ *S'* ⟶ *S* = *S'*)›

## 1.3 Consistency Properties

**locale** *Params* =

**fixes** *map-fm* :: ‹('x ⇒ 'x) ⇒ 'fm ⇒ 'fm›
**and** *params-fm* :: ‹'fm ⇒ 'x set›
**and** *is-param* :: ‹'x ⇒ bool›
**assumes** *map-fm-id*: ‹*map-fm id* = *id*›
**and** *finite-params-fm* [*simp*]: ‹⋀*p*. *finite* (*params-fm p*)›
**and** *map-params-fm*: ‹⋀*f g p*. (∀ *x* ∈ *params-fm p*. *f x* = *g x*) ⟹ *map-fm f p* = *map-fm g p*›
**begin**

**definition** *is-subst* :: ‹('x ⇒ 'x) ⇒ bool› **where**
‹*is-subst f* ≡ ∀ *x*. *is-param x* ⟷ *is-param* (*f x*)›

**lemma** *is-subst-id* [*intro*]: ‹*is-subst id*›
**unfolding** *is-subst-def* **by** *simp*

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

**lemma** *mk-alt-consistency-subset*: ‹*C* ⊆ *mk-alt-consistency C*›
**unfolding** *mk-alt-consistency-def*
**proof**
**fix** *x*
**assume** ‹*x* ∈ *C*›
**then have** ‹*map-fm id* ‘ *x* ∈ *C*›
**using** *map-fm-id* **by** *simp*
**then have** ‹∃ *f*. *is-subst f* ∧ *map-fm f* ‘ *x* ∈ *C*›
**by** *blast*
**then show** ‹*x* ∈ {*S*. ∃ *f*. *is-subst f* ∧ *map-fm f* ‘ *S* ∈ *C*}›
**by** *blast*
**qed**

**lemma** *mk-alt-consistency-closed*:
**assumes** ‹*subset-closed C*›
**shows** ‹*subset-closed* (*mk-alt-consistency C*)›
**unfolding** *subset-closed-def mk-alt-consistency-def*
**proof** *safe*
**fix** *S S' f*
**assume** ‹*is-subst f*› ‹*map-fm f* ‘ *S'* ∈ *C*› ‹*S* ⊆ *S'*›
**moreover have** ‹*map-fm f* ‘ *S* ⊆ *map-fm f* ‘ *S'*›
**using** ‹*S* ⊆ *S'*› **by** *blast*
**moreover have** ‹∀ *S'* ∈ *C*. ∀ *S* ⊆ *S'*. *S* ∈ *C*›
**using** ‹*subset-closed C*› **unfolding** *subset-closed-def* **by** *blast*
**ultimately show** ‹∃ *f*. *is-subst f* ∧ *map-fm f* ‘ *S* ∈ *C*›
**by** *blast*
**qed**

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

**lemma** *infinite-params*: ‹*infinite* (*U* − *params B*) ⟹ *infinite* (*U* − *params* (*set ps* ∪ *B*))›
**using** *finite-params-fm* **by** (*metis List.finite-set UN-Un finite-UN-I infinite-Diff-fin-Un*)

**lemma** *infinite-params-left*:
**assumes** ‹*infinite A*› ‹|*A*| ≤*o* |*U* − *params S*|›
**shows** ‹|*A*| ≤*o* |*U* − *params* (*set ps* ∪ *S*)|›
**proof** −
**have** ‹*infinite* (*U* − *params S*)›
**using** *assms card-of-ordLeq-infinite* **by** *blast*
**then have** ‹|*U* − *params S*| =*o* |*U* − *params* (*set ps* ∪ *S*)|›
**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* :: ‹'*fm set* ⇒ *bool*› **where**
‹*enough-new S* ≡ |*UNIV* :: '*fm set*| ≤*o* |*Collect is-param* − *params S*|›

**lemma** *enough-new-countable*:
**assumes** ‹∃ *to-nat* :: '*fm* ⇒ *nat*. *inj to-nat*› ‹*infinite* (*Collect is-param* − *params S*)›
**shows** ‹*enough-new S*›
**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** ‹|*UNIV* :: '*fm set*| ≤*o* |*UNIV* − *params S*|› ‹⋀*x*. *is-param x*›
**shows** ‹*enough-new S*›
**unfolding** *enough-new-def* **using** *assms* **by** (*simp add*: *Collect-cong*)

**end**

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

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

**inductive-cases** (**in** *Params*) *sat*$_E$*-CondE*[*elim*!]: ‹*sat*$_E$ (*Cond P H*) *C*›
**inductive-cases** (**in** *Params*) *sat*$_E$*-WitsE*[*elim*!]: ‹*sat*$_E$ (*Wits W*) *C*›

**inductive** (**in** *Params*) *sat*$_A$ :: ‹('*x*, '*fm*) *kind* ⇒ '*fm set set* ⇒ *bool*› **where**
*sat*$_A$*-Cond* [*intro*!]: ‹(⋀*S ps Q*. *S* ∈ *C* ⟹ *set ps* ⊆ *S* ⟹ *P ps Q* ⟹ *Q C S*) ⟹ *sat*$_A$ (*Cond P H*) *C*›

| $sat_A$-*Wits* [*intro*!]: ‹($\bigwedge S\ p\ x.\ S \in C \Longrightarrow p \in S \Longrightarrow x \notin params\ S \Longrightarrow is\text{-}param\ x \Longrightarrow set\ (W\ p\ x) \cup S \in C) \Longrightarrow sat_A\ (Wits\ W)\ C$›

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

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

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

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

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

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

**theorem** (**in** *Params*) $sat_H$-*Wits*: ‹$sat_E\ (Wits\ W)\ C \Longrightarrow S \in C \Longrightarrow maximal\ C\ S \Longrightarrow sat_H\ (Wits\ W)\ S$›
**unfolding** *maximal-def* **by** *fast*

### 1.3.1 Consistency Kinds

**locale** *Consistency-Kind* = *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** *K* :: ‹('*x*, '*fm*) *kind*›
**assumes** *respects-close*: ‹$\bigwedge C.\ sat_E\ K\ C \Longrightarrow sat_E\ K\ (close\ C)$›
**and** *respects-alt*: ‹$\bigwedge C.\ sat_E\ K\ C \Longrightarrow subset\text{-}closed\ C \Longrightarrow sat_A\ K\ (mk\text{-}alt\text{-}consistency\ C)$›
**and** *respects-fin*: ‹$\bigwedge C.\ subset\text{-}closed\ C \Longrightarrow sat_A\ K\ C \Longrightarrow sat_A\ K\ (mk\text{-}finite\text{-}char\ C)$›
**and** *hintikka*: ‹$\bigwedge C\ S.\ sat_E\ K\ C \Longrightarrow S \in C \Longrightarrow maximal\ C\ S \Longrightarrow sat_H\ K\ S$›

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

*is-param K*›
**begin**

**lemma** *sat*$_E$: ‹*K* ∈ *set Ks* ⟹ *prop*$_E$ *Ks C* ⟹ *sat*$_E$ *K C*›
**unfolding** *prop*$_E$*-def* **by** *blast*

**lemma** *prop*$_E$*-close*: ‹*prop*$_E$ *Ks C* ⟹ *prop*$_E$ *Ks* (*close C*)›
**unfolding** *prop*$_E$*-def* **using** *all-kinds Consistency-Kind.respects-close* **by** *fast*

**lemma** *prop*$_E$*-alt*: ‹*prop*$_E$ *Ks C* ⟹ *subset-closed C* ⟹ *prop*$_A$ *Ks* (*mk-alt-consistency C*)›
**unfolding** *prop*$_E$*-def prop*$_A$*-def* **using** *all-kinds Consistency-Kind.respects-alt* **by** *fast*

**lemma** *prop*$_E$*-fin*: ‹*subset-closed C* ⟹ *prop*$_A$ *Ks C* ⟹ *prop*$_A$ *Ks* (*mk-finite-char C*)›
**unfolding** *prop*$_A$*-def* **using** *all-kinds Consistency-Kind.respects-fin* **by** *fast*

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

**lemma** *mk-alt-fin-subset-closed*: ‹*subset-closed* (*mk-alt-fin C*)›
**unfolding** *mk-alt-fin-def* **using** *finite-char finite-char-closed* **by** *blast*

**lemma** *mk-alt-fin-finite-char*: ‹*finite-char* (*mk-alt-fin C*)›
**unfolding** *mk-alt-fin-def* **using** *finite-char* **by** *blast*

**lemma** *mk-alt-fin-in*: ‹*S* ∈ *C* ⟹ *S* ∈ *mk-alt-fin C*›
**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*$_E$: ‹*prop*$_E$ *Ks C* ⟹ *prop*$_A$ *Ks* (*mk-alt-fin C*)›
**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* :: ‹('*x*, '*fm*) *kind list* ⇒ '*fm* ⇒ '*fm set* ⇒ '*fm set*›
**where**
‹*witness-kinds* [] *p S* = {}›
| ‹*witness-kinds* (*Cond* - - # *Ks*) *p S* = *witness-kinds Ks p S*›
| ‹*witness-kinds* (*Wits W* # *Ks*) *p S* =
(*let*
*rest* = *witness-kinds Ks p S*;
*a* = *SOME x*. *x* ∈ *Collect is-param* − *params* (*rest* ∪ {*p*} ∪ *S*)
*in set* (*W p a*) ∪ *rest*)›

**lemma** (**in** *Params*) *witness-kinds-new*:

**assumes** ‹*infinite* (*UNIV* :: *'fm set*)› ‹*infinite* (*Collect is-param* − *params S*)›
**shows** ‹*infinite* (*Collect is-param* − *params* (*witness-kinds Ks p S* ∪ {*p*} ∪ *S*))›
**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*: ‹*aboveS a* ≠ {}›
**by** (*simp add*: *isLimOrd isLimOrd-aboveS*)

**lemma** *params-left*: ‹*enough-new S* ⟹ *enough-new* (*set ps* ∪ *S*)›
**unfolding** *enough-new-def* **using** *infinite-params-left inf-univ* **by** *blast*

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

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

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

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

**lemma** *adm-woL-extendL*: ‹*adm-woL* (*extendL C*)›
**unfolding** *extendL-def adm-woL-def* **by** *blast*

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

**lemma** *finite-witness-kinds*: ‹*finite* (*witness-kinds Qs p S*)›
**unfolding** *witness-def* **by** (*induct Qs p S rule*: *witness-kinds.induct*) (*simp-all add*: *Let-def*)

**lemma** *finite-witness*: ‹*finite* (*witness p S*)›
**unfolding** *witness-def* **using** *finite-witness-kinds* **.**

**lemma** *finite-witness-kinds-params*: ‹*finite* (*params* (*witness-kinds Qs p S*))›
**using** *finite-witness-kinds* **by** *simp*

**lemma** *finite-witness-params*: ‹*finite* (*params* (*witness p S*))›
**using** *finite-witness* **by** *simp*

**lemma** *extend-zero* [*simp*]: ‹*extend C S zero* = *S*›
**unfolding** *extend-def worecZSL-zero*[*OF adm-woL-extendL*] **..**

**lemma** *extend-succ* [*simp*]: ‹*extend C S* (*succ a*) =
(*if* {*a*} ∪ *extend C S a* ∈ *C then witness a* (*extend C S a*) ∪ {*a*} ∪ *extend C S a else extend C S a*)›
**unfolding** *extend-def extendS-def worecZSL-succ*[*OF adm-woL-extendL aboveS-ne*]
**..**

**lemma** *extend-isLim* [*simp*]:
**assumes** ‹*isLim a*› ‹*a* ≠ *zero*›
**shows** ‹*extend C S a* = (⋃ *b* ∈ *underS a*. *extend C S b*)›
**unfolding** *extend-def extendL-def worecZSL-isLim*[*OF adm-woL-extendL assms*]
..

**lemma** *extend-subset*: ‹*S* ⊆ *extend C S a*›
**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*: ‹*S* ⊆ *Extend C S*›
**unfolding** *Extend-def* **using** *extend-subset* **by** *fast*

**lemma** *extend-underS*: ‹*b* ∈ *underS a* ⟹ *extend C S b* ⊆ *extend C S a*›
**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** ‹*b* = *i* ∨ *b* ∈ *underS i*›
**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*: ‹*b* ∈ *under a* ⟹ *extend C S b* ⊆ *extend C S a*›
**using** *extend-underS supr-greater supr-under*
**by** (*metis emptyE in-Above-under set-eq-subset underS-I under-empty*)

**lemma** *params-origin*:
**assumes** ‹*x* ∈ *params* (*extend C S a*)›
**shows** ‹*x* ∈ *params S* ∨ (∃ *b* ∈ *underS a*. *x* ∈ *params* (*witness b* (*extend C S b*) ∪ {*b*}))›
**using** *assms*
**proof** (*induct a rule*: *well-order-inductZSL*)
**case** *Zero*
**then show** *?case*
**by** *simp*
**next**
**case** (*Suc i*)

```
  then consider
    (here) ‹x ∈ params ({i} ∪ witness i (extend C S i))› |
    (there) ‹x ∈ params (extend C S i)›
    using Suc(3) by (fastforce split: if-splits)
  then show ?case
  proof cases
    case here
    then show ?thesis
      using Suc(1) succ-diff succ-in by (metis sup-commute underS-I )
  next
    case there
    then show ?thesis
      using Suc by (metis in-mono underS-subset-under underS-succ)
  next
  qed
next
  case (Lim i)
  then obtain j where ‹j ∈ underS i› ‹x ∈ params (extend C S j)›
   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: ‹is-chain (extend C S)›
  by (simp add: extend-under is-chain-def)

lemma extend-in-C-step:
  assumes ‹prop_A Ks C› ‹{a} ∪ extend C S a ∈ C›
    and inf: ‹infinite (Collect is-param − params ({a} ∪ extend C S a))›
  shows ‹extend C S (succ a) ∈ C›
proof −
  have ‹set Qs ⊆ set Ks ⟹ witness-kinds Qs a (extend C S a) ∪ {a} ∪ extend C
S a ∈ C› for Qs
  proof (induct Qs)
    case Nil
    then show ?case
      using assms(2) by simp
  next
    case (Cons Q Qs)
    let ?S = ‹extend C S a›
    let ?rest = ‹witness-kinds Qs a ?S›

    have Q: ‹Q ∈ set Ks›
      using Cons.prems by simp

    have ∗: ‹?rest ∪ {a} ∪ ?S ∈ C›
      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 prop_A-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 ∗: ‹|underS i| <o |UNIV :: 'fm set|›
    using card-of-underS by (simp add: Cinfinite-r)

  let ?params = ‹λk. params ({k} ∪ witness k (extend C S k))›
  let ?X = ‹⋃ k ∈ underS i. ?params k›
  have ‹|?X| <o |UNIV :: 'fm set|›
  proof (cases ‹finite (underS i)›)
    case True
    then have ‹finite ?X›
      using finite-witness-params by simp
    then show ?thesis
     using Cinfinite-r unfolding cinfinite-def by (simp add: finite-ordLess-infinite)
  next
    case False
    moreover have ‹∀ k. finite (?params k)›
      using finite-witness-params by simp
    then have ‹∀ k. |?params k| <o |underS i|›
      using False by simp
    ultimately have ‹|?X| ≤o |underS i|›
      using card-of-infinite-smaller-Union by fast
    then show ?thesis
      using ∗ ordLeq-ordLess-trans by blast
  qed
  then have ‹|?X| <o |Collect is-param − params S|›
    using Suc(4) ordLess-ordLeq-trans unfolding enough-new-def by blast
  moreover have ‹infinite (Collect is-param − params S)›
    using Suc(4) Cinfinite-r unfolding cinfinite-def enough-new-def
    by (metis Field-card-of ordLeq-finite-Field)
  ultimately have ‹|Collect is-param − params S − ?X| =o |Collect is-param −
params S|›
    using card-of-Un-diff-infinite by blast
  moreover from this have ‹infinite (Collect is-param − params S − ?X)›
    using ‹infinite (Collect is-param − params S)› card-of-ordIso-finite by blast
  moreover have ‹⋀a. a ∈ params (extend C S i) ⟹ a ∈ params S ∨ a ∈ ?X›
    using params-origin by simp
  then have ‹params (extend C S i) ⊆ params S ∪ ?X›
    by fast
  ultimately have ‹infinite (Collect is-param − params (extend C S i))›
    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** ‹$prop_A$ *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 worecZSL-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 worecZSL-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*)

**qed**
**qed**

**lemma** *Extend-in-C*:
**assumes** ‹$prop_A$ *Ks C*› ‹*finite-char C*› ‹$S \in C$› ‹*enough-new S*›
**shows** ‹*Extend C S* $\in C$›
**unfolding** *Extend-def* **using** *assms chain-union-closed is-chain-extend extend-in-C*
**by** *simp*

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

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

**theorem** *Extend-witnessed*:
**assumes** ‹$prop_A$ *Ks C*› ‹*finite-char C*› ‹$S \in C$› ‹*enough-new S*›
**shows** ‹*witnessed* (*Extend C S*)›
**unfolding** *witnessed-def*
**proof** *safe*
**fix** *p*
**assume** ‹$p \in Extend\ C\ S$›

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

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

**theorem** *mk-mcs-rmaximal*: ‹*maximal C* (*mk-mcs C S*)›
**using** *Extend-maximal maximal-def mk-alt-fin-in mk-alt-fin-subset-closed* **by** *meson*

**theorem** *mk-mcs-witnessed*:
**assumes** ‹$prop_E$ *Ks C*› ‹*S* $\in$ *C*› ‹*enough-new S*›
**shows** ‹*witnessed* (*mk-mcs C S*)›
**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** ‹$prop_E$ *Ks C*› ‹*S* $\in$ *C*› ‹*enough-new S*›
**shows** ‹$prop_H$ *Ks* (*mk-mcs C S*)›
**unfolding** $prop_H$*-def*
**proof**
**fix** *K*
**assume** *K*: ‹*K* $\in$ *set Ks*›
**show** ‹$sat_H$ *K* (*mk-mcs C S*)›
**proof** (*cases K*)
**case** (*Cond P H*)
**moreover have** ‹*maximal* (*mk-alt-fin C*) (*mk-mcs C S*)›
**using** *Extend-maximal mk-alt-fin-subset-closed* **by** *blast*
**moreover have** ‹$prop_A$ *Ks* (*mk-alt-fin C*)›
**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** ‹$sat_E$ (*Cond P H*) (*mk-alt-fin C*)›
**using** ‹$prop_A$ *Ks* (*mk-alt-fin C*)› *Cond K* **unfolding** $prop_A$*-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*: ‹$prop_H$ *Ks H*›
**begin**

**lemma** $sat_H$: ‹*K* ∈ *set Ks* ⟹ $sat_H$ *K H*›
**using** *hintikka* **unfolding** $prop_H$*-def* **by** *blast*

**end**

**context** *Maximal-Consistency*
**begin**

**theorem** *mk-mcs-Hintikka*:
**assumes** ‹$prop_E$ *Ks C*› ‹*S* ∈ *C*› ‹*enough-new S*›
**shows** ‹*Hintikka map-fm params-fm is-param Ks* (*mk-mcs C S*)›
**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) ⟹ $sat_E$ *K* {*A*. *enough-new A* ∧ ⊢ *A*}›

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

**theorem** *Consistency*: ‹$prop_E$ *Ks* {*A*. *enough-new A* ∧ ⊢ *A*}›
**using** *all-consistent inf-univ* **unfolding** $prop_E$*-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*) ⟹ $sat_E$ *K* {*S*. ∃ *A*. *set A* = *S* ∧ ⊢ *A*}›

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

## 1.7 Conflicts

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

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

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

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

**declare** *hint.simps*[*simp*]

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

**end**

**sublocale** *Confl* ⊆ *Consistency-Kind map-fm params-fm is-param kind*
**proof**
  **fix** *C*
  **assume** *conflC*: ‹$sat_E$ kind C›
  **then show** ‹$sat_E$ 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›
    **then have** ∗: ‹set ps ⊆ S'›
      **using** ‹S ⊆ S'› **by** *blast*

    **assume** ∗∗: ‹ps ⇝✗ qs›
    **then have** ‹∀ q ∈ set qs. q ∉ S'›
      **using** *conflC S'* ∗ **by** *blast*
    **then have** ‹∀ q ∈ set qs. q ∉ S›
      **using** ‹S ⊆ S'› **by** *blast*
    **moreover assume** ‹q ∈ set qs› ‹q ∈ S›

```
    ultimately show ‹q ∈ {}›
      using ∗∗ by auto
  qed
next
  fix C
  assume conflC: ‹sat_E kind C›
  then show ‹sat_A kind (mk-alt-consistency C)›
  proof safe
    fix S ps qs q

    assume ‹S ∈ mk-alt-consistency C›
    then obtain f where 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 ↝✗ qs›
    then have ‹map (map-fm f) ps ↝✗ (map (map-fm f) qs)›
      using confl-map by blast
    then have ‹∀ q ∈ set  (map (map-fm f) qs). q ∉ ?S›
      using conflC f ∗ by blast
    then have ‹∀ q ∈ set qs. map-fm f q ∉ ?S›
      by simp
    then have ‹∀ q ∈ set qs. q ∉ S›
      by blast
    moreover assume ‹q ∈ set qs› ‹q ∈ S›
    ultimately show ‹q ∈ {}›
      by auto
  qed
next
  fix C
  assume conflAC: ‹sat_A kind C› and closedC: ‹subset-closed C›
  then show ‹sat_A kind (mk-finite-char C)›
  proof safe
    fix S ps qs q
    assume ‹S ∈ mk-finite-char C›
    then have finc: ‹∀ 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*› ‹*ps* ↝✗ *qs*› ‹*q* ∈ *set qs*› ‹*q* ∈ *S*›
**then have** ‹{*q*} ∪ *set ps* ∈ *C*›
**using** ‹*set ps* ⊆ *S*› *finc* **by** *simp*
**then have** ‹*q* ∈ {}›
**using** ∗ *conflAC* **by** *blast*
**then show** ‹*q* ∈ {}›
**by** *auto*
**qed**
**next**
**fix** *C S*
**assume** ∗: ‹*sat*$_E$ *kind C*› ‹*S* ∈ *C*› ‹*maximal C S*›
**show** ‹*sat*$_H$ *kind S*›
**proof** *safe*
**fix** *ps qs q*
**assume** ∗∗: ‹*set ps* ⊆ *S*› ‹*ps* ↝✗ *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** ‹↝✗› 50) +
**fixes** *consistent* :: ‹'*fm set* ⇒ *bool*› (‹⊢ -› [51] 50)
**assumes** *consistent*: ‹⋀*S ps qs x*. *set ps* ⊆ *S* ⟹ *ps* ↝✗ *qs* ⟹ *x* ∈ *set qs* ⟹ *x* ∈ *S* ⟹ ¬ ⊢ *S*›

**sublocale** *Derivational-Confl* ⊆ *Derivational-Kind map-fm params-fm is-param kind consistent*
**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** ‹↝✗› 50) +
**fixes** *consistent* :: ‹'*fm list* ⇒ *bool*› (‹⊢ -› [51] 50)
**assumes** *consistent*: ‹⋀*A ps qs x*. *set ps* ⊆ *set A* ⟹ *ps* ↝✗ *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** ‹⇝$_\alpha$›  50)
  **assumes** *alpha-map*: ‹⋀ps qs f. ps ⇝$_\alpha$ qs ⟶ map (map-fm f) ps ⇝$_\alpha$ map (map-fm f) qs›
**begin**

**inductive** *cond* **where**
  *cond* [*intro*!]: ‹ps ⇝$_\alpha$ 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 ⇝$_\alpha$ qs ⟹ set ps ⊆ H ⟹ q ∈ set qs ⟹ q ∈ H) ⟹ hint H›

**declare** *hint.simps*[*simp*]

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

**end**

**sublocale** *Alpha* ⊆ *Consistency-Kind map-fm params-fm is-param kind*
**proof**
  **fix** *C*
  **assume** *alphaC*: ‹sat$_E$ kind C›
  **then show** ‹sat$_E$ 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 ⇝$_\alpha$ qs›
    **then have** ∗: ‹set ps ⊆ S'›
      **using** ‹S ⊆ S'› **by** *blast*

    **assume** ∗∗: ‹ps ⇝$_\alpha$ 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: ‹sat_E kind C›
  then show ‹sat_A 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: ‹sat_A kind C› and closedC: ‹subset-closed C›
  then show ‹sat_A kind (mk-finite-char C)›
  proof safe
    fix S ps qs q
    assume ‹S ∈ mk-finite-char C›
    then have finc: ‹∀ 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** ‹$S' \subseteq$ *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** ‹*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** $*$: ‹$sat_E$ *kind* $C$› ‹$S \in C$› ‹*maximal* $C$ $S$›
**show** ‹$sat_H$ *kind* $S$›
**proof** *safe*
**fix** *ps qs q*
**assume** $**$: ‹*set ps* $\subseteq$ $S$› ‹*ps* $\rightsquigarrow_\alpha$ *qs*›
**then have** ‹*set qs* $\cup$ $S \in C$›
**using** $*$ **by** *blast*
**moreover assume** ‹$q \in$ *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 set*› **and**
*is-param* :: ‹*'x* $\Rightarrow$ *bool*› **and**
*classify* :: ‹*'fm list* $\Rightarrow$ *'fm list* $\Rightarrow$ *bool*› (**infix** ‹$\rightsquigarrow_\alpha$› *50*) +
**fixes** *consistent* :: ‹*'fm set* $\Rightarrow$ *bool*› (‹$\vdash$ -› [*51*] *50*)
**assumes** *consistent*: ‹$\bigwedge S$ *ps qs*. *set ps* $\subseteq S \Longrightarrow$ *ps* $\rightsquigarrow_\alpha$ *qs* $\Longrightarrow \vdash S \Longrightarrow \vdash$ *set qs* $\cup$ $S$›

**sublocale** *Derivational-Alpha* $\subseteq$ *Derivational-Kind map-fm params-fm is-param kind consistent*
**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 set*› **and**
*is-param* :: ‹*'x* $\Rightarrow$ *bool*› **and**
*classify* :: ‹*'fm list* $\Rightarrow$ *'fm list* $\Rightarrow$ *bool*› (**infix** ‹$\rightsquigarrow_\alpha$› *50*) +

**fixes** *consistent* :: ‹*'fm list ⇒ bool*› (‹⊢ -› [*51*] *50*)
**assumes** *consistent*: ‹⋀*A ps qs*. *set ps* ⊆ *set A* ⟹ *ps* ⇝$_\alpha$ *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* ⇝$_\alpha$ *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** ‹⇝$_\beta$› *50*)
**assumes** *beta-map*: ‹⋀*ps qs f*. *ps* ⇝$_\beta$ *qs* ⟶ *map* (*map-fm f*) *ps* ⇝$_\beta$ *map* (*map-fm f*) *qs*›
**begin**

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

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

**inductive** *hint* **where**
*hint* [*intro*!]: ‹(⋀*ps qs*. *ps* ⇝$_\beta$ *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 ‹S ∈ close C›
    then obtain S' where S': ‹S' ∈ C› and ‹S ⊆ S'›
      unfolding close-def by blast

    assume ‹set ps ⊆ S› ‹ps ↝β qs›
    then have ∗: ‹set ps ⊆ S'›
      using ‹S ⊆ S'› by blast

    assume ∗∗: ‹ps ↝β qs›
    then have ‹∃ q ∈ set qs. {q} ∪ S' ∈ C›
      using betaC S' ∗ by blast
    then show ‹∃ q ∈ set qs. insert q S ∈ close C›
      using ‹S ⊆ S'› subset-in-close by (metis insert-is-Un)
  qed
next
  fix C
  assume betaC: ‹sat_E kind C›
  then show ‹sat_A 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 ↝β qs›
    then have ‹map (map-fm f) ps ↝β (map (map-fm f) qs)›
      using beta-map by blast
    then have ‹∃ q ∈ set (map (map-fm f) qs). {q} ∪ ?S ∈ C›
      using betaC ∗ f by blast
    then show ‹∃ q ∈ set qs. insert q S ∈ ?C›
      unfolding mk-alt-consistency-def using f by (auto simp: image-Un)
  qed
next
  fix C
  assume betaAC: ‹sat_A kind C› and closedC: ‹subset-closed C›
  then show ‹sat_A kind (mk-finite-char C)›
  proof safe
    fix S ps qs q
    assume ‹S ∈ mk-finite-char C›
```

