Synthetic Completeness

Asta Halkjær From

January 10, 2023

Abstract

In this work, I provide an abstract framework for proving the completeness of a logical calculus using the synthetic method. The synthetic method is based on maximal consistent saturated sets (MCSs). A set of formulas is consistent (with respect to the calculus) when we cannot derive a contradiction from it. It is maximally consistent when it contains every formula that is consistent with it. For logics where it is relevant, it is saturated when it contains a witness for every existential formula. To prove completeness using these maximal consistent saturated sets, we prove a truth lemma: every formula in an MCS has a satisfying model. Here, Hintikka sets provide a useful stepping stone. These can be seen as characterizations of the MCSs based on simple subformula conditions rather than via the calculus. We then prove that every Hintikka set gives rise to a satisfying model and that MCSs are Hintikka sets. Now, assume a valid formula cannot be derived. Then its negation must be consistent and therefore satisfiable. This contradicts validity and the original formula must be derivable.

To start, I build maximal consistent saturated sets for any logic that satisfies a small set of assumptions. I do this using a transfinite version of Lindenbaum's lemma, which allows me to support languages of any cardinality. I then prove useful abstract results about derivations and refutations as they relate to MCSs. Finally, I show how Hintikka sets can be derived from the logic's semantics, outlining one way to prove the required truth lemma.

To demonstrate the versatility of the framework, I instantiate it with five different examples. The formalization contains soundness and completeness results for: a propositional tableau calculus, a propositional sequent calculus, an axiomatic system for modal logic, a labelled natural deduction system for hybrid logic and a natural deduction system for first-order logic. The tableau example uses custom Hintikka sets based on the calculus, but the other four examples derive them from the semantics in the style of the framework. The hybrid and first-order logic examples rely on saturated MCSs. This places requirements on the cardinalities of their languages to ensure that there are enough witnesses available. In both cases, the type of witnesses must be infinite and have cardinality at least that of the type of propositional/predicate symbols.

Contents

Chapter 1

Maximal Consistent Sets

theory *Maximal-Consistent-Sets* **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-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*) ⇒ '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 *split-finite-sets*:

assumes ‹*finite A*› ‹*finite B*›
assumes ‹$A \subseteq B \cup S$›
shows ‹$\exists B'\ C.$ *finite* $C \wedge (B' \cup C = A) \wedge B' \subseteq B \wedge C \subseteq S$›
using *assms subset-UnE* **by** *fastforce*

lemma *split-list*:
assumes ‹*set A* $\subseteq$ *set B* $\cup$ *S*›
shows ‹$\exists B'\ C.$ *set* $(B' @ C) =$ *set A* $\wedge$ *set* $B' \subseteq$ *set B* $\wedge$ *set* $C \subseteq S$›
using *assms split-finite-sets*[**where** *A*=‹*set A*› **and** *B*=‹*set B*› **and** *S*=*S*]
by (*metis List.finite-set finite-Un finite-list set-append*)

lemma *struct-split*:
assumes ‹$\bigwedge A\ B.$ *P A* $\Longrightarrow$ *set A* $\subseteq$ *set B* $\Longrightarrow$ *P B*› ‹*P A*› ‹*set A* $\subseteq$ *set B* $\cup$ *X*›
shows ‹$\exists C.$ *set* $C \subseteq X \wedge P\ (B @ C)$›
proof −
obtain $B'\ C$ **where** *C*: ‹*set* $(B' @ C) =$ *set A*› ‹*set* $B' \subseteq$ *set B*› ‹*set* $C \subseteq X$›
using *assms*(*3*) *split-list* **by** *meson*
then have ‹*P* $(B @ C)$›
using *assms*(*1*)[**where** *B*=‹*B* @ *C*›] *assms*(*2*) **by** *fastforce*
then show *?thesis*
using *C* **by** *blast*
qed

context *wo-rel* **begin**

lemma *underS-bound*: ‹$a \in$ *underS* $n \Longrightarrow b \in$ *underS* $n \Longrightarrow a \in$ *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* n› ‹$X \neq \{\}$›
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*› ‹$p \subseteq (\bigcup n \in$ *Field r*. *f n*)›
shows ‹$\exists m.\ p \subseteq (\bigcup n \in$ *under m*. *f n*)›

using *assms*
proof (*induct rule*: *finite-induct*)
case (*insert x p*)
then obtain *m* **where** ‹$p \subseteq (\bigcup n \in under\ m.\ f\ n)$›
by *fast*
moreover obtain m' **where** ‹$x \in f\ m'$› ‹$m' \in Field\ r$›
using *insert*(*4*) **by** *blast*
then have ‹$x \in (\bigcup n \in under\ m'.\ f\ n)$›
using *REFL Refl-under-in* **by** *fast*
ultimately have ‹$\{x\} \cup p \subseteq (\bigcup n \in under\ m.\ f\ n) \cup (\bigcup n \in under\ m'.\ f\ n)$›
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*)

end

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)

lemma *card-of-info-marker-lists*:
assumes ‹*infinite* (*UNIV* :: *'i set*)› ‹$|UNIV :: 'm\ set| \leq o\ |UNIV :: nat\ set|$›
shows ‹$|UNIV :: ('i + 'm \times nat)\ list\ set| \leq o\ |UNIV :: 'i\ set|$›
proof −
have ‹$(UNIV :: 'm\ set) \neq \{\}$›
by *simp*
then have ‹$|UNIV :: 'm\ set| *c\ |UNIV :: nat\ set| \leq o\ |UNIV :: nat\ set|$›
using *assms*(*2*) **by** (*simp add*: *cinfinite-def cprod-cinfinite-bound ordLess-imp-ordLeq*)
then have ‹$|UNIV :: ('m \times nat)\ set| \leq o\ |UNIV :: nat\ set|$›
unfolding *cprod-def* **by** *simp*
moreover have ‹$|UNIV :: nat\ set| \leq o\ |UNIV :: 'i\ set|$›
using *assms infinite-iff-card-of-nat* **by** *blast*
ultimately have ‹$|UNIV :: ('m \times nat)\ set| \leq o\ |UNIV :: 'i\ set|$›
using *ordLeq-transitive* **by** *blast*
moreover have ‹*Cinfinite* $|UNIV :: 'i\ set|$›
using *assms* **by** (*simp add*: *cinfinite-def*)
ultimately have ‹$|UNIV :: 'i\ set| +c\ |UNIV :: ('m \times nat)\ set| =o\ |UNIV :: 'i\ set|$›
using *csum-absorb1* **by** *blast*
then have ‹$|UNIV :: ('i + 'm \times nat)\ set| =o\ |UNIV :: 'i\ set|$›
unfolding *csum-def* **by** *simp*
then have ‹$|UNIV :: ('i + 'm \times nat)\ set| \leq o\ |UNIV :: 'i\ set|$›
using *ordIso-iff-ordLeq* **by** *blast*
moreover have ‹*infinite* $(UNIV :: ('i + 'm \times nat)\ set)$›

using *assms* **by** *simp*
then have ‹$|UNIV :: ('i + 'm \times nat)\ list\ set| =o |UNIV :: ('i + 'm \times nat)\ set|$›
by (*metis card-of-lists-infinite lists-UNIV*)
ultimately have ‹$|UNIV :: ('i + 'm \times nat)\ list\ set| \leq o\ |UNIV :: 'i\ set|$›
using *ordIso-ordLeq-trans* **by** *blast*
then show *?thesis*
using *ordLeq-transitive* **by** *blast*
qed

1.2 Base Locale

locale *MCS-Base* =
fixes *consistent* :: ‹$'a\ set \Rightarrow bool$›
assumes *consistent-hereditary*: ‹$\bigwedge S\ S'.\ consistent\ S \Longrightarrow S' \subseteq S \Longrightarrow consistent\ S'$›
and *inconsistent-finite*: ‹$\bigwedge S.\ \neg\ consistent\ S \Longrightarrow \exists\, S' \subseteq S.\ finite\ S' \wedge \neg\ consistent\ S'$›
begin

definition *maximal* :: ‹$'a\ set \Rightarrow bool$› **where**
‹$maximal\ S \equiv \forall\, p.\ consistent\ (\{p\} \cup S) \longrightarrow p \in S$›

end

1.3 Ordinal Locale

locale *MCS-Lim-Ord* = *MCS-Base* +
fixes *r* :: ‹$'a\ rel$›
assumes *WELL*: ‹*Well-order r*›
and *Cinfinite-r*: ‹*Cinfinite r*›
fixes *info* :: ‹$'a \Rightarrow 'i\ set$›
and *witness* :: ‹$'a \Rightarrow 'a\ set \Rightarrow 'a\ set$›
assumes *finite-info*: ‹$\bigwedge p.\ finite\ (info\ p)$›
and *finite-witness-info*: ‹$\bigwedge p\ S.\ finite\ (\bigcup (info\ `\ witness\ p\ S))$›
and *consistent-witness*: ‹$\bigwedge p\ S.\ consistent\ (\{p\} \cup S)$
$\Longrightarrow infinite\ (UNIV - \bigcup (info\ `\ S))$
$\Longrightarrow consistent\ (witness\ p\ S \cup \{p\} \cup S)$›
begin

lemma *wo-rel-r*: ‹*wo-rel r*›
by (*simp add*: *WELL wo-rel.intro*)

lemma *isLimOrd-r*: ‹*isLimOrd r*›
using *Cinfinite-r card-order-infinite-isLimOrd cinfinite-def* **by** *blast*

1.3.1 Lindenbaum Extension

abbreviation *infos* :: ‹$'a\ set \Rightarrow 'i\ set$› **where**
‹$infos\ S \equiv \bigcup (info\ `\ S)$›

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

definition *extendL* :: ‹('*a* ⇒ '*a set*) ⇒ '*a* ⇒ '*a set*› **where**
‹*extendL rec n* ≡ ⋃ *m* ∈ *underS r n*. *rec m*›

definition *extend* :: ‹'*a set* ⇒ '*a* ⇒ '*a set*› **where**
‹*extend S n* ≡ *worecZSL r S extendS extendL n*›

lemma *adm-woL-extendL*: ‹*adm-woL r extendL*›
unfolding *extendL-def wo-rel.adm-woL-def*[*OF wo-rel-r*] **by** *blast*

definition *Extend* :: ‹'*a set* ⇒ '*a set*› **where**
‹*Extend S* ≡ ⋃ *n* ∈ *Field r*. *extend S n*›

lemma *extend-subset*: ‹*n* ∈ *Field r* ⟹ *S* ⊆ *extend S n*›
proof (*induct n rule*: *wo-rel.well-order-inductZSL*[*OF wo-rel-r*])
case *1*
then show *?case*
unfolding *extend-def wo-rel.worecZSL-zero*[*OF wo-rel-r adm-woL-extendL*]
by *simp*
next
case (*2 i*)
moreover from *this* **have** ‹*i* ∈ *Field r*›
by (*meson FieldI1 wo-rel.succ-in wo-rel-r*)
ultimately show *?case*
unfolding *extend-def extendS-def wo-rel.worecZSL-succ*[*OF wo-rel-r adm-woL-extendL 2*(*1*)]
by *auto*
next
case (*3 i*)
then show *?case*
unfolding *extend-def extendL-def wo-rel.worecZSL-isLim*[*OF wo-rel-r adm-woL-extendL 3*(*1*−*2*)]
using *wo-rel.zero-in-Field*[*OF wo-rel-r*] *wo-rel.zero-smallest*[*OF wo-rel-r*]
by (*metis SUP-upper2 emptyE underS-I*)
qed

lemma *Extend-subset'*: ‹*Field r* ≠ {} ⟹ *S* ⊆ *Extend S*›
unfolding *Extend-def* **using** *extend-subset* **by** *fast*

lemma *extend-underS*: ‹*m* ∈ *underS r n* ⟹ *extend S m* ⊆ *extend S n*›
proof (*induct n rule*: *wo-rel.well-order-inductZSL*[*OF wo-rel-r*])
case *1*
then show *?case*
unfolding *extend-def* **using** *wo-rel.underS-zero*[*OF wo-rel-r*] **by** *fast*
next
case (*2 i*)
moreover from *this* **have** ‹*m* = *i* ∨ *m* ∈ *underS r i*›
by (*metis wo-rel.less-succ*[*OF wo-rel-r*] *underS-E underS-I*)
ultimately show *?case*

unfolding *extend-def extendS-def wo-rel.worecZSL-succ*[*OF wo-rel-r adm-woL-extendL 2*(*1*)] **by** *auto*
next
case (*3 i*)
then show *?case*
unfolding *extend-def extendL-def wo-rel.worecZSL-isLim*[*OF wo-rel-r adm-woL-extendL 3*(*1*−*2*)]
by *blast*
qed

lemma *extend-under*: ‹*m* ∈ *under r n* ⟹ *extend S m* ⊆ *extend S n*›
using *extend-underS wo-rel.supr-greater*[*OF wo-rel-r*] *wo-rel.supr-under*[*OF wo-rel-r*]
by (*metis emptyE in-Above-under set-eq-subset underS-I under-Field under-empty*)

1.3.2 Consistency

lemma *info-origin*:
assumes ‹*a* ∈ *infos* (*extend S n*)›
shows ‹*a* ∈ *infos S* ∨ (∃ *m* ∈ *underS r n*. *a* ∈ *infos* (*witness m* (*extend S m*) ∪ {*m*}))›
using *assms*
proof (*induct n rule*: *wo-rel.well-order-inductZSL*[*OF wo-rel-r*])
case *1*
then show *?case*
unfolding *extend-def wo-rel.worecZSL-zero*[*OF wo-rel-r adm-woL-extendL*]
by *blast*
next
case (*2 i*)
then consider (*here*) ‹*a* ∈ *infos* (*witness i* (*extend S i*) ∪ {*i*})› | (*there*) ‹*a* ∈ *infos* (*extend S i*)›
using *wo-rel.worecZSL-succ*[*OF wo-rel-r adm-woL-extendL 2*(*1*)] *extendS-def extend-def*
by (*metis* (*no-types*, *lifting*) *UN-Un UnE*)
then show *?case*
proof *cases*
case *here*
moreover have ‹*i* ∈ *Field r*›
by (*meson WELL 2*(*1*) *well-order-on-domain wo-rel.succ-in-diff*[*OF wo-rel-r*])
ultimately show *?thesis*
using *2*(*1*) **by** (*metis Refl-under-in wo-rel.underS-succ*[*OF wo-rel-r*] *wo-rel.REFL*[*OF wo-rel-r*])
next
case *there*
then show *?thesis*
using *2* **by** (*metis in-mono underS-subset-under wo-rel.underS-succ*[*OF wo-rel-r*])
next
qed
next
case (*3 i*)
then obtain *j* **where** ‹*j* ∈ *underS r i*› ‹*a* ∈ *infos* (*extend S j*)›
unfolding *extend-def extendL-def wo-rel.worecZSL-isLim*[*OF wo-rel-r adm-woL-extendL 3*(*1*−*2*)]
by *blast*
then show *?case*
using *3 wo-rel.underS-trans*[*OF wo-rel-r*, *of - j i*] **by** *meson*
qed

lemma *consistent-extend*:
assumes ‹*consistent S*› ‹*r* $\leq o$ |*UNIV* − *infos S*|›
shows ‹*consistent* (*extend S n*)›
using *assms*(*1*)
proof (*induct n rule*: *wo-rel.well-order-inductZSL*[*OF wo-rel-r*])
case *1*
then show *?case*
unfolding *extend-def wo-rel.worecZSL-zero*[*OF wo-rel-r adm-woL-extendL*]
by *blast*
next
case (*2 i*)
then have ‹*i* ∈ *Field r*›
by (*meson WELL well-order-on-domain wo-rel.succ-in-diff*[*OF wo-rel-r*])
then have ∗: ‹|*underS r i*| $<o$ *r*›
using *card-of-underS* **by** (*simp add*: *Cinfinite-r*)
let *?infos* = ‹λ*k*. *infos* (*witness k* (*extend S k*) ∪ {*k*})›
let *?X* = ‹$\bigcup$ *k* ∈ *underS r i*. *?infos k*›
have ‹|*?X*| $<o$ *r*›
proof (*cases* ‹*finite* (*underS r i*)›)
case *True*
then have ‹*finite ?X*›
by (*simp add*: *finite-info finite-witness-info*)
then show *?thesis*
using *Cinfinite-r assms*(*2*) **unfolding** *cinfinite-def* **by** (*simp add*: *finite-ordLess-infinite*)
next
case *False*
moreover have ‹∀ *k*. *finite* (*?infos k*)›
by (*simp add*: *finite-info finite-witness-info*)
then have ‹∀ *k*. |*?infos k*| $<o$ |*underS r i*|›
using *False* **by** *simp*
ultimately have ‹|*?X*| $\leq o$ |*underS r i*|›
using *card-of-infinite-smaller-Union* **by** *fast*
then show *?thesis*
using ∗ *ordLeq-ordLess-trans* **by** *blast*
qed
then have ‹|*?X*| $<o$ |*UNIV* − *infos S*|›
using *assms*(*2*) *ordLess-ordLeq-trans* **by** *blast*
moreover have ‹*infinite* (*UNIV* − *infos S*)›
using *assms*(*2*) *Cinfinite-r* **unfolding** *cinfinite-def* **by** (*metis Field-card-of ordLeq-finite-Field*)
ultimately have ‹|*UNIV* − *infos S* − *?X*| $=o$ |*UNIV* − *infos S*|›
using *card-of-Un-diff-infinite* **by** *blast*
moreover from *this* **have** ‹*infinite* (*UNIV* − *infos S* − *?X*)›
using ‹*infinite* (*UNIV* − *infos S*)› *card-of-ordIso-finite* **by** *blast*
moreover have ‹$\bigwedge$*a*. *a* ∈ *infos* (*extend S i*) ⟹ *a* ∈ *infos S* ∨ *a* ∈ *?X*›
using *info-origin* **by** *simp*
then have ‹*infos* (*extend S i*) ⊆ *infos S* ∪ *?X*›
by *fast*
ultimately have ‹*infinite* (*UNIV* − *infos* (*extend S i*))›

```
    using infinite-Diff-subset by (metis (no-types, lifting) Set-Diff-Un)
  with 2 show ?case
    unfolding extend-def extendS-def wo-rel.worecZSL-succ[OF wo-rel-r adm-woL-extendL 2(1)]
    using consistent-witness by auto
next
  case (3 i)
  show ?case
  proof (rule ccontr)
    assume ‹¬ consistent (extend S i)›
    then obtain S' where S': ‹finite S'› ‹S' ⊆ (⋃ n ∈ underS r i. extend S n)› ‹¬ consistent S'›
      unfolding extend-def extendL-def wo-rel.worecZSL-isLim[OF wo-rel-r adm-woL-extendL 3(1–2)]
      using inconsistent-finite by auto
    then obtain ns where ns: ‹S' ⊆ (⋃ n ∈ ns. extend S n)› ‹ns ⊆ underS r i› ‹finite ns›
      by (metis finite-subset-Union finite-subset-image)
    moreover have ‹ns ≠ {}›
      using S'(3) assms calculation(1) consistent-hereditary by auto
    ultimately obtain j where ‹∀ n ∈ ns. n ∈ under r j› ‹j ∈ underS r i›
      using wo-rel.finite-underS-bound wo-rel-r ns by (meson subset-iff)
    then have ‹∀ n ∈ ns. extend S n ⊆ extend S j›
      using extend-under by fast
    then have ‹S' ⊆ extend S j›
      using S' ns(1) by blast
    then show False
      using 3(3–) ‹¬ consistent S'› consistent-hereditary ‹j ∈ underS r i›
      by (meson BNF-Least-Fixpoint.underS-Field)
  qed
qed

lemma consistent-Extend:
  assumes ‹consistent S› ‹r ≤o |UNIV – infos S|›
  shows ‹consistent (Extend S)›
  unfolding Extend-def
proof (rule ccontr)
  assume ‹¬ consistent (⋃ n ∈ Field r. extend S n)›
  then obtain S' where ‹finite S'› ‹S' ⊆ (⋃ n ∈ Field r. extend S n)› ‹¬ consistent S'›
    using inconsistent-finite by metis
  then obtain m where ‹S' ⊆ (⋃ n ∈ under r m. extend S n)› ‹m ∈ Field r›
    using wo-rel.finite-bound-under[OF wo-rel-r] assms consistent-hereditary
    by (metis Sup-empty emptyE image-empty subsetI under-empty)
  then have ‹S' ⊆ extend S m›
    using extend-under by fast
  moreover have ‹consistent (extend S m)›
    using assms consistent-extend ‹m ∈ Field r› by blast
  ultimately show False
    using ‹¬ consistent S'› consistent-hereditary by blast
qed

lemma Extend-bound: ‹n ∈ Field r ⟹ extend S n ⊆ Extend S›
  unfolding Extend-def by blast
```

1.3.3 Maximality

definition *maximal′* :: ‹*'a set ⇒ bool*› **where**
‹*maximal′ S* ≡ ∀ *p* ∈ *Field r*. *consistent* ({*p*} ∪ *S*) ⟶ *p* ∈ *S*›

lemma *maximal′-Extend*: ‹*maximal′* (*Extend S*)›
unfolding *maximal′-def*
proof *safe*
fix *p*
assume ∗: ‹*p* ∈ *Field r*› ‹*consistent* ({*p*} ∪ *Extend S*)›
then have ‹{*p*} ∪ *extend S p* ⊆ {*p*} ∪ *Extend S*›
unfolding *Extend-def* **by** *blast*
then have ∗∗: ‹*consistent* ({*p*} ∪ *extend S p*)›
using ∗ *consistent-hereditary* **by** *blast*
moreover have *succ*: ‹*aboveS r p* ≠ {}›
using ∗ *isLimOrd-r wo-rel.isLimOrd-aboveS*[*OF wo-rel-r*] **by** *blast*
then have ‹*succ r p* ∈ *Field r*›
using *wo-rel.succ-in-Field*[*OF wo-rel-r*] **by** *blast*
moreover have ‹*p* ∈ *extend S* (*succ r p*)›
using ∗∗ **unfolding** *extend-def extendS-def wo-rel.worecZSL-succ*[*OF wo-rel-r adm-woL-extendL succ*]
by *simp*
ultimately show ‹*p* ∈ *Extend S*›
using *Extend-bound* **by** *fast*
qed

1.3.4 Saturation

definition *saturated′* :: ‹*'a set ⇒ bool*› **where**
‹*saturated′ S* ≡ ∀ *p* ∈ *S*. *p* ∈ *Field r* ⟶ (∃ *S′*. *witness p S′* ⊆ *S*)›

lemma *saturated′-Extend*:
assumes ‹*consistent* (*Extend S*)›
shows ‹*saturated′* (*Extend S*)›
unfolding *saturated′-def*
proof *safe*
fix *p*
assume ∗: ‹*p* ∈ *Extend S*› ‹*p* ∈ *Field r*›
then have ‹*extend S p* ⊆ *Extend S*›
unfolding *Extend-def* **by** *blast*
then have ‹*consistent* ({*p*} ∪ *extend S p*)›
using *assms*(*1*) ∗ *consistent-hereditary* **by** *auto*
moreover have *succ*: ‹*aboveS r p* ≠ {}›
using ∗ *isLimOrd-r wo-rel.isLimOrd-aboveS wo-rel-r* **by** *fast*
then have ‹*succ r p* ∈ *Field r*›
using *wo-rel-r* **by** (*simp add*: *wo-rel.succ-in-Field*)
ultimately have ‹*extend S* (*succ r p*) = *witness p* (*extend S p*) ∪ {*p*} ∪ *extend S p*›
unfolding *extend-def extendS-def wo-rel.worecZSL-succ*[*OF wo-rel-r adm-woL-extendL succ*]
by *simp*
moreover have ‹*extend S* (*succ r p*) ⊆ *Extend S*›

unfolding *Extend-def* **using** ‹*succ r p ∈ Field r*› **by** *blast*
ultimately show ‹∃ *a*. *witness p a ⊆ Extend S*›
by *fast*
qed

end

1.4 Locale with Saturation

locale *MCS-Saturation = MCS-Base +*
assumes *infinite-UNIV*: ‹*infinite* (*UNIV* :: *'a set*)›
fixes *info* :: ‹*'a ⇒ 'i set*›
and *witness* :: ‹*'a ⇒ 'a set ⇒ 'a set*›
assumes ‹⋀*p*. *finite* (*info p*)›
and ‹⋀*p S*. *finite* (⋃(*info ‘ witness p S*))›
and ‹⋀*p S*. *consistent* ({*p*} ∪ *S*) ⟹ *infinite* (*UNIV* − ⋃(*info ‘ S*))
⟹ *consistent* (*witness p S* ∪ {*p*} ∪ *S*)›

sublocale *MCS-Saturation ⊆ MCS-Lim-Ord* - ‹|*UNIV*|›
proof
show ‹*Well-order* |*UNIV*|›
by *simp*
next
show ‹*Cinfinite* |*UNIV* :: *'a set*|›
unfolding *cinfinite-def* **using** *infinite-UNIV* **by** *simp*
next
fix *p*
show ‹*finite* (*info p*)›
by (*metis MCS-Saturation-axioms MCS-Saturation-axioms-def MCS-Saturation-def*)
next
fix *p S*
show ‹*finite* (⋃ (*info ‘ witness p S*))›
by (*metis MCS-Saturation-axioms MCS-Saturation-axioms-def MCS-Saturation-def*)
next
fix *p S*
show ‹*consistent* ({*p*} ∪ *S*) ⟹
infinite (*UNIV* − ⋃ (*info ‘ S*)) ⟹
consistent (*witness p S* ∪ {*p*} ∪ *S*)›
by (*metis MCS-Saturation-axioms MCS-Saturation-axioms-def MCS-Saturation-def*)
qed

context *MCS-Saturation* **begin**

theorem *Extend-subset*: ‹*S ⊆ Extend S*›
by (*simp add*: *Extend-subset'*)

lemma *maximal-maximal'*: ‹*maximal S ⟷ maximal' S*›
unfolding *maximal-def maximal'-def* **by** *simp*

lemma *maximal-Extend*: ‹*maximal* (*Extend S*)›
using *maximal'-Extend maximal-maximal'* **by** *fast*

definition *saturated* :: ‹'*a set* ⇒ *bool*› **where**
‹*saturated S* ≡ ∀ *p* ∈ *S*. ∃ *S'*. *witness p S'* ⊆ *S*›

lemma *saturated-saturated'*: ‹*saturated S* ⟷ *saturated' S*›
unfolding *saturated-def saturated'-def* **by** *simp*

lemma *saturated-Extend*:
assumes ‹*consistent* (*Extend S*)›
shows ‹*saturated* (*Extend S*)›
using *assms saturated'-Extend saturated-saturated'* **by** *blast*

theorem *MCS-Extend*:
assumes ‹*consistent S*› ‹|*UNIV* :: '*a set*| ≤*o* |*UNIV* − *infos S*|›
shows ‹*consistent* (*Extend S*)› ‹*maximal* (*Extend S*)› ‹*saturated* (*Extend S*)›
using *assms consistent-Extend maximal-Extend saturated-Extend* **by** *blast+*

end

1.5 Locale without Saturation

locale *MCS-No-Saturation* = *MCS-Base* +
assumes ‹*infinite* (*UNIV* :: '*a set*)›

sublocale *MCS-No-Saturation* ⊆ *MCS-Saturation consistent* ‹λ-. {} :: '*a set*› ‹λ- -. {}›
proof
show ‹*infinite* (*UNIV* :: '*a set*)›
using *MCS-No-Saturation-axioms MCS-No-Saturation-axioms-def MCS-No-Saturation-def* **by** *blast*
next
show ‹*finite* {}› **..**
next
show ‹*finite* (⋃-∈{}. {})›
by *fast*
next
fix *p S*
show ‹*consistent* ({*p*} ∪ *S*) ⟹ *consistent* ({} ∪ {*p*} ∪ *S*)›
by *simp*
qed

context *MCS-No-Saturation* **begin**

lemma *always-saturated* [*simp*]: ‹*saturated H*›
unfolding *saturated-def* **by** *simp*

theorem *MCS-Extend'*:
assumes ‹*consistent S*›
shows ‹*consistent* (*Extend S*)› ‹*maximal* (*Extend S*)›

using *assms consistent-Extend maximal-Extend* **by** *simp-all*

end

1.6 Truth Lemma

locale *Truth-Base* =
fixes *interp* :: ‹'*model* ⇒ ('*model* ⇒ '*fm* ⇒ *bool*) ⇒ '*fm* ⇒ *bool*›
and *semantics* :: ‹'*model* ⇒ '*fm* ⇒ *bool*›
and *models-from* :: ‹'*a set* ⇒ '*model set*›
and *P* :: ‹'*a set* ⇒ '*model* ⇒ '*fm* ⇒ *bool*›
assumes *interp-semantics*: ‹*semantics M p* ⟷ *interp M semantics p*›
and *Hintikka-model*: ‹⋀*H M p*. ∀ *M* ∈ *models-from H*. ∀ *p*. *interp M* (*P H*) *p* ⟷ *P H M p* ⟹
M ∈ *models-from H* ⟹ *semantics M p* ⟷ *P H M p*›

locale *Truth-Saturation* = *MCS-Saturation* + *Truth-Base* +
assumes *MCS-Hintikka*: ‹⋀*H*. *consistent H* ⟹ *maximal H* ⟹ *saturated H* ⟹
∀ *M* ∈ *models-from H*. ∀ *p*. *interp M* (*P H*) *p* ⟷ *P H M p*›
begin

theorem *truth-lemma-saturation*:
assumes ‹*consistent H*› ‹*maximal H*› ‹*saturated H*› ‹*M* ∈ *models-from H*›
shows ‹*semantics M p* ⟷ *P H M p*›
using *Hintikka-model MCS-Hintikka assms* **.**

end

locale *Truth-No-Saturation* = *MCS-No-Saturation* + *Truth-Base* +
assumes *MCS-Hintikka*: ‹⋀*H*. *consistent H* ⟹ *maximal H* ⟹
∀ *M* ∈ *models-from H*. ∀ *p*. *interp M* (*P H*) *p* ⟷ *P H M p*›
begin

theorem *truth-lemma-no-saturation*:
assumes ‹*consistent H*› ‹*maximal H*› ‹*M* ∈ *models-from H*›
shows ‹*semantics M p* ⟷ *P H M p*›
using *Hintikka-model MCS-Hintikka assms* **.**

end

end

Chapter 2

Derivations

theory *Derivations* **imports** *Maximal-Consistent-Sets* **begin**

2.1 Rearranging Assumptions

locale *Derivations* =
  fixes *derive* :: ‹'*a list* ⇒ '*a* ⇒ *bool*›
  assumes *derive-struct*: ‹⋀*A B p*. *derive A p* ⟹ *set A* ⊆ *set B* ⟹ *derive B p*›
begin

theorem *derive-split*:
  assumes ‹*set A* ⊆ *set B* ∪ *X*› ‹*derive A p*›
  shows ‹∃ *C*. *set C* ⊆ *X* ∧ *derive* (*B* @ *C*) *p*›
  using *struct-split*[**where** *P*=‹λ*A*. *derive A p*›] *derive-struct assms* **by** *blast*

corollary *derive-split1*:
  assumes ‹*set A* ⊆ {*q*} ∪ *X*› ‹*derive A p*›
  shows ‹∃ *C*. *set C* ⊆ *X* ∧ *derive* (*q* # *C*) *p*›
  using *assms derive-split*[**where** *B*=‹[*q*]›] **by** *simp*

end

2.2 MCSs and Deriving Falsity

locale *Derivations-MCS* = *Derivations* + *MCS-Base* +
  fixes *fls*
  assumes *consistent-derive-fls*: ‹⋀*S*. *consistent S* = (∄ *S'*. *set S'* ⊆ *S* ∧ *derive S' fls*)›
begin

theorem *MCS-derive-fls*:
  assumes ‹*consistent S*› ‹*maximal S*›
  shows ‹*p* ∉ *S* ⟷ (∃ *S'*. *set S'* ⊆ *S* ∧ *derive* (*p* # *S'*) *fls*)›
proof
  assume ‹*p* ∉ *S*›

then show ‹$\exists S'.\ set\ S' \subseteq S \wedge derive\ (p\ \#\ S')\ fls$›
using *assms derive-split1 consistent-derive-fls* **unfolding** *maximal-def* **by** *metis*
next
assume ‹$\exists S'.\ set\ S' \subseteq S \wedge derive\ (p\ \#\ S')\ fls$›
then show ‹$p \notin S$›
using *assms consistent-derive-fls* **by** *fastforce*
qed

end

2.3 MCSs and Derivability

locale *Derivations-MCS-Cut = Derivations-MCS +*
assumes *derive-assm*: ‹$\bigwedge A\ p.\ p \in set\ A \Longrightarrow derive\ A\ p$›
and *derive-cut*: ‹$\bigwedge A\ B\ p\ q.\ derive\ A\ p \Longrightarrow derive\ (p\ \#\ B)\ q \Longrightarrow derive\ (A\ @\ B)\ q$›
begin

theorem *MCS-derive*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹$p \in S \longleftrightarrow (\exists S'.\ set\ S' \subseteq S \wedge derive\ S'\ p)$›
proof
assume ‹$p \in S$›
then show ‹$\exists S'.\ set\ S' \subseteq S \wedge derive\ S'\ p$›
using *derive-assm* **by** (*metis List.set-insert empty-set empty-subsetI insert-subset singletonI*)
next
assume ‹$\exists A.\ set\ A \subseteq S \wedge derive\ A\ p$›
then obtain *A* **where** *A*: ‹*set A* ⊆ *S*› ‹*derive A p*›
by *blast*
have ‹*consistent* ($\{p\} \cup S$)›
unfolding *consistent-derive-fls*
proof *safe*
fix *B*
assume *B*: ‹*set B* ⊆ {*p*} ∪ *S*› ‹*derive B fls*›
then obtain *C* **where** *C*: ‹*derive* (*p* # *C*) *fls*› ‹*set C* ⊆ *S*›
using *derive-split1* **by** *blast*
then have ‹*derive* (*A* @ *C*) *fls*›
using *A derive-cut* **by** *blast*
then show *False*
using *A*(*1*) *B*(*1*) *C assms*(*1*) *consistent-derive-fls* **by** *simp*
qed
then show ‹*p* ∈ *S*›
using *assms* **unfolding** *maximal-def* **by** *auto*
qed

end

end

Chapter 3

Refutations

theory *Refutations* **imports** *Maximal-Consistent-Sets* **begin**

3.1 Rearranging Refutations

locale *Refutations* =
  fixes *refute* :: ‹'*a list* ⇒ *bool*›
  assumes *refute-struct*: ‹⋀*A B*. *refute A* ⟹ *set A* ⊆ *set B* ⟹ *refute B*›
begin

theorem *refute-split*:
  assumes ‹*set A* ⊆ *set B* ∪ *X*› ‹*refute A*›
  shows ‹∃ *C*. *set C* ⊆ *X* ∧ *refute* (*B* @ *C*)›
  using *struct-split*[**where** *P*=*refute*] *refute-struct assms* **by** *blast*

corollary *refute-split1*:
  assumes ‹*set A* ⊆ {*q*} ∪ *X*› ‹*refute A*›
  shows ‹∃ *C*. *set C* ⊆ *X* ∧ *refute* (*q* # *C*)›
  using *assms refute-split*[**where** *B*=‹[*q*]›] **by** *simp*

end

3.2 MCSs and Refutability

locale *Refutations-MCS* = *Refutations* + *MCS-Base* +
  assumes *consistent-refute*: ‹⋀*S*. *consistent S* = (∄ *S'*. *set S'* ⊆ *S* ∧ *refute S'*)›
begin

theorem *MCS-refute*:
  assumes ‹*consistent S*› ‹*maximal S*›
  shows ‹*p* ∉ *S* ⟷ (∃ *S'*. *set S'* ⊆ *S* ∧ *refute* (*p* # *S'*))›
proof
  assume ‹*p* ∉ *S*›
  then show ‹∃ *S'*. *set S'* ⊆ *S* ∧ *refute* (*p* # *S'*)›

using *assms refute-split1 consistent-refute* **unfolding** *maximal-def* **by** *metis*
next
assume ‹$\exists S'.$ *set* $S' \subseteq S \wedge$ *refute* $(p \# S')$›
then show ‹$p \notin S$›
using *assms consistent-refute* **by** *fastforce*
qed

end

end

Chapter 4

Example: Propositional Tableau Calculus

theory *Example-Propositional-Tableau* **imports** *Refutations* **begin**

4.1 Syntax

datatype *'p fm*
  = *Pro 'p* (‹‡›)
  | *Neg* ‹*'p fm*› (‹¬ -› [*70*] *70*)
  | *Imp* ‹*'p fm*› ‹*'p fm*› (**infixr** ‹⟶› *55*)

4.2 Semantics

type-synonym *'p model* = ‹*'p* ⇒ *bool*›

fun *semantics* :: ‹*'p model* ⇒ *'p fm* ⇒ *bool*› (‹⟦-⟧›) **where**
  ‹⟦*I*⟧ (‡*P*) = *I P*›
| ‹⟦*I*⟧ (¬ *p*) = (¬ ⟦*I*⟧ *p*)›
| ‹⟦*I*⟧ (*p* ⟶ *q*) = (⟦*I*⟧ *p* ⟶ ⟦*I*⟧ *q*)›

4.3 Calculus

inductive *Calculus* :: ‹*'p fm list* ⇒ *bool*› (‹⊢$_T$ -› [*50*] *50*) **where**
  Axiom [*intro*]: ‹⊢$_T$ ‡*P* # ¬ ‡*P* # *A*›
| *NegI* [*intro*]: ‹⊢$_T$ *p* # *A* ⟹ ⊢$_T$ ¬ ¬ *p* # *A*›
| *ImpP* [*intro*]: ‹⊢$_T$ ¬ *p* # *A* ⟹ ⊢$_T$ *q* # *A* ⟹ ⊢$_T$ (*p* ⟶ *q*) # *A*›
| *ImpN* [*intro*]: ‹⊢$_T$ *p* # ¬ *q* # *A* ⟹ ⊢$_T$ ¬ (*p* ⟶ *q*) # *A*›
| *Weaken*: ‹⊢$_T$ *A* ⟹ *set A* ⊆ *set B* ⟹ ⊢$_T$ *B*›

lemma *Weaken2*:
  assumes ‹⊢$_T$ *p* # *A*› ‹⊢$_T$ *q* # *B*›
  shows ‹⊢$_T$ *p* # *A* @ *B* ∧ ⊢$_T$ *q* # *A* @ *B*›

using *assms Weaken*[**where** *A*=‹- # -› **and** *B*=‹- # *A* @ *B*›] **by** *fastforce*