```
    then have finc: ‹∀ 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 ⇝β qs›

    show ‹∃ q ∈ set qs. insert q S ∈ mk-finite-char C›
    proof (rule ccontr)
      assume ‹¬ ?thesis›
      then have ‹∀ q ∈ set qs. ∃ Sq. Sq ⊆ insert q S ∧ finite Sq ∧ Sq ∉ C›
        unfolding mk-finite-char-def by blast
      then obtain f where f: ‹∀ q ∈ set qs. f q ⊆ insert q S ∧ finite (f q) ∧ f q ∉
C›
        by metis

      let ?S' = ‹set ps ∪ ⋃{f q − {q} |q. q ∈ set qs}›

      have ‹?S' ⊆ S›
        using ∗ f by fast
      moreover have ‹finite ?S'›
        using f by auto
      ultimately have ‹?S' ∈ C›
        using finc by blast
      then have ‹∃ q ∈ set qs. {q} ∪ ?S' ∈ C›
        using ∗∗ betaAC by fast
      then have ‹∃ q ∈ set qs. f q ∈ C›
        using sc' by blast
      then show False
        using f by blast
    qed
  qed
next
  fix C S
  assume ∗: ‹sat_E kind C› ‹S ∈ C› ‹maximal C S›
  show ‹sat_H kind S›
  proof safe
    fix ps qs
    assume ∗∗: ‹set ps ⊆ S› ‹ps ⇝β qs›
    then have ‹∃ q ∈ set qs. {q} ∪ S ∈ C›
      using ∗ by blast
    then show ‹∃ q ∈ set qs. q ∈ S›
      using ‹maximal C S› unfolding maximal-def by fast
  qed
qed
```

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

**sublocale** *Derivational-Beta* ⊆ *Derivational-Kind map-fm params-fm is-param kind consistent*
**proof**
**assume** *inf*: ‹*infinite* (*UNIV* :: 'fm set)›
**then show** ‹*sat*$_E$ *kind* {*A*. *enough-new A* ∧ ⊢ *A*}›
**proof** *safe*
**fix** *S ps qs*
**assume** ∗: ‹*set ps* ⊆ *S*› ‹*ps* ⇝$_\beta$ *qs*› ‹⊢ *S*›
**then have** ‹∃ *q* ∈ *set qs*. ⊢ {*q*} ∪ *S*›
**using** *consistent* **by** *blast*
**moreover assume** ‹*enough-new S*›
**ultimately show** ‹∃ *q*∈*set qs*. *insert q S* ∈ {*A*. *enough-new A* ∧ ⊢ *A*}›
**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* :: ‹('x ⇒ 'x) ⇒ 'fm ⇒ 'fm› **and**
*params-fm* :: ‹'fm ⇒ 'x set› **and**
*is-param* :: ‹'x ⇒ bool› **and**
*classify* :: ‹'fm list ⇒ 'fm list ⇒ bool› (**infix** ‹⇝$_\beta$› 50) +
**fixes** *consistent* :: ‹'fm list ⇒ bool› (‹⊢ -› [51] 50)
**assumes** *consistent*: ‹⋀A ps qs. set ps ⊆ set A ⟹ ps ⇝$_\beta$ qs ⟹ ⊢ A ⟹ ∃ q ∈ set qs. ⊢ q # A›

**sublocale** *Weak-Derivational-Beta* ⊆ *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* ⇝$_\beta$ *qs*› ‹⊢ *A*›
**then have** ‹∃ *q* ∈ *set qs*. ⊢ *q* # *A*›
**using** *consistent* **by** *blast*
**then show** ‹∃ *q*∈*set qs*. *insert q* (*set A*) ∈ {*S*. ∃ *A*. *set A* = *S* ∧ ⊢ *A*}›
**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** ‹$\rightsquigarrow_\gamma$› 50)
  **assumes** *gamma-map*: ‹$\bigwedge$ps F qs f. ps $\rightsquigarrow_\gamma$ (F, qs) ⟹ (∃ G rs. map (map-fm f) ps $\rightsquigarrow_\gamma$ (G, rs) ∧
      (∀ S. map-tm f ‘ F S ⊆ G (map-fm f ‘ S)) ∧
      (∀ t. map (map-fm f) (qs t) = rs (map-tm f t)))›
    **and** *gamma-mono*: ‹$\bigwedge$ps F qs S S'. ps $\rightsquigarrow_\gamma$ (F, qs) ⟹ S ⊆ S' ⟹ F S ⊆ F S'›
    **and** *gamma-fin*: ‹$\bigwedge$ps F qs t A. ps $\rightsquigarrow_\gamma$ (F, qs) ⟹ t ∈ F A ⟹ ∃ B ⊆ A. finite B ∧ t ∈ F B›
**begin**

**inductive** *cond* **where**
  *cond* [*intro*!]: ‹ps $\rightsquigarrow_\gamma$ (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*!]: ‹($\bigwedge$ps F qs. ps $\rightsquigarrow_\gamma$ (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*: ‹$sat_E$ kind C›
  **then show** ‹$sat_E$ 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 ‹set ps ⊆ S›
    then have ∗: ‹set ps ⊆ S'›
      using ‹S ⊆ S'› by blast

    assume ∗∗: ‹ps ⇝γ (F, qs)› ‹t ∈ F S›
    then have ‹t ∈ F S'›
      using ∗∗ gamma-mono ‹S ⊆ S'› by blast
    then have ‹set (qs t) ∪ S' ∈ C›
      using gammaC S' ∗ ∗∗ by blast
    then have ‹set (qs t) ∪ S ∈ close C›
      using ‹S ⊆ S'› subset-in-close by blast
    then show ‹set (qs t) ∪ S ∈ close C›
      by (meson close-closed equalityE subset-closed-def sup.mono)
  qed
next
  fix C
  assume gammaC: ‹sat_E kind C›
  then show ‹sat_A kind (mk-alt-consistency C)›
  proof safe
    fix S ps F qs t

    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 ⇝γ (F, qs)› and t: ‹t ∈ F S›
    obtain rs G where 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)›
      using gamma-map ∗∗ by meson
    then have ‹set (rs (map-tm f t)) ∪ ?S ∈ C›
      using gammaC f ∗ t by blast
    then have ‹set (map (map-fm f) (qs t)) ∪ ?S ∈ C›
      using rs by simp
    then show ‹set (qs t) ∪ S ∈ ?C›
      unfolding mk-alt-consistency-def using f by (auto simp: image-Un)
  qed
next
  fix C
  assume gammaAC: ‹sat_A kind C› and closedC: ‹subset-closed C›
  then show ‹sat_A kind (mk-finite-char C)›
```

```
  proof safe
    fix S ps F qs t
    assume ‹S ∈ mk-finite-char C›
    then have finc: ‹∀ 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 ⇝_γ (F, qs)› and t: ‹t ∈ F S›

    show ‹set (qs t) ∪ S ∈ mk-finite-char C›
      unfolding mk-finite-char-def
    proof safe
      fix S'
      assume 1: ‹S' ⊆ set (qs t) ∪ S› and 2: ‹finite S'›

      obtain A where A: ‹A ⊆ S› ‹finite A› ‹S' ⊆ set (qs t) ∪ A›
        using 1 2 by (meson Diff-subset-conv equalityD2 finite-Diff)

      obtain B where B: ‹B ⊆ S› ‹finite B› ‹t ∈ F B›
        using ∗∗ t gamma-fin by meson

      let ?S = ‹set ps ∪ A ∪ B›

      have ‹finite ?S›
        using A(2) B(2) by blast
      moreover have ‹?S ⊆ S›
        using ∗ A(1) B(1) by simp
      ultimately have ‹?S ∈ C›
        using finc by simp

      moreover have ‹set ps ⊆ ?S›
        by blast
      moreover have ‹t ∈ F ?S›
        using ∗∗ B(3) gamma-mono by blast
      ultimately have ‹set (qs t) ∪ ?S ∈ C›
        using ∗∗ gammaAC by blast

      moreover have ‹S' ⊆ set (qs t) ∪ ?S›
        using A(3) by blast
      ultimately show ‹S' ∈ C›
        using sc by blast
    qed
  qed
next
  fix C S
```

**assume** $*$: ‹$sat_E$ *kind* $C$› ‹$S \in C$› ‹*maximal* $C$ $S$›
**show** ‹$sat_H$ *kind* $S$›
**proof** *safe*
**fix** *ps* $F$ *qs* $t$
**assume** $**$: ‹*set ps* $\subseteq S$› ‹*ps* $\rightsquigarrow_\gamma$ ($F$, *qs*)›
**then have** ‹$\forall\, t \in F\ S.$ *set* (*qs* $t$) $\cup$ $S \in C$›
**using** $*$ **by** *blast*
**then have** ‹$\forall\, t \in F\ S.$ *set* (*qs* $t$) $\subseteq S$›
**using** ‹*maximal* $C$ $S$› **unfolding** *maximal-def* **by** *fast*
**then show** ‹$t \in F\ S \Longrightarrow x \in$ *set* (*qs* $t$) $\Longrightarrow x \in S$› **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** ‹$\rightsquigarrow_\gamma$› *50*) +
**fixes** *consistent* :: ‹'*fm set* ⇒ *bool*› (‹⊢ -› [*51*] *50*)
**assumes** *consistent*: ‹$\bigwedge S$ *ps* $F$ *qs* $t$. *set ps* $\subseteq S \Longrightarrow$ *ps* $\rightsquigarrow_\gamma$ ($F$, *qs*) $\Longrightarrow t \in F\ S \Longrightarrow \vdash S \Longrightarrow \vdash$ *set* (*qs* $t$) $\cup$ $S$›

**sublocale** *Derivational-Gamma* ⊆ *Derivational-Kind map-fm params-fm is-param kind consistent*
**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** ‹$\rightsquigarrow_\gamma$› *50*) +
**fixes** *consistent* :: ‹'*fm list* ⇒ *bool*› (‹⊢ -› [*51*] *50*)
**assumes** *consistent*: ‹$\bigwedge A$ *ps* $F$ *qs* $t$. *set ps* $\subseteq$ *set* $A \Longrightarrow$ *ps* $\rightsquigarrow_\gamma$ ($F$, *qs*) $\Longrightarrow t \in F$ (*set* $A$) $\Longrightarrow \vdash A \Longrightarrow \vdash$ *qs* $t$ @ $A$›

**sublocale** *Weak-Derivational-Gamma* ⊆ *Weak-Derivational-Kind map-fm params-fm is-param kind consistent*
**proof**
**show** ‹$sat_E$ *kind* {$S.$ ∃ $A.$ *set* $A = S \wedge \vdash A$}›
**proof** *safe*

**fix** *ps qs A F t*
**assume** ∗: ‹*set ps* ⊆ *set A*› ‹*ps* ⇝$_\gamma$ (*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** ‹⇝$_\gamma$''› 50)
**assumes** *gamma-map-UNIV*: ‹⋀*ps qs f*. *ps* ⇝$_\gamma$' *qs* ⟹ ∃ *rs*. *map* (*map-fm f*) *ps* ⇝$_\gamma$' *rs* ∧
(∀ *t*. *map* (*map-fm f*) (*qs t*) = *rs* (*map-tm f t*))›
**begin**

**abbreviation** (*input*) *classify-UNIV* **where**
‹*classify-UNIV* ≡ λ*ps* (*F*, *qs*). (*F* = (λ-. *UNIV*)) ∧ *ps* ⇝$_\gamma$' *qs*›

**end**

**sublocale** *Gamma-UNIV* ⊆ *Gamma map-tm map-fm params-fm is-param classify-UNIV*
**proof**
**show** ‹⋀*ps F qs f*.
(*case* (*F*, *qs*) *of* (*F*, *qs*) ⇒ *F* = (λ-. *UNIV*) ∧ *ps* ⇝$_\gamma$' *qs*) ⟹
∃ *G rs*.
(*case* (*G*, *rs*) *of* (*F*, *qs*) ⇒ *F* = (λ-. *UNIV*) ∧ *map* (*map-fm f*) *ps* ⇝$_\gamma$' *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* ⇝$_\gamma$' *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* ⇝$_\gamma$' *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: ‹sat_E kind C›
  then show ‹sat_E 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)

  qed
next
  fix C
  assume deltaC: ‹sat_E kind C›
  then show ‹sat_A 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: ‹is-param y› ‹set (δ (map-fm f p) y) ∪ ?S ∈ C›
      using deltaC f ∗ by blast

    let ?g = ‹f(x := y)›

    have g: ‹is-subst ?g›
      using f x y unfolding is-subst-def by simp

    have ‹∀ x ∈ params S. f x = ?g x›
      using x by simp
    then have S: ‹∀ q ∈ S. map-fm ?g q = map-fm f q›
      using map-params-fm by simp
    then have ‹map-fm ?g p = map-fm f p›
      using ‹p ∈ S› by auto

    moreover have ‹set (δ (map-fm f p) (?g x)) ∪ ?S ∈ C›
      using y by simp
    ultimately have ‹set (map (map-fm ?g) (δ p x)) ∪ ?S ∈ C›
      using delta-map x g by metis
    then have ‹∃f. is-subst f ∧ set (map (map-fm f) (δ p x)) ∪ map-fm f ‘ S ∈
C›
      using S g by (metis image-cong)
    then show ‹set (δ p x) ∪ S ∈ ?C›
      unfolding mk-alt-consistency-def
      by (metis (mono-tags, lifting) image-Un image-set mem-Collect-eq)
  qed
next
  fix C
  assume deltaAC: ‹sat_A kind C› and closedC: ‹subset-closed C›
  then show ‹sat_A kind (mk-finite-char C)›
  proof safe
    fix S p qs x
    assume ‹S ∈ mk-finite-char C›
    then have finc: ‹∀ 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 ∗: ‹p ∈ S› and x: ‹x ∉ params S› ‹is-param x›
    show ‹set (δ p x) ∪ S ∈ mk-finite-char C›
      unfolding mk-finite-char-def
    proof safe
      fix S'
      let ?S' = ‹{p} ∪ (S' − set (δ p x))›

      assume ‹S' ⊆ set (δ p x) ∪ S› and ‹finite S'›
```

**then have** ‹*?S′* ⊆ *S*›
**using** ∗ **by** *fast*
**moreover have** ‹*finite ?S′*›
**using** ‹*finite S′*› **by** *blast*
**ultimately have** ‹*?S′* ∈ *C*›
**using** *finc* **by** *blast*
**moreover have** ‹∀ *a* ∈ *?S′*. *x* ∉ *params-fm a*›
**using** *x* ‹*?S′* ⊆ *S*› **by** *blast*
**ultimately have** ‹*set* (*δ p x*) ∪ *?S′* ∈ *C*›
**using** *x deltaAC* **by** *blast*
**then show** ‹*S′* ∈ *C*›
**using** *sc* **by** *fast*
**qed**
**qed**
**next**
**show** ‹⋀*C S*. *sat*$_E$ *kind C* ⟹ *S* ∈ *C* ⟹ *maximal C S* ⟹ *sat*$_H$ *kind S*›
**using** *sat*$_H$*-Wits* **.**
**qed**

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

**sublocale** *Derivational-Delta* ⊆ *Derivational-Kind map-fm params-fm is-param kind consistent*
**proof**
**assume** *inf*: ‹*infinite* (*UNIV* :: *'fm set*)›
**show** ‹*sat*$_E$ *kind* {*A*. *enough-new A* ∧ ⊢ *A*}›
**proof** *safe*
**fix** *S p*
**assume** ∗: ‹*p* ∈ *S*› ‹*enough-new S*› ‹⊢ *S*›
**then have** ‹*infinite* (*Collect is-param* − (*params* ({*p*} ∪ *S*)))›
**unfolding** *enough-new-def* **using** *card-of-ordLeq-finite inf* **by** *auto*
**then obtain** *x* **where** ‹*is-param x*› ‹*x* ∉ *params* ({*p*} ∪ *S*)›
**using** *infinite-imp-nonempty* **by** *blast*
**then have** ‹∃ *x*. *is-param x* ∧ ⊢ *set* (*δ p x*) ∪ *S*›
**using** ∗(*1*,*3*) *consistent* ‹⊢ *S*› **by** *fast*
**then show** ‹∃ *x*. *is-param x* ∧ *set* (*δ p x*) ∪ *S* ∈ {*A*. *enough-new A* ∧ ⊢ *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 δ*

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

**sublocale** *Weak-Derivational-Delta* ⊆ *Weak-Derivational-Kind map-fm params-fm is-param kind consistent*
**proof**
**assume** *inf*: ‹*infinite* (*Collect is-param* :: 'x *set*)›
**show** ‹$sat_E$ *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** ‹⇝$_\Box$› 50)
**and** *hint* :: ‹'fm set ⇒ bool›
**assumes** *modal-map*: ‹⋀ps F qs f. ps ⇝$_\Box$ (F, qs) ⟹ ∃ G. map (map-fm f) ps ⇝$_\Box$ (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 ⇝$_\Box$ (F, qs) ⟹ S ⊆ S' ⟹ F S ⊆ F S'›
**and** *modal-fin*: ‹⋀ps F qs S A. ps ⇝$_\Box$ (F, qs) ⟹ finite A ⟹ A ⊆ F S ⟹ ∃ S' ⊆ S. finite S' ∧ A ⊆ F S'›
**begin**

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

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

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

**end**

**locale** *ModalH* = *Modal map-fm params-fm is-param classify hint*
**for**
*map-fm* :: ‹('*x* $\Rightarrow$ '*x*) $\Rightarrow$ '*fm* $\Rightarrow$ '*fm*› **and**
*params-fm* :: ‹'*fm* $\Rightarrow$ '*x set*› **and**
*is-param* :: ‹'*x* $\Rightarrow$ *bool*› **and**
*classify* :: ‹'*fm list* $\Rightarrow$ ('*fm set* $\Rightarrow$ '*fm set*) $\times$ '*fm list* $\Rightarrow$ *bool*› (**infix** ‹$\rightsquigarrow_{\Box}$› *50*)
**and**
*hint* :: ‹'*fm set* $\Rightarrow$ *bool*› +
**assumes** *modal-hintikka*: ‹$\bigwedge$*C S*. *sat*$_E$ *kind C* $\Longrightarrow$ *S* $\in$ *C* $\Longrightarrow$ *maximal C S* $\Longrightarrow$ *sat*$_H$ *kind S*›

**sublocale** *ModalH* $\subseteq$ *Consistency-Kind map-fm params-fm is-param kind*
**proof**
**fix** *C*
**assume** *modalC*: ‹*sat*$_E$ *kind C*›
**then show** ‹*sat*$_E$ *kind* (*close C*)›
**proof** *safe*
**fix** *S ps F qs*
**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* $\rightsquigarrow_{\Box}$ (*F*, *qs*)›
**then have** ‹*set qs* $\cup$ *F S'* $\in$ *C*›
**using** *modalC S'* $*$ $**$ **by** *blast*
**then show** ‹*set qs* $\cup$ *F S* $\in$ *close C*›
**using** ‹*S* $\subseteq$ *S'*› *subset-in-close* $**$ *modal-mono* **by** *metis*
**qed**
**next**
**fix** *C*
**assume** *modalC*: ‹*sat*$_E$ *kind C*› **and** *closedC*: ‹*subset-closed C*›
**then show** ‹*sat*$_A$ *kind* (*mk-alt-consistency C*)›
**proof** *safe*
**fix** *S ps F 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 ⇝□ (F, qs)›
    obtain G where G:
      ‹map (map-fm f) ps ⇝□ (G, map (map-fm f) qs)›
      ‹∀ S. map-fm f ‘ F S ⊆ G (map-fm f ‘ S)›
      using modal-map ∗∗ by meson
    then have ‹set (map (map-fm f) qs) ∪ G ?S ∈ C›
      using modalC f ∗ by blast
    then have ‹set (map (map-fm f) qs) ∪ map-fm f ‘ F S ∈ C›
     using G closedC unfolding subset-closed-def by (meson Un-mono order-refl)
    then show ‹set qs ∪ F S ∈ ?C›
      unfolding mk-alt-consistency-def using f
      by (auto split: option.splits simp: image-Un)
  qed
next
  fix C
  assume modalAC: ‹sat_A kind C› and closedC: ‹subset-closed C›
  then show ‹sat_A kind (mk-finite-char C)›
  proof safe
    fix S ps F qs t
    assume S: ‹S ∈ mk-finite-char C›
    then have finc: ‹∀ 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 ⇝□ (F, qs)›

    show ‹set qs ∪ F S ∈ mk-finite-char C›
      unfolding mk-finite-char-def
    proof safe
      fix S'
      assume 1: ‹S' ⊆ set qs ∪ F S› and 2: ‹finite S'›

      obtain A where A: ‹A ⊆ S› ‹finite A› ‹S' ⊆ set qs ∪ F A›
```

**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** *finc* **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** ∗: ‹*sat*$_E$ *kind C*› ‹*S* ∈ *C*› ‹*maximal C S*›
**then show** ‹*sat*$_H$ *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** ‹⇝$_\Box$› *50*)
+
**fixes** *consistent* :: ‹'*fm set* ⇒ *bool*› (‹⊢ -› [*51*] *50*)
**assumes** *consistent*: ‹⋀*S ps F qs*. *set ps* ⊆ *S* ⟹ *ps* ⇝$_\Box$ (*F*, *qs*) ⟹ ⊢ *S* ⟹ ⊢ *set qs* ∪ *F S*›
**and** *params-subset*: ‹⋀*ps F qs S*. *ps* ⇝$_\Box$ (*F*, *qs*) ⟹ *params* (*F S*) ⊆ *params S*›

**sublocale** *Derivational-Modal* ⊆ *Derivational-Kind map-fm params-fm is-param kind consistent*
**proof**
**assume** *inf*: ‹*infinite* (*UNIV* :: '*fm set*)›
**then show** ‹*sat*$_E$ *kind* {*A*. *enough-new A* ∧ ⊢ *A*}›
**proof** *safe*

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

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

**sublocale** *Weak-Derivational-Modal* ⊆ *Weak-Derivational-Kind map-fm params-fm is-param kind consistent*
**proof**
**assume** *inf*: ‹*infinite* (*Collect is-param* :: '*x set*)›
**show** ‹$sat_E$ *kind* {*S*. ∃ *A*. *set A* = *S* ∧ ⊢ *A*}›
**proof** *safe*
**fix** *ps qs A F*
**assume** ‹*set ps* ⊆ *set A*› ‹*ps* $\rightsquigarrow_{\Box}$ (*F*, *qs*)› ‹⊢ *A*›
**then show** ‹∃ *B*. *set B* = *set qs* ∪ *F* (*set A*) ∧ ⊢ *B*›
**using** *consistent*[*of ps A F qs*] *F-size*
**by** (*metis card-of-ordLeq-infinite finite-list list.set-finite set-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* :: ‹('a, 'f, 'p) *model* ⇒ *bool*› **where**
  ‹*wf-model* (*Model U E F G*) ⟷ (∀ *n*. *E n* ∈ *U*) ∧ (∀ *f ts*. *F f ts* ∈ *U*)›

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

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

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

## 2.3 Operations

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

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

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

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

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

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

### 2.3.1 Lemmas

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

**lemma** *size-sub-fm* [*simp*]: ‹*size-fm* (*sub-fm s p*) = *size-fm p*›
**by** (*induct p arbitrary*: *s*) *simp-all*

**lemma** *upd-params-tm* [*simp*]: ‹*f* ∉ *params-tm t* ⟹ ⦇(*E*, *F*(*f* := *x*))⦈ *t* = ⦇(*E*, *F*)⦈ *t*›
**by** (*induct t*) (*auto cong*: *map-cong*)

**lemma** *upd-params-fm* [*simp*]:
**assumes** ‹*f* ∉ *params-fm p*›
**shows** ‹*Model U E* (*F*(*f* := *x*)) *G* ⊨ *p* ⟷ *Model U E F G* ⊨ *p*›
**using** *assms* **by** (*induct p arbitrary*: *E*) (*auto cong*: *map-cong*)

**lemma** *finite-params-tm* [*simp*]: ‹*finite* (*params-tm t*)›
**by** (*induct t*) *simp-all*

**lemma** *finite-params-fm′* [*simp*]: ‹*finite* (*params-fm p*)›
**by** (*induct p*) *simp-all*

**lemma** *env* [*simp*]: ‹*P* ((*x* ≫ *E*) *n*) = (*P x* ≫ λ*n*. *P* (*E n*)) *n*›
**by** (*induct n*) *simp-all*

**lemma** *lift-lemma*: ‹⦇(*x* ≫ *E*, *F*)⦈ (*lift-tm t*) = ⦇(*E*, *F*)⦈ *t*›
**by** (*induct t*) (*auto cong*: *map-cong*)

**lemma** *sub-tm-semantics*: ‹⦇(*E*, *F*)⦈ (*sub-tm s t*) = ⦇(λ*n*. ⦇(*E*, *F*)⦈ (*s n*), *F*)⦈ *t*›
**by** (*induct t*) (*auto cong*: *map-cong*)

**lemma** *sub-fm-semantics* [*simp*]:
‹*Model U E F G* ⊨ *sub-fm s p* ⟷ *Model U* (λ*n*. ⦇(*E*, *F*)⦈ (*s n*)) *F G* ⊨ *p*›
**by** (*induct p arbitrary*: *E s*) (*auto cong*: *map-cong simp*: *sub-tm-semantics lift-lemma*)

**lemma** *map-tm-sub-tm* [*simp*]: ‹*map-tm f* (*sub-tm g t*) = *sub-tm* (*map-tm f o g*) (*map-tm f t*)›
**by** (*induct t*) *simp-all*

**lemma** *map-tm-lift-tm* [*simp*]: ‹*map-tm f* (*lift-tm t*) = *lift-tm* (*map-tm f t*)›
**by** (*induct t*) *simp-all*

**lemma** *psub-sub-fm*: ‹*psub f* (*sub-fm g p*) = *sub-fm* (*map-tm f o g*) (*psub f p*)›
**by** (*induct p arbitrary*: *g*) (*simp-all add*: *comp-def*)

**lemma** *map-tm-inst-single*: ‹(*map-tm f o* (*u* ≫ #)) *t* = (*map-tm f u* ≫ #) *t*›
**by** (*induct t*) *auto*

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

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

## 2.4 Terms

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

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

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

### 2.4.1 Lemmas

**lemma** *terms-mono*: ‹*S* ⊆ *S'* ⟹ *terms S* ⊆ *terms S'*›
  **unfolding** *terms-def* **by** *blast*

**lemma** *terms-tm-refl* [*intro*]: ‹*t* ∈ *terms-tm t*›
  **by** (*induct t*) *simp-all*

**lemma** *terms-tm-trans* [*trans*]: ‹*s* ∈ *terms-tm t* ⟹ *r* ∈ *terms-tm s* ⟹ *r* ∈ *terms-tm t*›
  **by** (*induct t*) *auto*

**lemma** *map-terms-tm* [*simp*]: ‹*terms-tm* (*map-tm f t*) = *map-tm f* ‘ *terms-tm t*›
  **by** (*induct t*) *auto*

**lemma** *map-terms-fm* [*simp*]: ‹*terms-fm* (*map-fm f g p*) = *map-tm f* ‘ *terms-fm p*›
  **by** (*induct p*) *auto*

**lemma** *terms-tm-closed* [*dest*]: ‹*t* ∈ *terms-tm s* ⟹ *terms-tm t* ⊆ *terms-tm s*›
  **using** *terms-tm-trans* **by** *blast*

**lemma** *terms-fm-closed*: ‹*t* ∈ *terms-fm p* ⟹ *terms-tm t* ⊆ *terms-fm p*›
  **by** (*induct p*) *auto*

**lemma** *terms-source*: ‹*t* ∈ *terms S* ⟹ ∃*S'* ⊆ *S*. *finite S'* ∧ *t* ∈ *terms S'*›
  **unfolding** *terms-def* **using** *insert-subset* **by** *blast*

**lemma** *terms-tm-Fun*: ‹⋀*f ts* ∈ *terms-tm s* ⟹ *t* ∈ *set ts* ⟹ *t* ∈ *terms-tm s*›
  **using** *terms-tm-refl terms-tm-trans*
  **by** (*metis UN-I UnI2 list.set-map terms-tm.simps*(*2*))

**lemma** *terms-Fun*: ‹⋀*f ts* ∈ *terms S* ⟹ *set ts* ⊆ *terms S*›

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

## 2.5 Guard

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

**lemma** *guard-in*: ‹*S* ≠ {} ⟹ (*x* ∈? *S*) ∈ *S*›
**unfolding** *guard-def* **using** *some-in-eq* **by** *auto*

**lemma** *guard-refl* [*simp*]: ‹*t* ∈ *S* ⟹ *t* ∈? *S* = *t*›
**unfolding** *guard-def* **by** *simp*

## 2.6 Model Existence

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

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

**interpretation** *P*: *Params psub params-fm* ‹λ-. *True*›
**by** *unfold-locales* (*auto simp*: *fm.map-id0 cong*: *fm.map-cong0*)

**interpretation** *C*: *Confl psub params-fm* ‹λ-. *True*› *confl-class*
**by** *unfold-locales* (*auto elim*!: *confl-class.cases*)

**interpretation** *A*: *Alpha psub params-fm* ‹λ-. *True*› *alpha-class*
**by** *unfold-locales* (*auto elim*!: *alpha-class.cases*)

**interpretation** *B*: *Beta psub params-fm* ‹λ-. *True*› *beta-class*
**by** *unfold-locales* (*auto elim*!: *beta-class.cases*)

**interpretation** *G*: *Gamma map-tm psub params-fm* ‹λ-. *True*› *gamma-class*
**proof**
**show** ‹⋀*ps F qs t A*. *ps* ⇝$_\gamma$ (*F*, *qs*) ⟹ *t* ∈ *F A* ⟹ ∃ *B* ⊆ *A*. *finite B* ∧ *t* ∈ *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* ‹λ-. *True*› *δ*
**proof**
 **show** ‹⋀*f*. *δ* (*psub f p*) (*f x*) = *map* (*psub f*) (*δ p x*)› **for** *p* :: ‹('*f*, '*p*) *fm*› **and** *x*
  **by** (*induct p x rule*: *δ.induct*) *simp-all*
**qed**

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

**lemma** *prop$_E$-Kinds* [*intro*]:
  **assumes** ‹*P.sat$_E$ C.kind C*› ‹*P.sat$_E$ A.kind C*› ‹*P.sat$_E$ B.kind C*› ‹*P.sat$_E$ G.kind C*› ‹*P.sat$_E$ D.kind C*›
 **shows** ‹*P.prop$_E$ Kinds C*›
 **unfolding** *P.prop$_E$-def* **using** *assms* **by** *simp*

**interpretation** *Consistency-Kinds psub params-fm* ‹λ-. *True*› *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* ‹λ-. *True*› *Kinds*
**proof**
 **show** ‹*infinite* (*UNIV* :: ('*f*, '*p*) *fm set*)›
  **using** *infinite-UNIV-size*[*of* ‹λ*p*. *p* ⟶ *p*›] **by** *simp*
**qed** *simp*

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

**lemma** *wf-canonical*:
 **assumes** ‹*terms H* ≠ {}›
 **shows** ‹*wf-model* (*canonical H*)›
 **using** *assms guard-in* **by** (*metis* (*no-types*, *lifting*) *wf-model.simps*)

**lemma** *canonical-tm-id* [*simp*]:
 ‹*t* ∈ *terms H* ⟹ ⦇(λ*n*. #*n* ∈*?* *terms H*, λ*P ts*. ◯*P ts* ∈*?* *terms H*)⦈ *t* = *t*›
**proof** (*induct t*)
 **case** (*Fun f ts*)
 **moreover from** *this* **have** ‹*t* ∈ *set ts* ⟹ *t* ∈ *terms H*› **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* ⦇(λ*n*. #*n* ∈*?* *terms H*, λ*P ts*. ○*P 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* = *sat*$_H$[*of C.kind*] **and**
  *alpha* = *sat*$_H$[*of A.kind*] **and**
  *beta* = *sat*$_H$[*of B.kind*] **and**
  *gamma* = *sat*$_H$[*of G.kind*] **and**
  *delta* = *sat*$_H$[*of D.kind*]

**theorem** *model*: ‹(*p* ∈ *H* ⟶ *canonical H* ⊨ *p*) ∧ (¬ *p* ∈ *H* ⟶ ¬ *canonical H* ⊨ *p*)›
**proof** (*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** *Fls*
    **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.prop_E 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.prop*$_H$ *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** ‹⊫› *50*) **where**
‹*M* ⊫ *S* ≡ ∀ *p* ∈ *S*. *M* ⊨ *p*›

**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* ⇝✗ *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*: ‹$P.sat_E$ *A.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** *qs* **and** $S$ :: ‹('$f$, '$p$) *fm set*›
**assume** ‹*P.enough-new* $S$›
**then show** ‹*P.enough-new* (*set qs* ∪ $S$)›
**using** *params-left* **by** *blast*
**next**
**fix** *ps qs* **and** $S$ $S'$ :: ‹('$f$, '$p$) *fm set*›
**assume** ‹*ps* ∼$_\alpha$ *qs*› **and** ∗: ‹*set ps* ⊆ $S$›
‹∀ $S'$⊆$S$. *finite* $S'$ ⟶ (∃($U$ :: '*a set*) $E$ $F$ $G$. *wf-model* (*Model* $U$ $E$ $F$ $G$) ∧
*Model* $U$ $E$ $F$ $G$ ⊨ $S'$)›
‹$S'$ ⊆ *set qs* ∪ $S$› ‹*finite* $S'$›
**then show** ‹∃($U$ :: '*a set*) $E$ $F$ $G$. *wf-model* (*Model* $U$ $E$ $F$ $G$) ∧ *Model* $U$ $E$ $F$
$G$ ⊨ $S'$›
**proof** *cases*
**case** (*CImpN* $p$ $q$)
**let** *?S* = ‹{¬ ($p$ ⟶ $q$)} ∪ ($S'$ − {$p$, ¬ $q$})›
**have** ‹*?S* ⊆ $S$› ‹*finite ?S*›
**using** *CImpN* ∗ **by** *fastforce+*
**then obtain** $U$ :: ‹'*a set*› **and** $E$ $F$ $G$ **where**
$M$: ‹*wf-model* (*Model* $U$ $E$ $F$ $G$)› ‹*Model* $U$ $E$ $F$ $G$ ⊨ *?S*›
**using** ∗ **by** *meson*
**then have** ‹*Model* $U$ $E$ $F$ $G$ ⊨ {$p$, ¬ $q$} ∪ *?S*›
**by** *auto*
**then show** *?thesis*
**using** $M(1)$ **by** *blast*
**qed**
**qed**

**lemma** *compact-B*: ‹$P.sat_E$ *B.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** *ps qs* **and** $S$ $S'$ :: ‹('$f$, '$p$) *fm set*›
**assume** ‹*ps* ∼$_\beta$ *qs*› **and** ∗: ‹*set ps* ⊆ $S$› ‹*P.enough-new* $S$›
‹∀ $S'$⊆$S$. *finite* $S'$ ⟶ (∃($U$ :: '*a set*) $E$ $F$ $G$. *wf-model* (*Model* $U$ $E$ $F$ $G$) ∧
*Model* $U$ $E$ $F$ $G$ ⊨ $S'$)›
**then show** ‹∃ *q*∈*set qs*. *insert q S* ∈ {$S$. *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** *cases*
**case** (*CImpP* $p$ $q$)
**then have** $P$: ‹∀ *q*∈*set qs*. {$q$} ∪ $S$ ∈ {$S$. *P.enough-new* $S$}›
**using** ∗ *params-left* **by** (*metis List.set-insert insert-is-Un list.set*(*1*) *mem-Collect-eq*)

**show** *?thesis*

**proof** (*rule ccontr*)
**assume** ‹¬ (∃ *q*∈*set qs*. *insert q S* ∈ {*S*. *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'*))})›
**then have** ‹∀ *q*∈*set qs*. ∃ *S'* ⊆ {*q*} ∪ *S*. *finite S'* ∧ ¬(∃ (*U* :: '*a set*) *E F G*. *wf-model* (*Model U E F G*) ∧ *Model U E F G* ⊨ *S'*)›
**using** *P* **by** *simp*
**then have** ‹∀ *q*∈*set qs*. ∃ *S'* ⊆ {*q*} ∪ *S*. *finite S'* ∧ (∀ (*U* :: '*a set*) *E F G*. *wf-model* (*Model U E F G*) ⟶ ¬ *Model U E F G* ⊨ *S'*)›
**by** *meson*
**then obtain** *Sp Sq* **where**
*Sp*: ‹*Sp* ⊆ {¬ *p*} ∪ *S*› ‹*finite Sp*› ‹∀ (*U* :: '*a set*) *E F G*. *wf-model* (*Model U E F G*) ⟶ ¬ *Model U E F G* ⊨ *Sp*› **and**
*Sq*: ‹*Sq* ⊆ {*q*} ∪ *S*› ‹*finite Sq*› ‹∀ (*U* :: '*a set*) *E F G*. *wf-model* (*Model U E F G*) ⟶ ¬ *Model U E F G* ⊨ *Sq*›
**using** *CImpP* **by** *auto*

**let** *?S* = ‹{*p* ⟶ *q*} ∪ (*Sp* − {¬ *p*}) ∪ (*Sq* − {*q*})›
**have** ‹*finite ?S*›
**using** *Sp*(*2*) *Sq*(*2*) **by** *blast*
**moreover have** ‹*?S* ⊆ *S*›
**using** ∗(*1*) *CImpP*(*1*) *Sp*(*1*) *Sq*(*1*) **by** *auto*
**ultimately obtain** *U* :: ‹'*a set*› **and** *E F G* **where**
*M*: ‹*wf-model* (*Model U E F G*)› ‹*Model U E F G* ⊨ *?S*›
**using** ∗ **by** *meson*
**then have** ‹*Model U E F G* ⊨ *Sp* ∨ *Model U E F G* ⊨ *Sq*›
**by** *auto*
**then show** *False*
**using** *M*(*1*) *Sp*(*3*) *Sq*(*3*) **by** *blast*
**qed**
**qed**
**qed**

**lemma** *compact-G*: ‹*P.sat*$_E$ *G.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** *qs* :: ‹'*f tm* ⇒ ('*f*, '*p*) *fm list*› **and** *t* **and** *S* :: ‹('*f*, '*p*) *fm set*›
**assume** ‹*P.enough-new S*›
**then show** ‹*P.enough-new* (*set* (*qs t*) ∪ *S*)›
**using** *params-left* **by** *fast*
**next**
**fix** *ps qs F t* **and** *S S'* :: ‹('*f*, '*p*) *fm set*›
**assume** ‹*ps* ⇝$_\gamma$ (*F*, *qs*)› **and** ∗: ‹*set ps* ⊆ *S*› ‹*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'*)›
‹*t* ∈ *F S*› ‹*S'* ⊆ *set* (*qs t*) ∪ *S*› ‹*finite S'*›
**then show** ‹∃ (*U* :: '*a set*) *E F G*. *wf-model* (*Model U E F G*) ∧ *Model U E F G* ⊨ *S'*›

**proof** *cases*
**case** (*CAllP p*)
**let** *?S* = ‹{∀ *p*} ∪ (*S'* − *set* (*qs t*))›
**have** ‹*?S* ⊆ *S*› ‹*finite ?S*›
**using** *CAllP* ∗ **by** *fastforce+*
**then obtain** *U* :: ‹'*a set*› **and** *E F G* **where**
*M*: ‹*wf-model* (*Model U E F G*)› ‹*Model U E F G* ⊨ *?S*›
**using** ∗ **by** *meson*
**then have** ‹*Model U E F G* ⊨ *set* (*qs t*) ∪ *?S*›
**using** *CAllP* **by** *auto*
**then show** *?thesis*
**using** *M*(*1*) **by** *blast*
**qed**
**qed**

**lemma** *compact-D*: ‹*P.sat*$_E$ *D.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** *p* **and** *S* :: ‹('*f*, '*p*) *fm set*›
**assume** ∗: ‹*p* ∈ *S*› ‹*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'*)›
**then show** ‹∃ *x*. *True* ∧ *set* (*δ p x*) ∪ *S* ∈ {*S*. *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** (*induct p* - *rule*: *δ.induct*)
**case** (*1 p* -)
**then have** *P*: ‹∀ *x*. *set* (*δ* (¬ ∀ *p*) *x*) ∪ *S* ∈ {*S*. *P.enough-new S*}›
**using** ∗ *params-left* **by** *fast*

**show** *?case*
**proof** (*rule ccontr*)
**assume** ‹∄ *x*. *True* ∧ *set* (*δ* (¬ ∀ *p*) *x*) ∪ *S* ∈ {*S*. *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'*))}›
**then have** ‹∀ *x*. ∃ *S'* ⊆ {¬ ⟨⋆*x*⟩*p*} ∪ *S*. *finite S'* ∧ ¬(∃(*U* :: '*a set*) *E F G*. *wf-model* (*Model U E F G*) ∧ *Model U E F G* ⊨ *S'*)›
**using** *P* **by** *simp*
**moreover obtain** *x* **where** *x*: ‹*x* ∉ *P.params S*›
**using** ‹*P.enough-new S*› **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'*: ‹*S'* ⊆ {¬ ⟨⋆*x*⟩*p*} ∪ *S*› ‹*finite S'*› ‹∀(*U* :: '*a set*) *E F G*. *wf-model* (*Model U E F G*) ⟶ ¬ *Model U E F G* ⊨ *S'*›
**using** *1* **by** *meson*

**let** *?S* = ‹{¬ ∀ *p*} ∪ (*S'* − {¬ ⟨⋆*x*⟩*p*})›
**have** ‹*finite ?S*›

```
      using S'(2) by blast
    moreover have **: ‹?S ⊆ S›
      using *(1) 1(1) S'(1) by auto
    ultimately obtain U :: ‹'a set› and E F G where
      M: ‹wf-model (Model U E F G)› ‹Model U E F G ⊨ ?S›
      using * by meson
    then obtain z where z: ‹z ∈ U› ‹¬ Model U (z ≫ E) F G ⊨ p›
      by auto

    let ?F = ‹F(x := λ-. z)›
    have ‹¬ Model U (?F x [] ≫ E) ?F G ⊨ p›
      using M x z ** by auto
    moreover have ‹Model U E ?F G ⊨ ?S›
      using M x ** by (metis (no-types, lifting) UN-iff in-mono upd-params-fm)
    ultimately have ‹∃F. wf-model (Model U E F G) ∧ Model U E F G ⊨ S'›
      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: ‹P.prop_E Kinds {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'))}›
 using compact-C compact-A compact-B compact-G compact-D ..

theorem compactness:
 fixes S :: ‹('f, 'p) fm set›
 assumes ‹P.enough-new S›
 shows ‹(∃(U :: 'f tm set) E F G. wf-model (Model U E F G) ∧ Model U E F G
⊨ 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'))›
proof safe
 fix U :: ‹'f tm set› and E F G and S' :: ‹('f, 'p) fm set›
 assume ‹wf-model (Model U E F G)› ‹Model U E F G ⊨ S› ‹S' ⊆ S›
 then show ‹∃(U :: 'f tm set) E F G. wf-model (Model U E F G) ∧ Model U E
F G ⊨ S'›
   by blast
next
 let ?S = ‹{has-subterm} ∪ S›
 assume *: ‹∀S' ⊆ S. finite S' ⟶ (∃(U :: 'f tm set) E F G. wf-model (Model U
E F G) ∧ Model U E F G ⊨ S')›
 have ‹∀S' ⊆ ?S. finite S' ⟶ (∃(U :: 'f tm set) E F G. wf-model (Model U E
F G) ∧ Model U E F G ⊨ S')›
 proof safe
   fix S'
   assume ‹S' ⊆ {has-subterm} ∪ S› ‹finite S'›
   then obtain U :: ‹'f tm set› and E F G where M: ‹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** ‹∀ *x* ∈ *U*. *wf-model* (*Model U E* (*F*(*a* := λ-. *x*)) *G*)›
**using** *UniI*(*5*) **by** *simp*
**ultimately have** ‹∀ *x* ∈ *U*. *Model U E* (*F*(*a* := λ-. *x*)) *G* ⊨ ⟨⋆*a*⟩*p*›
**using** *UniI* **by** *meson*
**then show** *?case*
**using** *UniI* **by** *simp*
**qed** *auto*

### 2.8.2 Derivational Consistency

**lemma** *Boole*: ‹{¬ *p*} ∪ *A* ⊩ ⊥ ⟹ *A* ⊩ *p*›
**unfolding** *Neg-def* **using** *Clas FlsE* **by** *fast*

**sublocale** *DC*: *Derivational-Confl psub params-fm* ‹λ-. *True*› *confl-class* ‹λ*A*. ¬ *A* ⊩ ⊥›
**proof**
**fix** *A ps qs* **and** *q* :: ‹('*f*, '*p*) *fm*›
**assume** ‹*ps* ⇝✗ *qs*› ‹*set ps* ⊆ *A*› ‹*q* ∈ *set qs*› ‹*q* ∈ *A*›
**then show** ‹¬ ¬ *A* ⊩ ⊥›
**by** *cases auto*
**qed**

**sublocale** *DA*: *Derivational-Alpha psub params-fm* ‹λ-. *True*› *alpha-class* ‹λ*A*. ¬ *A* ⊩ ⊥›
**proof** (*standard*; *safe*)
**fix** *A* **and** *ps qs* :: ‹('*f*, '*p*) *fm list*›
**assume** ‹*ps* ⇝$_\alpha$ *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 psub params-fm* ‹λ-. *True*› *beta-class* ‹λ*A*. ¬ *A* ⊩ ⊥›
**proof**
**fix** *A* **and** *ps qs* :: ‹('*f*, '*p*) *fm list*›
**assume** ‹*ps* ⇝$_\beta$ *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 psub params-fm* ‹λ-. *True*› *gamma-class* ‹λ*A*. ¬ *A* ⊩ ⊥›
**proof**
**fix** *A F qs t* **and** *ps* :: ‹('*f*, '*p*) *fm list*›
**assume** ‹*ps* ⇝$_\gamma$ (*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 ‹λ*A*. ¬ *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* ‹λ*A*. ¬ *A* ⊩ ⊥›
**using** *prop*$_E$*-Kinds*[*OF DC.kind DA.kind DB.kind DG.kind DD.kind*] **by** *unfold-locales*

### 2.8.3 Strong Completeness

**lemma** *with-subterm-elim*: ‹*A* ⊩ *with-subterm p* ⟹ *A* ⊩ *p*›
**using** *Assm ImpE* **by** *blast*

**theorem** *strong-completeness*:
**fixes** *p* :: ‹('*f*, '*p*) *fm*›
**assumes** *mod*: ‹⋀(*U* :: '*f tm set*) *E F G*. *wf-model* (*Model U E F G*) ⟹

```
    (∀ q ∈ A. Model U E F G ⊨ q) ⟹ Model U E F G ⊨ p›
    and ‹P.enough-new A›
  shows ‹A ⊩ p›
proof (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** ‹*bounded K A P*› ‹$\bigwedge$*A*. *P A* $\Longrightarrow$ *Q A*›
**shows** ‹*bounded K A Q*›
**using** *assms* **unfolding** *bounded-def* **by** *simp*

**lemma** *bounded-two* [*elim*]:
**assumes** ‹*bounded K A P*› ‹*bounded K′ A Q*› ‹$\bigwedge$*A*. *P A* $\Longrightarrow$ *Q A* $\Longrightarrow$ *R A*›
**shows** ‹*bounded* (*K* @ *K′*) *A R*›
**using** *assms* **unfolding** *bounded-def* **by** *simp*

**lemma** *bounded-removeAll* [*dest*]:
**assumes** ‹*bounded K* ({*p*} ∪ *A*) *P*›
**shows** ‹*bounded* (*removeAll p K*) *A* (λ*B*. *P* (*p* # *B*))›
**using** *assms* **unfolding** *bounded-def*
**by** (*metis Diff-subset-conv insert-is-Un insert-mono list.simps*(*15*) *set-removeAll*)

**lemma** *bounded-terms*:
**assumes** ‹*t* ∈ *terms* ({*p*} ∪ *A*)›
**shows** ‹*t* ∈ *terms-fm p* ∧ *bounded* [] *A* (λ*B*. *t* ∈ *terms* (*set* (*p* # *B*))) ∨
(∃ *q* ∈ *A*. *t* ∈ *terms-fm q* ∧ *bounded* [*q*] *A* (λ*B*. *t* ∈ *terms* (*set* (*p* # *B*))))›
**using** *assms* **unfolding** *terms-def bounded-def* **by** *auto*

**lemma** *bounded-params*:
**assumes** ‹*a* ∉ *P.params* ({*p*} ∪ *A*)› ‹*bounded K A P*›
**shows** ‹*bounded K A* (λ*B*. *a* ∉ *P.params* (*set* (*p* # *B*)))›
**using** *assms* **unfolding** *bounded-def* **by** *auto*

**lemma** *finite-kernel*: ‹*A* ⊩ *p* $\Longrightarrow$ ∃ *K*. *bounded K A* (λ*B*. *B* ⊢ *p*)›
**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** ‹∃ *K*. *bounded K A* (λ*B*. (*p* ⟶ *q*) # *B* ⊢ *p*)›
**by** *fast*
**then show** *?case*
**using** *ND-List.Clas* **unfolding** *bounded-def* **by** *meson*
**qed** *fast*+

**corollary** *finite-assumptions*: ‹*A* ⊩ *p* $\Longrightarrow$ ∃ *B*. *set B* ⊆ *A* ∧ *B* ⊢ *p*›

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

**lemma** *to-set*: ‹$A \vdash p \Longrightarrow set\ A \Vdash p$›
**by** (*induct A p pred*: *ND-List*) (*auto intro*: *ND-Set.Clas*)

**corollary** *soundness-list*:
**assumes** ‹$A \vdash p$› ‹*wf-model* (*Model U E F G*)› ‹$\forall q \in set\ A.\ Model\ U\ E\ F\ G \models q$›
**shows** ‹*Model U E F G* $\models p$›
**using** *assms soundness-set to-set* **by** *fast*

**corollary** *soundness-nil*: ‹$[] \vdash p \Longrightarrow$ *wf-model* (*Model U E F G*) $\Longrightarrow$ *Model U E F G* $\models p$›
**using** *soundness-list* **by** (*metis empty-iff list.set*(*1*))

**corollary** ‹$\neg\ ([] \vdash \bot)$›
**using** *soundness-nil* **by** *fastforce*

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

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

**end**

## 2.9 Tableau

**locale** *Tableau*
**begin**

**inductive** *TC* :: ‹$('f, 'p)\ fm\ set \Rightarrow bool$› (‹$\vdash$ -› [*51*] *50*) **where**
*Axiom* [*simp*]: ‹$\neg \cdot P\ ts \in A \Longrightarrow \cdot P\ ts \in A \Longrightarrow \vdash A$›
| *FlsP* [*simp*]: ‹$\bot \in A \Longrightarrow \vdash A$›
| *FlsN* [*intro*]: ‹$\vdash A \Longrightarrow \vdash \{\neg\ \bot\} \cup A$›
| *ImpP* [*intro*]: ‹$\vdash \{\neg\ p\} \cup A \Longrightarrow \vdash \{q\} \cup A \Longrightarrow \vdash \{p \longrightarrow q\} \cup A$›
| *ImpN* [*intro*]: ‹$\vdash \{p, \neg\ q\} \cup A \Longrightarrow \vdash \{\neg\ (p \longrightarrow q)\} \cup A$›
| *UniP* [*intro*]: ‹$\vdash \{\langle t \rangle p\} \cup A \Longrightarrow t \in terms\ (\{p\} \cup A) \Longrightarrow \vdash \{\forall\ p\} \cup A$›

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

### 2.9.1 Soundness

**theorem** *soundness*:
  **assumes** ‹$\vdash A$› ‹*wf-model* (*Model U E F G*)›
  **shows** ‹$\exists\, q \in A.\ \neg\ Model\ U\ E\ F\ G \models q$›
  **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** ‹$\exists\, q \in \{\langle t\rangle\ p\} \cup A.\ \neg\ Model\ U\ E\ F\ G \models q$›
    **by** *blast*
  **moreover have** ‹$\langle\!|(E,\ F)|\!\rangle\ t \in U$›
    **using** *UniP.prems* **by** *auto*
  **then have** ‹$\neg\ Model\ U\ E\ F\ G \models \langle t\rangle\ p \Longrightarrow \neg\ Model\ U\ E\ F\ G \models \forall\, p$›
    **by** *auto*
  **ultimately show** *?case*
    **by** *blast*
**next**
  **case** (*UniN a p A*)
  **then have** ‹$\forall\, x \in U.\ wf\text{-}model\ (Model\ U\ E\ (F(a := \lambda\text{-}.\ x))\ G)$›
    **by** *simp*
  **then have** ‹$\forall\, x \in U.\ \exists\, q \in \{\neg\ \langle\star\ a\rangle\ p\} \cup A.\ \neg\ Model\ U\ E\ (F(a := \lambda\text{-}.\ x))\ G \models q$›
    **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* ‹$\lambda\text{-}.\ True$› *confl-class* ‹$\lambda A.\ \neg \vdash A$›
**proof**
  **fix** *A ps qs* **and** *q* :: ‹(*'f*, *'p*) *fm*›
  **assume** ‹$ps \rightsquigarrow\!\!\!\!\!\!\times\ qs$› ‹$set\ ps \subseteq A$› ‹$q \in set\ qs$› ‹$q \in A$›
  **then show** ‹$\neg\ \neg \vdash A$›
    **by** *cases auto*
**qed**

**sublocale** *DA*: *Derivational-Alpha psub params-fm* ‹λ-. *True*› *alpha-class* ‹λ*A*. ¬ ⊢ *A*›
**proof**
  **fix** *A* **and** *ps qs* :: ‹('*f*, '*p*) *fm list*›
  **assume** ‹*ps* ∼$_\alpha$ *qs*› **and** ∗: ‹*set ps* ⊆ *A*› ‹¬ ⊢ *A*›
  **then show** ‹¬ ⊢ *set qs* ∪ *A*›
  **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* ‹λ-. *True*› *beta-class* ‹λ*A*. ¬ ⊢ *A*›
**proof**
  **fix** *A* **and** *ps qs* :: ‹('*f*, '*p*) *fm list*›
  **assume** ‹*ps* ∼$_\beta$ *qs*› **and** ∗: ‹*set ps* ⊆ *A*› ‹¬ ⊢ *A*›
  **then show** ‹∃ *q* ∈ *set qs*. ¬ ⊢ {*q*} ∪ *A*›
  **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* ‹λ-. *True*› *gamma-class* ‹λ*A*. ¬ ⊢ *A*›
**proof**
  **fix** *A F qs t* **and** *ps* :: ‹('*f*, '*p*) *fm list*›
  **assume** ‹*ps* ∼$_\gamma$ (*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* ∗ *UniP*[*of t p A*]
      **by** (*auto simp*: *sup.order-iff*)
  **qed**
**qed**

**sublocale** *DD*: *Derivational-Delta psub params-fm* ‹λ-. *True*› δ ‹λ*A*. ¬ ⊢ *A*›
**proof**
  **fix** *A a* **and** *p* :: ‹('*f*, '*p*) *fm*›
  **assume** ‹*p* ∈ *A*› ‹*a* ∉ *P.params A*› ‹¬ ⊢ *A*›
  **then show** ‹¬ ⊢ *set* (δ *p a*) ∪ *A*›

**proof** (*induct p a rule*: *δ.induct*)
**case** (*1 p x*)
**then have** ‹$x \notin P.params\ (\{p\} \cup A)$›
**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* ‹$\lambda$-. *True*› *Kinds* ‹$\lambda A.\ \neg \vdash A$›
**using** *prop$_E$-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* :: ‹(*'f*, *'p*) *fm*›
**assumes** *mod*: ‹$\bigwedge$(*U* :: *'f tm set*) *E F G*. *wf-model* (*Model U E F G*) $\Longrightarrow$ ($\forall$ *q* $\in$ *A*. *Model U E F G* $\models$ *q*) $\Longrightarrow$ *Model U E F G* $\models$ *p*›
**and** ‹*P.enough-new A*›
**shows** ‹$\vdash$ {$\neg$ *with-subterm p*} $\cup$ *A*›
**proof** (*rule ccontr*)
**assume** $*$: ‹$\neg \vdash$ {$\neg$ *with-subterm p*} $\cup$ *A*›

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

**have** *ne*: ‹*terms ?S* $\neq$ {}›
**unfolding** *terms-def* **by** *simp*
**then have** ‹*terms* (*mk-mcs ?C ?S*) $\neq$ {}›
**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* $\in$ *?C*›
**using** $*$ **by** *simp*
**ultimately have** $*$: ‹$\forall$ *p* $\in$ *?S*. *?M* $\models$ *p*›
**using** *model-existence ne* **by** *blast*
**then have** ‹*?M* $\models$ *p*›
**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* (‹◯$_2$›)

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

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

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

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

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

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

*0*) [*t2*])))›

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

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

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

## 3.2 Semantics

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

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

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

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

**proposition** ‹(*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ (∀ (•*P* [# *0*]) ⟶ •*P* [⋆*a*])›
**by** (*simp add*: *shift-def*)

## 3.3 Operations

### 3.3.1 Shift

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

**lemma** *shift-eq* [*simp*]: ‹*n* = *m* ⟹ *E*⟨*n*:*x*⟩ *m* = *x*›
**by** (*simp add*: *shift-def*)

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

**by** (*simp add*: *shift-def*)

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

**lemma** *shift-commute* [*simp*]: ‹$(E\langle n{:}y\rangle\langle 0{:}x\rangle) = (E\langle 0{:}x\rangle\langle n+1{:}y\rangle)$›
**proof**
**fix** *m*
**show** ‹$(E\langle n{:}y\rangle\langle 0{:}x\rangle)\ m = (E\langle 0{:}x\rangle\langle n+1{:}y\rangle)\ m$›
**unfolding** *shift-def* **by** (*cases m*) *simp-all*
**qed**

**end**

### 3.3.2 Parameters

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

**lemma** *upd-params-sym* [*simp*]: ‹$f \notin$ *params-sym fn* $\Longrightarrow \langle\!|E_F, F(f := x)|\!\rangle_2$ *fn* = $\langle\!|E_F, F|\!\rangle_2$ *fn*›
**by** (*induct fn*) (*auto cong*: *map-cong*)

**lemma** *upd-params-tm* [*simp*]: ‹$f \notin$ *params-tm t* $\Longrightarrow \langle\!|E, E_F, C, F(f := x)|\!\rangle$ *t* = $\langle\!|E, E_F, C, F|\!\rangle$ *t*›
**by** (*induct t*) (*auto cong*: *map-cong*)

**lemma** *upd-params-tm-c* [*simp*]: ‹$c \notin$ *params-tm t* $\Longrightarrow \langle\!|E, E_F, C(c := x), F|\!\rangle$ *t* = $\langle\!|E, E_F, C, F|\!\rangle$ *t*›
**by** (*induct t*) (*auto cong*: *map-cong*)