4.4 Soundness

theorem *soundness*: ‹$\vdash_T A \Longrightarrow \exists p \in set\ A.\ \neg\ [\![I]\!]\ p$›
by (*induct A rule*: *Calculus.induct*) *auto*

corollary *soundness'*: ‹$\vdash_T [\neg\ p] \Longrightarrow [\![I]\!]\ p$›
using *soundness* **by** *fastforce*

corollary ‹$\neg\ (\vdash_T [])$›
using *soundness* **by** *fastforce*

4.5 Maximal Consistent Sets

definition *consistent* :: ‹*'p fm set* ⇒ *bool*› **where**
‹*consistent S* ≡ ∄*S'*. *set S'* ⊆ *S* ∧ $\vdash_T$ *S'*›

interpretation *MCS-No-Saturation consistent*
proof
fix *S S'* :: ‹*'p fm set*›
assume ‹*consistent S*› ‹*S'* ⊆ *S*›
then show ‹*consistent S'*›
unfolding *consistent-def* **by** *fast*
next
fix *S* :: ‹*'p fm set*›
assume ‹¬ *consistent S*›
then show ‹∃*S'*⊆*S*. *finite S'* ∧ ¬ *consistent S'*›
unfolding *consistent-def* **by** *blast*
next
show ‹*infinite* (*UNIV* :: *'p fm set*)›
using *infinite-UNIV-size*[*of* ‹λ*p*. *p* ⟶ *p*›] **by** *simp*
qed

interpretation *Refutations-MCS Calculus consistent*
proof
fix *A B* :: ‹*'p fm list*›
assume ‹$\vdash_T$ *A*› ‹*set A* ⊆ *set B*›
then show ‹$\vdash_T$ *B*›
using *Weaken* **by** *meson*
next
fix *S* :: ‹*'p fm set*›
show ‹*consistent S* ⟷ (∄*S'*. *set S'* ⊆ *S* ∧ $\vdash_T$ *S'*)›
unfolding *consistent-def* **..**
qed

4.6 Truth Lemma

abbreviation (*input*) *hmodel* :: ‹*'p fm set* ⇒ *'p model*› **where**
‹*hmodel H* ≡ λ*P*. ‡*P* ∈ *H*›

locale *Hintikka* =
fixes *H* :: ‹*'a fm set*›
assumes *AxiomH*: ‹⋀*P*. ‡*P* ∈ *H* ⟹ ¬ ‡*P* ∈ *H* ⟹ *False*›
and *NegIH*: ‹⋀*p*. ¬ ¬ *p* ∈ *H* ⟹ *p* ∈ *H*›
and *ImpPH*: ‹⋀*p q*. *p* ⟶ *q* ∈ *H* ⟹ ¬ *p* ∈ *H* ∨ *q* ∈ *H*›
and *ImpNH*: ‹⋀*p q*. ¬ (*p* ⟶ *q*) ∈ *H* ⟹ *p* ∈ *H* ∧ ¬ *q* ∈ *H*›

lemma *Hintikka-model*:
assumes ‹*Hintikka H*›
shows ‹(*p* ∈ *H* ⟶ ⟦*hmodel H*⟧ *p*) ∧ (¬ *p* ∈ *H* ⟶ ¬ ⟦*hmodel H*⟧ *p*)›
using *assms* **by** (*induct p*) (*unfold Hintikka-def semantics.simps*; *blast*)+

lemma *MCS-Hintikka*:
assumes ‹*consistent H*› ‹*maximal H*›
shows ‹*Hintikka H*›
proof
fix *P*
assume ‹‡*P* ∈ *H*› ‹¬ ‡*P* ∈ *H*›
then have ‹*set* [‡*P*, ¬ ‡*P*] ⊆ *H*›
by *simp*
moreover have ‹⊢$_T$ [‡*P*, ¬ ‡*P*]›
by *blast*
ultimately show *False*
using *assms* **unfolding** *consistent-def* **by** *blast*
next
fix *p*
assume ‹¬ ¬ *p* ∈ *H*›
then show ‹*p* ∈ *H*›
using *assms MCS-refute* **by** *blast*
next
fix *p q*
assume ∗: ‹*p* ⟶ *q* ∈ *H*›
show ‹¬ *p* ∈ *H* ∨ *q* ∈ *H*›
proof (*rule ccontr*)
assume ‹¬ (¬ *p* ∈ *H* ∨ *q* ∈ *H*)›
then have ‹¬ *p* ∉ *H*› ‹*q* ∉ *H*›
by *blast*+
then have ‹∃ *A*. *set A* ⊆ *H* ∧ ⊢$_T$ ¬ *p* # *A*› ‹∃ *A*. *set A* ⊆ *H* ∧ ⊢$_T$ *q* # *A*›
using *assms MCS-refute* **by** *blast*+
then have ‹∃ *A*. *set A* ⊆ *H* ∧ ⊢$_T$ ¬ *p* # *A* ∧ ⊢$_T$ *q* # *A*›
using *Weaken2*[**where** *p*=‹¬ *p*› **and** *q*=*q*] **by** (*metis Un-least set-append*)
then have ‹∃ *A*. *set A* ⊆ *H* ∧ ⊢$_T$ (*p* ⟶ *q*) # *A*›
by *blast*
then have ‹*p* ⟶ *q* ∉ *H*›

```
    using assms unfolding consistent-def by auto
  then show False
    using ∗ ..
 qed
next
 fix p q
 assume ∗: ‹¬ (p ⟶ q) ∈ H›
 show ‹p ∈ H ∧ ¬ q ∈ H›
 proof (rule ccontr)
  assume ‹¬ (p ∈ H ∧ ¬ q ∈ H)›
  then consider ‹p ∉ H› | ‹¬ q ∉ H›
    by blast
  then show False
  proof cases
    case 1
    then have ‹∃ A. set A ⊆ H ∧ ⊢T p # A›
      using assms MCS-refute by blast
    then have ‹∃ A. set A ⊆ H ∧ ⊢T p # ¬ q # A›
      using Weaken[where B=‹p # ¬ q # -›] by fastforce
    then have ‹∃ A. set A ⊆ H ∧ ⊢T ¬ (p ⟶ q) # A›
      by fast
    then have ‹¬ (p ⟶ q) ∉ H›
      using assms unfolding consistent-def by auto
    then show False
      using ∗ ..
  next
    case 2
    then have ‹∃ A. set A ⊆ H ∧ ⊢T ¬ q # A›
      using assms MCS-refute by blast
    then have ‹∃ A. set A ⊆ H ∧ ⊢T p # ¬ q # A›
      using Weaken by (metis set-subset-Cons)
    then have ‹∃ A. set A ⊆ H ∧ ⊢T ¬ (p ⟶ q) # A›
      by fast
    then have ‹¬ (p ⟶ q) ∉ H›
      using assms unfolding consistent-def by auto
    then show False
      using ∗ ..
  qed
 qed
qed

lemma truth-lemma:
 assumes ‹consistent H› ‹maximal H› ‹p ∈ H›
 shows ‹⟦hmodel H⟧ p›
 using Hintikka-model MCS-Hintikka assms by blast
```

4.7 Completeness

theorem *strong-completeness*:

assumes ‹$\forall M :: 'p\ model.\ (\forall q \in X.\ [\![M]\!]\ q) \longrightarrow [\![M]\!]\ p$›
shows ‹$\exists A.\ set\ A \subseteq X \wedge \vdash_T \neg\ p\ \#\ A$›
proof (*rule ccontr*)
assume ‹$\nexists A.\ set\ A \subseteq X \wedge \vdash_T \neg\ p\ \#\ A$›
then have $*$: ‹$\nexists A.\ set\ A \subseteq \{\neg\ p\} \cup X \wedge \vdash_T A$›
using *refute-split1* **by** *blast*

let *?S* = ‹$\{\neg\ p\} \cup X$›
let *?H* = ‹*Extend ?S*›

have ‹*consistent ?S*›
unfolding *consistent-def* **using** $*$ **by** *blast*
then have ‹*consistent ?H*› ‹*maximal ?H*›
using *MCS-Extend'* **by** *blast+*
then have ‹$p \in ?H \longrightarrow [\![hmodel\ ?H]\!]\ p$› **for** *p*
using *truth-lemma* **by** *fastforce*
then have ‹$p \in ?S \longrightarrow [\![hmodel\ ?H]\!]\ p$› **for** *p*
using *Extend-subset* **by** *blast*
then have ‹$[\![hmodel\ ?H]\!]\ (\neg\ p)$› ‹$\forall q \in X.\ [\![hmodel\ ?H]\!]\ q$›
by *blast+*
moreover from *this* **have** ‹$[\![hmodel\ ?H]\!]\ p$›
using *assms(1)* **by** *blast*
ultimately show *False*
by *simp*
qed

abbreviation *valid* :: ‹$'p\ fm \Rightarrow bool$› **where**
‹$valid\ p \equiv \forall M.\ [\![M]\!]\ p$›

theorem *completeness*:
assumes ‹*valid p*›
shows ‹$\vdash_T [\neg\ p]$›
using *assms strong-completeness*[**where** X=‹{}›] **by** *auto*

theorem *main*: ‹$valid\ p \longleftrightarrow \vdash_T [\neg\ p]$›
using *completeness soundness'* **by** *blast*

end

Chapter 5

Example: Propositional Sequent Calculus

theory *Example-Propositional-SC* **imports** *Derivations* **begin**

5.1 Syntax

datatype *'p fm*
  = *Fls* (‹⊥›)
  | *Pro 'p* (‹‡›)
  | *Imp* ‹*'p fm*› ‹*'p fm*› (**infixr** ‹⟶› *55*)

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

5.2 Semantics

type-synonym *'p model* = ‹*'p* ⇒ *bool*›

fun *semantics* :: ‹*'p model* ⇒ *'p fm* ⇒ *bool*› (‹⟦-⟧›) **where**
  ‹⟦-⟧ ⊥ = *False*›
| ‹⟦*I*⟧ (‡*P*) = *I P*›
| ‹⟦*I*⟧ (*p* ⟶ *q*) = (⟦*I*⟧ *p* ⟶ ⟦*I*⟧ *q*)›

5.3 Calculus

inductive *Calculus* :: ‹*'p fm list* ⇒ *'p fm list* ⇒ *bool*› (‹- ⊢$_S$ -› [*50*, *50*] *50*) **where**
  Axiom [*intro*]: ‹*p* # *A* ⊢$_S$ *p* # *B*›
| *FlsL* [*intro*]: ‹⊥ # *A* ⊢$_S$ *B*›
| *FlsR* [*dest*]: ‹*A* ⊢$_S$ ⊥ # *B* ⟹ *A* ⊢$_S$ *B*›
| *ImpL* [*intro*]: ‹*A* ⊢$_S$ *p* # *B* ⟹ *q* # *A* ⊢$_S$ *B* ⟹ (*p* ⟶ *q*) # *A* ⊢$_S$ *B*›
| *ImpR* [*intro*]: ‹*p* # *A* ⊢$_S$ *q* # *B* ⟹ *A* ⊢$_S$ (*p* ⟶ *q*) # *B*›
| *Cut* [*elim*]: ‹*A* ⊢$_S$ *p* # *B* ⟹ *p* # *C* ⊢$_S$ *D* ⟹ *A* @ *C* ⊢$_S$ *B* @ *D*›
| *WeakenL*: ‹*A* ⊢$_S$ *B* ⟹ *set A* ⊆ *set A'* ⟹ *A'* ⊢$_S$ *B*›

| *WeakenR*: ‹$A \vdash_S B \Longrightarrow set\ B \subseteq set\ B' \Longrightarrow A \vdash_S B'$›

lemma *Boole*: ‹$\neg\ p\ \#\ A \vdash_S [] \Longrightarrow A \vdash_S [p]$›
by (*metis Axiom Cut ImpR WeakenR append-self-conv2 self-append-conv set-subset-Cons*)

5.4 Soundness

theorem *soundness*: ‹$A \vdash_S B \Longrightarrow \forall\, q \in set\ A.\ [\![I]\!]\ q \Longrightarrow \exists\, p \in set\ B.\ [\![I]\!]\ p$›
by (*induct A B rule*: *Calculus.induct*) *auto*

corollary *soundness'*: ‹$[] \vdash_S [p] \Longrightarrow [\![I]\!]\ p$›
using *soundness* **by** *fastforce*

corollary ‹$\neg\ ([] \vdash_S [])$›
using *soundness* **by** *fastforce*

5.5 Maximal Consistent Sets

definition *consistent* :: ‹$'p\ fm\ set \Rightarrow bool$› **where**
‹$consistent\ S \equiv \nexists\, S'.\ set\ S' \subseteq S \wedge S' \vdash_S [\bot]$›

interpretation *MCS-No-Saturation consistent*
proof
fix $S\ S'$ :: ‹$'p\ fm\ set$›
assume ‹$consistent\ S$› ‹$S' \subseteq S$›
then show ‹$consistent\ S'$›
unfolding *consistent-def* **by** *fast*
next
fix S :: ‹$'p\ fm\ set$›
assume ‹$\neg\ consistent\ S$›
then show ‹$\exists\, S' \subseteq S.\ finite\ S' \wedge \neg\ consistent\ S'$›
unfolding *consistent-def* **by** *blast*
next
show ‹$infinite\ (UNIV :: 'p\ fm\ set)$›
using *infinite-UNIV-size*[*of* ‹$\lambda p.\ p \longrightarrow p$›] **by** *simp*
qed

interpretation *Derivations-MCS-Cut* ‹$\lambda A\ p.\ A \vdash_S [p]$› *consistent* ‹$\bot$›
proof
fix $A\ B$ **and** p :: ‹$'p\ fm$›
assume ‹$A \vdash_S [p]$› ‹$set\ A \subseteq set\ B$›
then show ‹$B \vdash_S [p]$›
using *WeakenL* **by** *blast*
next
fix S :: ‹$'p\ fm\ set$›
show ‹$consistent\ S \longleftrightarrow (\nexists\, S'.\ set\ S' \subseteq S \wedge S' \vdash_S [\bot])$›
unfolding *consistent-def* **..**
next

```
  fix A and p :: ⟨'p fm⟩
  assume ⟨p ∈ set A⟩
  then show ⟨A ⊢S [p]⟩
    by (metis Axiom WeakenL set-ConsD subsetI)
next
  fix A B and p q :: ⟨'p fm⟩
  assume ⟨A ⊢S [p]⟩ ⟨p # B ⊢S [q]⟩
  then show ⟨A @ B ⊢S [q]⟩
    using Cut by fastforce
qed
```

5.6 Truth Lemma

```
abbreviation hmodel :: ⟨'p fm set ⇒ 'p model⟩ where
  ⟨hmodel H ≡ λP. ‡P ∈ H⟩

fun interp :: ⟨'p model ⇒ ('p model ⇒ 'p fm ⇒ bool) ⇒ 'p fm ⇒ bool⟩ where
  ⟨interp - - ⊥ = False⟩
| ⟨interp I - (‡P) = I P⟩
| ⟨interp I X (p ⟶ q) = (X I p ⟶ X I q)⟩

fun rel :: ⟨'p fm set ⇒ 'p model ⇒ 'p fm ⇒ bool⟩ where
  ⟨rel H - p = (p ∈ H)⟩

theorem Hintikka-model':
  assumes ⟨⋀p. interp (hmodel H) (rel H) p ⟷ p ∈ H⟩
  shows ⟨p ∈ H ⟷ ⟦hmodel H⟧ p⟩
proof (induct p rule: wf-induct[where r=⟨measure size⟩])
  case 1
  then show ?case ..
next
  case (2 x)
  then show ?case
    using assms[of x] by (cases x) simp-all
qed

lemma Hintikka-Extend:
  assumes ⟨consistent H⟩ ⟨maximal H⟩
  shows ⟨interp (hmodel H) (rel H) p ⟷ p ∈ H⟩
proof (cases p)
  case Fls
  have ⟨⊥ ∉ H⟩
    using assms MCS-derive consistent-def by blast
  then show ?thesis
    using Fls by simp
next
  case (Imp p q)
  have ⟨A ⊢S [q] ⟹ A ⊢S [p ⟶ q]⟩ for A
    by (meson ImpR WeakenL set-subset-Cons)
```

moreover have ‹$p \# A \vdash_S [\bot] \Longrightarrow A \vdash_S [p \longrightarrow q]$› **for** A
by (*meson FlsR ImpR WeakenR set-subset-Cons*)
moreover have ‹$A \vdash_S [p \longrightarrow q] \Longrightarrow B \vdash_S [p] \Longrightarrow A @ B \vdash_S [q]$› **for** A B
using *Cut* **by** (*metis Axiom ImpL append-Nil append-Nil2*)
ultimately have ‹$(p \in H \longrightarrow q \in H) \longleftrightarrow p \longrightarrow q \in H$›
using *assms MCS-derive MCS-derive-fls Axiom*
by (*metis append-Cons append-Nil insert-subset list.simps*(15))
then show *?thesis*
using *Imp* **by** *simp*
qed *simp*

interpretation *Truth-No-Saturation consistent interp semantics* ‹$\lambda H.\ \{hmodel\ H\}$› *rel*
proof
fix p **and** M :: ‹$'p$ *model*›
show ‹$[\![M]\!]\ p \longleftrightarrow$ *interp M semantics p*›
by (*induct p*) *auto*
next
fix p **and** H :: ‹$'p$ *fm set*› **and** M :: ‹$'p$ *model*›
assume ‹$\forall M \in \{hmodel\ H\}.\ \forall p.\ interp\ M\ (rel\ H)\ p \longleftrightarrow rel\ H\ M\ p$› ‹$M \in \{hmodel\ H\}$›
then show ‹$[\![M]\!]\ p \longleftrightarrow$ *rel H M p*›
using *Hintikka-model′* **by** *auto*
next
fix H :: ‹$'p$ *fm set*›
assume ‹*consistent H*› ‹*maximal H*›
then show ‹$\forall M \in \{hmodel\ H\}.\ \forall p.\ interp\ M\ (rel\ H)\ p \longleftrightarrow rel\ H\ M\ p$›
using *Hintikka-Extend* **by** *auto*
qed