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

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

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

**lemma** *finite-params-sym* [*simp*]: ‹*finite* (*params-sym fn*)›
**by** (*induct fn*) *simp-all*

**lemma** *finite-params-tm* [*simp*]: ‹*finite* (*params-tm t*)›
**by** (*induct t*) *simp-all*

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

**by** (*induct p*) *simp-all*

### 3.3.3 Instantiation

**primrec** *lift-tm* (‹↑›) **where**
  ‹↑(#*n*) = #(*n*+*1*)›
| ‹↑(○*f ts*) = ○*f* (*map* ↑ *ts*)›
| ‹↑(⋆*c*) = ⋆ *c*›

**primrec** *lift-sym* (‹↑$_2$›) **where**
  ‹↑$_2$(#$_2$ *n*) = #$_2$ (*n*+*1*)›
| ‹↑$_2$(○$_2$ *p*) = ○$_2$ *p*›

**primrec** *lift-fn* (‹↑$_F$›) **where**
  ‹↑$_F$(#*n*) = #*n*›
| ‹↑$_F$(○*f ts*) = ○(↑$_2$ *f*) (*map* ↑$_F$ *ts*)›
| ‹↑$_F$(⋆ *c*) = ⋆ *c*›

**primrec** *inst-tm* (‹⟪-′/-⟫›) **where**
  ‹⟪*s*/*m*⟫(#*n*) = (*if n* < *m then* #*n else if n* = *m then s else* #(*n*−*1*))›
| ‹⟪*s*/*m*⟫(○*f ts*) = ○*f* (*map* ⟪*s*/*m*⟫ *ts*)›
| ‹⟪*s*/*m*⟫(⋆*c*) = ⋆*c*›

**primrec** *inst-sym* (‹⟪-′/-⟫$_2$›) **where**
  ‹⟪*s*/*m*⟫$_2$ (#$_2$ *n*) = (*if n* < *m then* #$_2$ *n else if n* = *m then s else* #$_2$ (*n*−*1*))›
| ‹⟪*s*/*m*⟫$_2$ (○$_2$ *p*) = ○$_2$ *p*›

**primrec** *inst-fn* (‹⟪-′/-⟫$_F$›) **where**
  ‹⟪*s*/*m*⟫$_F$(#*n*) = (#*n*)›
| ‹⟪*s*/*m*⟫$_F$(○*f ts*) = ○(⟪*s*/*m*⟫$_2$ *f*) (*map* ⟪*s*/*m*⟫$_F$ *ts*)›
| ‹⟪*s*/*m*⟫$_F$(⋆*c*) = (⋆*c*)›

**primrec** *inst-fm* (‹⟨-′/-⟩›) **where**
  ‹⟨-/-⟩⊥ = ⊥›
| ‹⟨*s*/*m*⟩(•*P ts*) = •*P* (*map* ⟪*s*/*m*⟫ *ts*)›
| ‹⟨*s*/*m*⟩(*p* ⟶ *q*) = ⟨*s*/*m*⟩*p* ⟶ ⟨*s*/*m*⟩*q*›
| ‹⟨*s*/*m*⟩(∀ *p*) = ∀ (⟨↑*s*/*m*+*1*⟩*p*)›
| ‹⟨*s*/*m*⟩(∀$_P$ *p*) = ∀$_P$(⟨*s*/*m*⟩*p*)›
| ‹⟨*s*/*m*⟩(∀$_F$ *p*) = ∀$_F$(⟨↑$_F$ *s*/*m*⟩*p*)›

**primrec** *inst-fm-P* (‹⟨-′/-⟩$_P$›) **where**
  ‹⟨-/-⟩$_P$⊥ = ⊥›
| ‹⟨*s*/*m*⟩$_P$(•*P ts*) = •(⟪*s*/*m*⟫$_2$ *P*) *ts*›
| ‹⟨*s*/*m*⟩$_P$(*p* ⟶ *q*) = ⟨*s*/*m*⟩$_P$*p* ⟶ ⟨*s*/*m*⟩$_P$*q*›
| ‹⟨*s*/*m*⟩$_P$(∀ *p*) = ∀ (⟨*s*/*m*⟩$_P$*p*)›
| ‹⟨*s*/*m*⟩$_P$(∀$_P$ *p*) = ∀$_P$(⟨↑$_2$ *s*/*m*+*1*⟩$_P$*p*)›
| ‹⟨*s*/*m*⟩$_P$(∀$_F$ *p*) = ∀$_F$(⟨*s*/*m*⟩$_P$*p*)›

**primrec** *inst-fm-F* (‹⟨-′/-⟩$_F$›) **where**

$\langle\langle$-/-$\rangle_F \bot = \bot\rangle$
| $\langle\langle s/m\rangle_F(\cdot P\ ts) = \cdot P\ (map\ \langle\!\langle s/m\rangle\!\rangle_F\ ts)\rangle$
| $\langle\langle s/m\rangle_F(p \longrightarrow q) = \langle s/m\rangle_F p \longrightarrow \langle s/m\rangle_F q\rangle$
| $\langle\langle s/m\rangle_F(\forall\, p) = \forall\, (\langle s/m\rangle_F p)\rangle$
| $\langle\langle s/m\rangle_F(\forall_P p) = \forall_P(\langle s/m\rangle_F p)\rangle$
| $\langle\langle s/m\rangle_F(\forall_F p) = \forall_F(\langle\uparrow_2\ s/m{+}1\rangle_F p)\rangle$

**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-semantics* [*simp*]: $\langle\langle\!| E,\ E_F,\ C,\ F|\!\rangle\ (\langle\!\langle s/m\rangle\!\rangle t) = \langle\!| E\langle m{:}\langle\!| E,\ E_F,\ C,\ F|\!\rangle\ s\rangle,\ E_F,\ C,\ F|\!\rangle\ t\rangle$
**by** (*induct t*) (*auto cong*: *map-cong*)

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

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

**lemma** *inst-fm-semantics-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-semantics* [*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-semantics-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**
$\langle$*size-fm* $\bot$ = *1*$\rangle$
| $\langle$*size-fm* (•- -) = *1*$\rangle$
| $\langle$*size-fm* ($p \longrightarrow q$) = *1* + *size-fm p* + *size-fm q*$\rangle$

| ‹*size-fm* (∀ *p*) = *1* + *size-fm p*›
| ‹*size-fm* (∀ $_P$*p*) = *1* + *size-fm p*›
| ‹*size-fm* (∀ $_F$*p*) = *1* + *size-fm p*›

**lemma** *size-inst-fm* [*simp*]: ‹*size-fm* (⟨*t*/*m*⟩*p*) = *size-fm p*›
**by** (*induct p arbitrary*: *m t*) *simp-all*

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

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

## 3.4 Model Existence

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

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

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

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

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

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

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

**interpretation** *P*: *Params map-fm params-fm* ‹λ-. *True*›
**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* ‹λ-. *True*› *confl-class*
**by** *unfold-locales* (*auto elim*!: *confl-class.cases intro*: *confl-class.intros*)

**interpretation** *A*: *Alpha map-fm params-fm ‹λ-. True› 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 ‹λ-. True› 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-tm f* (⟪*t*/*n*⟫ *x*) = ⟪*map-tm f t*/*n*⟫ (*map-tm f x*)
**by** (*induct x*) *simp-all*

**lemma** *map-tm-lift-tm* [*simp*]: *map-tm f* (↑ *t*) = ↑ (*map-tm f t*)
**by** (*induct t*) *simp-all*

**lemma** *map-sym-lift-fn* [*simp*]: ‹*map-sym f* (↑$_2$ *t*) = ↑$_2$ (*map-sym f t*)›
**by** (*induct t*) *auto*

**lemma** *map-tm-lift-fn* [*simp*]: *map-tm f* (↑$_F$ *t*) = ↑$_F$ (*map-tm f t*)
**by** (*induct t*) *simp-all*

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

**lemma** *map-sym-inst-sym* [*simp*]: ‹*map-sym f* (⟪*t*/*m*⟫$_2$ *p*) = ⟪*map-sym f t*/*m*⟫$_2$ (*map-sym f p*)›
**by** (*induct p arbitrary*: *t m*) *simp-all*

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

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

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

**interpretation** *G*: *Gamma-UNIV map-tm map-fm params-fm ‹λ-. True› 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 ‹λ-. True› 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 ‹λ-. True› gamma-class-F*
**by** *unfold-locales* (*fastforce elim*: *gamma-class-F.cases intro*: *gamma-class-F.intros*)+

**interpretation** *D*: *Delta map-fm params-fm ‹λ-. True› δ*
**proof**
**show** ‹$\bigwedge f.\ \delta\ (map\text{-}fm\ f\ p)\ (f\ x) = map\ (map\text{-}fm\ f)\ (\delta\ p\ x)$› **for** $p :: {'x\ fm}$ **and** $x$
**by** (*induct p x rule*: *δ.induct*) *simp-all*
**qed**

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

**lemma** $prop_E$*-Kinds*:
**assumes** ‹$P.sat_E$ *C.kind C*› ‹$P.sat_E$ *A.kind C*› ‹$P.sat_E$ *B.kind C*› ‹$P.sat_E$ *G.kind C*› ‹$P.sat_E$ $G_P$*.kind C*›
‹$P.sat_E$ $G_F$*.kind C*› ‹$P.sat_E$ *D.kind C*›
**shows** ‹$P.prop_E$ *Kinds C*›
**unfolding** $P.prop_E$*-def* **using** *assms* **by** *simp*

**interpretation** *Consistency-Kinds map-fm params-fm ‹λ-. True› Kinds*
**using** *P.Params-axioms C.Consistency-Kind-axioms A.Consistency-Kind-axioms B.Consistency-Kind-axioms*
*G.Consistency-Kind-axioms* $G_P$*.Consistency-Kind-axioms* $G_F$*.Consistency-Kind-axioms*
*D.Consistency-Kind-axioms*
**by** (*auto intro*: *Consistency-Kinds.intro simp*: *Consistency-Kinds-axioms-def*)

**interpretation** *Maximal-Consistency map-fm params-fm ‹λ-. True› Kinds*
**proof**
**show** ‹*infinite* (*UNIV* :: *'x fm set*)›
**using** *infinite-UNIV-size*[*of* ‹$\lambda p.\ p \longrightarrow p$›] **by** *simp*
**qed** *simp*

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

**abbreviation** *hpred* **where** $hpred\ H == \lambda P\ ts.\ \cdot(\bigcirc_2\ P)\ ts \in H$

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

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

**abbreviation** *hfun* **where** $hfun == \ \lambda f.\ \bigcirc\ (\bigcirc_2\ f)$

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

**abbreviation** (*input*) *hmodel* (‹⟦-⟧›) **where** ‹⟦*H*⟧ ≡ (#, $henv_F$, $henv_P$ *H*, ⋆, *hfun*, *hpred H*, $hdom_P$ *H*, $hdom_F$)›

**lemma** *semantics-tm-id* [*simp*]: ‹⦇#, $henv_F$ , ⋆ , $\lambda f.\ \bigcirc\ (\bigcirc_2\ f)$ ⦈ *t* = *t*›
**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*]: ‹*map* ⦇#, λ*f*. ○ (#$_2$ *f*) , ⋆, λ*f*. ○ (○$_2$ *f*) ⦈ *ts* = *ts*›
**by** (*auto cong*: *map-cong*)

**lemma** *semantics-fn-h* [*simp*]: ‹⦇*henv*$_P$ *S*, *hpred S*⦈$_2$ *P ts* ⟷ •*P ts* ∈ *S*›
**by** (*cases P*) *simp-all*

**lemma** *canonical-henv*$_P$:
‹∀ *t*. (#, *henv*$_F$, (*henv*$_P$ *S*)⟨*0*:⦇*henv*$_P$ *S*, *hpred S*⦈$_2$ *t*⟩, ⋆, *hfun*, *hpred S*, *hdom*$_P$ *S*, *hdom*$_F$) ⊨ *p* ⟹
(#, *henv*$_F$, (*henv*$_P$ *S*)⟨*0*:*henv*$_P$ *S n*⟩, ⋆, *hfun*, *hpred S*, *hdom*$_P$ *S*, *hdom*$_F$) ⊨ *p*›
**by** (*metis semantics-fn.simps*(*1*))

**lemma** *canonical-hpred*:
‹∀ *t*. (#, *henv*$_F$, (*henv*$_P$ *S*)⟨*0*:⦇*henv*$_P$ *S*, *hpred S*⦈$_2$ *t*⟩, ⋆, *hfun*, *hpred S*, *hdom*$_P$ *S*, *hdom*$_F$) ⊨ *p* ⟹
(#, *henv*$_F$, (*henv*$_P$ *S*)⟨*0*:*hpred S P*⟩, ⋆, *hfun*, *hpred S*, *hdom*$_P$ *S*, *hdom*$_F$) ⊨ *p*›
**by** (*metis semantics-fn.simps*(*2*))

**lemma** *canonical-henv*$_F$:
‹∀ *t*. (#, *henv*$_F$⟨*0*:⦇*henv*$_F$, *hfun*⦈$_2$ *t*⟩, *henv*$_P$ *S*, ⋆, *hfun*, *hpred S*, *hdom*$_P$ *S*, *hdom*$_F$) ⊨ *p* ⟹
(#, *henv*$_F$⟨*0*:*henv*$_F$ *f*⟩, *henv*$_P$ *S*, ⋆, *hfun*, *hpred S*, *hdom*$_P$ *S*, *hdom*$_F$) ⊨ *p*›
**by** (*metis semantics-fn.simps*(*1*))

**lemma** *canonical-hfun*:
‹∀ *t*. (#, *henv*$_F$⟨*0*:⦇*henv*$_F$, *hfun*⦈$_2$ *t*⟩, *henv*$_P$ *S*, ⋆, *hfun*, *hpred S*, *hdom*$_P$ *S*, *hdom*$_F$) ⊨ *p* ⟹
(#, *henv*$_F$⟨*0*:*hfun f*⟩, *henv*$_P$ *S*, ⋆, *hfun*, *hpred S*, *hdom*$_P$ *S*, *hdom*$_F$) ⊨ *p*›
**by** (*metis semantics-fn.simps*(*2*))

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

**lemmas**
*confl* = $sat_H$[*of C.kind*] **and**
*alpha* = $sat_H$[*of A.kind*] **and**
*beta* = $sat_H$[*of B.kind*] **and**
*gammaFO* = $sat_H$[*of G.kind*] **and**
*gamma2P* = $sat_H$[*of* $G_P$*.kind*] **and**
*gamma2F* = $sat_H$[*of* $G_F$*.kind*] **and**
*delta* = $sat_H$[*of D.kind*]

**theorem** *model*: ‹$(p \in S \longrightarrow [\![S]\!] \models p) \wedge (\neg\ p \in S \longrightarrow \neg\ [\![S]\!] \models p)$›
**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** *Falsity*
**then show** *?thesis*
**using** *confl* **by** (*force intro*: *CFls*)
**next**
**case** (*Pre P ts*)
**then show** *?thesis*
**proof** (*safe del*: *notI*)
**assume** ‹$x = \cdot P\ ts$› ‹$\cdot P\ ts \in S$›
**then show** ‹$[\![S]\!] \models (\cdot P\ ts)$›
**by** *simp*
**next**
**assume** ‹$x = \cdot P\ ts$› ‹$\neg\ \cdot P\ ts \in S$›
**then have** ‹$\cdot P\ ts \notin S$›
**using** *confl* **by** (*force intro*: *CNeg*)
**then show** ‹$\neg\ [\![S]\!] \models (\cdot P\ ts)$›
**by** *simp*
**qed**
**next**
**case** (*Imp p q*)
**then show** *?thesis*
**proof** (*safe del*: *notI*)
**assume** ‹$x = p \longrightarrow q$› ‹$p \longrightarrow q \in S$›
**then have** ‹$\neg\ p \in S \vee q \in S$›
**using** *beta* **by** (*force intro*: *CImpP*)
**then show** ‹$[\![S]\!] \models (p \longrightarrow q)$›
**using** *2 Imp* **by** *auto*
**next**
**assume** ‹$x = p \longrightarrow q$› ‹$\neg\ (p \longrightarrow q) \in S$›
**then have** ‹$p \in S \wedge \neg\ q \in S$›
**using** *alpha* **by** (*force intro*: *CImpN*)
**then show** ‹$\neg\ [\![S]\!] \models (p \longrightarrow q)$›

```
    using 2 Imp by auto
  qed
next
  case (Uni p)
  then show ?thesis
  proof (safe del: notI)
    assume ‹x = ∀ p› ‹∀ p ∈ S›
    then have ‹∀ t. ⟨t/0⟩p ∈ S›
      using gammaFO by (fastforce intro: CAllP)
    moreover have ‹∀ t. (⟨t/0⟩p, ∀ p) ∈ measure size-fm›
      by simp
    ultimately have ‹∀ t. ⟦S⟧ ⊨ (⟨t/0⟩p)›
      using 2 ‹x = ∀ p› by blast
    then show ‹⟦S⟧ ⊨ (∀ p)›
      by simp
  next
    assume ‹x = ∀ p› ‹¬ ∀ p ∈ S›
    then obtain a where ‹¬ ⟨⋆a/0⟩p ∈ S›
      using delta by auto
    moreover have ‹(⟨⋆a/0⟩p, ∀ p) ∈ measure size-fm›
      by simp
    ultimately have ‹¬ ⟦S⟧ ⊨ (⟨⋆a/0⟩p)›
      using 2 ‹x = ∀ p› by blast
    then show ‹¬ ⟦S⟧ ⊨ (∀ p)›
      by auto
  qed
next
  case (UniP p)
  then show ?thesis
  proof (safe del: notI)
    assume ‹x = ∀ P p› ‹∀ P p ∈ S›
    then have ‹∀ t. ⟨t/0⟩ P p ∈ S›
      using gamma2P by (fastforce intro: CAllPP)
    moreover have ∗: ‹∀ t. (⟨t/0⟩ P p, ∀ P p) ∈ measure size-fm›
      by simp
    ultimately have ‹∀ t. ⟦S⟧ ⊨ (⟨t/0⟩ P p)›
      using 2 ‹x = ∀ P p› by blast
    then show ‹⟦S⟧ ⊨ ∀ P p›
      using canonical-henv P canonical-hpred by auto
  next
    assume ‹x = ∀ P p› ‹¬ ∀ P p ∈ S›
    then obtain a where ‹¬ ⟨◯2 a/0⟩ P p ∈ S›
      using delta by auto
    moreover have ‹(⟨◯2 a/0⟩ P p, ∀ P p) ∈ measure size-fm›
      by simp
    ultimately have ‹¬ ⟦S⟧ ⊨ (⟨◯2a/0⟩ P p)›
      using 2 ‹x = ∀ P p› by blast
    then show ‹¬ ⟦S⟧ ⊨ (∀ P p)›
      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-henv_F canonical-hfun unfolding hdom_F-def by auto
    next
      assume ‹x = ∀_F p› ‹¬ ∀_F p ∈ S›
      then obtain a where ‹¬ ⟨◯_2 a/0⟩_F p ∈ S›
        using delta by auto
      moreover have ‹(⟨◯_2 a/0⟩_F p, ∀_F p) ∈ measure size-fm›
        by simp
      ultimately have ‹¬ ⟦S⟧ ⊨ (⟨◯_2 a/0⟩_F p)›
        using 2 ‹x = ∀_F p› by blast
      then show ‹¬ ⟦S⟧ ⊨ (∀_F p)›
        by (auto simp: hdom_F-def)
    qed
  qed
qed

end

theorem model-existence:
  fixes S :: ‹'x fm set›
  assumes ‹P.prop_E 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.prop_H 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**
  ‹*boolean* - - ⊥ = *False*›
| ‹*boolean G* - (·*P ts*) = *G P ts*›
| ‹*boolean G A* (*p* ⟶ *q*) = (*boolean G A p* ⟶ *boolean G A q*)›
| ‹*boolean* - *A* (∀ *p*) = *A* (∀ *p*)›
| ‹*boolean* - *A* (∀ $_P$*p*) = *A* (∀ $_P$*p*)›
| ‹*boolean* - *A* (∀ $_F$*p*) = *A* (∀ $_F$*p*)›

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

**proposition** ‹*tautology* (∀ (·*P* [#*0*]) ⟶ ∀ (·*P* [#*0*]))›
  **by** *simp*

**lemma** *boolean-semantics*: ‹*boolean* (λ*a*. ⦇*E*$_P$,*G*⦈$_2$ *a* ∘ *map* ⦇*E*, *E*$_F$, *C*, *F*⦈) (λ*p*. (*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*) = (λ*p*. (*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*)›
**proof**
  **fix** *p*
  **show** ‹*boolean* (λ*a*. ⦇*E*$_P$, *G*⦈$_2$ *a* ∘ *map* ⦇*E*,*E*$_F$, *C*, *F*⦈) ((⊨) (*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*)) *p* = ((*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*)›
    **by** (*induct p*) *simp-all*
**qed**

**lemma** *tautology*[*simp*]: ‹*tautology p* ⟹ (*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*›
  **using** *boolean-semantics* **by** *metis*

**proposition** ‹∃ *p*. (∀ *E E*$_F$ *E*$_P$ *C F G PS FS*. (*E*, *E*$_F$, *E*$_P$, *C*, *F*, *G*, *PS*, *FS*) ⊨ *p*) ∧ ¬ *tautology p*›
  **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* (‹⊢ -› [*50*] *50*) **where**
  *TA*: ‹*tautology p* ⟹ ⊢ *p*›
| *IA*: ‹⊢ ∀ *p* ⟶ ⟨*t*/*0*⟩*p*›
| *IA*$_P$: ‹⊢ ∀ $_P$*p* ⟶ ⟨*s*/*0*⟩$_P$*p*›
| *IA*$_F$: ‹⊢ ∀ $_F$*p* ⟶ ⟨*s*/*0*⟩$_F$*p*›
| *MP*: ‹⊢ *p* ⟶ *q* ⟹ ⊢ *p* ⟹ ⊢ *q*›
| *GR*: ‹⊢ *q* ⟶ ⟨⋆*a*/*0*⟩*p* ⟹ *a* ∉ *params* {*p*, *q*} ⟹ ⊢ *q* ⟶ ∀ *p*›
| *GR*$_P$: ‹⊢ *q* ⟶ ⟨◯$_2$ *a*/*0*⟩$_P$*p* ⟹ *a* ∉ *params* {*p*, *q*} ⟹ ⊢ *q* ⟶ ∀ $_P$*p*›
| *GR*$_F$: ‹⊢ *q* ⟶ ⟨◯$_2$ *a*/*0*⟩$_F$*p* ⟹ *a* ∉ *params* {*p*, *q*} ⟹ ⊢ *q* ⟶ ∀ $_F$*p*›

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

**primrec** *imply* (**infixr** ‹$\rightsquigarrow$› *56*) **where**
‹([] $\rightsquigarrow$ *q*) = *q*›
| ‹(*p* # *ps* $\rightsquigarrow$ *q*) = (*p* $\longrightarrow$ *ps* $\rightsquigarrow$ *q*)›

**abbreviation** *Axiomatic-assms* (‹- $\vdash$ -› [*50*, *50*] *50*) **where**
‹*ps* $\vdash$ *q* $\equiv$ $\vdash$ *ps* $\rightsquigarrow$ *q*›

## 3.7 Soundness

**fun** *wf-model* **where**
‹*wf-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*›

**theorem** *soundness*:
**shows** ‹$\vdash$ *p* $\Longrightarrow$ *wf-model* (*E*, $E_F$, $E_P$, *C*, *F*, *G*, *PS*, *FS*) $\Longrightarrow$ (*E*, $E_F$, $E_P$, *C*, *F*, *G*, *PS*, *FS*) $\models$ *p*›
**proof** (*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** ‹(*E*, $E_F$, $E_P$, *C*(*a* := *x*), *F*, *G*, *PS*, *FS*) $\models$ (*q* $\longrightarrow$ $\langle\star a/0\rangle p$)› **for** *x*
**unfolding** *wf-model.simps* **by** *meson*
**ultimately show** *?case*
**by** *fastforce*
**next**
**case** ($GR_P$ *q a p*)
**moreover from** *this* **have** ‹$\forall$ *x*. *x* $\in$ *PS* $\longrightarrow$ (*E*, $E_F$, $E_P$, *C*, *F*, *G*(*a* := *x*), *PS*,*FS*) $\models$ (*q* $\longrightarrow$ $\langle\bigcirc_2 a/0\rangle_P p$)›
**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** ‹$\forall x.\ x \in FS \longrightarrow (E,\ E_F,\ E_P,\ C,\ F(a := x),\ G,\ PS,FS) \models (q \longrightarrow \langle \bigcirc_2 a/0\rangle_F p)$›
**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** ‹$\neg\ (\vdash \bot)$›
**using** *soundness*[**where** $p{=}\bot$ **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*: ‹$\vdash q \longrightarrow p \longrightarrow q$›
**and** *Imp2*: ‹$\vdash (p \longrightarrow q \longrightarrow r) \longrightarrow (p \longrightarrow q) \longrightarrow p \longrightarrow r$›
**and** *Neg*: ‹$\vdash \neg\ \neg\ p \longrightarrow p$›
**by** (*auto intro*: *TA*)

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

**lemma** *Tran′*: ‹$\vdash (q \longrightarrow r) \longrightarrow (p \longrightarrow q) \longrightarrow p \longrightarrow r$›
**by** (*meson Imp1 Imp2 MP*)

**lemma** *Swap*: ‹$\vdash (p \longrightarrow q \longrightarrow r) \longrightarrow q \longrightarrow p \longrightarrow r$›
**by** (*meson Imp1 Imp2 Tran′ MP*)

**lemma** *Tran*: ‹$\vdash (p \longrightarrow q) \longrightarrow (q \longrightarrow r) \longrightarrow p \longrightarrow r$›
**by** (*meson Swap Tran′ MP*)

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

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

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

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

**lemma** $GR_P'$: ‹$\vdash \neg \langle \bigcirc_2 P/0\rangle_P p \longrightarrow q \Longrightarrow P \notin params\ \{p, q\} \Longrightarrow \vdash \neg (\forall_P p) \longrightarrow q$›
**proof** −
**assume** ∗: ‹$\vdash \neg \langle \bigcirc_2 P/0\rangle_P p \longrightarrow q$› **and** *a*: ‹$P \notin params\ \{p, q\}$›
**have** ‹$\vdash \neg q \longrightarrow \neg \neg \langle \bigcirc_2 P/0\rangle_P p$›
**using** ∗ *Tran MP* **by** *metis*
**then have** ‹$\vdash \neg q \longrightarrow \langle \bigcirc_2 P/0\rangle_P p$›
**using** *Neg Tran MP* **by** *metis*
**then have** ‹$\vdash \neg q \longrightarrow \forall_P p$›
**using** *a* **by** (*auto intro*: $GR_P$)
**then have** ‹$\vdash \neg (\forall_P p) \longrightarrow \neg \neg q$›
**using** *Tran MP* **by** *metis*
**then show** *?thesis*
**using** *Neg Tran MP* **by** *metis*
**qed**

**lemma** $GR_F'$: ‹$\vdash \neg \langle \bigcirc_2 F/0\rangle_F p \longrightarrow q \Longrightarrow F \notin params\ \{p, q\} \Longrightarrow \vdash \neg (\forall_F p) \longrightarrow q$›
**proof** −
**assume** ∗: ‹$\vdash \neg \langle \bigcirc_2 F/0\rangle_F p \longrightarrow q$› **and** *a*: ‹$F \notin params\ \{p, q\}$›
**have** ‹$\vdash \neg q \longrightarrow \neg \neg \langle \bigcirc_2 F/0\rangle_F p$›
**using** ∗ *Tran MP* **by** *metis*
**then have** ‹$\vdash \neg q \longrightarrow \langle \bigcirc_2 F/0\rangle_F p$›
**using** *Neg Tran MP* **by** *metis*
**then have** ‹$\vdash \neg q \longrightarrow \forall_F p$›
**using** *a* **by** (*auto intro*: $GR_F$)
**then have** ‹$\vdash \neg (\forall_F p) \longrightarrow \neg \neg q$›
**using** *Tran MP* **by** *metis*
**then show** *?thesis*
**using** *Neg Tran MP* **by** *metis*
**qed**

**lemma** *imply-ImpE*: ‹$\vdash ps \rightsquigarrow p \longrightarrow ps \rightsquigarrow (p \longrightarrow q) \longrightarrow ps \rightsquigarrow q$›
**proof** (*induct ps*)
**case** *Nil*
**then show** *?case*
**by** (*metis Imp1 Swap MP imply.simps*(*1*))
**next**
**case** (*Cons r ps*)
**have** ‹$\vdash (r \longrightarrow ps \rightsquigarrow p) \longrightarrow r \longrightarrow ps \rightsquigarrow (p \longrightarrow q) \longrightarrow ps \rightsquigarrow q$›
**by** (*meson Cons.hyps Imp1 Imp2 MP*)
**then have** ‹$\vdash (r \longrightarrow ps \rightsquigarrow p) \longrightarrow (r \longrightarrow ps \rightsquigarrow (p \longrightarrow q)) \longrightarrow r \longrightarrow ps \rightsquigarrow q$›
**by** (*meson Imp1 Imp2 MP*)

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

**lemma** *MP′*: ‹*ps* ⊢ *p* ⟶ *q* ⟹ *ps* ⊢ *p* ⟹ *ps* ⊢ *q*›
**using** *imply-ImpE MP* **by** *metis*

**lemma** *imply-Cons* [*intro*]: ‹*ps* ⊢ *q* ⟹ *p* # *ps* ⊢ *q*›
**by** (*auto intro*: *MP Imp1*)

**lemma** *imply-head* [*intro*]: ‹*p* # *ps* ⊢ *p*›
**by** (*induct ps*) (*metis Imp1 Imp2 MP imply.simps*, *metis Imp1 Imp2 MP imply.simps*(*2*))

**lemma** *add-imply* [*simp*]: ‹⊢ *q* ⟹ *ps* ⊢ *q*›
**using** *imply-head* **by** (*metis MP imply.simps*(*2*))

**lemma** *imply-mem* [*simp*]: ‹*p* ∈ *set ps* ⟹ *ps* ⊢ *p*›
**using** *imply-head imply-Cons* **by** (*induct ps*) *fastforce*+

**lemma** *deduct1*: ‹*ps* ⊢ *p* ⟶ *q* ⟹ *p* # *ps* ⊢ *q*›
**by** (*meson MP′ imply-Cons imply-head*)

**lemma** *imply-append* [*iff*]: ‹(*ps* @ *qs* ⇝ *r*) = (*ps* ⇝ *qs* ⇝ *r*)›
**by** (*induct ps*) *simp-all*

**lemma** *imply-swap-append*: ‹*ps* @ *qs* ⊢ *r* ⟹ *qs* @ *ps* ⊢ *r*›
**proof** (*induct qs arbitrary*: *ps*)
**case** *Cons*
**then show** *?case*
**by** (*metis deduct1 imply.simps*(*2*) *imply-append*)
**qed** *simp*

**lemma** *deduct2*: ‹*p* # *ps* ⊢ *q* ⟹ *ps* ⊢ *p* ⟶ *q*›
**by** (*metis imply.simps imply-append imply-swap-append*)

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

**lemma** *cut*: ‹*p* # *ps* ⊢ *r* ⟹ *q* # *ps* ⊢ *p* ⟹ *q* # *ps* ⊢ *r*›
**by** (*meson MP′ deduct*(*2*) *imply-Cons*)

**lemma** *Boole*: ‹(¬ *p*) # *ps* ⊢ ⊥ ⟹ *ps* ⊢ *p*›
**by** (*meson MP′ Neg add-imply deduct*(*2*))

**lemma** *imply-weaken*: ‹*ps* ⊢ *q* ⟹ *set ps* ⊆ *set ps′* ⟹ *ps′* ⊢ *q*›
**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 ‹λ-. True› confl-class ‹λA. ¬ A ⊢ ⊥›*
**proof**
  **fix** *A ps qs* **and** *q* :: *‹'x fm›*
  **assume** *‹ps ⇝✗ qs› ‹set ps ⊆ set A› ‹q ∈ set qs› ‹q ∈ set A›*
  **then show** *‹¬ ¬ A ⊢ ⊥›*
    **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 ‹λ-. True› alpha-class ‹λA. ¬ A ⊢ ⊥›*
**proof** (*standard*; *safe*)
  **fix** *A* **and** *ps qs* :: *‹'x fm list›*
  **assume** *‹ps ⇝α qs›* **and** ∗: *‹set ps ⊆ set A› ‹¬ A ⊢ ⊥› ‹qs @ A ⊢ ⊥›*
  **then show** *False*
  **proof** *cases*
    **case** (*CImpN p q*)
    **then have** *‹A ⊢ ¬ (p ⟶ q)›*
      **using** ∗(*1*) **by** *simp*
    **moreover have** *‹A ⊢ p ⟶ q›*
      **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 ‹λ-. True› beta-class ‹λA. ¬ A ⊢ ⊥›*
**proof**
  **fix** *A* **and** *ps qs* :: *‹'x fm list›*
  **assume** *‹ps ⇝β qs›* **and** ∗: *‹set ps ⊆ set 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 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 ‹λ-. True› G.classify-UNIV ‹λA. ¬ A ⊢ ⊥›*
**proof** (*unfold-locales*; *safe*)
  **fix** *A qs t* **and** *ps* :: *‹'x fm list›*

**assume** ‹$ps \rightsquigarrow_{\gamma} qs$› **and** ∗: ‹$set\ ps \subseteq set\ A$› ‹$\neg\ A \vdash \bot$› ‹$qs\ t\ @\ A \vdash \bot$›
**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* ‹*λ-. True*› $G_P$*.classify-UNIV* ‹$\lambda A.\ \neg\ A \vdash \bot$›
**proof** (*unfold-locales*; *safe*)
**fix** *A qs t* **and** *ps* :: ‹*'x fm list*›
**assume** ‹$ps \rightsquigarrow_{\gamma P} qs$› **and** ∗: ‹$set\ ps \subseteq set\ A$› ‹$\neg\ A \vdash \bot$› ‹$qs\ t\ @\ A \vdash \bot$›
**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* ‹*λ-. True*› $G_F$*.classify-UNIV* ‹$\lambda A.\ \neg\ A \vdash \bot$›
**proof** (*unfold-locales*; *safe*)
**fix** *A qs t* **and** *ps* :: ‹*'x fm list*›
**assume** ‹$ps \rightsquigarrow_{\gamma F} qs$› **and** ∗: ‹$set\ ps \subseteq set\ A$› ‹$\neg\ A \vdash \bot$› ‹$qs\ t\ @\ A \vdash \bot$›
**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*: ‹$params\text{-}fm\ (ps \rightsquigarrow q) = params\text{-}fm\ q \cup (\bigcup p \in set\ ps.\ params\text{-}fm\ p)$›
**by** (*induct ps*) *auto*

**interpretation** *DD*: *Weak-Derivational-Delta map-fm params-fm* ‹*λ-. True*› *δ* ‹$\lambda A.\ \neg\ A \vdash \bot$›
**proof** (*standard*; *safe*)
**fix** *A a* **and** *p* :: ‹*'x fm*›

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

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

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

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

**have** $\langle infinite\ (UNIV :: 'x\ set) \rangle$
**using** *assms(2) card-of-ordLeq-infinite finite-subset inf-univ subset-UNIV*
**unfolding** *P.enough-new-def* **by** *blast*
**then have** $\langle P.prop_E\ Kinds\ ?C \rangle$
**using** *Consistency* **by** *blast*
**moreover have** $\langle ?S \in ?C \rangle$
**using** $*$ **by** *blast*
**moreover have** $\langle P.enough\text{-}new\ ?S \rangle$
**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** $hdom_F$*-def* **by** *auto*
**then show** *False*
**using** $*$ **by** *simp*
**qed**

**theorem** *completeness*:
**fixes** $p :: \langle 'x\ fm \rangle$
**assumes** $\langle \forall (E :: nat \Rightarrow 'x\ tm)\ E_P\ E_F\ C\ F\ G\ PS\ FS.\ \ wf\text{-}model\ (E, E_F, E_P, C, F, G, PS, FS) \longrightarrow (E, E_F, E_P, C, F, G, PS, FS) \models p \rangle$
**and** $\langle |UNIV :: 'x\ fm\ set| \leq o\ \ |UNIV :: 'x\ set| \rangle$
**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* :: $\langle 'x\ fm\ set \Rightarrow 'x\ fm \Rightarrow bool \rangle$ (**infix** $\langle \vdash \rangle$ *50*) **where**

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

### 3.10.1 Soundness

**theorem** *soundness-set*:
**assumes** ‹$A \Vdash p$› ‹*wf-model* ($E$, $E_F$, $E_P$, $C$, $F$, $G$, $PS$, $FS$)›
**shows** ‹$\forall\, q \in A.\ (E, E_F, E_P, C, F, G, PS, FS) \models q \Longrightarrow (E, E_F, E_P, C, F, G, PS, FS) \models p$›
**using** *assms*
**proof** (*induct A p arbitrary*: *C F G pred*: *ND-Set*)
**case** (*UniI A a p*)
**have** ‹$\forall\, q \in A.\ (E, E_F, E_P, C(a := x), F, G, PS, FS) \models q$› **for** $x$
**using** *UniI*(3−) **by** *simp*
**moreover have** ‹*wf-model* ($E$, $E_F$, $E_P$, $C(a := x)$, $F$, $G$, $PS$, $FS$)› **for** $x$
**using** *UniI*(5) **by** *simp*
**ultimately have** ‹$(E, E_F, E_P, C(a := x), F, G, PS, FS) \models \langle\star a/0\rangle p$› **for** $x$
**using** *UniI* **by** *meson*
**then show** *?case*
**using** *UniI* **by** *simp*
**next**
**case** ($UniI_P$ *A a p*)
**have** ‹$\forall\, x \in PS.\ \forall\, q \in A.\ (E, E_F, E_P, C, F, G(a := x), PS, FS) \models q$›
**using** $UniI_P$(3−) **by** *simp*
**moreover have** ‹$\forall\, x \in PS.$ *wf-model* ($E$, $E_F$, $E_P$, $C$, $F$, $G(a := x)$, $PS$, $FS$)›
**using** $UniI_P$(5) **by** *auto*
**ultimately have** ‹$\forall\, x \in PS.\ (E, E_F, E_P, C, F, G(a := x), PS, FS) \models \langle\bigcirc_2\ a/0\rangle_P\ p$›
**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** ‹$\forall\, x \in FS.\ \forall\, q \in A.\ (E, E_F, E_P, C, F(a := x), G, PS, FS) \models q$›
**using** $UniI_F$(3−) **by** *simp*
**moreover have** ‹$\forall\, x \in FS.$ *wf-model* ($E$, $E_F$, $E_P$, $C$, $F(a := x)$, $G$, $PS$, $FS$)›

**using** $UniI_F(5)$ **by** *auto*
**ultimately have** ‹$\forall x \in FS.\ (E,\ E_F,\ E_P,\ C,\ F(a := x),\ G,\ PS,\ FS) \models \langle \bigcirc_2 a/0 \rangle_F\ p$›
**using** $UniI_F$ **by** *meson*
**then show** *?case*
**using** $UniI_F$ **by** *simp*
**next**
**case** ($UniE_F$ *A p s*)
**then show** *?case*
**by** (*cases s*) (*fastforce intro*!: *rangeI*)+
**qed** *auto*

### 3.10.2 Derivational Consistency

**lemma** *Boole*: ‹$\{\neg p\} \cup A \Vdash \bot \Longrightarrow A \Vdash p$›
**using** *Clas FlsE* **by** *blast*

**sublocale** *DC*: *Derivational-Confl map-fm params-fm* ‹$\lambda$-. *True*› *confl-class* ‹$\lambda A.\ \neg A \Vdash \bot$›
**proof**
**fix** *A ps qs* **and** *q* :: ‹'*x fm*›
**assume** ‹$ps \rightsquigarrow\!\!\!\!\!\times\ qs$› ‹*set ps* $\subseteq A$› ‹$q \in$ *set qs*› ‹$q \in A$›
**then show** ‹$\neg \neg A \Vdash \bot$›
**by** *cases auto*
**qed**

**sublocale** *DA*: *Derivational-Alpha map-fm params-fm* ‹$\lambda$-. *True*› *alpha-class* ‹$\lambda A.\ \neg A \Vdash \bot$›
**proof** (*standard*; *safe*)
**fix** *A* **and** *ps qs* :: ‹'*x fm list*›
**assume** ‹$ps \rightsquigarrow_\alpha qs$› **and** $*$: ‹*set ps* $\subseteq A$› ‹$\neg A \Vdash \bot$› ‹*set qs* $\cup A \Vdash \bot$›
**then show** *False*
**proof** *cases*
**case** (*CImpN p q*)
**then have** ‹$A \Vdash \neg (p \longrightarrow q)$›
**using** $*(1)$ **by** *auto*
**moreover have** ‹$A \Vdash p \longrightarrow q$›
**using** *CImpN*(*2*) $*$ *Boole*[*of q* ‹$\{p\} \cup A$›] **by** *auto*
**ultimately show** *?thesis*
**using** $*$ **by** *blast*
**qed**
**qed**

**sublocale** *DB*: *Derivational-Beta map-fm params-fm* ‹$\lambda$-. *True*› *beta-class* ‹$\lambda A.\ \neg A \Vdash \bot$›
**proof**
**fix** *A* **and** *ps qs* :: ‹'*x fm list*›
**assume** ‹$ps \rightsquigarrow_\beta qs$› **and** $*$: ‹*set ps* $\subseteq A$› ‹$\neg A \Vdash \bot$›
**then show** ‹$\exists q \in$ *set qs*. $\neg \{q\} \cup A \Vdash \bot$›

**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* ‹λ-. *True*› *G.classify-UNIV* ‹λ*A*. ¬ *A* ⊩ ⊥›
**proof** (*unfold-locales*; *safe*)
**fix** *A qs t* **and** *ps* :: ‹'*x fm list*›
**assume** ‹*ps* ∼$_\gamma$ *qs*› **and** ∗: ‹*set ps* ⊆ *A*› ‹¬ *A* ⊩ ⊥› ‹*set* (*qs t*) ∪ *A* ⊩ ⊥›
**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* ‹λ-. *True*› *G$_P$.classify-UNIV* ‹λ*A*. ¬ *A* ⊩ ⊥›
**proof** (*unfold-locales*; *safe*)
**fix** *A qs t* **and** *ps* :: ‹'*x fm list*›
**assume** ‹*ps* ∼$_{\gamma P}$ *qs*› **and** ∗: ‹*set ps* ⊆ *A*› ‹¬ *A* ⊩ ⊥› ‹*set* (*qs t*) ∪ *A* ⊩ ⊥›
**then show** *False*
**proof** *cases*
**case** (*CAllPP p*)
**then show** *?thesis*
**using** ∗ *UniE$_P$*[*of A p t*] *ImpI* **by** *auto*
**qed**
**qed**

**sublocale** *DGF*: *Derivational-Gamma map-sym map-fm params-fm* ‹λ-. *True*› *G$_F$.classify-UNIV* ‹λ*A*. ¬ *A* ⊩ ⊥›
**proof** (*unfold-locales*; *safe*)
**fix** *A qs t* **and** *ps* :: ‹'*x fm list*›
**assume** ‹*ps* ∼$_{\gamma F}$ *qs*› **and** ∗: ‹*set ps* ⊆ *A*› ‹¬ *A* ⊩ ⊥› ‹*set* (*qs t*) ∪ *A* ⊩ ⊥›
**then show** *False*
**proof** *cases*
**case** (*CAllFP p*)
**then show** *?thesis*
**using** ∗ *UniE$_F$*[*of A p t*] *ImpI* **by** *auto*
**qed**
**qed**

**sublocale** *DD*: *Derivational-Delta map-fm params-fm* ‹λ-. *True*› δ ‹λ*A*. ¬ *A* ⊩ ⊥›
**proof** (*standard*; *safe*)

**fix** $A$ $a$ **and** $p$ :: ‹$'x$ $fm$›
**assume** ‹$p \in A$› ‹$a \notin P.params\ A$› ‹$\neg\ A \Vdash \bot$› ‹$set\ (\delta\ p\ a) \cup A \Vdash \bot$›
**then show** *False*
**proof** (*induct p a rule*: $\delta.induct$)
**case** (*1 p x*)
**then have** ‹$x \notin P.params\ (\{p\} \cup A)$›
**by** *auto*
**moreover have** ‹$A \Vdash \langle\star x/0\rangle\ p$›
**using** *1*(*4*) *Boole* **by** *auto*
**ultimately show** *?case*
**using** *1 UniI* **by** *blast*
**next**
**case** (*2 p x*)
**then have** ‹$x \notin P.params\ (\{p\} \cup A)$›
**by** *auto*
**moreover have** ‹$A \Vdash \langle\bigcirc_2\ x/0\rangle_P\ p$›
**using** *2*(*4*) *Boole* **by** *auto*
**ultimately show** *?case*
**using** *2* $UniI_P$ **by** *blast*
**next**
**case** (*3 p x*)
**then have** ‹$x \notin P.params\ (\{p\} \cup A)$›
**by** *auto*
**moreover have** ‹$A \Vdash \langle\bigcirc_2\ x/0\rangle_F\ p$›
**using** *3*(*4*) *Boole* **by** *auto*
**ultimately show** *?case*
**using** *3* $UniI_F$ **by** *blast*
**qed** *simp-all*
**qed**

**sublocale** *Derivational-Consistency map-fm params-fm* ‹$\lambda$-. *True*› *Kinds* ‹$\lambda A.\ \neg\ A \Vdash \bot$›
**using** $prop_E$-*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** $p$ :: ‹$'x$ $fm$›
**assumes** *mod*: ‹$\bigwedge(E :: nat \Rightarrow 'x\ tm)\ E_F\ E_P\ C\ F\ G\ PS\ FS.\ wf\text{-}model\ (E,\ E_F,\ E_P,\ C,\ F,\ G,\ PS,\ FS) \Longrightarrow$
$(\forall\ q \in A.\ (E,\ E_F,\ E_P,\ C,\ F,\ G,\ PS,\ FS) \models q) \Longrightarrow (E,\ E_F,\ E_P,\ C,\ F,\ G,\ PS,\ FS) \models p$›
**and** ‹$P.enough\text{-}new\ A$›
**shows** ‹$A \Vdash p$›
**proof** (*rule ccontr*)
**assume** ‹$\neg\ A \Vdash p$›
**then have** $*$: ‹$\neg\ \{\neg\ p\} \cup A \Vdash \bot$›
**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 hdom_F-def by simp

  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 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* ⇒ *'a set* ⇒ *'b set* ⇒ (*'a* ⇒ *'b*) *set*
(‹(‹*indent=3 notation=‹binder* Π∈››Π -∈-./ -)› *10*)

## 4.1 Syntax

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

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

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

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

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

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

## 4.2 Semantics

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

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

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

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

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

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

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

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

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

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

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

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

**lemma** *wf-inter*: ‹$X \in \Pi\ M \Longrightarrow Y \in \Pi\ M \Longrightarrow X \cap Y \in \Pi\ M$›
**using** *wf* **unfolding** *wf-model-def wf-gframe-def admissible-def unfolds* **by** *blast*

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

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

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

**lemma** *wf-*$\Pi$: ‹$P \in \Pi\ M \Longrightarrow P \subseteq \mathcal{W}\ M$›
**using** *wf* **unfolding** *wf-gframe-def wf-model-def unfolds* **by** *fast*

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

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

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

**lemma** *wf-*$\mathcal{V}'$: ‹$\mathcal{V}\ M\ n \subseteq \mathcal{W}\ M$›
**using** *wf wf-*$\mathcal{V}$ *wf-*$\Pi$ **by** *blast*

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

**lemma** *wf-*$\mathfrak{V}'$: ‹$\mathfrak{V}\ M\ n \subseteq \mathcal{W}\ M$›
**using** *wf wf-*$\mathfrak{V}$ *wf-*$\Pi$ **by** *blast*

**end**

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

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

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

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

**lemma** ‹(M, w) ⊨ p ⟶ q ⟷ (M, w) ⊨ p ⟶ (M, w) ⊨ q›
**by** *simp*

**lemma** ‹(M, w) ⊨ ↓(·(#0))›
**by** (*cases M*) *simp*

**lemma** ‹(M, w) ⊨ ∀ (@(○k) (·(#0))) ⟶ @(○k) (∀ (·(#0)))›
**by** (*cases M*) *auto*

**lemma** ‹(M, w) ⊨ @(○k) (∀ (·(#0))) ⟶ ∀ (@(○k) (·(#0)))›
**by** (*cases M*) *auto*

**lemma** ‹*wf-model M* ⟹ (M, w) ⊨ ∀ (·(#0)) ⟶ ·P›
**unfolding** *wf-model-def wf-env-def* **by** (*fastforce split*: *tm.splits*)

**lemma** ‹*wf-model M* ⟹ (M, w) ⊨ ∀ (@(○k) (·(#0))) ⟶ ↓(·(○k) ⟶ @(#0) (·P))›
**unfolding** *wf-model-def wf-env-def* **by** (*fastforce split*: *tm.splits*)

## 4.3 Operations

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

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

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

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

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

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

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

## Nominals

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

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

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

## Propositions

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

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

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

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

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

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

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

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

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

| ‹*qs-fm* (∀ *p*) = *qs-fm p* + *1*›

### 4.3.1 Lemmas

#### Finite

**lemma** *finite-symbols-tm* [*simp*]: ‹*finite* (*symbols-tm t*)›
**by** (*induct t*) *simp-all*

**lemma** *finite-symbols-fm* [*simp*]: ‹*finite* (*symbols-fm p*)›
**by** (*induct p*) *simp-all*

**lemma** *finite-symbols-lbd*: ‹*finite* (*symbols-lbd p*)›
**by** (*cases p*) *simp*

#### Terms

**lemma** *env* [*simp*]: ‹*P* ((*x* ≫ *E*) *n*) = (*P x* ≫ λ*n*. *P* (*E n*)) *n*›
**by** (*induct n*) *simp-all*

**lemma** *lift-lemma* [*simp*]: ‹*case-tm* (*x* ≫ *e*) *s* (*lift-tm 0 t*) = *case-tm e s t*›
**by** (*induct t*) *auto*

**lemma** *sub-tm-semantics* [*simp*]: ‹*case-tm e g* (*sub-tm s t*) = *case-tm* (λ*n*. *case-tm e g* (*s n*)) *g t*›
**by** (*induct t*) (*auto split*: *tm.splits*)

**lemma** *semantics-tm-id* [*simp*]: ‹*case-tm* # ○ *t* = *t*›
**by** (*induct t*) *auto*

**lemma** *semantics-tm-id-map* [*simp*]: ‹*map* (*case-tm* # ○) *ts* = *ts*›
**by** (*auto cong*: *map-cong*)

**lemma** *map-tm-sub-tm* [*simp*]: ‹*map-tm f* (*sub-tm g t*) = *sub-tm* (*map-tm f o g*) (*map-tm f t*)›
**by** (*induct t*) *simp-all*

**lemma** *map-tm-lift-tm* [*simp*]: ‹*map-tm f* (*lift-tm m t*) = *lift-tm m* (*map-tm f t*)›
**by** (*induct t*) *simp-all*

**lemma** *semantics-tm-fresh* [*simp*]: ‹*x* ∉ *symbols-tm t* ⟹ *case-tm e* (*g*(*x* := *a*)) *t* = *case-tm e g t*›
**by** (*induct t*) *auto*

**lemma** *map-tm-inst-single*: ‹(*map-tm f o* (*u* ≫ #)) *t* = (*map-tm f u* ≫ #) *t*›
**by** (*induct t*) *auto*

#### Nominals

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

**by** (*induct p arbitrary*: *s*) *simp-all*

**lemma** *semantics-symbols-cong-nom*:
**assumes** ‹∀ *i* ∈ *symbols-fm p*. *N i* = *N′ i*›
**shows** ‹(*Model W R PI N e V f*, *w*) ⊨ *p* ⟷ (*Model W R PI N′ e V f*, *w*) ⊨ *p*›
**using** *assms* **by** (*induct p arbitrary*: *e f w*) (*simp-all split*: *tm.splits*)

**corollary** *semantics-symbols-other-nom* [*simp*]:
**assumes** ‹*i* ∉ *symbols-fm p*›
**shows** ‹(*Model W R PI* (*N*(*i* := *x*)) *e V f*, *w*) ⊨ *p* ⟷ (*Model W R PI N e V f*, *w*) ⊨ *p*›
**using** *assms* **by** (*simp add*: *semantics-symbols-cong-nom*)

**lemma** *sub-nom-semantics* [*simp*]:
‹(*Model W R PI N e V f*, *w*) ⊨ *sub-nom s p* ⟷ (*Model W R PI N* (λ*n*. *case-tm e N* (*s n*)) *V f*, *w*) ⊨ *p*›
**by** (*induct p arbitrary*: *e f s w*) (*simp-all split*: *tm.splits*)

**lemma** *map-fm-sub-nom*: ‹*map-fm f* (*sub-nom s p*) = *sub-nom* (*map-tm f o s*) (*map-fm f p*)›
**by** (*induct p arbitrary*: *s*) (*simp-all add*: *comp-def*)

**lemma** *map-fm-inst-single-nom* [*simp*]: ‹*map-fm f* (⟨*t*⟩$_i$ *p*) = ⟨*map-tm f t*⟩$_i$ (*map-fm f p*)›
**unfolding** *map-fm-sub-nom map-tm-inst-single* **..**

## Propositions

**lemma** *semantics-symbols-cong-pro*:
‹∀ *P* ∈ *symbols-fm p*. *V P* = *V′ P* ⟹
(*Model W R PI N e V f*, *w*) ⊨ *p* ⟷ (*Model W R PI N e V′ f*, *w*) ⊨ *p*›
**by** (*induct p arbitrary*: *e f w*) (*simp-all split*: *tm.splits*)

**corollary** *semantics-symbols-other-pro* [*simp*]:
**assumes** ‹*P* ∉ *symbols-fm p*›
**shows** ‹(*Model W R PI N e* (*V*(*P* := *x*)) *f*, *w*) ⊨ *p* ⟷ (*Model W R PI N e V f*, *w*) ⊨ *p*›
**using** *assms* **by** (*simp add*: *semantics-symbols-cong-pro*)

**lemma** *semantics-symbols-lbd-cong-pro*:
‹∀ *P* ∈ *symbols-lbd* (*i*, *p*). *V P* = *V′ P* ⟹
(*Model W R PI N e V f*, *w*) ⊨ *p* ⟷ (*Model W R PI N e V′ f*, *w*) ⊨ *p*›
**by** (*simp add*: *semantics-symbols-cong-pro*)

**lemma** *map-lift-fm-pro* [*simp*]: ‹*map-fm f* (*lift-fm-pro m p*) = *lift-fm-pro m* (*map-fm f p*)›
**by** (*induct p arbitrary*: *m*) *simp-all*

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

(*map-fm f p*)›
**by** (*induct p arbitrary*: *m*) *simp-all*

**lemma** *map-fm-sub-pro*: ‹*map-fm f* (*sub-pro s p*) = *sub-pro* (*map-fm f o s*) (*map-fm f p*)›
**by** (*induct p arbitrary*: *s*) (*simp-all add*: *comp-def split*: *tm.splits*)

**lemma** *map-fm-inst-single*: ‹(*map-fm f o* (*q* ≫ · *o* #)) *p* = (*map-fm f q* ≫ · *o* #) *p*›
**by** (*induct p*) *auto*

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

### 4.3.2 softqdf

**lemma** *softqdf-map-fm* [*simp*]: ‹*softqdf* (*map-fm f p*) ⟷ *softqdf p*›
**by** (*induct p*) *auto*

**lemma** *softqdf-lift-fm-nom* [*simp*]: ‹*softqdf* (*lift-fm-nom m q*) ⟷ *softqdf q*›
**by** (*induct q arbitrary*: *m*) *auto*

**lemma** *softqdf-lift-fm-pro* [*simp*]: ‹*softqdf* (*lift-fm-pro m q*) ⟷ *softqdf q*›
**by** (*induct q*) *auto*

### 4.3.3 Add env

**lemma** *range-add-env*:
**assumes** ‹*range f* ⊆ *A*› ‹*a* ∈ *A*›
**shows** ‹*range* (*a* ≫ *f*) ⊆ *A*›
**proof** *safe*
**fix** *x n*
**show** ‹(*a* ≫ *f*) *n* ∈ *A*›
**using** *assms* **by** (*induct n*) *auto*
**qed**

**lemma** *softqdf-add-env*: ‹*softqdf q* ⟹ *softqdf-sub* (*q* ≫ · *o* #)›
**by** (*metis add-env.simps comp-apply not0-implies-Suc softqdf.simps*(*2*))

**lemma** *wf-env-add-nom*: ‹*wf-env* (*Model W R PI N e V f*) ⟹ *w* ∈ *W* ⟹
*wf-env* (⦇$\mathcal{W}$ = *W*, $\mathcal{R}$ = *R*, Π = *PI*, $\mathcal{N}$ = *N*, $\mathfrak{N}$ = *w* ≫ *e*, $\mathcal{V}$ = *V*, $\mathfrak{V}$ = *f*⦈)›
**unfolding** *wf-env-def unfolds* **using** *range-add-env* **by** *meson*

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

*wf-env* (⦇$\mathcal{W}$ = *W*, $\mathcal{R}$ = *R*, Π = *PI*, $\mathcal{N}$ = *N*, $\mathfrak{N}$ = *e*, $\mathcal{V}$ = *V*, $\mathfrak{V}$ = *P* ≫ *f*⦈)›
**unfolding** *wf-env-def unfolds* **using** *range-add-env* **by** *meson*

**lemma** *wf-model-add-pro*:
‹*wf-model* (*Model W R PI N e V f*) ⟹ *P* ∈ *PI* ⟹ *wf-model* (*Model W R PI N e V* (*P* ≫ *f*))›
**using** *wf-env-add-pro* **unfolding** *wf-model-def wf-env-def wf-frame-def wf-gframe-def admissible-def unfolds* **by** *meson*

**lemma** *softqdf-lift-fm-nom-add-env* [*simp*]:
‹*softqdf p* ⟹ (*Model W R PI N* (*v* ≫ *e*) *V f*, *w*) ⊨ *lift-fm-nom 0 p* ⟷ (*Model W R PI N e V f*, *w*) ⊨ *p*›
**by** (*induct p arbitrary*: *e w*) (*simp-all split*: *tm.splits*)

**lemma** *softqdf-lift-fm-pro-add-env* [*simp*]:
‹*softqdf p* ⟹ (*Model W R PI N e V* (*P* ≫ *f*), *w*) ⊨ *lift-fm-pro 0 p* ⟷ (*Model W R PI N e V f*, *w*) ⊨ *p*›
**by** (*induct p arbitrary*: *e w*) (*simp-all split*: *tm.splits*)

### 4.3.4 Sizes

**lemma** *qs-fm-sub-nom* [*simp*]: ‹*qs-fm* (*sub-nom s p*) = *qs-fm p*›
**by** (*induct p arbitrary*: *s*) *auto*

**lemma** *softqdf-qs-fm* [*simp*]: ‹*softqdf q* ⟹ *qs-fm q* = *0*›
**by** (*induct q*) *simp-all*

**lemma** *softqdf-sub-pro*: ‹*softqdf-sub s* ⟹ *softqdf-sub* (•(#*0*) ≫ λ*n*. *lift-fm-pro 0* (*s n*))›
**by** (*metis add-env.simps not0-implies-Suc softqdf.simps*(*2*) *softqdf-lift-fm-pro*)

**lemma** *qs-fm-sub-pro* [*simp*]: ‹*softqdf-sub s* ⟹ *qs-fm* (*sub-pro s p*) = *qs-fm p*›
**proof** (*induct p arbitrary*: *s*)
**case** (*All p*)
**then show** *?case*
**using** *softqdf-sub-pro* **by** *force*
**qed** (*auto split*: *tm.splits*)

## 4.4 Propositional Quantification

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

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

**using** *assms* **unfolding** *worlds-def wf-model-def wf-frame-def wf-gframe-def unfolds* **by** *auto*

**lemma** *softqdf-worlds*:
**assumes** ‹*wf-model M*› ‹*softqdf p*›
**shows** ‹*worlds M p* ∈ Π *M*›
**using** *assms*
**proof** (*induct p*)
**case** (*TmI x*)
**then show** *?case*
**by** *simp*
**next**
**case** (*TmP x*)
**then have** ‹$\mathcal{V}$ *M P* ∈ Π *M*› ‹$\mathfrak{V}$ *M n* ∈ Π *M*› **for** *P n*
**using** *wf-*$\mathcal{V}$ *wf-*$\mathfrak{V}$ **by** *fastforce+*
**moreover have** ‹$\mathcal{W}$ *M* ∈ Π *M*›
**using** *TmP wf-univ* **by** *fastforce*
**ultimately have** ‹$\mathcal{V}$ *M P* ∩ $\mathcal{W}$ *M* ∈ Π *M*› ‹$\mathfrak{V}$ *M n* ∩ $\mathcal{W}$ *M* ∈ Π *M*› **for** *P n*
**using** *TmP wf-inter* **by** *fastforce+*
**then have** ‹{*w* ∈ $\mathcal{W}$ *M*. *w* ∈ $\mathcal{V}$ *M P*} ∈ Π *M*› ‹{*w* ∈ $\mathcal{W}$ *M*. *w* ∈ $\mathfrak{V}$ *M n*} ∈ Π *M*› **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** ‹(∀ *w*. (*M*, *w*) ⊨ **@***k p*) ∨ (∄ *w*. (*M*, *w*) ⊨ **@***k p*)›
**by** (*auto split*: *tm.splits*)
**then have** ‹*worlds M* (**@***k p*) = {} ∨ *worlds M* (**@***k p*) = $\mathcal{W}$ *M*›
**unfolding** *worlds-def* **by** *blast*
**then show** *?case*
**using** *Sat*(*2*) *wf-univ wf-empty* **by** *fastforce*
**next**
**case** (*Glo p*)
**then have** ‹(∀ *w*. (*M*, *w*) ⊨ **A** *p*) ∨ (∄ *w*. (*M*, *w*) ⊨ **A** *p*)›
**by** *auto*