5.7 Completeness

theorem *strong-completeness*:
assumes ‹$\forall M$ :: $'p$ *model*. $(\forall q \in X.\ [\![M]\!]\ q) \longrightarrow [\![M]\!]\ p$›
shows ‹$\exists A.$ *set* $A \subseteq X \wedge A \vdash_S [p]$›
proof (*rule ccontr*)
assume ‹$\nexists A.$ *set* $A \subseteq X \wedge A \vdash_S [p]$›
then have ‹$\nexists A.$ *set* $A \subseteq X \wedge \neg\ p \# A \vdash_S []$›
using *Boole* **by** *blast*
then have ‹$\nexists A.$ *set* $A \subseteq X \wedge \neg\ p \# A \vdash_S [\bot]$›
by *fast*
then have $*$: ‹$\nexists A.$ *set* $A \subseteq \{\neg\ p\} \cup X \wedge A \vdash_S [\bot]$›
using *derive-split1* **by** *blast*

let *?S* = ‹$\{\neg\ p\} \cup X$›
let *?H* = ‹*Extend ?S*›

have ‹*consistent ?S*›
unfolding *consistent-def* **using** $*$ **by** *blast*
then have ‹*consistent ?H*› ‹*maximal ?H*›
using *MCS-Extend′* **by** *blast*+

```
  then have ‹p ∈ ?H ⟷ ⟦hmodel ?H⟧ p› for p
    using truth-lemma-no-saturation by fastforce
  then have ‹p ∈ ?S ⟶ ⟦hmodel ?H⟧ p› for p
    using Extend-subset by blast
  then have ‹⟦hmodel ?H⟧ (¬ p)› ‹∀ q ∈ X. ⟦hmodel ?H⟧ q›
    by blast+
  moreover from this have ‹⟦hmodel ?H⟧ p›
    using assms(1) by blast
  ultimately show False
    by simp
qed

abbreviation valid :: ‹'p fm ⇒ bool› where
  ‹valid p ≡ ∀ M. ⟦M⟧ p›

theorem completeness:
  assumes ‹valid p›
  shows ‹[] ⊢S [p]›
  using assms strong-completeness[where X=‹{}›] by auto

theorem main: ‹valid p ⟷ [] ⊢S [p]›
  using completeness soundness' by blast

end
```

Chapter 6

Example: Modal Logic

theory *Example-Modal-Logic* **imports** *Derivations* **begin**

6.1 Syntax

datatype $('i, 'p)$ *fm*
  = *Fls* (‹⊥›)
  | *Pro* $'p$
  | *Imp* ‹$('i, 'p)$ *fm*› ‹$('i, 'p)$ *fm*› (**infixr** ‹⟶› *55*)
  | *Box* $'i$ ‹$('i, 'p)$ *fm*› (‹□›)

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

6.2 Semantics

record $('i, 'p, 'w)$ *kripke* =
  $\mathcal{W}$ :: ‹$'w$ *set*›
  $\mathcal{K}$ :: ‹$'i \Rightarrow 'w \Rightarrow 'w$ *set*›
  π :: ‹$'w \Rightarrow 'p \Rightarrow$ *bool*›

type-synonym $('i, 'p, 'w)$ *ctx* = ‹$('i, 'p, 'w)$ *kripke* × $'w$›

fun *semantics* :: ‹$('i, 'p, 'w)$ *ctx* ⇒ $('i, 'p)$ *fm* ⇒ *bool*› (‹- ⊨ -› [*50*, *50*] *50*) **where**
  ‹- ⊨ ⊥ ⟷ *False*›
| ‹$(M, w) \models$ *Pro* $x \longleftrightarrow \pi\ M\ w\ x$›
| ‹$(M, w) \models p \longrightarrow q \longleftrightarrow (M, w) \models p \longrightarrow (M, w) \models q$›
| ‹$(M, w) \models \Box\ i\ p \longleftrightarrow (\forall v \in \mathcal{W}\ M \cap \mathcal{K}\ M\ i\ w.\ (M, v) \models p)$›

6.3 Calculus

primrec *eval* :: ‹$('p \Rightarrow$ *bool*$) \Rightarrow (('i, 'p)$ *fm* ⇒ *bool*$) \Rightarrow ('i, 'p)$ *fm* ⇒ *bool*› **where**
  ‹*eval* - - ⊥ = *False*›
| ‹*eval g* - (*Pro x*) = *g x*›

| ‹*eval g h* (*p* ⟶ *q*) = (*eval g h p* ⟶ *eval g h q*)›
| ‹*eval* - *h* (□ *i p*) = *h* (□ *i p*)›

abbreviation ‹*tautology p* ≡ ∀ *g h*. *eval g h p*›

inductive *SystemK* :: ‹('*i*, '*p*) *fm* ⇒ *bool*› (‹⊢□ -› [*50*] *50*) **where**
A1: ‹*tautology p* ⟹ ⊢□ *p*›
| *A2*: ‹⊢□ □ *i* (*p* ⟶ *q*) ⟶ □ *i p* ⟶ □ *i q*›
| *R1*: ‹⊢□ *p* ⟹ ⊢□ *p* ⟶ *q* ⟹ ⊢□ *q*›
| *R2*: ‹⊢□ *p* ⟹ ⊢□ □ *i p*›

primrec *imply* :: ‹('*i*, '*p*) *fm list* ⇒ ('*i*, '*p*) *fm* ⇒ ('*i*, '*p*) *fm*› (**infixr** ‹⇝› *56*) **where**
‹([] ⇝ *q*) = *q*›
| ‹(*p* # *ps* ⇝ *q*) = (*p* ⟶ *ps* ⇝ *q*)›

abbreviation *K-assms* (‹- ⊢□ -› [*50*, *50*] *50*) **where**
‹*G* ⊢□ *p* ≡ ⊢□ *G* ⇝ *p*›

6.4 Soundness

lemma *eval-semantics*:
‹*eval* (*pi w*) (λ*q*. (⦇𝒲 = *W*, 𝒦 = *r*, π = *pi*⦈, *w*) ⊨ *q*) *p* = ((⦇𝒲 = *W*, 𝒦 = *r*, π = *pi*⦈, *w*) ⊨ *p*)›
by (*induct p*) *simp-all*

lemma *tautology*:
assumes ‹*tautology p*›
shows ‹(*M*, *w*) ⊨ *p*›
proof −
from *assms* **have** ‹*eval* (*g w*) (λ*q*. (⦇𝒲 = *W*, 𝒦 = *r*, π = *g*⦈, *w*) ⊨ *q*) *p*› **for** *W g r*
by *simp*
then have ‹(⦇𝒲 = *W*, 𝒦 = *r*, π = *g*⦈, *w*) ⊨ *p*› **for** *W g r*
using *eval-semantics* **by** *fast*
then show ‹(*M*, *w*) ⊨ *p*›
by (*metis kripke.cases*)
qed

theorem *soundness*:
assumes ‹⋀*M w p*. *A p* ⟹ *w* ∈ 𝒲 *M* ⟹ (*M*, *w*) ⊨ *p*›
shows ‹⊢□ *p* ⟹ *w* ∈ 𝒲 *M* ⟹ (*M*, *w*) ⊨ *p*›
by (*induct p arbitrary*: *w rule*: *SystemK.induct*) (*auto simp*: *assms tautology*)

6.5 Derived rules

lemma *K-imply-head*: ‹*p* # *ps* ⊢□ *p*›
proof −
have ‹*tautology* (*p* # *ps* ⇝ *p*)›
by (*induct ps*) *simp-all*
then show *?thesis*

```
    using A1 by blast
qed

lemma K-imply-Cons:
  assumes ‹ps ⊢□ q›
  shows ‹p # ps ⊢□ q›
proof −
  have ‹⊢□ (ps ⇝ q ⟶ p # ps ⇝ q)›
    by (simp add: A1)
  with R1 assms show ?thesis .
qed

lemma K-right-mp:
  assumes ‹ps ⊢□ p› ‹ps ⊢□ p ⟶ q›
  shows ‹ps ⊢□ q›
proof −
  have ‹tautology (ps ⇝ p ⟶ ps ⇝ (p ⟶ q) ⟶ ps ⇝ q)›
    by (induct ps) simp-all
  with A1 have ‹⊢□ ps ⇝ p ⟶ ps ⇝ (p ⟶ q) ⟶ ps ⇝ q› .
  then show ?thesis
    using assms R1 by blast
qed

lemma deduct1: ‹ps ⊢□ p ⟶ q ⟹ p # ps ⊢□ q›
  by (meson K-right-mp K-imply-Cons K-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 K-ImpI: ‹p # ps ⊢□ q ⟹ ps ⊢□ p ⟶ q›
  by (metis imply.simps imply-append imply-swap-append)

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

lemma add-imply [simp]: ‹⊢□ q ⟹ ps ⊢□ q›
  using K-imply-head R1 by auto

lemma K-imply-weaken: ‹ps ⊢□ q ⟹ set ps ⊆ set ps' ⟹ ps' ⊢□ q›
  by (induct ps arbitrary: q) (simp, metis K-right-mp K-ImpI imply-mem insert-subset list.set(2))

lemma K-Boole:
```

```
  assumes ‹(¬ p) # G ⊢□ ⊥›
  shows ‹G ⊢□ p›
proof −
  have ‹G ⊢□ ¬ ¬ p›
    using assms K-ImpI by blast
  moreover have ‹tautology (G ⇝ ¬ ¬ p ⟶ G ⇝ p)›
    by (induct G) simp-all
  then have ‹⊢□ (G ⇝ ¬ ¬ p ⟶ G ⇝ p)›
    using A1 by blast
  ultimately show ?thesis
    using R1 by blast
qed

lemma K-distrib-K-imp:
  assumes ‹⊢□ □ i (G ⇝ q)›
  shows ‹map (□ i) G ⊢□ □ i q›
proof −
  have ‹⊢□ □ i (G ⇝ q) ⟶ map (□ i) G ⇝ □ i q›
  proof (induct G)
    case Nil
    then show ?case
      by (simp add: A1)
  next
    case (Cons a G)
    have ‹⊢□ □ i (a # G ⇝ q) ⟶ □ i a ⟶ □ i (G ⇝ q)›
      by (simp add: A2)
    moreover have
      ‹⊢□ ((□ i (a # G ⇝ q) ⟶ □ i a ⟶ □ i (G ⇝ q)) ⟶
        (□ i (G ⇝ q) ⟶ map (□ i) G ⇝ □ i q) ⟶
        (□ i (a # G ⇝ q) ⟶ □ i a ⟶ map (□ i) G ⇝ □ i q))›
      by (simp add: A1)
    ultimately have ‹⊢□ □ i (a # G ⇝ q) ⟶ □ i a ⟶ map (□ i) G ⇝ □ i q›
      using Cons R1 by blast
    then show ?case
      by simp
  qed
  then show ?thesis
    using assms R1 by blast
qed

interpretation Derivations K-assms
proof
  fix A B and p :: ‹('i, 'p) fm›
  assume ‹⊢□ A ⇝ p› ‹set A ⊆ set B›
  then show ‹⊢□ B ⇝ p›
    using K-imply-weaken by blast
qed
```

6.6 Maximal Consistent Sets

definition *consistent* :: ‹('i, 'p) *fm set* ⇒ *bool*› **where**
‹*consistent S* ≡ ∄*S'*. *set S'* ⊆ *S* ∧ *S'* ⊢□ ⊥›

interpretation *MCS-No-Saturation consistent*
proof
fix *S S'* :: ‹('i, 'p) *fm set*›
assume ‹*consistent S*› ‹*S'* ⊆ *S*›
then show ‹*consistent S'*›
unfolding *consistent-def* **by** *fast*
next
fix *S* :: ‹('i, 'p) *fm set*›
assume ‹¬ *consistent S*›
then show ‹∃*S'* ⊆ *S*. *finite S'* ∧ ¬ *consistent S'*›
unfolding *consistent-def* **by** *blast*
next
show ‹*infinite* (*UNIV* :: ('i, 'p) *fm set*)›
using *infinite-UNIV-size*[*of* ‹λ*p*. *p* ⟶ *p*›] **by** *simp*
qed

interpretation *Derivations-MCS-Cut K-assms consistent* ‹⊥›
proof
fix *S* :: ‹('i, 'p) *fm set*›
show ‹*consistent S* = (∄*S'*. *set S'* ⊆ *S* ∧ *S'* ⊢□ ⊥)›
unfolding *consistent-def* **..**
next
fix *A* **and** *p* :: ‹('i, 'p) *fm*›
assume ‹*p* ∈ *set A*›
then show ‹*A* ⊢□ *p*›
by (*metis K-imply-head K-imply-weaken Un-upper2 set-append split-list-first*)
next
fix *A B* **and** *p q* :: ‹('i, 'p) *fm*›
assume ‹*A* ⊢□ *p*› ‹*p* # *B* ⊢□ *q*›
then show ‹*A* @ *B* ⊢□ *q*›
by (*metis K-imply-head K-right-mp R1 imply.simps*(2) *imply-append*)
qed

lemma *exists-finite-inconsistent*:
assumes ‹¬ *consistent* ({¬ *p*} ∪ *V*)›
obtains *W* **where** ‹{¬ *p*} ∪ *W* ⊆ {¬ *p*} ∪ *V*› ‹(¬ *p*) ∉ *W*› ‹*finite W*› ‹¬ *consistent* ({¬ *p*} ∪ *W*)›
proof −
obtain *W'* **where** *W'*: ‹*set W'* ⊆ {¬ *p*} ∪ *V*› ‹*W'* ⊢□ ⊥›
using *assms* **unfolding** *consistent-def* **by** *blast*
let *?S* = ‹*removeAll* (¬ *p*) *W'*›
have ‹¬ *consistent* ({¬ *p*} ∪ *set ?S*)›
unfolding *consistent-def* **using** *W'*(2) **by** *auto*
moreover have ‹*finite* (*set ?S*)›
by *blast*

moreover have ‹{¬ *p*} ∪ *set ?S* ⊆ {¬ *p*} ∪ *V*›
using *W'*(*1*) **by** *auto*
moreover have ‹(¬ *p*) ∉ *set ?S*›
by *simp*
ultimately show *?thesis*
by (*meson that*)
qed

lemma *MCS-consequent*:
assumes ‹*consistent V*› ‹*maximal V*› ‹(*p* ⟶ *q*) ∈ *V*› ‹*p* ∈ *V*›
shows ‹*q* ∈ *V*›
using *assms MCS-derive*
by (*metis* (*mono-tags*, *lifting*) *K-imply-Cons K-imply-head K-right-mp insert-subset list.simps*(*15*))

theorem *deriv-in-maximal*:
assumes ‹*consistent V*› ‹*maximal V*› ‹⊢□ *p*›
shows ‹*p* ∈ *V*›
using *assms R1 derive-split1* **unfolding** *consistent-def maximal-def* **by** (*metis imply.simps*(*2*))

theorem *exactly-one-in-maximal*:
assumes ‹*consistent V*› ‹*maximal V*›
shows ‹*p* ∈ *V* ⟷ (¬ *p*) ∉ *V*›
using *assms MCS-derive MCS-derive-fls* **by** (*metis K-Boole K-imply-Cons K-imply-head K-right-mp*)

6.7 Truth Lemma

abbreviation *pi* :: ‹('*i*, '*p*) *fm set* ⇒ '*p* ⇒ *bool*› **where**
‹*pi V x* ≡ *Pro x* ∈ *V*›

abbreviation *known* :: ‹('*i*, '*p*) *fm set* ⇒ '*i* ⇒ ('*i*, '*p*) *fm set*› **where**
‹*known V i* ≡ {*p*. □ *i p* ∈ *V*}›

abbreviation *reach* :: ‹'*i* ⇒ ('*i*, '*p*) *fm set* ⇒ ('*i*, '*p*) *fm set set*› **where**
‹*reach i V* ≡ {*W*. *known V i* ⊆ *W*}›

abbreviation *mcss* :: ‹('*i*, '*p*) *fm set set*› **where**
‹*mcss* ≡ {*W*. *consistent W* ∧ *maximal W*}›

abbreviation *canonical* :: ‹('*i*, '*p*, ('*i*, '*p*) *fm set*) *kripke*› **where**
‹*canonical* ≡ (|*W* = *mcss*, *K* = *reach*, π = *pi*|)›

fun *interp* ::
‹('*i*, '*p*, '*w*) *ctx* ⇒ (('*i*, '*p*, '*w*) *ctx* ⇒ ('*i*, '*p*) *fm* ⇒ *bool*) ⇒ ('*i*, '*p*) *fm* ⇒ *bool*› **where**
‹*interp* - - ⊥ = *False*›
| ‹*interp* (*M*, *w*) - (*Pro x*) = π *M w x*›
| ‹*interp* (*M*, *w*) *X* (*p* ⟶ *q*) = (*X* (*M*, *w*) *p* ⟶ *X* (*M*, *w*) *q*)›
| ‹*interp* (*M*, *w*) *X* (□ *i p*) = (∀ *v* ∈ *W M* ∩ *K M i w*. *X* (*M*, *v*) *p*)›

fun *rel* :: ‹('*i*, '*p*) *fm set* ⇒ ('*i*, '*p*, ('*i*, '*p*) *fm set*) *ctx* ⇒ ('*i*, '*p*) *fm* ⇒ *bool*› **where**

```
  ‹rel - (-, w) p = (p ∈ w)›

lemma Hintikka-model':
  fixes V :: ‹('i, 'p) fm set›
  assumes ‹⋀(V :: ('i, 'p) fm set) p. V ∈ mcss ⟹ interp (canonical, V) (rel H) p ⟷ p ∈ V›
  shows ‹V ∈ mcss ⟹ (canonical, V) ⊨ p ⟷ p ∈ V›
proof (induct p arbitrary: V rule: wf-induct[where r=‹measure size›])
  case 1
  then show ?case ..
next
  case (2 x)
  then show ?case
    using assms[of V x] by (cases x) auto
qed

lemma maximal-extension:
  fixes V :: ‹('i, 'p) fm set›
  assumes ‹consistent V›
  shows ‹∃ W. V ⊆ W ∧ consistent W ∧ maximal W›
  using assms MCS-Extend' Extend-subset by meson

lemma Hintikka-canonical:
  fixes V :: ‹('i, 'p) fm set›
  assumes ‹V ∈ mcss›
  shows ‹interp (canonical, V) (rel H) p = rel H (canonical, V) p›
proof (cases p)
  case Fls
  have ‹⊥ ∉ V›
    using assms MCS-derive unfolding consistent-def by blast
  then show ?thesis
    using Fls by simp
next
  case (Imp p q)
  have ‹(p ∈ V ⟶ q ∈ V) ⟷ p ⟶ q ∈ V›
    using assms MCS-derive MCS-derive-fls MCS-consequent
    by (metis (no-types, lifting) CollectD K-Boole K-ImpI K-imply-Cons)
  then show ?thesis
    using Imp by simp
next
  case (Box i p)
  have ‹(∀ v ∈ mcss ∩ reach i V. p ∈ v) = (□ i p ∈ V)›
  proof
    assume ‹□ i p ∈ V›
    then show ‹∀ v ∈ mcss ∩ reach i V. p ∈ v›
      by auto
  next
    assume ∗: ‹∀ v ∈ mcss ∩ reach i V. p ∈ v›

    have ‹consistent V› ‹maximal V›
```

```
    using ‹V ∈ mcss› by blast+

  have ‹¬ consistent ({¬ p} ∪ known V i)›
  proof
    assume ‹consistent ({¬ p} ∪ known V i)›
    then obtain W where W: ‹{¬ p} ∪ known V i ⊆ W› ‹consistent W› ‹maximal W›
      using ‹V ∈ mcss› maximal-extension by blast
    then have ‹(canonical, W) ⊨ ¬ p›
      using ∗ exactly-one-in-maximal by auto
    moreover have ‹W ∈ reach i V› ‹W ∈ mcss›
      using W by simp-all
    ultimately have ‹(canonical, V) ⊨ ¬ □ i p›
      by auto
    then show False
      using ∗ W(1) ‹W ∈ mcss› exactly-one-in-maximal by auto
  qed

  then obtain W where W:
    ‹{¬ p} ∪ W ⊆ {¬ p} ∪ known V i› ‹(¬ p) ∉ W› ‹finite W› ‹¬ consistent ({¬ p} ∪ W)›
    using exists-finite-inconsistent by metis

  obtain L where L: ‹set L = W›
    using ‹finite W› finite-list by blast
  then have ‹⊢□ L ⇝ p›
    using W(4) derive-split1 unfolding consistent-def by (meson K-Boole K-imply-weaken)
  then have ‹⊢□ □ i (L ⇝ p)›
    using R2 by fast
  then have ‹map (□ i) L ⊢□ □ i p›
    using K-distrib-K-imp by fast
  then have ‹(map (□ i) L ⇝ □ i p) ∈ V›
    using deriv-in-maximal ‹V ∈ mcss› by blast
  then show ‹□ i p ∈ V›
    using L W(1−2)
  proof (induct L arbitrary: W)
    case (Cons a L)
    then have ‹□ i a ∈ V›
      by auto
    then have ‹(map (□ i) L ⇝ □ i p) ∈ V›
      using Cons(2) ‹consistent V› ‹maximal V› MCS-consequent by auto
    then show ?case
      using Cons by auto
  qed simp
 qed
 then show ?thesis
  using Box by simp
qed simp

interpretation Truth-No-Saturation consistent interp semantics
 ‹λ-. {(canonical, V) |V. V ∈ mcss}› rel
```