```
  then have ‹worlds M (A p) = {} ∨ worlds M (A p) = 𝒲 M›
    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 :: ‹('x, 'w) model ⇒ 'w set set› where
  ‹sqdfs M ≡ { worlds M p |p. softqdf p }›

lemma sqdfs:
  assumes ‹wf-model M›
  shows ‹sqdfs M ⊆ Π M›
  unfolding sqdfs-def using assms softqdf-worlds by blast

lemma sqdfs-admissible:
  assumes ‹wf-model M›
  shows ‹admissible (frame.truncate M) (sqdfs M)›
  unfolding admissible-def
proof safe
  fix X
  assume ‹X ∈ sqdfs M›
  then obtain p where ‹X = worlds M p› ‹softqdf p›
    unfolding sqdfs-def by fast
  moreover from this have ‹worlds M (¬ p) ∈ sqdfs M›
    unfolding sqdfs-def by auto
  ultimately show ‹𝒲 (frame.truncate M) − X ∈ sqdfs M›
    unfolding sqdfs-def unfolds using assms by simp
next
  fix X Y
  assume ‹X ∈ sqdfs M› ‹Y ∈ sqdfs M›
  then obtain p q where ‹X = worlds M p› ‹softqdf p› ‹Y = worlds M q› ‹softqdf
q›
    unfolding sqdfs-def by fast
  moreover from this have ‹worlds M (p ∧ q) ∈ sqdfs M›
    unfolding sqdfs-def by auto
  ultimately show ‹X ∩ Y ∈ sqdfs M›
    unfolding sqdfs-def using assms by simp
next
  fix X
  assume ‹X ∈ sqdfs M›
  then obtain p where ‹X = worlds M p› ‹softqdf p›
```

**unfolding** *sqdfs-def* **by** *fast*
**moreover from** *this* **have** ‹*worlds M* (□ *p*) ∈ *sqdfs M*›
**unfolding** *sqdfs-def* **by** *auto*
**ultimately show** ‹{*w* ∈ *𝒲* (*frame.truncate M*). ∀ *v* ∈ *ℛ* (*frame.truncate M*) *w*. *v* ∈ *X*} ∈ *sqdfs M*›
**unfolding** *sqdfs-def unfolds* **using** *assms* **by** *simp*
**qed**

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

**lemma** *sub-pro-with-worlds*:
**assumes** ‹*wf-model* (*Model W R PI N e V f*)› ‹*w* ∈ *W*› ‹*softqdf-sub s*›
**shows** ‹(*Model W R PI N e V f*, *w*) ⊨ *sub-pro s p* ⟷ (*with-worlds* (*Model W R PI N e V f*) *s*, *w*) ⊨ *p*›
**using** *assms*
**proof** (*induct p arbitrary*: *s e f w*)
**case** (*Box p*)
**moreover from** *this* **have** ‹*v* ∈ *R w* ⟹ *v* ∈ *W*› **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** ‹*case-tm e N x* ∈ *W*›
**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* = ‹*lift-fm-nom 0 o s*›

**have** ‹*wf-model* (*Model W R PI N* (*w* ≫ *e*) *V f*)›
**using** *Dwn* **by** (*simp add*: *wf-model-add-nom*)
**moreover have** ‹*softqdf-sub ?s*›
**using** *Dwn* **by** *simp*
**ultimately have** ‹(*Model W R PI N* (*w* ≫ *e*) *V f*, *w*) ⊨ *sub-pro ?s p* ⟷
(*with-worlds* (*Model W R PI N* (*w* ≫ *e*) *V f*) *?s*, *w*) ⊨ *p*›
**using** *Dwn* **by** *fastforce*
**moreover have** ‹*worlds* (*Model W R PI N* (*w* ≫ *e*) *V f*) *o ?s* = *worlds* (*Model W R PI N e V f*) *o s*›
**unfolding** *worlds-def comp-def* **using** *Dwn*(*4*) **by** *simp*
**ultimately have** ∗: ‹(*Model W R PI N* (*w* ≫ *e*) *V f*, *w*) ⊨ *sub-pro ?s p* ⟷
(*Model W R PI N* (*w* ≫ *e*) *V* (*worlds* (*Model W R PI N e V f*) ∘ *s*), *w*) ⊨ *p*›
**unfolding** *with-worlds-def* **by** *simp*

```
  moreover have ‹(with-worlds (Model W R PI N e V f) ?s, w) ⊨ ↓ p ⟷
     (Model W R PI N (w ≫ e) V (worlds (Model W R PI N e V f) o ?s), w) ⊨
p›
    unfolding with-worlds-def by simp
  then have ** :‹(with-worlds (Model W R PI N e V f) ?s, w) ⊨ ↓ p ⟷
     (Model W R PI N (w ≫ e) V (worlds (Model W R PI N e V f) o ?s), w) ⊨
p›
    by metis

  ultimately show ?case
    unfolding with-worlds-def by simp
next
  case (All p)
  let ?s = ‹•(#0) ≫ λn. lift-fm-pro 0 (s n)›

  have ‹softqdf (?s n)› for n
    using All(4) by (induct n) auto
  then have ‹softqdf-sub ?s›
    by blast
  moreover have ‹∀ P ∈ PI. wf-model (Model W R PI N e V (P ≫ f))›
    using All(2) wf-model-add-pro by blast
  ultimately have ‹(∀ P ∈ PI. (Model W R PI N e V (P ≫ f), w) ⊨ sub-pro ?s
p) ⟷
    (∀ P ∈ PI. (with-worlds (Model W R PI N e V (P ≫ f)) ?s, w) ⊨ p)›
    using All by blast

  moreover have ‹∀ P ∈ PI. P ⊆ W›
    using All(2) unfolding wf-model-def wf-gframe-def unfolds by blast
  then have ‹P ∈ PI ⟹
    (worlds (Model W R PI N e V (P ≫ f)) o ?s) n = (P ≫ worlds (Model W R
PI N e V f) o s) n› for P n
    unfolding worlds-def using All(4) by (induct n) auto
  then have ‹P ∈ PI ⟹
     (worlds (Model W R PI N e V (P ≫ f)) o ?s) = (P ≫ worlds (Model W R
PI N e V f) o s)› 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 ‹wf-model (Model W R PI N e V f)›
 shows ‹worlds (Model W R PI N e V f) (• (# n)) = f n›
  using wf-𝔘'[OF assms] unfolding worlds-def unfolds by auto

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

*PI N e V f) q ≫ f)*›
**unfolding** *comp-def env worlds-id-sub*[*OF assms*] **..**

**corollary** *inst-single-worlds*:
**assumes** ‹*wf-model* (*Model W R PI N e V f*)› ‹*w* ∈ *W*› ‹*softqdf q*›
**shows**
‹(*Model W R PI N e V f*, *w*) ⊨ ⟨*q*⟩$_p$ *p* ⟷
(*Model W R PI N e V* (*worlds* (*Model W R PI N e V f*) *q* ≫ *f*), *w*) ⊨ *p*›
**proof** −
**have**
‹(*Model W R PI N e V f*, *w*) ⊨ ⟨*q*⟩$_p$ *p* ⟷
(*Model W R PI N e V* (*worlds* (*Model W R PI N e V f*) *o* (*q* ≫ • *o* #)), *w*)
⊨ *p*›
**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** *confl-class* :: ‹'*x lbd list* ⇒ '*x lbd list* ⇒ *bool*› (**infix** ‹⇝✗› *50*) **where**
*CNegP*: ‹[ (*i*, ¬ •*P*) ] ⇝✗ [(*i*, •*P*)]›
| *CNegI*: ‹[ (*i*, ¬ •*k*) ] ⇝✗ [(*i*, •*k*)]›

**inductive** *alpha-class* :: ‹'*x lbd list* ⇒ '*x lbd list* ⇒ *bool*› (**infix** ‹⇝$_\alpha$› *50*) **where**
*CSym*: ‹[ (*i*, •*k*) ] ⇝$_\alpha$ [(*k*, •*i*)]›
| *CNom*: ‹[ (*i*, •*k*), (*i*, *p*) ] ⇝$_\alpha$ [(*k*, *p*)]›
| *CNegN*: ‹[ (*i*, ¬ ¬ *p*) ] ⇝$_\alpha$ [(*i*, *p*)]›
| *CConP*: ‹[ (*i*, *p* ∧ *q*) ] ⇝$_\alpha$ [(*i*, *p*), (*i*, *q*)]›
| *CSatP*: ‹[ (*i*, @*k p*) ] ⇝$_\alpha$ [(*k*, *p*)]›
| *CSatN*: ‹[ (*i*, ¬ @*k p*) ] ⇝$_\alpha$ [(*k*, ¬ *p*)]›
| *CBoxP*: ‹[ (*i*, □*p*), (*i*, ◇(•*k*)) ] ⇝$_\alpha$ [(*k*, *p*)]›
| *CDwnP*: ‹[ (*i*, ↓ *p*) ] ⇝$_\alpha$ [ (*i*, ⟨*i*⟩$_i$ *p*) ]›
| *CDwnN*: ‹[ (*i*, ¬ ↓ *p*) ] ⇝$_\alpha$ [ (*i*, ¬ ⟨*i*⟩$_i$ *p*) ]›

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

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

**inductive** *gamma-class-fm* :: ‹'*x lbd list* ⇒ ('*x lbd set* ⇒ '*x fm set*) × ('*x fm* ⇒ -) ⇒ *bool*› (**infix** ‹⇝$_{\gamma p}$› *50*) **where**
*CAllP*: ‹[ (*i*, ∀ *p*) ] ⇝$_{\gamma p}$ (λ-. {*q*. *softqdf q*}, λ*q*. [ (*i*, ⟨*q*⟩$_p$ *p*) ])›

**fun** δ :: ‹'*x lbd* ⇒ '*x* ⇒ '*x lbd list*› **where**

*CBoxN*: ‹$\delta$ $(i, \neg\ \Box p)$ $k = [(\bigcirc k, \neg\ p), (i, \Diamond\ (\cdot\ (\bigcirc\ k)))]$›
| *CGloN*: ‹$\delta$ $(i, \neg\ \mathbf{A}\ p)$ $k = [\ (\bigcirc k, \neg\ p)\ ]$›
| *CAllN*: ‹$\delta$ $(i, \neg\ \forall\ p)$ $P = [\ (i, \neg\ \langle\cdot(\bigcirc P)\rangle_p\ p)\ ]$›
| ‹$\delta$ - - = []›

**interpretation** *P*: *Params map-lbd symbols-lbd* ‹$\lambda$-. *True*›
**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* ‹$\lambda$-. *True*› *confl-class*
**by** *unfold-locales* (*auto elim*!: *confl-class.cases intro*: *confl-class.intros*)

**interpretation** *A*: *Alpha map-lbd symbols-lbd* ‹$\lambda$-. *True*› *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* ‹$\lambda$-. *True*› *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* ‹$\lambda$-. *True*› *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* ‹$\lambda$-. *True*› *gamma-class-fm*
**by** *unfold-locales* (*fastforce elim*: *gamma-class-fm.cases intro*: *gamma-class-fm.intros*)+

**interpretation** *D*: *Delta map-lbd symbols-lbd* ‹$\lambda$-. *True*› $\delta$
**proof**
**show** ‹$\bigwedge f.\ \delta\ (map\text{-}lbd\ f\ p)\ (f\ x) = map\ (map\text{-}lbd\ f)\ (\delta\ p\ x)$› **for** $p$ :: ‹$'x$ *lbd*› **and** $x$
**by** (*induct p x rule*: $\delta$.*induct*) *simp-all*
**qed**

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

**lemma** *prop$_E$-Kinds*:
**assumes** ‹*P.sat$_E$ C.kind C*› ‹*P.sat$_E$ A.kind C*› ‹*P.sat$_E$ B.kind C*› ‹*P.sat$_E$ GI.kind C*› ‹*P.sat$_E$ GP.kind C*› ‹*P.sat$_E$ D.kind C*›
**shows** ‹*P.prop$_E$ Kinds C*›
**unfolding** *P.prop$_E$-def* **using** *assms* **by** *simp*

**interpretation** *Consistency-Kinds map-lbd symbols-lbd* ‹$\lambda$-. *True*› *Kinds*
**using** *P.Params-axioms C.Consistency-Kind-axioms A.Consistency-Kind-axioms B.Consistency-Kind-axioms GI.Consistency-Kind-axioms GP.Consistency-Kind-axioms D.Consistency-Kind-axioms*
**by** (*auto intro*: *Consistency-Kinds.intro simp*: *Consistency-Kinds-axioms-def*)

**interpretation** *Maximal-Consistency map-lbd symbols-lbd* ‹$\lambda$-. *True*› *Kinds*
**proof**

**have** ‹*infinite* (*UNIV* :: *'x fm set*)›
  **using** *infinite-UNIV-size*[*of* ‹$\lambda p.\ p \longrightarrow p$›] **by** *simp*
**then show** ‹*infinite* (*UNIV* :: *'x lbd set*)›
  **using** *finite-prod* **by** *blast*
**qed** *simp*

**context begin**

**lemma** ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \cdot P) \notin S \vee (i, \neg \cdot P) \notin S$›
  **using** $sat_E$[*of C.kind*] **by** (*force intro*: *CNegP*)

**lemma** ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, p \wedge q) \in S \Longrightarrow \{(i, p), (i, q)\} \cup S \in C$›
  **using** $sat_E$[*of A.kind*] **by** (*force intro*: *CConP*)

**lemma** ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \neg (p \wedge q)) \in S \Longrightarrow \{(i, \neg p)\} \cup S \in C \vee \{(i, \neg q)\} \cup S \in C$›
  **using** $sat_E$[*of B.kind*] **by** (*force intro*: *CConN*)

**lemma** ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \Box p) \in S \Longrightarrow (i, \Diamond(\cdot k)) \in S \Longrightarrow \{(k, p)\} \cup S \in C$›
  **using** $sat_E$[*of A.kind*] **by** (*force intro*: *CBoxP*)

**lemma** ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \neg \Box p) \in S \Longrightarrow \exists k.\ \{(k, \neg p), (i, \Diamond (\cdot k))\} \cup S \in C$›
  **using** $sat_E$[*of D.kind*] **by** *fastforce*

**lemma** ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow \{\ (i, \cdot i)\ \} \cup S \in C$›
  **using** $sat_E$[*of GI.kind*] **by** (*force intro*: *CRefl*)

**lemma** ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \mathbf{A}\ p) \in S \Longrightarrow \{\ (k, p)\ \} \cup S \in C$›
  **using** $sat_E$[*of GI.kind*] **by** (*force intro*: *CGloP*)

**lemma** ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \neg\ \mathbf{A}\ p) \in S \Longrightarrow \exists k.\ \{\ (k, \neg p)\ \} \cup S \in C$›
  **using** $sat_E$[*of D.kind*] **by** *fastforce*

**lemma** ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \forall\ p) \in S \Longrightarrow$ *softqdf* $q \Longrightarrow \{\ (i, \langle q \rangle_p\ p)\ \} \cup S \in C$›
  **using** $sat_E$[*of GP.kind*] **by** (*force intro*: *CAllP*)

**lemma** ‹$P.prop_E$ *Kinds* $C \Longrightarrow S \in C \Longrightarrow (i, \neg\ \forall\ p) \in S \Longrightarrow \exists P.\ \{\ (i, \neg \langle \cdot(\bigcirc P) \rangle_p\ p)\ \} \cup S \in C$›
  **using** $sat_E$[*of D.kind*] **by** *fastforce*

**end**

**definition** *equiv-nom* :: ‹*'x lbd set* ⇒ *'x tm* ⇒ *'x tm* ⇒ *bool*› **where**
  ‹*equiv-nom S i k* ≡ $(i, \cdot k) \in S$›

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

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

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

**lemma** *range-val-ne*: ‹range (val S) ≠ {}›
**unfolding** *val-def* **by** *blast*

**lemma** *admissible-Pow*: ‹admissible F (Pow (𝒲 F))›
**unfolding** *admissible-def* **by** *blast*

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

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

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

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

**lemma** *mono-grow*: ‹mono (grow F B)›
**unfolding** *adm-δ-def mono-def* **by** (*auto simp*: *subset-eq*)

**lemma** *admissible-δ*: ‹admissible F B ⟷ adm-δ F B ⊆ B›
**unfolding** *admissible-def adm-δ-def* **by** *auto*

**lemma** *admit-B*: ‹B ⊆ admit F B›
**unfolding** *admit-def* **by** (*meson le-sup-iff lfp-greatest*)

**lemma** *admit-Pow*: ‹B ⊆ Pow (𝒲 F) ⟹ admit F B ⊆ Pow (𝒲 F)›
**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*: ‹admissible F B ⟷ grow F B B = B›
**using** *admissible-δ* **by** *auto*

**lemma** *lfp-grow*: ‹grow F B (lfp (grow F B)) = lfp (grow F B)›
**using** *lfp-unfold mono-grow* **by** *blast*

**lemma** *admit-admissible*: ‹*admissible F* (*admit F B*)›
**unfolding** *admit-def*
**proof** −
**have** *B* ∪ *lfp* (*grow F B*) = *lfp* (*grow F B*)
**using** *lfp-grow* **by** *blast*
**then show** *admissible F* (*lfp* (*grow F B*))
**using** *lfp-grow*[*of B F*] **by** (*simp add*: *admissible-δ sup.order-iff*)
**qed**

**lemma** *admits-admit*: ‹*B* ≠ {} ⟹ *B* ⊆ *Pow* (*W F*) ⟹ *admits F B* (*admit F B*)›
**by** (*metis admit-B admit-Pow admit-admissible admits-def subset-empty*)

**lemma** *admissible-admit*:
**assumes** ‹*B* ≠ {}› ‹*B* ⊆ *Pow* (*W F*)›
**shows**
‹*admit F B* ≠ {}›
‹*admit F B* ⊆ *Pow* (*W F*)›
‹*admissible F* (*admit F B*)›
**using** *assms admits-admit* **unfolding** *admits-def* **by** *metis*+

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

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

**definition** *canonical* :: ‹'*x lbd set* ⇒ ('*x*, '*x tm*) *model*› **where**
‹*canonical S* ≡
*gframe.extend* (*canonical-gframe S*)
(| *N* = λ*i*. [◯*i*]*S*,
𝔑 = λ*i*. [#*i*]*S*,
*V* = *val S o* ◯,
𝔙 = *val S o* #
|)›

**lemma** *wf-canonical-frame*: ‹*wf-frame* (*canonical-frame S*)›
**unfolding** *wf-frame-def unfolds reach-def* **by** *auto*

**lemma** *val-Pow*: ‹*range* (*val S*) ⊆ *Pow* (*W* (*canonical-frame S*))›
**unfolding** *val-def* **by** *auto*

**lemma** *wf-cannonical-gframe*: ‹*wf-gframe* (*canonical-gframe S*)›
**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*: ‹*admits* (*canonical-frame S*) (*range* (*val S*)) *PI* ⟹ *val S P* ∈ *PI*›
**unfolding** *admits-def* **by** *blast*

**lemma** *admit-val*: ‹*val S P* ∈ *admit* (*canonical-frame S*) (*range* (*val S*))›
**using** *admits-val admits-admit val-Pow* **by** (*simp add*: *admit-B range-subsetD*)

**lemma** *wf-canonical-env*: ‹*wf-env* (*canonical S*)›
**unfolding** *wf-env-def canonical-def unfolds* **using** *admit-val* **by** *auto*

**lemma** *wf-gframe-canonical*: ‹*wf-gframe* (*gframe.truncate* (*canonical S*))›
**using** *wf-cannonical-gframe* **unfolding** *canonical-def unfolds* **.**

**lemma** *wf-canonical*: ‹*wf-model* (*canonical S*)›
**unfolding** *wf-model-def* **using** *wf-gframe-canonical wf-canonical-env* **by** *blast*

**lemma** *admissible-sqdfs*: ‹*admissible* (*canonical-frame S*) (*sqdfs* (*canonical S*))›
**using** *sqdfs-admissible*[*OF wf-canonical*[*of S*]]
**unfolding** *canonical-def unfolds* **.**

**lemma** *sqdfs-Pow*: ‹*sqdfs* (*canonical S*) ⊆ *Pow* (*𝒲* (*canonical-frame S*))›
**unfolding** *sqdfs-def canonical-def unfolds worlds-def* **by** *blast*

**lemma** *val-sqdfs*: ‹*val S P* ∈ *sqdfs* (*canonical S*)›
**unfolding** *val-def sqdfs-def canonical-def unfolds worlds-def*
**by** (*auto split*: *tm.splits intro*!: *exI*[*of* - ‹• *P*›])

**lemma** *admits-canonical-sqdfs*: ‹*admits* (*canonical-frame S*) (*range* (*val S*)) (*sqdfs* (*canonical S*))›
**unfolding** *admits-def* **using** *admissible-sqdfs sqdfs-Pow val-sqdfs* **by** *blast*

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

**lemma** *sqdfs-canonical*: ‹*sqdfs* (*canonical S*) = Π (*canonical S*)›
**proof**
**show** ‹*sqdfs* (*canonical S*) ⊆ Π (*canonical S*)›
**using** *sqdfs wf-canonical* **by** *blast*
**next**
**have** ‹*grow* (*canonical-frame S*) (*range* (*val S*)) (*sqdfs* (*canonical S*)) ⊆ *sqdfs* (*canonical S*)›
**using** *admissible-grow*[*of* ‹*canonical-frame S*› ‹*sqdfs* -›] *admits-canonical-sqdfs*[*of S*] **unfolding** *admits-def*
**by** (*metis* (*no-types*, *lifting*) *equalityE le-iff-sup*)
**then have** ‹*admit* (*canonical-frame S*) (*range* (*val S*)) ⊆ *sqdfs* (*canonical S*)›
**by** (*simp add*: *admit-def lfp-lowerbound*)
**then show** ‹Π (*canonical S*) ⊆ *sqdfs* (*canonical S*)›
**unfolding** *canonical-def unfolds* **.**
**qed**

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

**corollary** *canonical-tm-eta′* [*simp*]: ‹*case-tm* (𝔑 (*canonical S*)) (𝒩 (*canonical S*)) *k* = [*k*]$_S$›
**unfolding** *canonical-def unfolds* **by** *simp*

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

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

**lemma** *Nom-refl*: ‹(*i*, •*i*) ∈ *S*›
**using** *gammaI* **by** (*fastforce intro*: *CRefl*)

**lemma** *Nom-sym*:
**assumes** ‹(*i*, •*k*) ∈ *S*›
**shows** ‹(*k*, •*i*) ∈ *S*›
**using** *assms alpha* **by** (*fastforce intro*: *CSym*)

**lemma** *Nom-trans*:
**assumes** ‹(*i*, •*j*) ∈ *S*› ‹(*j*, •*k*) ∈ *S*›
**shows** ‹(*i*, •*k*) ∈ *S*›
**using** *assms*
**proof** −
**have** ‹(*j*, •*i*) ∈ *S*›
**using** *assms Nom-sym* **by** *blast*
**then show** *?thesis*
**using** *assms*(*2*) *alpha* **by** (*fastforce intro*: *CNom*)
**qed**

**lemma** *equiv-nom-ne*: ‹{*k*. *equiv-nom S i k*} ≠ {}›
**unfolding** *equiv-nom-def* **using** *Nom-refl* **by** *blast*

**lemma** *equiv-nom-assign*: ‹*equiv-nom S i* ([*i*]$_S$)›
**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** ‹*equiv-nom S i k*› ‹(*i*, *p*) ∈ *S*›

```
  shows ‹(k, p) ∈ S›
proof −
  have ‹(i, •k) ∈ S›
    using assms unfolding equiv-nom-def by blast
  then show ?thesis
    using assms alpha by (fastforce intro: CNom)
qed

lemma assign-in-W: ‹[i]_S ∈ 𝒲 (canonical S)›
  unfolding canonical-def gframe.defs frame.defs unfolds by blast

theorem model: ‹((i, p) ∈ S ⟶ ⟦S, i⟧ ⊨ p) ∧ ((i, ¬ p) ∈ S ⟶ ¬ ⟦S, i⟧ ⊨ p)›
proof (induct p arbitrary: i rule: wf-induct[where r=‹measures [qs-fm, sz-fm]›])
  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 ‹x = •k› ‹(i, •k) ∈ S›
      moreover from this have ‹(k, •i) ∈ S›
        using Nom-sym by fast
      ultimately have ‹[i]_S = [k]_S›
        using Nom-trans unfolding assign-def equiv-nom-def by metis
      then show ‹⟦S, i⟧ ⊨ •k›
        unfolding canonical-ctx-def canonical-def gframe.defs
        by (simp split: tm.splits)
    next
      assume ‹x = •k› ‹(i, ¬ •k) ∈ S›
      then have ‹(i, •k) ∉ S›
        using confl by (fastforce intro: CNegI)
      then have ‹¬ equiv-nom S i k›
        unfolding equiv-nom-def by blast
      moreover have ‹(i, •k) ∉ S›
        using ‹(i, ¬ •k) ∈ S› using confl by (fastforce intro: CNegI)
      ultimately have ‹[i]_S ≠ [k]_S›
        using Nom-sym Nom-trans equiv-nom-assign unfolding equiv-nom-def by
metis
      then show ‹¬ ⟦S, i⟧ ⊨ •k›
        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 ‹x = ·P› ‹(i, ·P) ∈ S›
    then show ‹⟦S, i⟧ ⊨ ·P›
      unfolding canonical-ctx-def canonical-def val-def gframe.defs
      by (auto split: tm.splits)
  next
    assume ‹x = ·P› ‹(i, ¬ ·P) ∈ S›
    then have ‹(i, ·P) ∉ S›
      using confl by (fastforce intro: CNegP)
    then have ‹⋀k. [k]_S = [i]_S ⟹ (k, ·P) ∉ S›
      by (metis Nom-refl equiv-nom-Nom equiv-nom-assign equiv-nom-def)
    then show ‹¬ ⟦S, i⟧ ⊨ ·P›
      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 ‹x = ¬ p› ‹(i, ¬ p) ∈ S›
    then show ‹⟦S, i⟧ ⊨ ¬ p›
      using 2 unfolding canonical-ctx-def by simp
  next
    assume ‹x = ¬ p› ‹(i, ¬ ¬ p) ∈ S›
    then have ‹(i, p) ∈ S›
      using alpha by (fastforce intro: CNegN)
    then show ‹¬ ⟦S, i⟧ ⊨ ¬ p›
      using 2 Neg unfolding canonical-ctx-def by auto
  qed
next
  case (Con p q)
  then show ?thesis
  proof (safe del: notI)
    assume ‹x = (p ∧ q)› ‹(i, p ∧ q) ∈ S›
    then have ‹(i, p) ∈ S ∧ (i, q) ∈ S›
      using alpha by (fastforce intro: CConP)
    moreover have
      ‹(p, x) ∈ measures [qs-fm, sz-fm]›
      ‹(q, x) ∈ measures [qs-fm, sz-fm]›
      using Con by auto
    ultimately show ‹⟦S, i⟧ ⊨ (p ∧ q)›
      using 2 unfolding canonical-ctx-def by auto
  next
    assume ‹x = (p ∧ q)› ‹(i, ¬ (p ∧ q)) ∈ S›
    then have ‹(i, ¬ p) ∈ S ∨ (i, ¬ q) ∈ S›
      using beta by (fastforce intro: CConN)
    moreover have
      ‹(¬ p, ¬ x) ∈ measures [qs-fm, sz-fm]›
      ‹(¬ q, ¬ x) ∈ measures [qs-fm, sz-fm]›
```

```
    using Con by auto
  ultimately show ‹¬ ⟦S, i⟧ ⊨ (p ∧ q)›
    using 2 unfolding canonical-ctx-def by auto
 qed
next
 case (Box p)
 then show ?thesis
 proof (safe del: notI)
  assume ‹x = □ p› ‹(i, □ p) ∈ S›
  {
   fix k
   assume ‹[k]_S ∈ reach S ([i]_S)›
   then obtain k' where ‹([i]_S, ◇ (·k')) ∈ S› ‹[k']_S = [k]_S›
    unfolding reach-def by auto
   then have ‹(i, ◇ (·k')) ∈ S›
    using Nom-sym equiv-nom-Nom equiv-nom-assign equiv-nom-def by blast
   then have ‹(k', p) ∈ S›
    using ‹(i, □ p) ∈ S› alpha by (fastforce intro: CBoxP)
   then have ‹⟦S, k'⟧ ⊨ p›
    using 2 Box by simp
   then have ‹⟦S, k⟧ ⊨ p›
    unfolding canonical-ctx-def canonical-def using ‹[k']_S = [k]_S› by simp
  }
  then show ‹⟦S, i⟧ ⊨ □ p›
   unfolding canonical-ctx-def canonical-def reach-def gframe.defs frame.defs
   by auto
 next
  assume ‹x = □ p› ‹(i, ¬ (□ p)) ∈ S›
  then obtain k where k: ‹(k, ¬ p) ∈ S› ‹(i, ◇ (· k)) ∈ S›
   using δ by force
  then have ‹¬ ⟦S, k⟧ ⊨ p›
   using 2 Box by simp
  moreover have ‹([i]_S, ◇ (·k)) ∈ S›
   using k(2) equiv-nom-Nom equiv-nom-assign by fastforce
  then have ‹[k]_S ∈ reach S ([i]_S)›
   unfolding reach-def by blast
  ultimately show ‹¬ ⟦S, i⟧ ⊨ □ p›
   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 ‹x = @k p› ‹(i, @k p) ∈ S›
  then have ‹(k, p) ∈ S›
   using alpha by (fastforce intro: CSatP)
  then have ‹⟦S, k⟧ ⊨ p›
   using 2 Sat by simp
```

```
    then show ‹⟦S, i⟧ ⊨ @k p›
      unfolding canonical-ctx-def canonical-def gframe.defs
      by (auto split: tm.splits)
  next
    assume ‹x = @k p› ‹(i, ¬ @k p) ∈ S›
    then have ‹(k, ¬ p) ∈ S›
      using alpha by (fastforce intro: CSatN)
    then have ‹¬ ⟦S, k⟧ ⊨ p›
      using 2 Sat by simp
    then show ‹¬ ⟦S, i⟧ ⊨ @k p›
      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 ‹x = A p› ‹(i, A p) ∈ S›
    then have ‹(k, p) ∈ S› for k
      using gammaI by (fastforce intro: CGloP)
    then have ‹⟦S, k⟧ ⊨ p› for k
      using 2 Glo by simp
    then show ‹⟦S, i⟧ ⊨ A p›
      unfolding canonical-ctx-def canonical-def gframe.defs frame.defs
      by auto
  next
    assume ‹x = A p› ‹(i, ¬ A p) ∈ S›
    then have ‹∃ k. (k, ¬ p) ∈ S›
      using δ by fastforce
    then have ‹∃ k. ¬ ⟦S, k⟧ ⊨ p›
      using 2 Glo by auto
    then show ‹¬ ⟦S, i⟧ ⊨ A p›
      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 ‹x = ↓ p› ‹(i, ↓ p) ∈ S›
    then have ‹(i, ⟨i⟩_i p) ∈ S›
      using alpha by (fastforce intro: CDwnP)
    moreover have ‹(⟨i⟩_i p, x) ∈ measures [qs-fm, sz-fm]›
      using Dwn by simp
    ultimately have ‹⟦S, i⟧ ⊨ ⟨i⟩_i p›
      using 2 by (meson in-measure)
    then show ‹⟦S, i⟧ ⊨ ↓ p›
      unfolding canonical-ctx-def canonical-def gframe.defs
      by simp
```

```
    next
      assume ‹x = ↓ p› ‹(i, ¬ ↓ p) ∈ S›
      then have ‹(i, ¬ ⟨i⟩_i p) ∈ S›
        using alpha by (fastforce intro: CDwnN)
      moreover have ‹(⟨i⟩_i p, x) ∈ measures [qs-fm, sz-fm]›
        using Dwn by simp
      ultimately have ‹¬ ⟦S, i⟧ ⊨ ⟨i⟩_i p›
        using 2 by (meson in-measure)
      then show ‹¬ ⟦S, i⟧ ⊨ ↓ p›
        unfolding canonical-ctx-def canonical-def gframe.defs
        by simp
    qed
  next
    case (All p)
    then show ?thesis
    proof (safe del: notI)
      assume ‹x = ∀  p› ‹(i, ∀  p) ∈ S›
      then have ‹softqdf q ⟹ (i, ⟨q⟩_p p) ∈ S› for q
        using gammaP by (fastforce intro: CAllP)
      moreover have ‹softqdf q ⟹ (⟨q⟩_p p, x) ∈ measures [qs-fm, sz-fm]› for q
       using All by (metis less-add-one measures-less qs-fm.simps(9) qs-fm-sub-pro
softqdf-add-env)
      ultimately have ∗: ‹softqdf q ⟹ ⟦S, i⟧ ⊨ ⟨q⟩_p p› for q
        using 2 by (meson in-measure)

      moreover note wf-canonical[of S] assign-in-W[of i]
      ultimately have ‹softqdf q ⟹
        ((canonical S)(|𝔙 := (worlds (canonical S) q ≫ 𝔙 (canonical S))|), [i]_S) ⊨
p› for q
       using inst-single-worlds[where W=‹𝒲 (canonical S)› and q=q] unfolding
canonical-ctx-def canonical-def unfolds by fastforce
     then have ‹(∀ P ∈ sqdfs (canonical S). ((canonical S)(|𝔙 := (P ≫ 𝔙 (canonical
S))|), [i]_S) ⊨ p)›
       unfolding sqdfs-def by blast
     then have ‹(∀ P ∈ Π (canonical S). ((canonical S)(|𝔙 := (P ≫ 𝔙 (canonical
S))|), [i]_S) ⊨ p)›
       using sqdfs-canonical by blast
      then show ‹⟦S, i⟧ ⊨ ∀  p›
        unfolding canonical-ctx-def by simp
    next
      assume ‹x = ∀  p› ‹(i, ¬ ∀  p) ∈ S›
      then obtain P where ‹(i, ¬ ⟨• (○ P)⟩_p p) ∈ S›
        using δ by auto
      moreover have ‹(¬ ⟨• (○ P)⟩_p p, x) ∈ measures [qs-fm, sz-fm]›
        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 ‹¬ ⟦S, i⟧ ⊨ ⟨• (○ P)⟩_p p›
        using 2 unfolding canonical-ctx-def by auto
```

**moreover have** ‹*softqdf* (• (○ *P*))›
**by** *simp*
**moreover note** *wf-canonical*[*of S*] *assign-in-W*[*of i*]
**ultimately have** ‹*softqdf* (• (○ *P*)) ∧
¬ ((*canonical S*)⦇𝔙 := (*worlds* (*canonical S*) (• (○ *P*)) ≫ 𝔙 (*canonical S*))⦈, [*i*]$_S$) ⊨ *p*›
**using** *inst-single-worlds*[**where** *W*=‹𝒲 (*canonical S*)›] **unfolding** *canonical-ctx-def canonical-def unfolds*
**by** *fastforce*
**then have** ‹(∃ *P* ∈ *sqdfs* (*canonical S*). ¬ ((*canonical S*)⦇𝔙 := (*P* ≫ 𝔙 (*canonical S*))⦈, [*i*]$_S$) ⊨ *p*)›
**unfolding** *sqdfs-def* **by** *blast*
**then have** ‹(∃ *P* ∈ Π (*canonical S*). ¬ ((*canonical S*)⦇𝔙 := (*P* ≫ 𝔙 (*canonical S*))⦈, [*i*]$_S$) ⊨ *p*)›
**using** *sqdfs-canonical* **by** *blast*
**then show** ‹¬ ⟦*S*, *i*⟧ ⊨ ∀ *p*›
**unfolding** *canonical-ctx-def* **by** *simp*
**qed**
**qed**
**qed**

**end**

**theorem** *model-existence*:
**fixes** *S* :: ‹'*x lbd set*›
**assumes** ‹*P*.*prop*$_E$ *Kinds C*›
**and** ‹*S* ∈ *C*›
**and** ‹*P*.*enough-new S*›
**and** ‹(*i*, *p*) ∈ *S*›
**shows** ‹⟦*mk-mcs C S*, *i*⟧ ⊨ *p*›
**proof** −
**have** ∗: ‹*MyHintikka* (*mk-mcs C S*)›
**proof**
**show** ‹*P*.*prop*$_H$ *Kinds* (*mk-mcs C S*)›
**using** *mk-mcs-Hintikka*[*OF assms*(1−3)] *Hintikka.hintikka* **by** *blast*
**qed**
**moreover have** ‹(*i*, *p*) ∈ *mk-mcs C S*›
**using** *assms*(4) *Extend-subset* **by** *blast*
**ultimately show** *?thesis*
**using** *MyHintikka.model* **by** *blast*
**qed**

## 4.6 Natural Deduction

**inductive** *Calculus-Set* :: ‹'*x lbd set* ⇒ '*x lbd* ⇒ *bool*› (‹- ⊩ -› [50, 50] 50) **where**
*Assm* [*dest*]: ‹(*i*, *p*) ∈ *A* ⟹ *A* ⊩ (*i*, *p*)›
| *Ref* [*simp*]: ‹*A* ⊩ (*i*, •*i*)›
| *Nom* [*dest*]: ‹*A* ⊩ (*i*, •*k*) ⟹ *A* ⊩ (*i*, *p*) ⟹ *A* ⊩ (*k*, *p*)›
| *NotI* [*intro*]: ‹{(*i*, *p*)} ∪ *A* ⊩ (*i*, ⊥) ⟹ *A* ⊩ (*i*, ¬ *p*)›

| *NotE* [*elim*]: ‹$A \Vdash (i,\ \neg\ p) \Longrightarrow A \Vdash (i,\ p) \Longrightarrow A \Vdash (k,\ q)$›
| *AndI* [*intro*]: ‹$A \Vdash (i,\ p) \Longrightarrow A \Vdash (i,\ q) \Longrightarrow A \Vdash (i,\ p \wedge q)$›
| *AndD1* [*dest*]: ‹$A \Vdash (i,\ p \wedge q) \Longrightarrow A \Vdash (i,\ p)$›
| *AndD2* [*dest*]: ‹$A \Vdash (i,\ p \wedge q) \Longrightarrow A \Vdash (i,\ q)$›
| *SatI* [*intro*]: ‹$A \Vdash (i,\ p) \Longrightarrow A \Vdash (k,\ @i\ p)$›
| *SatE* [*dest*]: ‹$A \Vdash (i,\ @k\ p) \Longrightarrow A \Vdash (k,\ p)$›
| *BoxI* [*intro*]: ‹$\{(i,\ \Diamond\ (\cdot\ (\bigcirc k)))\} \cup A \Vdash (\bigcirc k,\ p) \Longrightarrow k \notin$ *symbols* $(\{(i,\ p)\} \cup A) \Longrightarrow A \Vdash (i,\ \Box\ p)$›
| *BoxE* [*elim*]: ‹$A \Vdash (i,\ \Box\ p) \Longrightarrow A \Vdash (i,\ \Diamond\ (\cdot k)) \Longrightarrow A \Vdash (k,\ p)$›
| *GloI* [*intro*]: ‹$A \Vdash (\bigcirc k,\ p) \Longrightarrow k \notin$ *symbols* $(\{(i,\ p)\} \cup A) \Longrightarrow A \Vdash (i,\ \mathbf{A}\ p)$›
| *GloE* [*dest*]: ‹$A \Vdash (i,\ \mathbf{A}\ p) \Longrightarrow A \Vdash (k,\ p)$›
| *DwnI* [*intro*]: ‹$A \Vdash (i,\ \langle i\rangle_i\ p) \Longrightarrow A \Vdash (i,\ \downarrow p)$›
| *DwnE* [*dest*]: ‹$A \Vdash (i,\ \downarrow p) \Longrightarrow A \Vdash (i,\ \langle i\rangle_i\ p)$›
| *AllI* [*intro*]: ‹$A \Vdash (i,\ \langle\cdot(\bigcirc P)\rangle_p\ p) \Longrightarrow P \notin$ *symbols* $(\{(i,\ p)\} \cup A) \Longrightarrow A \Vdash (i,\ \forall\ p)$›
| *AllE* [*dest*]: ‹$A \Vdash (i,\ \forall\ p) \Longrightarrow$ *softqdf* $q \Longrightarrow A \Vdash (i,\ \langle q\rangle_p\ p)$›
| *Clas*: ‹$\{(i,\ \neg\ p)\} \cup A \Vdash (i,\ p) \Longrightarrow A \Vdash (i,\ p)$›

### 4.6.1 Soundness

**theorem** *soundness*:
  **assumes** ‹$A \Vdash (i,\ p)$› ‹$\forall (k,\ q) \in A.\ (Model\ W\ R\ PI\ N\ e\ V\ f,\ case\text{-}tm\ e\ N\ k) \models q$›
    ‹*wf-model* $(Model\ W\ R\ PI\ N\ e\ V\ f)$›
  **shows** ‹$(Model\ W\ R\ PI\ N\ e\ V\ f,\ case\text{-}tm\ e\ N\ i) \models p$›
  **using** *assms*
**proof** (*induct A* ‹$(i,\ p)$› *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*: ‹$v \in R\ (case\text{-}tm\ e\ N\ i)$›
    **moreover have** ‹$case\text{-}tm\ e\ N\ i \in W$›
      **using** *BoxI*(*5*) **unfolding** *wf-env-def wf-model-def unfolds* **by** (*auto split*:

*tm.splits*)
    **ultimately have** ‹$v \in W$›
      **using** *BoxI*(*5*) **unfolding** *wf-model-def wf-gframe-def wf-frame-def unfolds*
**by** *blast*