proof
fix *p* **and** *M* :: ‹('*i*, '*p*, ('*i*, '*p*) *fm set*) *ctx*›
show ‹(*M* ⊨ *p*) = *interp M semantics p*›
by (*cases M*, *induct p*) *simp-all*
next
fix *p* **and** *H* :: ‹('*i*, '*p*) *fm set*› **and** *M* :: ‹('*i*, '*p*, ('*i*, '*p*) *fm set*) *ctx*›
assume ‹∀ *M*∈{(*canonical*, *V*) | *V*. *V* ∈ *mcss*}. ∀ *p*. *interp M* (*rel H*) *p* = *rel H M p*›
‹*M* ∈ {(*canonical*, *V*) | *V*. *V* ∈ *mcss*}›
then show ‹(*M* ⊨ *p*) = *rel H M p*›
using *Hintikka-model'*[*of H - p*] **by** *auto*
next
fix *H* :: ‹('*i*, '*p*) *fm set*›
assume ‹*consistent H*› ‹*maximal H*›
then show ‹∀ *M*∈{(*canonical*, *V*) | *V*. *V* ∈ *mcss*}. ∀ *p*. *interp M* (*rel H*) *p* = *rel H M p*›
using *Hintikka-canonical* **by** *blast*
qed

lemma *Truth-lemma*:
fixes *p* :: ‹('*i*, '*p*) *fm*›
assumes ‹*consistent V*› ‹*maximal V*›
shows ‹(*canonical*, *V*) ⊨ *p* ⟷ *p* ∈ *V*›
using *assms truth-lemma-no-saturation* **by** *fastforce*

lemma *canonical-model*:
assumes ‹*consistent S*› ‹*p* ∈ *S*›
defines ‹*V* ≡ *Extend S*› **and** ‹*M* ≡ *canonical*›
shows ‹(*M*, *V*) ⊨ *p*› ‹*consistent V*› ‹*maximal V*›
proof −
have ‹*consistent V*›
using ‹*consistent S*› **unfolding** *V-def* **using** *consistent-Extend* **by** *auto*
have ‹*maximal V*›
unfolding *V-def* **using** *maximal-Extend* **by** *blast*
{ fix *x*
assume ‹*x* ∈ *S*›
then have ‹*x* ∈ *V*›
unfolding *V-def* **using** *Extend-subset* **by** *blast*
then have ‹(*M*, *V*) ⊨ *x*›
unfolding *M-def* **using** *Truth-lemma* ‹*consistent V*› ‹*maximal V*› **by** *blast* **}**
then show ‹(*M*, *V*) ⊨ *p*›
using ‹*p* ∈ *S*› **by** *blast+*
show ‹*consistent V*› ‹*maximal V*›
by *fact+*
qed

6.8 Completeness

abbreviation *valid* :: ‹('*i*, '*p*) *fm set* ⇒ ('*i*, '*p*) *fm* ⇒ *bool*›
(‹- ⊫ -› [*50*, *50*] *50*) **where**
‹*G* ⊫ *p* ≡ ∀ *M* :: ('*i*, '*p*, ('*i*, '*p*) *fm set*) *kripke*.

$(\forall\, w \in \mathcal{W}\ M.\ (\forall\, q \in G.\ (M,\, w) \models q) \longrightarrow (M,\, w) \models p)\rangle$

theorem *strong-completeness*:
assumes $\langle G \Vdash p\rangle$
shows $\langle \exists\, qs.\ set\ qs \subseteq G \wedge qs \vdash_\Box p\rangle$
proof (*rule ccontr*)
assume $\langle \nexists\, qs.\ set\ qs \subseteq G \wedge qs \vdash_\Box p\rangle$
then have $*$: $\langle \forall\, qs.\ set\ qs \subseteq G \longrightarrow \neg\ (\neg\ p)\ \#\ qs \vdash_\Box \bot\rangle$
using *K-Boole* **by** *blast*

let *?S* = $\langle \{\neg\ p\} \cup G\rangle$
let *?V* = $\langle$*Extend ?S*$\rangle$

have $\langle$*consistent ?S*$\rangle$
using $*$ *derive-split1* **unfolding** *consistent-def* **by** *meson*
then have $\langle (canonical,\ ?V) \models (\neg\ p)\rangle$ $\langle \forall\, q \in G.\ (canonical,\ ?V) \models q\rangle$
using *canonical-model* **by** *fastforce+*
moreover have $\langle ?V \in mcss\rangle$
using $\langle$*consistent ?S*$\rangle$ *maximal-Extend canonical-model*(*2*) **by** *blast*
ultimately have $\langle (canonical,\ ?V) \models p\rangle$
using *assms* **by** *simp*
then show *False*
using $\langle (canonical,\ ?V) \models (\neg\ p)\rangle$ **by** *simp*
qed

corollary *completeness*: $\langle \{\} \Vdash p \Longrightarrow \vdash_\Box p\rangle$
using *strong-completeness*[**where** *G*=$\langle \{\}\rangle$] **by** *simp*

theorem *main*: $\langle \{\} \Vdash p \longleftrightarrow \vdash_\Box p\rangle$
using *soundness completeness* **by** *fast*

end

Chapter 7

Example: Hybrid Logic

theory *Example-Hybrid-Logic* **imports** *Derivations* **begin**

7.1 Syntax

datatype (*nominals-fm*: *'i*, *'p*) *fm*
= *Fls* (‹⊥›)
| *Pro 'p*
| *Nom 'i*
| *Imp* ‹(*'i*, *'p*) *fm*› ‹(*'i*, *'p*) *fm*› (**infixr** ‹⟶› *55*)
| *Dia* ‹(*'i*, *'p*) *fm*› (‹◇›)
| *Sat 'i* ‹(*'i*, *'p*) *fm*› (‹@›)

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

type-synonym (*'i*, *'p*) *lbd* = ‹*'i* × (*'i*, *'p*) *fm*›

primrec *nominals-lbd* :: ‹(*'i*, *'p*) *lbd* ⇒ *'i set*› **where**
‹*nominals-lbd* (*i*, *p*) = {*i*} ∪ *nominals-fm p*›

abbreviation *nominals* :: ‹(*'i*, *'p*) *lbd set* ⇒ *'i set*› **where**
‹*nominals S* ≡ ⋃ *ip* ∈ *S*. *nominals-lbd ip*›

lemma *finite-nominals-fm*: ‹*finite* (*nominals-fm p*)›
by (*induct p*) *simp-all*

lemma *finite-nominals-lbd*: ‹*finite* (*nominals-lbd p*)›
by (*cases p*) (*simp add*: *finite-nominals-fm*)

7.2 Semantics

datatype (*'w*, *'p*) *model* =
Model (*R*: ‹*'w* ⇒ *'w set*›) (*V*: ‹*'w* ⇒ *'p* ⇒ *bool*›)

type-synonym $('i, 'p, 'w)\ ctx = \langle('w, 'p)\ model \times ('i \Rightarrow 'w) \times 'w\rangle$

fun *semantics* :: $\langle('i, 'p, 'w)\ ctx \Rightarrow ('i, 'p)\ fm \Rightarrow bool\rangle$ $(\langle\text{-} \models \text{-}\rangle\ [50, 50]\ 50)$ **where**
  $\langle(M, g, w) \models \bot \longleftrightarrow False\rangle$
| $\langle(M, \text{-}, w) \models Pro\ x \longleftrightarrow V\ M\ w\ x\rangle$
| $\langle(\text{-}, g, w) \models Nom\ i \longleftrightarrow w = g\ i\rangle$
| $\langle(M, g, w) \models (p \longrightarrow q) \longleftrightarrow ((M, g, w) \models p \longrightarrow (M, g, w) \models q)\rangle$
| $\langle(M, g, w) \models \Diamond\ p \longleftrightarrow (\exists\, v \in R\ M\ w.\ (M, g, v) \models p)\rangle$
| $\langle(M, g, \text{-}) \models @i\ p \longleftrightarrow ((M, g, g\ i) \models p)\rangle$

lemma *semantics-fresh*: $\langle i \notin nominals\text{-}fm\ p \Longrightarrow ((M, g, w) \models p) = ((M, g(i := v), w) \models p)\rangle$
  by (*induct p arbitrary*: *w*) *auto*

lemma *semantics-fresh-lbd*:
  $\langle k \notin nominals\text{-}lbd\ (i, p) \Longrightarrow ((M, g, w) \models p) = ((M, g(k := v), w) \models p)\rangle$
  by (*induct p arbitrary*: *w*) *auto*

7.3 Calculus

abbreviation *list-member-syntax* :: $\langle'a \Rightarrow 'a\ list \Rightarrow bool\rangle$ $(\langle\text{-} [\in] \text{-}\rangle\ [51, 51]\ 50)$ **where**
  $\langle x\ [\in]\ A \equiv x \in set\ A\rangle$

inductive *Calculus* :: $\langle('i, 'p)\ lbd\ list \Rightarrow ('i, 'p)\ lbd \Rightarrow bool\rangle$ $(\langle\text{-} \vdash_@ \text{-}\rangle\ [50, 50]\ 50)$ **where**
  Assm [*intro*]: $\langle(i, p)\ [\in]\ A \Longrightarrow A \vdash_@ (i, p)\rangle$
| *Ref* [*intro*]: $\langle A \vdash_@ (i, Nom\ i)\rangle$
| *Nom* [*intro*]: $\langle A \vdash_@ (i, Nom\ k) \Longrightarrow A \vdash_@ (i, p) \Longrightarrow A \vdash_@ (k, p)\rangle$
| *FlsE* [*elim*]: $\langle A \vdash_@ (i, \bot) \Longrightarrow A \vdash_@ (k, p)\rangle$
| *ImpI* [*intro*]: $\langle(i, p) \#\ A \vdash_@ (i, q) \Longrightarrow A \vdash_@ (i, p \longrightarrow q)\rangle$
| *ImpE* [*elim*]: $\langle A \vdash_@ (i, p \longrightarrow q) \Longrightarrow A \vdash_@ (i, p) \Longrightarrow A \vdash_@ (i, q)\rangle$
| *SatI* [*intro*]: $\langle A \vdash_@ (i, p) \Longrightarrow A \vdash_@ (k, @i\ p)\rangle$
| *SatE* [*elim*]: $\langle A \vdash_@ (i, @k\ p) \Longrightarrow A \vdash_@ (k, p)\rangle$
| *DiaI* [*intro*]: $\langle A \vdash_@ (i, \Diamond\ (Nom\ k)) \Longrightarrow A \vdash_@ (k, p) \Longrightarrow A \vdash_@ (i, \Diamond\ p)\rangle$
| *DiaE* [*elim*]: $\langle A \vdash_@ (i, \Diamond\ p) \Longrightarrow k \notin nominals\ (\{(i, p), (j, q)\} \cup set\ A) \Longrightarrow$
    $(k, p) \#\ (i, \Diamond\ (Nom\ k)) \#\ A \vdash_@ (j, q) \Longrightarrow A \vdash_@ (j, q)\rangle$
| *Clas*: $\langle(i, p \longrightarrow q) \#\ A \vdash_@ (i, p) \Longrightarrow A \vdash_@ (i, p)\rangle$
| *Weak*: $\langle A \vdash_@ (i, p) \Longrightarrow (k, q) \#\ A \vdash_@ (i, p)\rangle$

7.4 Soundness

theorem *soundness*: $\langle A \vdash_@ (i, p) \Longrightarrow list\text{-}all\ (\lambda(i, p).\ (M, g, g\ i) \models p)\ A \Longrightarrow (M, g, g\ i) \models p\rangle$
proof (*induct* $\langle(i, p)\rangle$ *arbitrary*: *i p g rule*: *Calculus.induct*)
  case (*Nom A i k p*)
  then show *?case*
    by *fastforce*
next
  case (*DiaE A i p k j q*)
  then have $\langle(M, g, g\ i) \models \Diamond\ p\rangle$
    by *blast*

then obtain v **where** v: ‹$v \in R\ M\ (g\ i)$› ‹$(M,\ g,\ v) \models p$›
by *auto*
let $?g = ‹g(k := v)›$
have ‹$(M,\ ?g,\ ?g\ k) \models p$› ‹$(M,\ ?g,\ ?g\ i) \models \Diamond\ (Nom\ k)$›
using *v fun-upd-same DiaE(3) semantics-fresh-lbd* **by** *fastforce+*
moreover have ‹*list-all* $(\lambda(i,\ p).\ (M,\ ?g,\ ?g\ i) \models p)\ A$›
using *DiaE.prems DiaE.hyps(3) semantics-fresh-lbd* **by** (*induct A*) *fastforce+*
ultimately have ‹*list-all* $(\lambda(i,\ p).\ (M,\ ?g,\ ?g\ i) \models p)\ ((k,\ p) \#\ (i,\ \Diamond\ (Nom\ k)) \#\ A)$›
by *simp*
then have ‹$(M,\ ?g,\ ?g\ j) \models q$›
using *DiaE.hyps* **by** *fast*
then show *?case*
using *DiaE.hyps(3) semantics-fresh-lbd* **by** *fastforce*
qed (*auto simp*: *list-all-iff*)

corollary *soundness'*: ‹$[] \vdash_@ (i,\ p) \Longrightarrow i \notin nominals\text{-}fm\ p \Longrightarrow (M,\ g,\ w) \models p$›
using *soundness semantics-fresh* **by** (*metis fun-upd-same list.pred-inject(1)*)

corollary ‹$\neg\ ([] \vdash_@ (i,\ \bot))$›
using *soundness'* **by** *fastforce*

7.5 Derived Rules

lemma *Assm-head*: ‹$(p,\ i) \#\ A \vdash_@ (p,\ i)$›
by *auto*

lemma *deduct1*: ‹$A \vdash_@ (i,\ p \longrightarrow q) \Longrightarrow (i,\ p) \#\ A \vdash_@ (i,\ q)$›
by (*meson ImpE Weak Assm-head*)

lemma *Boole*: ‹$(i,\ \neg\ p) \#\ A \vdash_@ (i,\ \bot) \Longrightarrow A \vdash_@ (i,\ p)$›
using *Clas FlsE* **by** *meson*

lemma *Weak'*: ‹$A \vdash_@ (i,\ p) \Longrightarrow B\ @\ A \vdash_@ (i,\ p)$›
proof (*induct B*)
case (*Cons b B*)
then show *?case*
by (*cases b*) (*metis Cons Weak append-Cons*)
qed *simp*

lemma *ImpI'*:
assumes ‹$(k,\ q) \#\ A \vdash_@ (i,\ p)$›
shows ‹$A \vdash_@ (i,\ (@k\ q) \longrightarrow p)$›
using *assms*
proof −
have ‹$(k,\ q) \#\ A \vdash_@ (k,\ @i\ p)$›
using *assms* **by** *fast*
then show *?thesis*
by (*meson Assm-head ImpE ImpI SatE Weak*)
qed

lemma *Weaken*: ‹$A \vdash_@ (i, p) \Longrightarrow set\ A \subseteq set\ B \Longrightarrow B \vdash_@ (i, p)$›
proof (*induct A arbitrary*: *i p*)
  case *Nil*
  then show *?case*
    using *Weak'* **by** *fastforce*
next
  case (*Cons kq A*)
  then show *?case*
  proof (*cases kq*)
    case (*Pair k q*)
    then show *?thesis*
      using *Cons* **by** (*meson Assm ImpI' ImpE SatI list.set-intros*(*1*) *set-subset-Cons subset-eq*)
  qed
qed

interpretation *Derivations Calculus*
proof
  fix *A B* **and** *p* :: ‹('*i*, '*p*) *lbd*›
  assume ‹$A \vdash_@ p$› ‹*set A* ⊆ *set B*›
  then show ‹$B \vdash_@ p$›
    by (*cases p*) (*metis Weaken*)
qed