    **let** *?N* = ‹$N(k := v)$›
    **have** ‹$\forall (i, p) \in A.$ (*Model W R PI ?N e V f*, *case-tm e ?N i*) $\models p$›
      **using** *BoxI* **by** *fastforce*
    **moreover have** ‹(*Model W R PI ?N e V f*, *case-tm e ?N i*) $\models \Diamond$ ($\cdot$ ($\bigcirc$ *k*))›
      **using** *v BoxI.hyps*(*3*) **by** (*simp split*: *tm.splits*)
    **moreover have** ‹*wf-model* (*Model W R PI ?N e V f*)›
     **using** *BoxI.prems*(*2*) ‹$v \in W$› **unfolding** *wf-env-def wf-model-def unfolds* **by**
*auto*
    **ultimately have** ‹(*Model W R PI ?N e V f*, *?N k*) $\models p$›
      **using** *BoxI.hyps*(*2*) **by** *fastforce*
  **}**
  **then have** ‹$\forall v \in R$ (*case-tm e N i*). (*Model W R PI* ($N(k := v)$) *e V f*, *v*) $\models$
*p*›
    **by** *simp*
  **then have** ‹$\forall v \in R$ (*case-tm e N i*). (*Model W R PI N e V f*, *v*) $\models p$›
    **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** ‹$v \in W$›
    **let** *?N* = ‹$N(k := v)$›
    **have** ‹$\forall v.\ \forall (i, p) \in A.$ (*Model W R PI ?N e V f*, *case-tm e ?N i*) $\models p$›
      **using** *GloI.prems*(*1*) *GloI.hyps*(*3*) **by** *fastforce*
    **moreover have** ‹*wf-model* (*Model W R PI ?N e V f*)›
      **using** *GloI.prems*(*2*) ‹$v \in W$› **unfolding** *wf-env-def wf-model-def unfolds*
**by** *auto*
    **ultimately have** ‹(*Model W R PI ?N e V f*, *?N k*) $\models p$›
      **using** *GloI.hyps*(*2*) **by** *fastforce*
  **}**
  **then have** ‹$\forall v \in W.$ (*Model W R PI* ($N(k := v)$) *e V f*, *v*) $\models p$›
    **by** *simp*
  **then have** ‹$\forall v \in W.$ (*Model W R PI N e V f*, *v*) $\models p$›
    **using** *GloI.hyps*(*3*) **by** *simp*
  **then show** *?case*
    **by** *simp*
**next**
  **case** (*GloE A i p k*)

```
  then show ?case
    using wf-𝔑 wf-𝒩
    by (cases k) fastforce+
next
  case (AllI A i P p)
  {
    fix X
    assume ‹X ∈ PI›
    let ?V = ‹V(P := X)›
    have ‹∀ v. ∀(i, p) ∈ A. (Model W R PI N e ?V f, case-tm e N i) ⊨ p›
      using AllI.prems(1) AllI.hyps(3) by fastforce
    moreover have ∗: ‹wf-model (Model W R PI N e ?V f)›
     using AllI.prems(2) ‹X ∈ PI› unfolding wf-env-def wf-model-def unfolds by
auto
    ultimately have ‹(Model W R PI N e ?V f, case-tm e N i) ⊨ ⟨• (○ P)⟩_p p›
      using AllI.hyps(2) by fast
    moreover have ‹case-tm e N i ∈ W›
       using AllI.prems(2) unfolding wf-model-def wf-env-def unfolds by (auto
split: tm.splits)
    ultimately have ‹(Model W R PI N e ?V (worlds (Model W R PI N e ?V f)
(• (○ P)) ≫ f), case-tm e N i) ⊨ p›
      using inst-single-worlds ∗ by fastforce
    then have ‹(Model W R PI N e ?V ({ w ∈ W. w ∈ X } ≫ f), case-tm e N
i) ⊨ p›
      unfolding worlds-def by simp
    moreover have ‹X ∈ Pow W›
     using AllI.prems(2) ‹X ∈ PI› unfolding wf-model-def wf-gframe-def unfolds
by auto
    then have ‹{ w ∈ W. w ∈ X } = X›
      by blast
    ultimately have ‹(Model W R PI N e ?V (X ≫ f), case-tm e N i) ⊨ p›
      by simp
  }
  then have ‹∀ X ∈ PI. (Model W R PI N e (V(P := X)) (X ≫ f), case-tm e N
i) ⊨ p›
    by simp
  then have ‹∀ X ∈ PI. (Model W R PI N e V (X ≫ f), case-tm e N i) ⊨ p›
    using AllI.hyps(3) by simp
  then show ?case
    by simp
next
  case (AllE A i p q)
  then show ?case
  proof (cases i)
    case (Var x1)
    then show ?thesis
      using AllE(2−) inst-single-worlds softqdf-worlds wf-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** ‹{} ⊩ (○*i*, *p*)› ‹*i* ∉ *symbols-fm p*›
**and** ‹*wf-model M*› ‹*w* ∈ 𝒲 *M*›
**shows** ‹(*M*, *w*) ⊨ *p*›
**proof** (*cases M*)
**case** (*fields W R PI N e V f*)
**let** *?N* = ‹*N*(*i* := *w*)›
**have** ‹*wf-model* (*Model W R PI ?N e V f*)›
**using** *fields assms*(3−4) **unfolding** *wf-env-def wf-model-def unfolds* **by** *auto*
**then have** ‹(*Model W R PI ?N e V f*, *case-tm e ?N* (○*i*)) ⊨ *p*›
**using** *assms*(1) *soundness* **by** *blast*
**then have** ‹(*Model W R PI ?N e V f*, *w*) ⊨ *p*›
**by** *simp*
**then show** *?thesis*
**using** *assms*(2) *fields* **by** *simp*
**qed**

**lemma** *no-bot*: ‹¬ (*M*, *w*) ⊨ ⊥›
**by** *simp*

**corollary** ‹¬ ({} ⊩ (○*i*, ⊥))›
**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*]: ‹{(*p*, *i*)} ∪ *A* ⊩ (*p*, *i*)›
**using** *Assm* **by** *blast*

**lemma** *Boole*: ‹{(*i*, ¬ *p*)} ∪ *A* ⊩ (*i*, ⊥) ⟹ *A* ⊩ (*i*, *p*)›
**using** *Clas AndD1 AndD2 NotE* **by** *meson*

**lemma** *FlsE* [*dest*]: ‹*A* ⊩ (*i*, ⊥) ⟹ *A* ⊩ (*k*, *p*)›
**by** (*meson Assm-head NotE NotI*)

### 4.6.3 Derivational Consistency

**lemma** *calculus-confl*:
**assumes** ‹*ps* ⇝✗ *qs*› ‹*set ps* ⊆ *A*› ‹*q* ∈ *set qs*› ‹*q* ∈ *A*›
**shows** ‹*A* ⊩ (*i*, ⊥)›
**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*, $\bot$)›
**shows** ‹*A* $\Vdash$ (*i*, $\bot$)›
**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*, $\bot$)›
**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*, $\bot$)›
**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* ⊆ *A*› ‹∀ *q* ∈ *set qs*. {*q*} ∪ *A* ⊩ (*i*, ⊥)›
**shows** ‹*A* ⊩ (*i*, ⊥)›
**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* ⊆ *A*› ‹*set* (*qs k*) ∪ *A* ⊩ (*i*, ⊥)›
**shows** ‹*A* ⊩ (*i*, ⊥)›
**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* ⊆ *A*› ‹*k* ∈ *F A*› ‹*set* (*qs k*) ∪ *A* ⊩ (*i*, ⊥)›
**shows** ‹*A* ⊩ (*i*, ⊥)›
**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** ‹$p \in A$› ‹$k \notin$ *symbols* $A$› ‹*set* ($\delta$ $p$ $k$) $\cup$ $A \Turnstile (a, \bot)$›
**shows** ‹$A \Turnstile (a, \bot)$›
**using** *assms*
**proof** (*induct p k rule*: *δ.induct*)
**case** (*1 i p k*)
**then have** ‹$\{(\bigcirc k, \neg\ p), (i, \Diamond\ (\bullet(\bigcirc k)))\} \cup A \Turnstile (a, \bot)$› ‹$(i, \neg\ \Box\ p) \in A$›
**by** *simp-all*
**then have** ‹$\{(i, \Diamond\ (\bullet(\bigcirc k)))\} \cup A \Turnstile (\bigcirc k, p)$›
**using** *Boole* **by** *auto*
**moreover have** ‹$k \notin$ *symbols* ($\{(i, p)\} \cup A$)›
**using** *1* **by** *auto*
**ultimately have** ‹$A \Turnstile (i, \Box\ p)$›
**using** *BoxI* **by** *blast*
**then show** *?thesis*
**using** ‹$(i, \neg\ \Box\ p) \in A$› **by** (*meson Assm NotE*)
**next**
**case** (*2 i p k*)
**then have** ‹$\{(\bigcirc k, \neg\ p)\} \cup A \Turnstile (a, \bot)$› ‹$(i, \neg\ \mathbf{A}\ p) \in A$›
**by** *simp-all*
**then have** ‹$\{(\bigcirc k, \neg\ p)\} \cup A \Turnstile (\bigcirc k, \bot)$›
**by** *fast*
**then have** ‹$A \Turnstile (\bigcirc k, p)$›
**by** (*meson Boole*)
**moreover have** ‹$k \notin$ *symbols* ($\{(i, p)\} \cup A$)›
**using** *2 CGloN* **by** *auto*
**ultimately have** ‹$A \Turnstile (i, \mathbf{A}\ p)$›
**by** *blast*
**then show** *?thesis*
**using** ‹$(i, \neg\ \mathbf{A}\ p) \in A$› *Assm NotE* **by** *meson*
**next**
**case** (*3 i p P*)
**then have** ‹$\{(i, \neg\ \langle\bullet\ (\bigcirc\ P)\rangle_p\ p)\} \cup A \Turnstile (a, \bot)$› ‹$(i, \neg\ \forall\ p) \in A$›
**by** *simp-all*
**then have** ‹$A \Turnstile (i, \langle\bullet\ (\bigcirc\ P)\rangle_p\ p)$›
**using** *Clas* **by** *blast*
**moreover have** ‹$P \notin$ *symbols* ($\{(i, p)\} \cup A$)›
**using** *3* **by** *auto*
**ultimately have** ‹$A \Turnstile (i, \forall\ p)$›
**by** *blast*
**then show** *?thesis*

**using** ‹$(i, \neg \forall\ p) \in A$› *Assm NotE* **by** *meson*
**qed** *simp-all*

**interpretation** *DC*: *Derivational-Confl map-lbd symbols-lbd* ‹$\lambda$-. *True*› *confl-class* ‹$\lambda A.\ \neg\ A \Vdash (a, \bot)$›
**using** *calculus-confl* **by** *unfold-locales blast*

**interpretation** *DA*: *Derivational-Alpha map-lbd symbols-lbd* ‹$\lambda$-. *True*› *alpha-class* ‹$\lambda A.\ \neg\ A \Vdash (a, \bot)$›
**using** *calculus-alpha* **by** *unfold-locales blast*

**interpretation** *DB*: *Derivational-Beta map-lbd symbols-lbd* ‹$\lambda$-. *True*› *beta-class* ‹$\lambda A.\ \neg\ A \Vdash (a, \bot)$›
**using** *calculus-beta* **by** *unfold-locales blast*

**interpretation** *DGI*: *Derivational-Gamma map-tm map-lbd symbols-lbd* ‹$\lambda$-. *True*› *GI.classify-UNIV* ‹$\lambda A.\ \neg\ A \Vdash (a, \bot)$›
**using** *calculus-gammaI* **by** *unfold-locales blast*

**interpretation** *DGP*: *Derivational-Gamma map-fm map-lbd symbols-lbd* ‹$\lambda$-. *True*› *gamma-class-fm* ‹$\lambda A.\ \neg\ A \Vdash (a, \bot)$›
**using** *calculus-gammaP* **by** *unfold-locales blast*

**interpretation** *DD*: *Derivational-Delta map-lbd symbols-lbd* ‹$\lambda$-. *True*› $\delta$ ‹$\lambda A.\ \neg\ A \Vdash (a, \bot)$›
**by** *unfold-locales* (*meson calculus-*$\delta$)

**interpretation** *Derivational-Consistency map-lbd symbols-lbd* ‹$\lambda$-. *True*› *Kinds* ‹$\lambda A.\ \neg\ A \Vdash (a, \bot)$›
**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** $p$ :: ‹$'x$ *fm*›
**assumes** *mod*: ‹$\bigwedge(M :: ('x, 'x\ tm)\ model)\ w.\ wf\text{-}model\ M \Longrightarrow w \in \mathcal{W}\ M \Longrightarrow$
$\forall (k, q) \in A.\ (M, case\text{-}tm\ (\mathfrak{N}\ M)\ (\mathcal{N}\ M)\ k) \models q \Longrightarrow$
$(M, w) \models p$›
**and** ‹*P.enough-new A*›
**shows** ‹$A \Vdash (i, p)$›
**proof** (*rule ccontr*)
**assume** ‹$\neg\ A \Vdash (i, p)$›
**then have** $*$: ‹$\neg\ \{(i, \neg\ p)\} \cup A \Vdash (i, \bot)$›
**using** *Boole* **by** *blast*

**let** *?S* = ‹$\{(i, \neg\ p)\} \cup A$›
**let** *?C* = ‹$\{A.\ P.enough\text{-}new\ A \wedge \neg\ A \Vdash (undefined, \bot)\}$›
**let** *?H* = ‹*mk-mcs ?C ?S*›

**let** *?M* = ‹*canonical ?H*›

**have** ‹*P.prop*$_E$ *Kinds ?C*›
**using** *Consistency* **by** *fast*
**moreover have** ‹*?S* ∈ *?C*›
**using** ∗ *FlsE params-left assms*(*2*)
**by** (*metis* (*no-types*, *lifting*) *ext List.set-insert empty-set mem-Collect-eq* )
**moreover from** *this* **have** ‹*P.enough-new ?S*›
**by** *blast*
**ultimately have** ∗∗: ‹∀ (*k*, *q*) ∈ *?S*. ⟦*?H*, *k*⟧ ⊨ *q*›
**using** *model-existence*[*of ?C ?S*] **by** *blast*
**then have** ‹∀ (*k*, *q*) ∈ *?S*. (*?M*, *case-tm* (𝔑 *?M*) (𝒩 *?M*) *k*) ⊨ *q*›
**unfolding** *canonical-tm-eta′ canonical-ctx-def* **by** *blast*

**moreover have** ‹*wf-model ?M*›
**using** *wf-canonical* **by** *blast*
**moreover have** ‹*assign i ?H* ∈ 𝒲 *?M*›
**unfolding** *canonical-def unfolds* **by** *auto*

**ultimately have** ‹⟦*?H*, *i*⟧ ⊨ *p*›
**using** *mod* **unfolding** *canonical-ctx-def* **by** *auto*

**moreover have** ‹¬ ⟦*?H*, *i*⟧ ⊨ *p*›
**using** ∗∗ **unfolding** *canonical-ctx-def* **by** *simp*
**ultimately show** *False*
**by** *simp*
**qed**

## 4.7 Natural Deduction with Lists

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

$(i, \forall\ p)$›
| *AllE* [*dest*]: ‹$A \vdash (i, \forall\ p) \Longrightarrow$ *softqdf* $q \Longrightarrow A \vdash (i, \langle q \rangle_p\ p)$›
| *Clas*: ‹$(i, \neg\ p) \#\ A \vdash (i, p) \Longrightarrow A \vdash (i, p)$›

**definition** *bounded* :: ‹$'a$ *list* ⇒ $'a$ *set* ⇒ ($'a$ *list* ⇒ *bool*) ⇒ *bool*› **where**
‹*bounded* $K\ A\ P \equiv$ *set* $K \subseteq A \wedge (\forall B.$ *set* $K \subseteq$ *set* $B \longrightarrow$ *set* $B \subseteq A \longrightarrow P\ B)$›

**lemma** *bounded-one* [*elim*]:
**assumes** ‹*bounded* $K\ A\ P$› ‹$\bigwedge A.\ P\ A \Longrightarrow Q\ A$›
**shows** ‹*bounded* $K\ A\ Q$›
**using** *assms* **unfolding** *bounded-def* **by** *simp*

**lemma** *bounded-two* [*elim*]:
**assumes** ‹*bounded* $K\ A\ P$› ‹*bounded* $K'\ A\ Q$› ‹$\bigwedge A.\ P\ A \Longrightarrow Q\ A \Longrightarrow R\ A$›
**shows** ‹*bounded* $(K\ @\ K')\ A\ R$›
**using** *assms* **unfolding** *bounded-def* **by** *simp*

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

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

**lemma** *finite-kernel*: ‹$A \Vdash (i, p) \Longrightarrow \exists K.$ *bounded* $K\ A\ (\lambda B.\ B \vdash (i, p))$›
**proof** (*induct* $A$ ‹$(i, p)$› *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** ‹$\exists K.$ *bounded* $K\ A\ (\lambda B.\ (i, p)\ \#\ B \vdash (i, \bot))$›
**by** *blast*
**then show** *?case*
**by** *fast*

**next**
**case** (*BoxI i k A p*)
**then have** ‹∃ *K*. *bounded K A* (λ*B*. (*i*, ◇ (• (○ *k*))) # *B* ⊢ (○ *k*, *p*))›
**by** *blast*
**moreover from** *this* **have** ‹∃ *K*. *bounded K A* (λ*B*. *k* ∉ *P.params* ({(*i*, *p*)} ∪ *set B*))›
**using** *BoxI*(*2*−*3*) *bounded-params* **by** *fastforce*
**ultimately show** *?case*
**by** *fastforce*
**next**
**case** (*GloI A k p i*)
**then have** ‹∃ *K*. *bounded K A* (λ*B*. *B* ⊢ (○ *k*, *p*))›
**by** *blast*
**moreover from** *this* **have** ‹∃ *K*. *bounded K A* (λ*B*. *k* ∉ *P.params* ({(*i*, *p*)} ∪ *set B*))›
**using** *GloI*(*3*) *bounded-params* **by** *fastforce*
**ultimately show** *?case*
**by** *fastforce*
**next**
**case** (*AllI A i P p*)
**then have** ‹∃ *K*. *bounded K A* (λ*B*. *B* ⊢ (*i*, ⟨• (○ *P*)⟩$_p$ *p*))›
**by** *blast*
**moreover from** *this* **have** ‹∃ *K*. *bounded K A* (λ*B*. *P* ∉ *P.params* ({(*i*, *p*)} ∪ *set B*))›
**using** *AllI*(*3*) *bounded-params* **by** *fastforce*
**ultimately show** *?case*
**by** *fastforce*
**next**
**case** (*Clas i p A*)
**then have** ‹∃ *K*. *bounded K A* (λ*B*. (*i*, ¬ *p*) # *B* ⊢ (*i*, *p*))›
**by** *fast*
**then show** *?case*
**using** *Calculus.Clas* **by** *force*
**qed** *fast+*

**corollary** *finite-assumptions*: ‹*A* ⊩ (*i*, *p*) ⟹ ∃ *B*. *set B* ⊆ *A* ∧ *B* ⊢ (*i*, *p*)›
**using** *finite-kernel* **unfolding** *bounded-def* **by** *blast*

**lemma** *calculus-set*: ‹*A* ⊢ (*i*, *p*) ⟹ *set A* ⊩ (*i*, *p*)›
**proof** (*induct A* ‹(*i*, *p*)› *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** ‹*A* ⊢ (*i*, *p*)› ‹∀ (*k*, *q*) ∈ *set A*. (*M*, *case-tm* (𝔑 *M*) (𝒩 *M*) *k*) ⊨ *q*›
**and** ‹*wf-model M*›
**shows** ‹(*M*, *case-tm* (𝔑 *M*) (𝒩 *M*) *i*) ⊨ *p*›

**using** *assms soundness calculus-set*
**by** (*metis* (*mono-tags*, *lifting*) *model.surjective unit.exhaust split-cong*)

**corollary** *soundness-nil*:
‹[] ⊢ (○*i*, *p*) ⟹ *i* ∉ *symbols-fm p* ⟹ *wf-model M* ⟹ *w* ∈ 𝒲 *M* ⟹ (*M*, *w*) ⊨ *p*›
**by** (*metis calculus-set empty-set soundness′*)

**corollary** ‹¬ ([] ⊢ (*i*, ⊥))›
**by** (*metis equals0D no-bot set-empty2 soundness-list wf-canonical*)

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

**theorem** *main*:
**fixes** *p* :: ‹′*x fm*›
**assumes** ‹*i* ∉ *symbols-fm p*› ‹|*UNIV* :: ′*x lbd set*| ≤*o* |*UNIV* :: ′*x set*|›
**shows** ‹[] ⊢ (○*i*, *p*) ⟷ (∀(*M* :: (′*x*, ′*x tm*) *model*). ∀ *w* ∈ 𝒲 *M*. *wf-model M* ⟶ (*M*, *w*) ⊨ *p*)›
**using** *assms strong-completeness-list*[**where** *A*=‹{}›] *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* :: *int* ⇒ *nat* ⇒ *int set* **where**
*arith-prog a b* = {*x*. [*x* = *a*] (*mod int b*)}

**lemma** *arith-prog-0-right* [*simp*]: *arith-prog a 0* = {*a*}
**by** (*simp add*: *arith-prog-def*)

**lemma** *arith-prog-Suc-0-right* [*simp*]: *arith-prog a* (*Suc 0*) = *UNIV*
**by** (*auto simp*: *arith-prog-def*)

**lemma** *in-arith-progI* [*intro*]: [*x* = *a*] (*mod b*) ⟹ *x* ∈ *arith-prog a b*
**by** (*auto simp*: *arith-prog-def*)

**lemma** *arith-prog-disjoint*:
**assumes** [*a* ≠ *a′*] (*mod int b*) **and** *b* > *0*
**shows** *arith-prog a b* ∩ *arith-prog a′ b* = {}
**using** *assms* **by** (*auto simp*: *arith-prog-def cong-def*)

**lemma** *arith-prog-dvd-mono*: *b dvd b′ ⟹ arith-prog a b′ ⊆ arith-prog a b*
**by** (*auto simp*: *arith-prog-def cong-dvd-modulus*)

**lemma** *bij-betw-arith-prog*:
**assumes** *b > 0*
**shows** *bij-betw* (*λn. a + int b ∗ n*) *UNIV* (*arith-prog a b*)
**proof** (*rule bij-betwI*[*of* - - - *λx.* (*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 ‹b > 0›* **in** ‹*auto simp*: *arith-prog-def*›)

**lemma** *arith-prog-altdef*: *arith-prog a b = range* (*λn. a + 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 ⟹ infinite* (*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** *−arith-prog a b* = (⋃ *i*∈{*1*..<*b*}. *arith-prog* (*a + int i*) *b*)
**proof** −
**have** *disjoint*: *x ∉ arith-prog a b* **if** *x ∈ arith-prog* (*a + int i*) *b i ∈* {*1*..<*b*} **for** *x i*
**proof** −
**have** [*a ≠ a + int i*] (*mod int b*)
**proof**
**assume** [*a = a + int i*] (*mod int b*)
**hence** [*a + 0 = a + int i*] (*mod int b*) **by** *simp*
**hence** [*0 = int i*] (*mod 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*] ‹*b > 0*› *that* **by** *auto*
**qed**

**have** *covering*: *x ∈ arith-prog a b ∨ x ∈* (⋃ *i*∈{*1*..<*b*}. *arith-prog* (*a + int i*) *b*)
**for** *x*
**proof** −
**define** *i* **where** *i = nat* ((*x − a*) *mod* (*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*: ‹$k \in$ *arith-prog* $a\ b \Longrightarrow$ *arith-prog* $k\ b =$ *arith-prog* $a\ b$›
**unfolding** *arith-prog-def* **by** (*simp add*: *cong-def*)

**lemma** *arith-prog-mod*: ‹*arith-prog* (*a mod int b*) $b =$ *arith-prog* $a\ b$›
**unfolding** *arith-prog-def* **by** *auto*

**lemma** *mod-bounds*: ‹$b > 0 \Longrightarrow a$ *mod int* $b \geq 0 \wedge a$ *mod int* $b < b$›
**by** *simp*

**lemma** *mod-range*: ‹$\{a$ *mod int* $b \mid a.\ b > 0\} \subseteq \{0..<$*int* $b\}$›
**using** *mod-bounds* **by** *auto*

**lemma** *finite-mod-range*: ‹*finite* $\{a$ *mod int* $b \mid a.\ b > 0\}$›
**using** *mod-range* **by** (*meson finite-atLeastLessThan-int finite-subset*)

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

**lemma** *arith-mono*: ‹*arith* $B\ U \Longrightarrow B \subseteq C \Longrightarrow$ *arith* $C\ U$›
**unfolding** *arith-def* **by** *blast*

**lemma** *arith-empty-steps*: ‹*arith* $B\ U \Longrightarrow B = \{\} \Longrightarrow U = \{\}$›
**unfolding** *arith-def* **by** *blast*

**lemma** *arith-ne-steps*: ‹*arith* $B\ U \Longrightarrow U \neq \{\} \Longrightarrow B \neq \{\}$›
**using** *arith-empty-steps* **by** *blast*

**lemma** *arith-decomp*:
**assumes** ‹*arith* $B\ U$›
**obtains** *abs* **where**
‹*finite* $B \Longrightarrow$ *finite abs*›
‹$\bigwedge a\ b.\ (a, b) \in abs \Longrightarrow b > 0 \wedge b \in B$›
‹$U = (\bigcup (a, b) \in abs.$ *arith-prog* $a\ b)$›
**proof** −
**have** ‹$\forall\, a \in U.\ \exists\, b \in B.\ b > 0 \wedge$ *arith-prog* $(a$ *mod int* $b)\ b \subseteq U$›
**using** ‹*arith* $B\ U$› **unfolding** *arith-def* **using** *arith-prog-mod* **by** *simp*
**then obtain** $f$ **where** $f$: ‹$\forall\, a \in U.\ \exists\, b \in B.\ f\ a = b \wedge b > 0 \wedge$ *arith-prog* $(a$ *mod int* $b)\ b \subseteq U$›
**by** *metis*

**let** *?abs* = ‹$\{(a$ *mod int* $(f\ a), f\ a) \mid a.\ a \in U\}$›

**have** *abs*: ‹$U = (\bigcup (a, b) \in$ *?abs*. *arith-prog* $a\ b)$›
**using** $f$ **by** *fastforce*

**have** ‹$\{a$ *mod int* $(f\ a) \mid a.\ a \in U\} \subseteq (\bigcup b \in B.\ \{a$ *mod int* $b \mid a.\ b > 0\})$›
**using** $f$ **by** *blast*

**moreover have** ‹*finite* ($\bigcup b \in B.\ \{a\ mod\ int\ b\ |a.\ b > 0\}$)› **if** ‹*finite B*›
**using** *that finite-mod-range* **by** *fast*
**ultimately have** ‹*finite* $\{a\ mod\ int\ (f\ a)\ |a.\ a \in U\}$› **if** ‹*finite B*›
**using** *that* **by** (*meson finite-subset*)

**moreover have** ‹*?abs* $\subseteq \{a\ mod\ int\ (f\ a)\ |a.\ a \in U\} \times f\ `\ U$›
**by** *blast*
**moreover have** *fin-f*: ‹*finite* ($f\ `\ U$)› **if** ‹*finite B*›
**using** *that f* **by** (*meson finite-subset image-subsetI*)
**ultimately have** *fin-abs*: ‹*finite ?abs*› **if** ‹*finite B*›
**using** *that fin-f* **by** (*meson finite-SigmaI finite-subset*)

**have** *pos*: ‹$\bigwedge a\ b.\ (a, b) \in$ *?abs* $\Longrightarrow b > 0 \wedge b \in B$›
**using** *f* **by** *blast*

**show** *?thesis*
**using** *fin-abs pos abs that* **by** *meson*
**qed**

**lemma** *arith-UNIV*: ‹*arith* $\{1\}$ *UNIV*›
**unfolding** *arith-def* **by** *blast*

**lemma** *arith-empty*: *arith B* $\{\}$
**unfolding** *arith-def* **by** *blast*

**lemma** *finite-case-prod-lcm*: ‹*finite B* $\Longrightarrow$ *finite C* $\Longrightarrow$ *finite* (*case-prod lcm* ‘ ($B \times C$)›
**by** *blast*

**lemma** *arith-inter*:
**assumes** *U*: ‹*arith B U*› **and** *V*: ‹*arith C V*›
**shows** ‹*arith* (*case-prod lcm* ‘ ($B \times C$)) ($U \cap V$)›
**unfolding** *arith-def*
**proof** *safe*
**fix** *a*
**assume** *a*: ‹$a \in U$› ‹$a \in V$›

**from** *a U* **obtain** *b* **where** *b*: ‹$b \in B$› ‹$b > 0$› ‹*arith-prog a b* $\subseteq U$›
**unfolding** *arith-def* **by** *auto*
**from** *a V* **obtain** *c* **where** *c*: ‹$c \in C$› ‹$c > 0$› ‹*arith-prog a c* $\subseteq V$›
**unfolding** *arith-def* **by** *auto*

**with** *a b c U V* **have** ‹*arith-prog a* (*lcm b c*) $\subseteq U \cap V$›
**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** ‹*lcm b c* $> 0$›
**using** *lcm-pos-nat* **by** *blast*
**ultimately show** ‹$\exists\, b \in (\lambda(x, y).\ lcm\ x\ y)\ `\ (B \times C).\ 0 < b \wedge$ *arith-prog a b* $\subseteq U \cap V$›

**using** *b c* **by** *blast*
**qed**

**lemma** *arith-Inter*:
**assumes** ‹*finite X*› **and** *X*: ‹∀ *U* ∈ *X*. *arith B U*› ‹*X* ≠ {}›
**shows** ‹∃ *B'*. (*finite B* ⟶ *finite B'*) ∧ *arith B'* (⋂ *X*)›
**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** ‹*arith B U*› ‹*arith C V*›
**shows** ‹*arith* (*B* ∪ *C*) (*U* ∪ *V*)›
**using** *assms* **unfolding** *arith-def* **by** (*metis Un-iff subset-trans sup-ge1 sup-ge2*)

**lemma** *arith-ne-infinite*:
**assumes** ‹*arith B U*› ‹*U* ≠ {}›
**shows** ‹*infinite U*›
**using** *assms* **unfolding** *arith-def*
**by** (*meson equals0I infinite-arith-prog rev-finite-subset*)

**lemma** *arith-prog-arith* [*intro*]:
**assumes** ‹*b* > *0*›
**shows** ‹*arith* {*b*} (*arith-prog a b*)›
**unfolding** *arith-def* **using** *assms arith-prog-offset-in* **by** *blast*

**lemma** *arith-prog-complement-arith* [*intro*]:
**assumes** ‹*b* > *0*›
**shows** ‹*arith* {*b*} (− *arith-prog a b*)›
**proof** −
**have** −*arith-prog a b* = (⋃ *i*∈{*1*..<*b*}. *arith-prog* (*a* + *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** ‹*arith B U*› ‹*finite B*›
**shows** ‹∃ *B'*. *finite B'* ∧ *arith B'* (− *U*)›
**proof** −

**obtain** *abs* **where**
*abs*: ‹$\bigwedge a\ b.\ (a,\ b) \in abs \Longrightarrow b > 0 \wedge b \in B$› ‹*finite abs*› **and**
*U*: ‹$U = (\bigcup (a,\ b) \in abs.\ arith\text{-}prog\ a\ b)$›
**using** *assms arith-decomp* **by** *metis*
**then have** ∗: ‹$\bigwedge a\ b.\ (a,\ b) \in abs \Longrightarrow -\ arith\text{-}prog\ a\ b = (\bigcup i{\in}\{1..{<}b\}.\ arith\text{-}prog\ (a + int\ i)\ b)$›
**using** *arith-prog-complement* **by** *simp*

**have** ‹$-\ U = (\bigcap (a,\ b) \in abs.\ -\ arith\text{-}prog\ a\ b)$›
**using** *U* **by** *blast*
**also have** ‹$\ldots = (\bigcap (a,\ b) \in abs.\ (\bigcup i{\in}\{1..{<}b\}.\ arith\text{-}prog\ (a + int\ i)\ b))$›
**using** ∗ **by** *blast*
**finally have** ∗∗: ‹$-\ U = (\bigcap (a,\ b) \in abs.\ (\bigcup i{\in}\{1..{<}b\}.\ arith\text{-}prog\ (a + int\ i)\ b))$› **.**

**have** ‹$\bigwedge a\ b.\ (a,\ b) \in abs \Longrightarrow arith\ \{b\}\ (\bigcup i{\in}\{1..{<}b\}.\ arith\text{-}prog\ (a + int\ i)\ b)$›
**using** *abs* ∗ **by** (*metis arith-prog-complement-arith*)
**then have** ∗∗∗: ‹$\bigwedge a\ b.\ (a,\ b) \in abs \Longrightarrow arith\ B\ (\bigcup i{\in}\{1..{<}b\}.\ arith\text{-}prog\ (a + int\ i)\ b)$›
**using** *arith-mono abs* **by** *fast*

**define** *X* **where** *X*: ‹$X \equiv (\lambda(a,\ b).\ \bigcup i{\in}\{1..{<}b\}.\ arith\text{-}prog\ (a + int\ i)\ b)$ ‘ *abs*›

**from** *X* **have** ‹*finite X*›
**using** *abs*(*2*) **by** *simp*
**moreover have** ‹$\forall\ U{\in}X.\ arith\ B\ U$›
**using** *X* ∗∗∗ **by** *fast*
**ultimately have** ‹$\exists\ B'.\ finite\ B' \wedge arith\ B'\ (\bigcap\ X)$›
**using** *arith-Inter*[*of X*] ‹*finite B*› *arith-UNIV* **by** *fastforce*

**moreover from** *X* **have** ‹$-\ U = \bigcap X$›
**using** ∗∗ **by** *simp*
**ultimately show** *?thesis*
**by** *simp*
**qed**

**lemma** *arith-distinguish*:
**assumes** ‹$x \neq y$›
**shows** ‹$\exists\ B\ U\ V.\ finite\ B \wedge arith\ B\ U \wedge arith\ B\ V \wedge x \in U \wedge y \in V \wedge U \cap V = \{\}$›
**proof** −
**obtain** *a b c* **where**
*b*: ‹$0 < b$› **and** *x*: ‹$x \in arith\text{-}prog\ a\ b$› **and** *y*: ‹$y \in arith\text{-}prog\ c\ b$› **and**
∗: ‹$arith\text{-}prog\ a\ b \cap arith\text{-}prog\ c\ b = \{\}$›
**using** *assms arith-prog-distinguish* **by** *meson*

**let** *?B* = ‹$\{b\}$›
**let** *?U* = ‹*arith-prog a b*›
**let** *?V* = ‹*arith-prog c b*›

**have** ‹*finite ?B* ∧ *arith ?B ?U* ∧ *arith ?B ?V* ∧ *x* ∈ *?U* ∧ *y* ∈ *?V* ∧ *?U* ∩ *?V* = {}›
**using** *b x y* ∗
**by** *blast*
**then show** *?thesis*
**by** *blast*
**qed**

## Finite Unions of Arithmetic Progressions

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

**lemma** *fin-arith-UNIV* [*intro*]: ‹*fin-arith UNIV*›
**unfolding** *fin-arith-def* **using** *arith-UNIV* **by** *force*

**lemma** *fin-arith-empty* [*intro*]: ‹*fin-arith* {}›
**unfolding** *fin-arith-def* **using** *arith-empty* **by** *blast*

**lemma** *fin-arith-inter* [*intro*]: ‹*fin-arith U* ⟹ *fin-arith V* ⟹ *fin-arith* (*U* ∩ *V*)›
**unfolding** *fin-arith-def* **using** *arith-inter finite-case-prod-lcm* **by** *metis*

**lemma** *fin-arith-union* [*intro*]: ‹*fin-arith U* ⟹ *fin-arith V* ⟹ *fin-arith* (*U* ∪ *V*)›
**unfolding** *fin-arith-def* **using** *arith-union* **by** *blast*

**lemma** *fin-arith-compl* [*intro*]: ‹*fin-arith U* ⟹ *fin-arith* (− *U*)›
**unfolding** *fin-arith-def* **by** *blast*

**lemma** *fin-arith-distinguish*:
**assumes** ‹*x* ≠ *y*›
**shows** ‹∃ *U V*. *fin-arith U* ∧ *fin-arith V* ∧ *x* ∈ *U* ∧ *y* ∈ *V* ∧ *U* ∩ *V* = {}›
**using** *assms arith-distinguish* **unfolding** *fin-arith-def* **by** *meson*

## Singletons

**lemma** *arith-prog-singleton*: ‹⋂ {*arith-prog a b* |*a b*. *b* > *0* ∧ *x* ∈ *arith-prog a b*} = {*x*}›
**proof**
**show** ‹⋂ {*arith-prog a b* |*a b*. *0* < *b* ∧ *x* ∈ *arith-prog a b*} ⊆ {*x*}›
**using** *arith-prog-distinguish* **by** *fast*
**qed** *fast*

**lemma** *fin-arith-Inter-singleton*: ‹⋂ {*U* |*U*. *fin-arith U* ∧ *x* ∈ *U*} = {*x*}›
**proof** −
**have** ‹{*arith-prog a b* |*a b*. *b* > *0* ∧ *x* ∈ *arith-prog a b*} ⊆ {*U* |*U b*. *arith* {*b*} *U* ∧ *x* ∈ *U*}›
**using** *arith-prog-arith* **by** *blast*
**also have** ‹. . . ⊆ {*U* |*U B*. *finite B* ∧ *arith B U* ∧ *x* ∈ *U*}›
**by** *force*

**finally have** ‹$\{arith\text{-}prog\ a\ b\ |a\ b.\ b > 0 \wedge x \in arith\text{-}prog\ a\ b\} \subseteq \{U\ |U.\ fin\text{-}arith\ U \wedge x \in U\}$›
**unfolding** *fin-arith-def* **by** *auto*
**then have** ‹$\bigcap \{U\ |U.\ fin\text{-}arith\ U \wedge x \in U\} \subseteq \bigcap \{arith\text{-}prog\ a\ b\ |a\ b.\ b > 0 \wedge x \in arith\text{-}prog\ a\ b\}$›
**by** *blast*
**then show** *?thesis*
**using** *arith-prog-singleton* **by** *auto*
**qed**

**lemma** *singleton-not-finarith*: ‹$\neg\ fin\text{-}arith\ \{x\}$›
**unfolding** *fin-arith-def* **using** *arith-ne-infinite* **by** *blast*

### 4.8.2 Counterexample

**definition** *Pss* :: ‹*int set set*› **where**
‹$Pss \equiv \{U.\ fin\text{-}arith\ U\}$›

**lemma** *Pss-empty*: ‹$\{\} \in Pss$›
**unfolding** *Pss-def* **by** *blast*

**lemma** *Pss-UNIV*: ‹$UNIV \in Pss$›
**unfolding** *Pss-def* **by** *blast*

**lemma** *Pss-union*: ‹$X \in Pss \Longrightarrow Y \in Pss \Longrightarrow X \cup Y \in Pss$›
**unfolding** *Pss-def* **by** *blast*

**lemma** *Pss-inter*: ‹$X \in Pss \Longrightarrow Y \in Pss \Longrightarrow X \cap Y \in Pss$›
**unfolding** *Pss-def* **by** *blast*

**lemma** *Pss-compl*: ‹$X \in Pss \Longrightarrow -\ X \in Pss$›
**unfolding** *Pss-def* **by** *blast*

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

**lemma** *wf-frame-mygframe*: ‹*wf-frame* (*frame.truncate my-gframe*)›
**unfolding** *wf-frame-def unfolds my-gframe-def* **by** *blast*

**lemma** *admissible-mygframe*: ‹*admissible* (*frame.truncate my-gframe*) (Π *my-gframe*)›
**unfolding** *admissible-def unfolds my-gframe-def Pss-def*
**by** (*auto simp*: *Compl-eq-Diff-UNIV*[*symmetric*])

**lemma** *wf-mygframe*: ‹*wf-gframe my-gframe*›
**using** *wf-frame-mygframe admissible-mygframe*
**unfolding** *wf-gframe-def unfolds my-gframe-def Pss-def* **by** *fast*

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

$\mathfrak{V} = \lambda n.\ \{\}\rangle\!\rangle$›

**lemma** *wf-mymodel*: ‹*wf-model my-model*›
  **using** *wf-mygframe* **unfolding** *wf-model-def wf-env-def unfolds my-model-def my-gframe-def Pss-def*
  **by** *blast*

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

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

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

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

Finite unions of arithmetic progressions are either empty or infinite.

**lemma** *fin-arith-nowhere-or-twice*:
  **assumes** ‹*fin-arith U*›
  **shows** ‹*U* = {} ∨ (∃ *x y*. *x* ∈ *U* ∧ *y* ∈ *U* ∧ *x* ≠ *y*)›
  **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*: ‹(*my-model*, *x*) ⊨ ∀ (*nowhere-or-twice* (·(#*0*)))›
  **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* ≡ ∀ ( **@**(○*x*) (·(#*0*)) ⟶ ·(#*0*) )›

**lemma** *singleton*: ‹((*my-model*, *x*) ⊨ *singleton y*) ⟷ *x* = *y*›
  **unfolding** *my-model-def my-gframe-def unfolds Pss-def*
  **using** *fin-arith-Inter-singleton* **by** *auto*

**lemma** *fin-arith-distinguish'*:
  ‹∀ *P*. *fin-arith P* ⟶ *y* ∈ *P* ⟶ *v* ∈ *P* ⟹ *v* ≠ *w* ⟹ ∃ *P*. *y* ∈ *P* ∧ *fin-arith P* ∧ *w* ∉ *P*›
  **by** (*metis disjoint-iff fin-arith-distinguish*)

The singleton does not hold nowhere-or-twice.

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

**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** ‹¬ (*my-model*, *x*) ⊨ ∀ (*nowhere-or-twice* (•(#*0*))) ⟶ *nowhere-or-twice* (*singleton y*)›
**using** *nowhere-or-twice-admissible not-nowhere-or-twice-singleton*
**by** (*meson semantics.simps*(*3*,*4*))

**end**