7.6 Maximal Consistent Sets

definition *consistent* :: ‹('*i*, '*p*) *lbd set* ⇒ *bool*› **where**
  ‹*consistent S* ≡ ∄*S' a*. *set S'* ⊆ *S* ∧ $S' \vdash_@ (a, \bot)$›

lemma *consistent-add-witness*:
  assumes ‹*consistent S*› ‹$(i, \Diamond\ p) \in S$› ‹$k \notin nominals\ S$›
  shows ‹*consistent* ({(k, p), $(i, \Diamond\ (Nom\ k))$} ∪ *S*)›
  unfolding *consistent-def*
proof
  assume ‹∃*S' a*. *set S'* ⊆ {(k, p), $(i, \Diamond\ (Nom\ k))$} ∪ *S* ∧ $S' \vdash_@ (a, \bot)$›
  then obtain *S' a* **where** ‹*set S'* ⊆ *S*› ‹$(k, p) \#\ (i, \Diamond\ (Nom\ k)) \#\ S' \vdash_@ (a, \bot)$›
    using *assms derive-split*[**where** *X*=*S* **and** *B*=‹[(k, p), $(i, \Diamond\ (Nom\ k))$]›]
    by (*metis append-Cons append-Nil list.simps*(*15*) *set-empty*)
  then have ‹$(k, p) \#\ (i, \Diamond\ (Nom\ k)) \#\ S' \vdash_@ (i, \bot)$›
    by *fast*
  then have ‹$(k, p) \#\ (i, \Diamond\ (Nom\ k)) \#\ (i, \Diamond\ p) \#\ S' \vdash_@ (i, \bot)$›
    by (*fastforce intro*: *Weaken*)
  moreover have ‹$k \notin nominals$ ({(i, p), $(i, \bot)$} ∪ *set* ($(i, \Diamond\ p) \#\ S'$))›
    using ‹*set S'* ⊆ *S*› *assms*(*2*−*3*) **by** *auto*
  moreover have ‹$(i, \Diamond\ p) \#\ S' \vdash_@ (i, \Diamond\ p)$›
    by *auto*
  ultimately have ‹$(i, \Diamond\ p) \#\ S' \vdash_@ (i, \bot)$›
    by *fastforce*
  moreover have ‹*set* ($(i, \Diamond\ p) \#\ S'$) ⊆ *S*›

```
    using ‹set S' ⊆ S› assms(2) by simp
  ultimately show False
    using assms(1) unfolding consistent-def by blast
qed

fun witness :: ‹('i, 'p) lbd ⇒ ('i, 'p) lbd set ⇒ ('i, 'p) lbd set› where
  ‹witness (i, ◇ p) S = (let k = (SOME k. k ∉ nominals ({(i, p)} ∪ S)) in {(k, p), (i, ◇ (Nom k))})›
| ‹witness (-, -) - = {}›

lemma consistent-witness':
  assumes ‹consistent ({(i, p)} ∪ S)› ‹infinite (UNIV − nominals S)›
  shows ‹consistent (witness (i, p) S ∪ {(i, p)} ∪ S)›
  using assms
proof (induct ‹(i, p)› S arbitrary: i p rule: witness.induct)
  case (1 i p S)
  have ‹infinite (UNIV − nominals ({(i, p)} ∪ S))›
    using 1(2) finite-nominals-lbd
    by (metis UN-Un finite.emptyI finite.insertI finite-UN-I infinite-Diff-fin-Un)
  then have ‹∃ k. k ∉ nominals ({(i, p)} ∪ S)›
    by (simp add: not-finite-existsD set-diff-eq)
  then have ‹(SOME k. k ∉ nominals ({(i, p)} ∪ S)) ∉ nominals ({(i, p)} ∪ S)›
    by (rule someI-ex)
  then obtain k where ‹witness (i, ◇ p) S = {(k, p), (i, ◇ (Nom k))}›
    ‹k ∉ nominals ({(i, ◇ p)} ∪ S)›
    by (simp add: Let-def)
  then show ?case
    using 1(1−2) consistent-add-witness[where S=‹{(i, ◇ p)} ∪ S›] by simp
qed (auto simp: assms)

interpretation MCS-Saturation consistent nominals-lbd witness
proof
  fix S S' :: ‹('i, 'p) lbd set›
  assume ‹consistent S› ‹S' ⊆ S›
  then show ‹consistent S'›
    unfolding consistent-def by fast
next
  fix S :: ‹('i, 'p) lbd set›
  assume ‹¬ consistent S›
  then show ‹∃ S'⊆S. finite S' ∧ ¬ consistent S'›
    unfolding consistent-def by blast
next
  fix ip :: ‹('i, 'p) lbd›
  show ‹finite (nominals-lbd ip)›
    using finite-nominals-fm by (cases ip) simp
next
  fix ip :: ‹('i, 'p) lbd› and S :: ‹('i, 'p) lbd set›
  show ‹finite (nominals (witness ip S))›
    by (induct ip S rule: witness.induct) (auto simp: finite-nominals-fm Let-def)
next
```

fix *ip* **and** *S* :: ‹('*i*, '*p*) *lbd set*›
assume ‹*consistent* ({*ip*} ∪ *S*)› ‹*infinite* (*UNIV* − *nominals S*)›
then show ‹*consistent* (*witness ip S* ∪ {*ip*} ∪ *S*)›
using *consistent-witness'* **by** (*cases ip*) *simp*
next
have ‹*infinite* (*UNIV* :: ('*i*, '*p*) *fm set*)›
using *infinite-UNIV-size*[*of* ‹◇›] **by** *simp*
then show ‹*infinite* (*UNIV* :: ('*i*, '*p*) *lbd set*)›
using *finite-prod* **by** *blast*
qed

interpretation *Derivations-MCS-Cut Calculus consistent* ‹(*undefined*, ⊥)›
proof
fix *S* :: ‹('*i*, '*p*) *lbd set*›
show ‹*consistent S* = (∄*S'*. *set S'* ⊆ *S* ∧ *S'* ⊢@ (*undefined*, ⊥))›
unfolding *consistent-def* **by** *fast*
next
fix *A* **and** *p* :: ‹('*i*, '*p*) *lbd*›
assume ‹*p* [∈] *A*›
then show ‹*A* ⊢@ *p*›
by (*metis Assm surj-pair*)
next
fix *A B* **and** *p q* :: ‹('*i*, '*p*) *lbd*›
assume ‹*A* ⊢@ *p*› ‹*p* # *B* ⊢@ *q*›
then have ‹*A* @ *B* ⊢@ *p*› ‹*p* # *A* @ *B* ⊢@ *q*›
by (*simp-all add*: *derive-struct subsetI*)
then show ‹*A* @ *B* ⊢@ *q*›
by (*cases p*, *cases q*) (*meson ImpE ImpI' SatI*)
qed

lemma *saturated*: ‹*saturated H* ⟹ (*i*, ◇*p*) ∈ *H* ⟹ ∃*k*. (*i*, ◇ (*Nom k*)) ∈ *H* ∧ (*k*, *p*) ∈ *H*›
unfolding *saturated-def* **by** (*metis insert-subset witness.simps*(*1*))

7.7 Nominals

lemma *MCS-Nom-refl*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹(*i*, *Nom i*) ∈ *S*›
using *assms Ref* **by** (*metis MCS-derive MCS-derive-fls*)

lemma *MCS-Nom-sym*:
assumes ‹*consistent S*› ‹*maximal S*› ‹(*i*, *Nom j*) ∈ *S*›
shows ‹(*j*, *Nom i*) ∈ *S*›
using *assms Nom Ref* **by** (*metis MCS-derive*)

lemma *MCS-Nom-trans*:
assumes ‹*consistent S*› ‹*maximal S*› ‹(*i*, *Nom j*) ∈ *S*› ‹(*j*, *Nom k*) ∈ *S*›
shows ‹(*i*, *Nom k*) ∈ *S*›
proof −

```
  have ‹∃ S'. set S' ⊆ S ∧ S' ⊢@ (i, Nom j)›
    using assms MCS-derive by blast
  then have ‹∃ S'. set S' ⊆ S ∧ S' ⊢@ (i, Nom j) ∧ S' ⊢@ (j, Nom k)›
    by (metis Assm-head Calculus.intros(12) assms(4) insert-subset list.set(2))
  then have ‹∃ S'. set S' ⊆ S ∧ S' ⊢@ (i, Nom k)›
    using Nom Ref by metis
  then show ?thesis
    using assms MCS-derive by blast
qed
```

7.8 Truth Lemma

```
fun interp :: ‹('i, 'p, 'w) ctx ⇒ (('i, 'p, 'w) ctx ⇒ ('i, 'p) fm ⇒ bool) ⇒ ('i, 'p) fm ⇒ bool›
  where
    ‹interp (M, g, w) X ⊥ ⟷ False›
  | ‹interp (M, -, w) X (Pro x) ⟷ V M w x›
  | ‹interp (-, g, w) X (Nom i) ⟷ w = g i›
  | ‹interp (M, g, w) X (p ⟶ q) ⟷ X (M, g, w) p ⟶ X (M, g, w) q›
  | ‹interp (M, g, w) X (◇ p) ⟷ (∃ v ∈ R M w. X (M, g, v) p)›
  | ‹interp (M, g, -) X (@ i p) ⟷ X (M, g, g i) p›

fun rel :: ‹('i, 'p) lbd set ⇒ ('i, 'p, 'i) ctx ⇒ ('i, 'p) fm ⇒ bool› where
  ‹rel H (-, -, i) p = ((i, p) ∈ H)›

definition val :: ‹('i, 'p) lbd set ⇒ 'i ⇒ 'p ⇒ bool› where
  ‹val H i x ≡ (i, Pro x) ∈ H›

definition hequiv :: ‹('i, 'p) lbd set ⇒ 'i ⇒ 'i ⇒ bool› where
  ‹hequiv H i j ≡ (i, Nom j) ∈ H›

lemma hequiv-reflp:
  assumes ‹consistent H› ‹maximal H›
  shows ‹reflp (hequiv H)›
  unfolding hequiv-def reflp-def using assms MCS-Nom-refl by fast

lemma hequiv-symp:
  assumes ‹consistent H› ‹maximal H›
  shows ‹symp (hequiv H)›
  unfolding hequiv-def symp-def using assms MCS-Nom-sym by fast

lemma hequiv-transp:
  assumes ‹consistent H› ‹maximal H›
  shows ‹transp (hequiv H)›
  unfolding hequiv-def transp-def using assms MCS-Nom-trans by fast

lemma hequiv-equivp:
  assumes ‹consistent H› ‹maximal H›
  shows ‹equivp (hequiv H)›
  using assms by (simp add: equivpI hequiv-reflp hequiv-symp hequiv-transp)
```

definition *assign* :: ‹('i, 'p) lbd set ⇒ 'i ⇒ 'i› **where**
‹*assign H i* ≡ *minim* (|*UNIV*|) {*j*. *hequiv H i j*}›

lemma *hequiv-ne*:
assumes ‹*consistent H*› ‹*maximal H*›
shows ‹{*j*. *hequiv H i j*} ≠ {}›
unfolding *hequiv-def* **using** *assms MCS-Nom-refl* **by** *fast*

lemma *hequiv-assign*:
assumes ‹*consistent H*› ‹*maximal H*›
shows ‹*hequiv H i* (*assign H i*)›
unfolding *assign-def* **using** *assms hequiv-ne wo-rel.minim-in*
by (*metis Field-card-of card-of-well-order-on mem-Collect-eq top.extremum wo-rel-def*)

lemma *hequiv-Nom*:
assumes ‹*consistent H*› ‹*maximal H*› ‹*hequiv H i j*› ‹(*i*, *p*) ∈ *H*›
shows ‹(*j*, *p*) ∈ *H*›
proof −
have ‹∃ *A*. *set A* ⊆ *H* ∧ *A* ⊢@ (*i*, *p*)›
using *assms MCS-derive* **by** *fast*
then have ‹∃ *A*. *set A* ⊆ *H* ∧ *A* ⊢@ (*i*, *p*) ∧ *A* ⊢@ (*i*, *Nom j*)›
using *assms*(*3*) **unfolding** *hequiv-def* **by** (*metis Assm-head Weak insert-subset list.simps*(*15*))
then have ‹∃ *A*. *set A* ⊆ *H* ∧ *A* ⊢@ (*j*, *p*)›
using *Nom* **by** *fast*
then show *?thesis*
using *assms MCS-derive* **by** *fast*
qed

definition *reach* :: ‹('i, 'p) lbd set ⇒ 'i ⇒ 'i set› **where**
‹*reach H i* ≡ {*assign H j* |*j*. (*i*, ◇ (*Nom j*)) ∈ *H*}›

abbreviation *canonical* :: ‹('i × ('i, 'p) *fm*) *set* ⇒ 'i ⇒ ('i, 'p, 'i) *ctx*› **where**
‹*canonical H i* ≡ (*Model* (*reach H*) (*val H*), *assign H*, *assign H i*)›

lemma *Hintikka-model'*:
assumes ‹⋀*i p*. *interp* (*canonical H i*) (*rel H*) *p* = *rel H* (*canonical H i*) *p*›
shows ‹(*canonical H i* ⊨ *p*) = *rel H* (*canonical H i*) *p*›
proof (*induct p arbitrary*: *i rule*: *wf-induct*[**where** *r*=‹*measure size*›])
case *1*
then show *?case* **..**
next
case (*2 x*)
then show *?case*
using *assms*[*of i x*] **by** (*cases x*) (*auto simp*: *reach-def*)
qed

lemma *Hintikka-Extend'*:
assumes ‹*consistent H*› ‹*maximal H*› ‹*saturated H*›

shows ‹*interp* (*canonical H i*) (*rel H*) *p* = *rel H* (*canonical H i*) *p*›
proof (*cases p*)
case *Fls*
have ‹(*assign H i*, ⊥) ∉ *H*›
using *assms*(1−2) *MCS-derive* **unfolding** *consistent-def* **by** *fast*
then show *?thesis*
using *Fls* **by** *simp*
next
case (*Pro x*)
have ‹*val H* (*assign H i*) *x* = ((*assign H i*, *Pro x*) ∈ *H*)›
unfolding *val-def* **..**
then show *?thesis*
using *Pro* **by** *simp*
next
case (*Nom k*)
have ‹(*assign H i* = *assign H k*) = ((*assign H i*, *Nom k*) ∈ *H*)›
using *assms*(1−2) *hequiv-equivp hequiv-assign* **by** (*metis assign-def equivp-def hequiv-def*)
then show *?thesis*
using *Nom* **by** *simp*
next
case (*Imp p q*)
have ‹(*i*, *p*) # *A* ⊢@ (*a*, ⊥) ⟹ *A* ⊢@ (*i*, *p* ⟶ *q*)› ‹*A* ⊢@ (*i*, *q*) ⟹ *A* ⊢@ (*i*, *p* ⟶ *q*)› **for** *A a i*
by (*auto simp*: *Weak*)
moreover have ‹*A* ⊢@ (*i*, *p* ⟶ *q*) ⟹ (*i*, *p*) # *A* ⊢@ (*i*, *q*)› **for** *A i*
using *deduct1* **.**
ultimately have ‹((*assign H i*, *p*) ∈ *H* ⟶ (*assign H i*, *q*) ∈ *H*) ⟷ (*assign H i*, *p* ⟶ *q*) ∈ *H*›
using *assms*(1−2) *MCS-derive MCS-derive-fls* **by** (*metis insert-subset list.simps*(15))
then show *?thesis*
using *Imp* **by** *simp*
next
case (*Dia p*)
have ‹(∃ *k* ∈ *reach H* (*assign H i*). (*k*, *p*) ∈ *H*) = ((*assign H i*, ◇ *p*) ∈ *H*)›
proof
assume ‹∃ *k* ∈ *reach H* (*assign H i*). (*k*, *p*) ∈ *H*›
then have ‹∃ *k*. (*assign H i*, ◇ (*Nom k*)) ∈ *H* ∧ (*assign H k*, *p*) ∈ *H*›
unfolding *reach-def* **by** *fast*
then have ‹∃ *k*. (*assign H i*, ◇ (*Nom k*)) ∈ *H* ∧ (*k*, *p*) ∈ *H*›
by (*metis assms*(1−2) *hequiv-Nom hequiv-assign hequiv-symp sympD*)
then have ‹∃ *k*. ∃ *A*. *set A* ⊆ *H* ∧ *A* ⊢@ (*assign H i*, ◇ (*Nom k*)) ∧ *A* ⊢@ (*k*, *p*)›
by (*metis Assm-head Weak assms*(1−2) *MCS-derive insert-subset list.simps*(15))
then have ‹∃ *A*. *set A* ⊆ *H* ∧ *A* ⊢@ (*assign H i*, ◇ *p*)›
by *fast*
then show ‹(*assign H i*, ◇ *p*) ∈ *H*›
by (*simp add*: *assms*(1−2) *MCS-derive*)
next
assume ‹(*assign H i*, ◇ *p*) ∈ *H*›
then have ‹∃ *k*. (*assign H i*, ◇ (*Nom k*)) ∈ *H* ∧ (*k*, *p*) ∈ *H*›
using *assms*(3) *saturated* **by** *fast*
then have ‹∃ *k*. (*assign H i*, ◇ (*Nom k*)) ∈ *H* ∧ (*assign H k*, *p*) ∈ *H*›

```
    by (meson assms(1−2) hequiv-Nom hequiv-assign)
  then show ‹∃ k ∈ reach H (assign H i). (k, p) ∈ H›
    unfolding reach-def by fast
 qed
 then show ?thesis
  using Dia by simp
next
 case (Sat k p)
 have ‹(assign H k, p) ∈ H ⟷ (assign H i, @k p) ∈ H›
  by (metis SatE SatI assms(1−2) MCS-derive hequiv-Nom hequiv-assign hequiv-symp sympD)
 then show ?thesis
  using Sat by simp
qed

interpretation Truth-Saturation
 ‹consistent :: ('i, 'p) lbd set ⇒ bool›
 nominals-lbd witness interp semantics ‹λH. {canonical H i |i. True}› rel
proof unfold-locales
 fix p and M :: ‹('i, 'p, 'w) ctx›
 show ‹(M ⊨ p) = interp M semantics p›
  by (cases M, induct p) auto
next
 fix p H and M :: ‹('i, 'p, 'i) ctx›
 assume ‹∀ M ∈ {canonical H i |i. True}. ∀ p. interp M (rel H) p = rel H M p›
  ‹M ∈ {canonical H i |i. True}›
 then show ‹(M ⊨ p) = rel H M p›
  using Hintikka-model′ by fast
next
 fix H :: ‹('i, 'p) lbd set›
 assume ‹consistent H› ‹maximal H› ‹saturated H›
 then show ‹∀ M∈{canonical H i |i. True}. ∀ p. interp M (rel H) p = rel H M p›
  using Hintikka-Extend′ by fast
qed

lemma Truth-lemma:
 assumes ‹consistent H› ‹maximal H› ‹saturated H›
 shows ‹(canonical H i ⊨ p) ⟷ (i, p) ∈ H›
proof −
 have ‹(canonical H i ⊨ p) ⟷ (assign H i, p) ∈ H›
  using truth-lemma-saturation assms by fastforce
 then show ?thesis
  using assms by (meson MCS-Nom-sym hequiv-Nom hequiv-assign hequiv-def)
qed
```

7.9 Cardinalities

```
datatype marker = FlsM | ImpM | DiaM | SatM

type-synonym ('i, 'p) enc = ‹('i + 'p) + marker × nat›
```

abbreviation ‹*NOM i* ≡ *Inl* (*Inl i*)›
abbreviation ‹*PRO x* ≡ *Inl* (*Inr x*)›
abbreviation ‹*FLS* ≡ *Inr* (*FlsM*, *0*)›
abbreviation ‹*IMP n* ≡ *Inr* (*FlsM*, *n*)›
abbreviation ‹*DIA* ≡ *Inr* (*DiaM*, *0*)›
abbreviation ‹*SAT* ≡ *Inr* (*SatM*, *0*)›

primrec *encode* :: ‹('*i*, '*p*) *fm* ⇒ ('*i*, '*p*) *enc list*› **where**
  ‹*encode* ⊥ = [*FLS*]›
| ‹*encode* (*Pro x*) = [*PRO x*]›
| ‹*encode* (*Nom i*) = [*NOM i*]›
| ‹*encode* (*p* ⟶ *q*) = *IMP* (*length* (*encode p*)) # *encode p* @ *encode q*›
| ‹*encode* (◇ *p*) = *DIA* # *encode p*›
| ‹*encode* (@ *i p*) = *SAT* # *NOM i* # *encode p*›

lemma *encode-ne* [*simp*]: ‹*encode p* ≠ []›
  by (*induct p*) *auto*

lemma *inj-encode'*: ‹*encode p* = *encode q* ⟹ *p* = *q*›
proof (*induct p arbitrary*: *q*)
  case *Fls*
  then show *?case*
    by (*cases q*) *auto*
next
  case (*Pro x*)
  then show *?case*
    by (*cases q*) *auto*
next
  case (*Nom x*)
  then show *?case*
    by (*cases q*) *auto*
next
  case (*Imp p1 p2*)
  then show *?case*
    by (*cases q*) *auto*
next
  case (*Dia p*)
  then show *?case*
    by (*cases q*) *auto*
next
  case (*Sat x1a p*)
  then show *?case*
    by (*cases q*) *auto*
qed

lemma *inj-encode*: ‹*inj encode*›
  unfolding *inj-def* **using** *inj-encode'* **by** *blast*

primrec *encode-lbd* :: ‹('*i*, '*p*) *lbd* ⇒ ('*i*, '*p*) *enc list*› **where**
‹*encode-lbd* (*i*, *p*) = *NOM i* # *encode p*›

lemma *inj-encode-lbd'*: ‹*encode-lbd* (*i*, *p*) = *encode-lbd* (*k*, *q*) ⟹ *i* = *k* ∧ *p* = *q*›
using *inj-encode'* **by** *auto*

lemma *inj-encode-lbd*: ‹*inj encode-lbd*›
unfolding *inj-def* **using** *inj-encode-lbd'* **by** *auto*

lemma *finite-marker*: ‹*finite* (*UNIV* :: *marker set*)›
proof −
have ‹*p* ∈ {*FlsM*, *ImpM*, *DiaM*, *SatM*}› **for** *p*
by (*cases p*) *auto*
then show *?thesis*
by (*meson ex-new-if-finite finite.emptyI finite-insert*)
qed

lemma *card-of-lbd*:
assumes ‹*infinite* (*UNIV* :: '*i set*)›
shows ‹|*UNIV* :: ('*i*, '*p*) *lbd set*| ≤*o* |*UNIV* :: '*i set*| +*c* |*UNIV* :: '*p set*|›
proof −
have ‹|*UNIV* :: *marker set*| ≤*o* |*UNIV* :: *nat set*|›
using *finite-marker* **by** (*simp add*: *ordLess-imp-ordLeq*)
moreover have ‹*infinite* (*UNIV* :: ('*i* + '*p*) *set*)›
using *assms* **by** *simp*
ultimately have ‹|*UNIV* :: ('*i*, '*p*) *enc list set*| ≤*o* |*UNIV* :: ('*i* + '*p*) *set*|›
using *card-of-info-marker-lists* **by** *blast*
moreover have ‹|*UNIV* :: ('*i*, '*p*) *lbd set*| ≤*o* |*UNIV* :: ('*i*, '*p*) *enc list set*|›
using *card-of-ordLeq inj-encode-lbd* **by** *blast*
ultimately have ‹|*UNIV* :: ('*i*, '*p*) *lbd set*| ≤*o* |*UNIV* :: ('*i* + '*p*) *set*|›
using *ordLeq-transitive* **by** *blast*
then show *?thesis*
unfolding *csum-def* **by** *simp*
qed

7.10 Completeness

theorem *strong-completeness*:
fixes *p* :: ‹('*i*, '*p*) *fm*›
assumes ‹∀ *M* :: ('*i*, '*p*) *model*. ∀ *g*. (∀ (*k*, *q*) ∈ *X*. (*M*, *g*, *g k*) ⊨ *q*) ⟶ (*M*, *g*, *g i*) ⊨ *p*›
‹*infinite* (*UNIV* :: '*i set*)›
‹|*UNIV* :: '*i set*| +*c* |*UNIV* :: '*p set*| ≤*o* |*UNIV* − *nominals X*|›
shows ‹∃ *A*. *set A* ⊆ *X* ∧ *A* ⊢@ (*i*, *p*)›
proof (*rule ccontr*)
assume ‹∄ *A*. *set A* ⊆ *X* ∧ *A* ⊢@ (*i*, *p*)›
then have ∗: ‹∄ *A*. ∃ *a*. *set A* ⊆ *X* ∧ ((*i*, ¬ *p*) # *A* ⊢@ (*a*, ⊥))›
using *Boole FlsE* **by** *metis*

let *?S* = ‹{(*i*, ¬ *p*)} ∪ *X*›

```
  let ?H = ‹Extend ?S›

  have ‹consistent ?S›
    unfolding consistent-def using ∗ derive-split1 by metis
  moreover have ‹infinite (UNIV − nominals X)›
    using assms(2−3)
    by (metis Cinfinite-csum Cnotzero-UNIV Field-card-of cinfinite-def cinfinite-mono)
  then have ‹|UNIV :: 'i set| +c |UNIV :: 'p set| ≤o |UNIV − nominals X − nominals-lbd (i, ¬ p)|›
    using assms(3) finite-nominals-lbd card-of-infinite-diff-finite
    by (metis ordIso-iff-ordLeq ordLeq-transitive)
  then have ‹|UNIV :: 'i set| +c |UNIV :: 'p set| ≤o |UNIV − (nominals X ∪ nominals-lbd (i, ¬ p))|›
    by (metis Set-Diff-Un)
  then have ‹|UNIV :: 'i set| +c |UNIV :: 'p set| ≤o |UNIV − nominals ?S|›
    by (metis UN-insert insert-is-Un sup-commute)
  then have ‹|UNIV :: ('i, 'p) lbd set| ≤o |UNIV − nominals ?S|›
    using assms card-of-lbd ordLeq-transitive by blast
  ultimately have ‹consistent ?H› ‹maximal ?H› ‹saturated ?H›
    using MCS-Extend by fast+
  then have ‹canonical ?H i ⊨ p ⟷ (i, p) ∈ ?H› for i p
    using Truth-lemma by fast
  then have ‹(i, p) ∈ ?S ⟹ canonical ?H i ⊨ p› for i p
    using Extend-subset by blast
  then have ‹canonical ?H i ⊨ ¬ p› ‹∀ (k, q) ∈ X. canonical ?H k ⊨ q›
    by blast+
  moreover from this have ‹canonical ?H i ⊨ p›
    using assms(1) by blast
  ultimately show False
    by simp
qed

abbreviation valid :: ‹('i, 'p) fm ⇒ bool› where
  ‹valid p ≡ ∀ (M :: ('i, 'p) model) g i. (M, g, g i) ⊨ p›

theorem completeness:
  fixes p :: ‹('i, 'p) fm›
  assumes ‹valid p› ‹infinite (UNIV :: 'i set)› ‹|UNIV :: 'p set| ≤o |UNIV :: 'i set|›
  shows ‹[] ⊢@ (i, p)›
proof −
  have ‹|UNIV :: 'i set| +c |UNIV :: 'p set| ≤o |UNIV :: 'i set|›
    using assms(2−3) by (simp add: cinfinite-def csum-absorb1 ordIso-imp-ordLeq)
  then show ?thesis
    using assms strong-completeness[where X=‹{}› and p=p] infinite-UNIV-listI by auto
qed

corollary completeness':
  fixes p :: ‹('i, 'i) fm›
  assumes ‹valid p› ‹infinite (UNIV :: 'i set)›
  shows ‹[] ⊢@ (i, p)›
  using assms completeness[of p] by simp
```

theorem *main*:
fixes $p :: \langle('i, 'p)\ fm\rangle$
assumes $\langle i \notin$ *nominals-fm* $p\rangle$ $\langle$*infinite* $(UNIV :: 'i\ set)\rangle$ $\langle|UNIV :: 'p\ set| \leq o\ |UNIV :: 'i\ set|\rangle$
shows $\langle$*valid* $p \longleftrightarrow [] \vdash_@ (i, p)\rangle$
using *assms completeness soundness'* **by** *metis*

corollary *main'*:
fixes $p :: \langle('i, 'i)\ fm\rangle$
assumes $\langle i \notin$ *nominals-fm* $p\rangle$ $\langle$*infinite* $(UNIV :: 'i\ set)\rangle$
shows $\langle$*valid* $p \longleftrightarrow [] \vdash_@ (i, p)\rangle$
using *assms completeness' soundness'* **by** *metis*

end

Chapter 8

Example: First-Order Logic

theory *Example-First-Order-Logic* **imports** *Derivations* **begin**

8.1 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* (‹¬ -› [*70*] *70*) **where** ‹¬ *p* ≡ *p* ⟶ ⊥›

8.2 Semantics

type-synonym (*'a*, *'f*, *'p*) *model* = ‹(*nat* ⇒ *'a*) × (*'f* ⇒ *'a list* ⇒ *'a*) × (*'p* ⇒ *'a list* ⇒ *bool*)›

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*› (‹⟦-⟧›) **where**
  ‹⟦-⟧ ⊥ = *False*›
| ‹⟦(*E*, *F*, *G*)⟧ (‡*P ts*) = *G P* (*map* ⟦(*E*, *F*)⟧ *ts*)›

| ‹⟦(E, F, G)⟧ (p ⟶ q) = (⟦(E, F, G)⟧ p ⟶ ⟦(E, F, G)⟧ q)›
| ‹⟦(E, F, G)⟧ (∀ p) = (∀ x. ⟦(x ⨟ E, F, G)⟧ p)›

8.3 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 ⨟ λn. *lift-tm* (*s n*)) *p*)›

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

abbreviation ‹*params'* S ≡ ⋃p ∈ S. *params-fm p*›

abbreviation ‹*params l* ≡ *params'* (*set l*)›

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*]: ‹*f* ∉ *params-fm p* ⟹ ⟦(E, F(f := x), G)⟧ *p* = ⟦(E, F, G)⟧ *p*›
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* ⨟ λn. *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*) = ⦇(λn. ⦇(E, F)⦈ (*s n*), F)⦈ *t*›
by (*induct t*) (*auto cong*: *map-cong*)

lemma *sub-fm-semantics* [*simp*]: ‹⟦(E, F, G)⟧ (*sub-fm s p*) = ⟦(λn. ⦇(E, F)⦈ (*s n*), F, G)⟧ *p*›
by (*induct p arbitrary*: *E s*) (*auto cong*: *map-cong simp*: *sub-tm-semantics lift-lemma*)

lemma *sub-tm-Var*: ‹*sub-tm* # *t* = *t*›
by (*induct t*) (*auto cong*: *map-cong*)

lemma *reduce-Var* [*simp*]: ‹(# *0* ⨟ λ*n*. # (*Suc n*)) = #›
proof (*rule ext*)
fix *n*
show ‹(# *0* ⨟ λ*n*. # (*Suc n*)) *n* = #*n*›
by (*induct n*) *simp-all*
qed

lemma *sub-fm-Var* [*simp*]:
fixes *p* :: ‹('*f*, '*p*) *fm*›
shows ‹*sub-fm* # *p* = *p*›
proof (*induct p*)
case (*Pre P ts*)
then show *?case*
by (*auto cong*: *map-cong simp*: *sub-tm-Var*)
qed *simp-all*

lemma *semantics-tm-id* [*simp*]: ‹⦇(#, †)⦈ *t* = *t*›
by (*induct t*) (*auto cong*: *map-cong*)

lemma *semantics-tm-id-map* [*simp*]: ‹*map* ⦇(#, †)⦈ *ts* = *ts*›
by (*auto cong*: *map-cong*)

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

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*›

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

8.4 Calculus

inductive *Calculus* :: ‹('*f*, '*p*) *fm list* ⇒ ('*f*, '*p*) *fm* ⇒ *bool*› (‹- ⊢∀ -› [*50*, *50*] *50*) **where**
Assm [*intro*]: ‹*p* ∈ *set A* ⟹ *A* ⊢∀ *p*›
| *FlsE* [*elim*]: ‹*A* ⊢∀ ⊥ ⟹ *A* ⊢∀ *p*›
| *ImpI* [*intro*]: ‹*p* # *A* ⊢∀ *q* ⟹ *A* ⊢∀ *p* ⟶ *q*›
| *ImpE* [*elim*]: ‹*A* ⊢∀ *p* ⟶ *q* ⟹ *A* ⊢∀ *p* ⟹ *A* ⊢∀ *q*›
| *UniI* [*intro*]: ‹*A* ⊢∀ ⟨⋆*a*⟩*p* ⟹ *a* ∉ *params* (*p* # *A*) ⟹ *A* ⊢∀ ∀ *p*›
| *UniE* [*elim*]: ‹*A* ⊢∀ ∀ *p* ⟹ *A* ⊢∀ ⟨*t*⟩*p*›
| *Clas*: ‹(*p* ⟶ *q*) # *A* ⊢∀ *p* ⟹ *A* ⊢∀ *p*›
| *Weak*: ‹*A* ⊢∀ *p* ⟹ *q* # *A* ⊢∀ *p*›

8.5 Soundness

theorem *soundness*: ‹$A \vdash_\forall p \Longrightarrow$ *list-all* $[\![(E, F, G)]\!]\ A \Longrightarrow [\![(E, F, G)]\!]\ p$›
proof (*induct p arbitrary*: *F rule*: *Calculus.induct*)
  case (*UniI A a p*)
  moreover have ‹*list-all* $[\![(E, F(a := x), G)]\!]\ A$› **for** *x*
    using *UniI*(*3*−*4*) **by** (*simp add*: *list.pred-mono-strong*)
  then have ‹$[\![(E, F(a := x), G)]\!]$ $(\langle\star a\rangle p)$› **for** *x*
    using *UniI* **by** *blast*
  ultimately show *?case*
    by *fastforce*
qed (*auto simp*: *list-all-iff*)

corollary *soundness′*: ‹$[] \vdash_\forall p \Longrightarrow [\![M]\!]\ p$›
  using *soundness* **by** (*cases M*) *fastforce*

corollary ‹$\neg$ $([] \vdash_\forall \bot)$›
  using *soundness′* **by** *fastforce*

8.6 Derived Rules

lemma *Assm-head*: ‹$p \# A \vdash_\forall p$›
  by *auto*

lemma *deduct1*: ‹$A \vdash_\forall p \longrightarrow q \Longrightarrow p \# A \vdash_\forall q$›
  by (*meson ImpE Weak Assm-head*)

lemma *Boole*: ‹$(\neg p) \# A \vdash_\forall \bot \Longrightarrow A \vdash_\forall p$›
  using *Clas FlsE* **by** *blast*

lemma *Weak′*: ‹$A \vdash_\forall p \Longrightarrow B @ A \vdash_\forall p$›
  by (*induct B*) (*simp*, *metis Weak append-Cons*)

lemma *Weaken*: ‹$A \vdash_\forall p \Longrightarrow$ *set* $A \subseteq$ *set* $B \Longrightarrow B \vdash_\forall p$›
proof (*induct A arbitrary*: *p*)
  case *Nil*
  then show *?case*
    using *Weak′* **by** *fastforce*
next
  case (*Cons q A*)
  then show *?case*
    by (*metis Assm ImpE ImpI list.set-intros*(*1*−*2*) *subset-code*(*1*))
qed

interpretation *Derivations Calculus*
proof
  fix *A B* **and** *p* :: ‹('*f*, '*p*) *fm*›
  assume ‹$A \vdash_\forall p$› ‹*set* $A \subseteq$ *set* B›
  then show ‹$B \vdash_\forall p$›

using *Weaken* **by** *blast*
qed

8.7 Maximal Consistent Sets

definition *consistent* :: ‹('f, 'p) *fm set* ⇒ *bool*› **where**
‹*consistent S* ≡ ∄*S'*. *set S'* ⊆ *S* ∧ *S'* ⊢∀ ⊥›

fun *witness* :: ‹('f, 'p) *fm* ⇒ ('f, 'p) *fm set* ⇒ ('f, 'p) *fm set*› **where**
‹*witness* (¬ (∀ *p*)) *S* = {¬ ⟨⋆(*SOME a*. *a* ∉ *params'* ({*p*} ∪ *S*))⟩*p*}›
| ‹*witness* - - = {}›

lemma *consistent-add-instance*:
assumes ‹*consistent S*› ‹∀ *p* ∈ *S*›
shows ‹*consistent* ({⟨*t*⟩*p*} ∪ *S*)›
unfolding *consistent-def*
proof
assume ‹∃*S'*. *set S'* ⊆ {⟨*t*⟩*p*} ∪ *S* ∧ *S'* ⊢∀ ⊥›
then obtain *S'* **where** ‹*set S'* ⊆ *S*› ‹⟨*t*⟩*p* # *S'* ⊢∀ ⊥›
using *assms derive-split1* **by** *metis*
then have ‹∀ *p* # *S'* ⊢∀ ¬ ⟨*t*⟩*p*›
using *Weak* **by** *blast*
moreover have ‹∀ *p* # *S'* ⊢∀ ⟨*t*⟩*p*›
using *Assm-head* **by** *fast*
ultimately have ‹∀ *p* # *S'* ⊢∀ ⊥›
by *fast*
moreover have ‹*set* ((∀ *p*) # *S'*) ⊆ *S*›
using ‹*set S'* ⊆ *S*› *assms*(*2*) **by** *simp*
ultimately show *False*
using *assms*(*1*) **unfolding** *consistent-def* **by** *blast*
qed

lemma *consistent-add-witness*:
assumes ‹*consistent S*› ‹¬ (∀ *p*) ∈ *S*› ‹*a* ∉ *params' S*›
shows ‹*consistent* ({¬ ⟨⋆*a*⟩*p*} ∪ *S*)›
unfolding *consistent-def*
proof
assume ‹∃*S'*. *set S'* ⊆ {¬ ⟨⋆*a*⟩*p*} ∪ *S* ∧ *S'* ⊢∀ ⊥›
then obtain *S'* **where** ‹*set S'* ⊆ *S*› ‹(¬ ⟨⋆*a*⟩*p*) # *S'* ⊢∀ ⊥›
using *assms derive-split1* **by** *metis*
then have ‹*S'* ⊢∀ ⟨⋆*a*⟩*p*›
using *Boole* **by** *blast*
moreover have ‹*a* ∉ *params-fm p*› ‹∀ *p* ∈ *set S'*. *a* ∉ *params-fm p*›
using ‹*set S'* ⊆ *S*› *assms*(*2*−*3*) **by** *auto*
then have ‹*a* ∉ *params'* ({*p*} ∪ *set S'*)›
using *calculation* **by** *fast*
ultimately have ‹*S'* ⊢∀ ∀ *p*›
by *fastforce*
then have ‹¬ (∀ *p*) # *S'* ⊢∀ ⊥›

```
    using Weak Assm-head by fast
  moreover have ‹set ((¬ (∀ p)) # S') ⊆ S›
    using ‹set S' ⊆ S› assms(2) by simp
  ultimately show False
    using assms(1) unfolding consistent-def by blast
qed

lemma consistent-witness':
  assumes ‹consistent ({p} ∪ S)› ‹infinite (UNIV − params' S)›
  shows ‹consistent (witness p S ∪ {p} ∪ S)›
  using assms
proof (induct p S rule: witness.induct)
  case (1 p S)
  have ‹infinite (UNIV − params' ({p} ∪ S))›
    using 1(2) finite-params-fm by (simp add: infinite-Diff-fin-Un)
  then have ‹∃ a. a ∉ params' ({p} ∪ S)›
    by (simp add: not-finite-existsD set-diff-eq)
  then have ‹(SOME a. a ∉ params' ({p} ∪ S)) ∉ params' ({p} ∪ S)›
    by (rule someI-ex)
  then obtain a where a: ‹witness (¬ (∀ p)) S = {¬ ⟨⋆a⟩p}› ‹a ∉ params' ({¬∀ p} ∪ S)›
    by simp
  then show ?case
    using 1(1−2) a(1) consistent-add-witness[where S=‹{¬∀ p} ∪ S›] by fastforce
qed (auto simp: assms)

interpretation MCS-Saturation consistent params-fm witness
proof
  fix S S' :: ‹('f, 'p) fm set›
  assume ‹consistent S› ‹S' ⊆ S›
  then show ‹consistent S'›
    unfolding consistent-def by fast
next
  fix S :: ‹('f, 'p) fm set›
  assume ‹¬ consistent S›
  then show ‹∃ S'⊆S. finite S' ∧ ¬ consistent S'›
    unfolding consistent-def by blast
next
  fix p :: ‹('f, 'p) fm›
  show ‹finite (params-fm p)›
    by simp
next
  fix p and S :: ‹('f, 'p) fm set›
  show ‹finite (params' (witness p S))›
    by (induct p S rule: witness.induct) simp-all
next
  fix p and S :: ‹('f, 'p) fm set›
  assume ‹consistent ({p} ∪ S)› ‹infinite (UNIV − params' S)›
  then show ‹consistent (witness p S ∪ {p} ∪ S)›
    using consistent-witness' by fast
```

next
show ‹*infinite* (*UNIV* :: ('*f*, '*p*) *fm set*)›
using *infinite-UNIV-size*[*of* ‹λ*p*. *p* ⟶ *p*›] **by** *simp*
qed

interpretation *Derivations-MCS-Cut Calculus consistent* ‹⊥›
proof
fix *S* :: ‹('*f*, '*p*) *fm set*›
show ‹*consistent S* = (∄*S'*. *set S'* ⊆ *S* ∧ *S'* ⊢∀ ⊥)›
unfolding *consistent-def* **..**
next
fix *A* **and** *p* :: ‹('*f*, '*p*) *fm*›
assume ‹*p* ∈ *set A*›
then show ‹*A* ⊢∀ *p*›
by *blast*
next
fix *A B* **and** *p q* :: ‹('*f*, '*p*) *fm*›
assume ‹*A* ⊢∀ *p*› ‹*p* # *B* ⊢∀ *q*›
then show ‹*A* @ *B* ⊢∀ *q*›
using *Weaken ImpI ImpE* **by** (*metis Un-upper2 inf-sup-ord*(*3*) *set-append*)
qed

8.8 Truth Lemma

abbreviation *hmodel* :: ‹('*f*, '*p*) *fm set* ⇒ ('*f tm*, '*f*, '*p*) *model*› **where**
‹*hmodel H* ≡ (#, †, λ*P ts*. ‡*P ts* ∈ *H*)›

fun *interp* ::
‹('*a*, '*f*, '*p*) *model* ⇒ (('*a*, '*f*, '*p*) *model* ⇒ ('*f*, '*p*) *fm* ⇒ *bool*) ⇒ ('*f*, '*p*) *fm* ⇒ *bool*› **where**
‹*interp* - - ⊥ = *False*›
| ‹*interp* (*E*, *F*, *G*) *X* (‡*P ts*) = *G P* (*map* (⦇(*E*, *F*)⦈) *ts*)›
| ‹*interp* (*E*, *F*, *G*) *X* (*p* ⟶ *q*) = (*X* (*E*, *F*, *G*) *p* ⟶ *X* (*E*, *F*, *G*) *q*)›
| ‹*interp* (*E*, *F*, *G*) *X* (∀ *p*) = (∀ *x*. *X* (*x* ⨟ *E*, *F*, *G*) *p*)›

fun *rel* :: ‹('*f*, '*p*) *fm set* ⇒ ('*f tm*, '*f*, '*p*) *model* ⇒ ('*f*, '*p*) *fm* ⇒ *bool*› **where**
‹*rel H* (*E*, -, -) *p* = (*sub-fm E p* ∈ *H*)›

theorem *Hintikka-model'*:
assumes ‹⋀*p*. *interp* (*hmodel H*) (*rel H*) *p* ⟷ *p* ∈ *H*›
shows ‹*p* ∈ *H* ⟷ ⟦*hmodel H*⟧ *p*›
proof (*induct p rule*: *wf-induct*[**where** *r*=‹*measure size-fm*›])
case *1*
then show *?case* **..**
next
case (*2 x*)
then show *?case*
using *assms*[*of x*] **by** (*cases x*) *simp-all*
qed

lemma *Hintikka-Extend*:
assumes ‹*consistent H*› ‹*maximal H*› ‹*saturated H*›
shows ‹*interp* (*hmodel H*) (*rel H*) *p* ⟷ *p* ∈ *H*›
proof (*cases p*)
case *Fls*
have ‹⊥ ∉ *H*›
using *assms MCS-derive* **unfolding** *consistent-def* **by** *blast*
then show *?thesis*
using *Fls* **by** *simp*
next
case (*Imp p q*)
have ‹*p* # *A* ⊢∀ ⊥ ⟹ *A* ⊢∀ *p* ⟶ *q*› ‹*A* ⊢∀ *q* ⟹ *A* ⊢∀ *p* ⟶ *q*› **for** *A*
by (*auto simp*: *Weak*)
moreover have ‹*A* ⊢∀ *p* ⟶ *q* ⟹ *p* # *A* ⊢∀ *q*› **for** *A*
using *deduct1* **.**
ultimately have ‹(*p* ∈ *H* ⟶ *q* ∈ *H*) ⟷ *p* ⟶ *q* ∈ *H*›
using *assms*(*1*−*2*) *MCS-derive MCS-derive-fls* **by** (*metis insert-subset list.simps*(*15*))
then show *?thesis*
using *Imp* **by** *simp*
next
case (*Uni p*)
have ‹(∀ *x*. ⟨*x*⟩*p* ∈ *H*) ⟷ (∀ *p* ∈ *H*)›
proof
assume ‹∀ *x*. ⟨*x*⟩*p* ∈ *H*›
show ‹∀ *p* ∈ *H*›
proof (*rule ccontr*)
assume ‹∀ *p* ∉ *H*›
then have ‹*consistent* ({¬ ∀ *p*} ∪ *H*)›
using *Boole assms*(*1*−*2*) *MCS-derive derive-split1* **by** (*metis consistent-derive-fls*)
then have ‹¬ ∀ *p* ∈ *H*›
using *assms*(*2*) **unfolding** *maximal-def* **by** *blast*
then obtain *a* **where** ‹¬ ⟨⋆*a*⟩*p* ∈ *H*›
using *assms*(*3*) **unfolding** *saturated-def* **by** *fastforce*
moreover have ‹⟨⋆*a*⟩*p* ∈ *H*›
using ‹∀ *x*. ⟨*x*⟩*p* ∈ *H*› **by** *blast*
ultimately show *False*
using *assms*(*1*−*2*) *MCS-derive* **by** (*metis consistent-def deduct1 insert-subset list.simps*(*15*))
qed
next
assume ‹∀ *p* ∈ *H*›
then show ‹∀ *x*. ⟨*x*⟩*p* ∈ *H*›
using *assms*(*1*−*2*) *consistent-add-instance maximal-def* **by** *blast*
qed
then show *?thesis*
using *Uni* **by** *simp*
qed *simp*

interpretation *Truth-Saturation*
consistent params-fm witness interp semantics-fm ‹λ*H*. {*hmodel H*}› *rel*

proof *unfold-locales*
fix *p* **and** *M* :: ‹('*a tm*, '*f*, '*p*) *model*›
show ‹⟦*M*⟧ *p* ⟷ *interp M semantics-fm p*›
by (*cases M*, *induct p*) *auto*
next
fix *p* **and** *H* :: ‹('*f*, '*p*) *fm set*› **and** *M* :: ‹('*f tm*, '*f*, '*p*) *model*›
assume ‹∀ *M* ∈ {*hmodel H*}. ∀ *p*. *interp M* (*rel H*) *p* ⟷ *rel H M p*› ‹*M* ∈ {*hmodel H*}›
then show ‹⟦*M*⟧ *p* ⟷ *rel H M p*›
using *Hintikka-model'* **by** *auto*
next
fix *H* :: ‹('*f*, '*p*) *fm set*›
assume ‹*consistent H*› ‹*maximal H*› ‹*saturated H*›
then show ‹∀ *M* ∈ {*hmodel H*}. ∀ *p*. *interp M* (*rel H*) *p* ⟷ *rel H M p*›
using *Hintikka-Extend* **by** *auto*
qed

8.9 Cardinalities

datatype *marker* = *VarM* | *FunM* | *TmM* | *FlsM* | *PreM* | *ImpM* | *UniM*

type-synonym ('*f*, '*p*) *enc* = ‹('*f* + '*p*) + *marker* × *nat*›

abbreviation ‹*FUNS f* ≡ *Inl* (*Inl f*)›
abbreviation ‹*PRES p* ≡ *Inl* (*Inr p*)›

abbreviation ‹*VAR n* ≡ *Inr* (*VarM*, *n*)›
abbreviation ‹*FUN n* ≡ *Inr* (*FunM*, *n*)›
abbreviation ‹*TM n* ≡ *Inr* (*TmM*, *n*)›

abbreviation ‹*PRE n* ≡ *Inr* (*PreM*, *n*)›
abbreviation ‹*FLS* ≡ *Inr* (*FlsM*, *0*)›
abbreviation ‹*IMP n* ≡ *Inr* (*FlsM*, *n*)›
abbreviation ‹*UNI* ≡ *Inr* (*UniM*, *0*)›

primrec
encode-tm :: ‹'*f tm* ⇒ ('*f*, '*p*) *enc list*› **and**
encode-tms :: ‹'*f tm list* ⇒ ('*f*, '*p*) *enc list*› **where**
‹*encode-tm* (#*n*) = [*VAR n*]›
| ‹*encode-tm* (†*f ts*) = *FUN* (*length ts*) # *FUNS f* # *encode-tms ts*›
| ‹*encode-tms* [] = []›
| ‹*encode-tms* (*t* # *ts*) = *TM* (*length* (*encode-tm t*)) # *encode-tm t* @ *encode-tms ts*›

lemma *encode-tm-ne* [*simp*]: ‹*encode-tm t* ≠ []›
by (*induct t*) *auto*

lemma *inj-encode-tm'*:
‹(*encode-tm t* :: ('*f*, '*p*) *enc list*) = *encode-tm s* ⟹ *t* = *s*›
‹(*encode-tms ts* :: ('*f*, '*p*) *enc list*) = *encode-tms ss* ⟹ *ts* = *ss*›
proof (*induct t* **and** *ts arbitrary*: *s* **and** *ss rule*: *encode-tm.induct encode-tms.induct*)

```
  case (Var n)
  then show ?case
    by (cases s) auto
next
  case (Fun f fts)
  then show ?case
    by (cases s) auto
next
  case Nil-tm
  then show ?case
    by (cases ss) auto
next
  case (Cons-tm t ts)
  then show ?case
    by (cases ss) auto
qed

lemma inj-encode-tm: ‹inj encode-tm›
  unfolding inj-def using inj-encode-tm' by blast

primrec encode-fm :: ‹('f, 'p) fm ⇒ ('f, 'p) enc list› where
  ‹encode-fm ⊥ = [FLS]›
| ‹encode-fm (‡P ts) = PRE (length ts) # PRES P # encode-tms ts›
| ‹encode-fm (p ⟶ q) = IMP (length (encode-fm p)) # encode-fm p @ encode-fm q›
| ‹encode-fm (∀ p) = UNI # encode-fm p›

lemma encode-fm-ne [simp]: ‹encode-fm p ≠ []›
  by (induct p) auto

lemma inj-encode-fm': ‹encode-fm p = encode-fm q ⟹ p = q›
proof (induct p arbitrary: q)
  case Fls
  then show ?case
    by (cases q) auto
next
  case (Pre P ts)
  then show ?case
    by (cases q) (auto simp: inj-encode-tm')
next
  case (Imp p1 p2)
  then show ?case
    by (cases q) auto
next
  case (Uni p)
  then show ?case
    by (cases q) auto
qed

lemma inj-encode-fm: ‹inj encode-fm›
```

unfolding *inj-def* **using** *inj-encode-fm'* **by** *blast*

lemma *finite-marker*: ‹*finite* (*UNIV* :: *marker set*)›
proof −
have ‹$p \in \{VarM, FunM, TmM, FlsM, PreM, ImpM, UniM\}$› **for** *p*
by (*cases p*) *auto*
then show *?thesis*
by (*meson ex-new-if-finite finite.emptyI finite-insert*)
qed

lemma *card-of-fm*:
assumes ‹*infinite* (*UNIV* :: *'f set*)›
shows ‹$|UNIV :: ('f, 'p)\ fm\ set| \leq_o |UNIV :: 'f\ set| +_c |UNIV :: 'p\ set|$›
proof −
have ‹$|UNIV :: marker\ set| \leq_o |UNIV :: nat\ set|$›
using *finite-marker* **by** (*simp add*: *ordLess-imp-ordLeq*)
moreover have ‹*infinite* (*UNIV* :: (*'f* + *'p*) *set*)›
using *assms* **by** *simp*
ultimately have ‹$|UNIV :: ('f, 'p)\ enc\ list\ set| \leq_o |UNIV :: ('f + 'p)\ set|$›
using *card-of-info-marker-lists* **by** *blast*
moreover have ‹$|UNIV :: ('f, 'p)\ fm\ set| \leq_o |UNIV :: ('f, 'p)\ enc\ list\ set|$›
using *card-of-ordLeq inj-encode-fm* **by** *blast*
ultimately have ‹$|UNIV :: ('f, 'p)\ fm\ set| \leq_o |UNIV :: ('f + 'p)\ set|$›
using *ordLeq-transitive* **by** *blast*
then show *?thesis*
unfolding *csum-def* **by** *simp*
qed

8.10 Completeness

theorem *strong-completeness*:
assumes ‹$\forall M :: ('f\ tm, 'f, 'p)\ model.\ (\forall q \in X.\ [\![M]\!]\ q) \longrightarrow [\![M]\!]\ p$›
‹*infinite* (*UNIV* :: *'f set*)›
‹$|UNIV :: 'f\ set| +_c |UNIV :: 'p\ set| \leq_o |UNIV - params'\ X|$›
shows ‹$\exists A.\ set\ A \subseteq X \wedge A \vdash_\forall p$›
proof (*rule ccontr*)
assume ‹$\nexists A.\ set\ A \subseteq X \wedge A \vdash_\forall p$›
then have ∗: ‹$\nexists A.\ set\ A \subseteq X \wedge \neg\ p\ \#\ A \vdash_\forall \bot$›
using *Boole* **by** *blast*

let *?S* = ‹$\{\neg\ p\} \cup X$›
let *?H* = ‹*Extend ?S*›

have ‹*consistent ?S*›
using ∗ **by** (*meson consistent-def derive-split1*)
moreover have ‹*infinite* (*UNIV* − *params'* *X*)›
using *assms*(*2*−*3*)
by (*metis Cinfinite-csum Cnotzero-UNIV Field-card-of cinfinite-def cinfinite-mono*)
then have ‹$|UNIV :: 'f\ set| +_c |UNIV :: 'p\ set| \leq_o |UNIV - params'\ X - params\text{-}fm\ (\neg\ p)|$›

```
    using assms(3) finite-params-fm card-of-infinite-diff-finite
    by (metis ordIso-iff-ordLeq ordLeq-transitive)
  then have ‹|UNIV :: 'f set| +c |UNIV :: 'p set| ≤o |UNIV − (params' X ∪ params-fm (¬ p))|›
    by (metis Set-Diff-Un)
  then have ‹|UNIV :: 'f set| +c |UNIV :: 'p set| ≤o |UNIV − params' ?S|›
    by (metis UN-insert insert-is-Un sup-commute)
  then have ‹|UNIV :: ('f, 'p) fm set| ≤o |UNIV − params' ?S|›
    using assms card-of-fm ordLeq-transitive by blast
  ultimately have ‹consistent ?H› ‹maximal ?H› ‹saturated ?H›
    using MCS-Extend by fast+
  then have ‹p ∈ ?H ⟷ ⟦hmodel ?H⟧ p› for p
    using truth-lemma-saturation by fastforce
  then have ‹p ∈ ?S ⟶ ⟦hmodel ?H⟧ p› for p
    using Extend-subset by blast
  then have ‹⟦hmodel ?H⟧ (¬ p)› ‹∀ q ∈ X. ⟦hmodel ?H⟧ q›
    by blast+
  moreover from this have ‹⟦hmodel ?H⟧ p›
    using assms(1) by blast
  ultimately show False
    by simp
qed

abbreviation valid :: ‹('f, 'p) fm ⇒ bool› where
  ‹valid p ≡ ∀ M :: ('f tm, -, -) model. ⟦M⟧ p›

theorem completeness:
  fixes p :: ‹('f, 'p) fm›
  assumes ‹valid p› ‹infinite (UNIV :: 'f set)› ‹|UNIV :: 'p set| ≤o |UNIV :: 'f set|›
  shows ‹[] ⊢∀ p›
proof −
  have ‹|UNIV :: 'f set| +c |UNIV :: 'p set| ≤o |UNIV :: 'f set|›
    using assms(2−3) by (simp add: cinfinite-def csum-absorb1 ordIso-imp-ordLeq)
  then show ?thesis
    using assms strong-completeness[where X=‹{}›] infinite-UNIV-listI by auto
qed

corollary completeness':
  fixes p :: ‹('f, 'f) fm›
  assumes ‹valid p› ‹infinite (UNIV :: 'f set)›
  shows ‹[] ⊢∀ p›
  using assms completeness[of p] by simp

theorem main:
  fixes p :: ‹('f, 'p) fm›
  assumes ‹infinite (UNIV :: 'f set)› ‹|UNIV :: 'p set| ≤o |UNIV :: 'f set|›
  shows ‹valid p ⟷ [] ⊢∀ p›
  using assms completeness soundness' by blast

corollary main':
```

fixes $p :: \langle('f, 'f)\ fm\rangle$
assumes $\langle infinite\ (UNIV :: 'f\ set)\rangle$
shows $\langle valid\ p \longleftrightarrow [] \vdash_\forall\ p\rangle$
using *assms completeness' soundness'* **by** *blast*

end

Bibliography

[1] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001.

[2] J. C. Blanchette, A. Popescu, and D. Traytel. Cardinals in Isabelle/HOL. In G. Klein and R. Gamboa, editors, *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8558 of *Lecture Notes in Computer Science*, pages 111–127. Springer, 2014.

[3] T. Braüner. *Hybrid Logic and its Proof-Theory*. Springer Dordrecht, first edition, 2011.

[4] C. C. Chang and H. J. Keisler. *Model theory, Third Edition*, volume 73 of *Studies in logic and the foundations of mathematics*. North-Holland, 1992.

[5] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[6] R. M. Smullyan. *First-order logic*. Dover Publications, 1995.