Synthetic Completeness

Asta Halkjær From

March 17, 2025

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 witnessed 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 witnessed when it contains a witness for every existential formula. To prove completeness using these maximal consistent witnessed sets, we prove a truth lemma: every formula in an MCS has a satisfying model. Here, saturated 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 saturated set gives rise to a satisfying model and that MCSs are saturated 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 witnessed 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 saturated 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 (downwards saturated) sets based on the calculus, but the other four examples derive their notion of saturation from the semantics in the style of the framework. The hybrid and first-order logic examples rely on witnessed 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*$) \Rightarrow 'a$›
  assumes ‹$\bigwedge x.$ *size* $x <$ *size* $(f\, x)$›
  shows ‹*infinite* $(UNIV$ :: $'a$ *set*$)$›
proof
  assume ‹*finite* $(UNIV$ :: $'a$ *set*$)$›
  then obtain x :: $'a$ **where** ‹$\forall\, y$ :: $'a.$ *size* $y \leq$ *size* x›
    using *finite-bound* **by** *fastforce*
  moreover have ‹*size* $x <$ *size* $(f\, x)$›
    using *assms* **.**
  ultimately show *False*
    using *leD* **by** *blast*
qed

context *wo-rel* **begin**

lemma *underS-bound*: ‹$a \in$ *underS* $c \Longrightarrow b \in$ *underS* $c \Longrightarrow a \in$ *under* $b \vee b \in$ *under* a›
  by (*meson BNF-Least-Fixpoint.underS-Field REFL Refl-under-in in-mono under-ofilter ofilter-linord*)

lemma *finite-underS-bound*:
  assumes ‹*finite* X› ‹$X \subseteq$ *underS* c› ‹$X \neq \{\}$›

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

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

lemma *underS-trans*: ‹$a \in \mathit{underS}\ b \Longrightarrow b \in \mathit{underS}\ c \Longrightarrow a \in \mathit{underS}\ c$›
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-params-marker-lists*:
assumes ‹*infinite* (*UNIV* :: *'i set*)› ‹$|\mathit{UNIV} :: \mathit{'m\ set}| \leq o\ |\mathit{UNIV} :: \mathit{nat\ set}|$›
shows ‹$|\mathit{UNIV} :: (\mathit{'i} + \mathit{'m} \times \mathit{nat})\ \mathit{list\ set}| \leq o\ |\mathit{UNIV} :: \mathit{'i\ set}|$›
proof −
have ‹$(\mathit{UNIV} :: \mathit{'m\ set}) \neq \{\}$›
by *simp*
then have ‹$|\mathit{UNIV} :: \mathit{'m\ set}| *c |\mathit{UNIV} :: \mathit{nat\ set}| \leq o\ |\mathit{UNIV} :: \mathit{nat\ set}|$›
using *assms*(*2*) **by** (*simp add*: *cinfinite-def cprod-cinfinite-bound ordLess-imp-ordLeq*)
then have ‹$|\mathit{UNIV} :: (\mathit{'m} \times \mathit{nat})\ \mathit{set}| \leq o\ |\mathit{UNIV} :: \mathit{nat\ set}|$›
unfolding *cprod-def* **by** *simp*
moreover have ‹$|\mathit{UNIV} :: \mathit{nat\ set}| \leq o\ |\mathit{UNIV} :: \mathit{'i\ set}|$›

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 Locales

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

locale *MCS-Witness* = *MCS-Base consistent*
for *consistent* :: ‹$'a\ set \Rightarrow bool$› +
fixes *witness* :: ‹$'a \Rightarrow 'a\ set \Rightarrow 'a\ set$›
and *params* :: ‹$'a \Rightarrow 'i\ set$›
assumes *finite-params*: ‹$\bigwedge p.\ finite\ (params\ p)$›
and *finite-witness-params*: ‹$\bigwedge p\ S.\ finite\ (\bigcup q \in witness\ p\ S.\ params\ q)$›
and *consistent-witness*: ‹$\bigwedge p\ S.\ consistent\ (\{p\} \cup S)$
$\Longrightarrow infinite\ (UNIV - (\bigcup q \in S.\ params\ q))$
$\Longrightarrow consistent\ (\{p\} \cup S \cup witness\ p\ S)$›
begin

definition *witnessed* :: ‹$'a\ set \Rightarrow bool$› **where**
‹$witnessed\ S \equiv \forall\, p \in S.\ \exists\, S'.\ witness\ p\ S' \subseteq S$›

abbreviation *MCS* :: ‹$'a\ set \Rightarrow bool$› **where**
‹$MCS\ S \equiv consistent\ S \wedge maximal\ S \wedge witnessed\ S$›

end

locale *MCS-No-Witness* = *MCS-Base consistent* **for** *consistent* :: ‹$'a\ set \Rightarrow bool$›

sublocale *MCS-No-Witness* ⊆ *MCS-Witness consistent* ‹λ- -. {}› ‹λ-. {}›
proof qed *simp-all*

1.3 Ordinal Locale

locale *MCS-Lim-Ord* = *MCS-Witness consistent witness params*
  for *consistent* :: ‹'*a set* ⇒ *bool*›
    and *witness* :: ‹'*a* ⇒ '*a set* ⇒ '*a set*›
    and *params* :: ‹'*a* ⇒ '*i set*› +
  fixes *r* :: ‹'*a rel*›
  assumes *Cinfinite-r*: ‹*Cinfinite r*›
begin

lemma *WELL*: ‹*Well-order r*›
  using *Cinfinite-r* **by** *simp*

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*

lemma *nonempty-Field-r*: ‹*Field r* ≠ {}›
  using *Cinfinite-r cinfinite-def infinite-imp-nonempty* **by** *blast*

1.3.1 Lindenbaum Extension

abbreviation *paramss* :: ‹'*a set* ⇒ '*i set*› **where**
  ‹*paramss S* ≡ ⋃ *p* ∈ *S*. *params p*›

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

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

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

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* ≡ ⋃ *a* ∈ *Field r*. *extend S a*›

lemma *extend-subset*: ‹*a* ∈ *Field r* ⟹ *S* ⊆ *extend S a*›
proof (*induct a 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*: ‹*S* ⊆ *Extend S*›
unfolding *Extend-def* **using** *extend-subset nonempty-Field-r* **by** *fast*

lemma *extend-underS*: ‹*b* ∈ *underS r a* ⟹ *extend S b* ⊆ *extend S a*›
proof (*induct a 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** ‹*b* = *i* ∨ *b* ∈ *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*: ‹*b* ∈ *under r a* ⟹ *extend S b* ⊆ *extend S a*›
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-empty*)

1.3.2 Consistency

lemma *params-origin*:
assumes ‹*x* ∈ *paramss* (*extend S a*)›
shows ‹*x* ∈ *paramss S* ∨ (∃ *b* ∈ *underS r a*. *x* ∈ *paramss* ({*b*} ∪ *witness b* (*extend S b*)))›
using *assms*
proof (*induct a 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*) ‹*x* ∈ *paramss* ({*i*} ∪ *witness i* (*extend S i*))› | (*there*) ‹*x* ∈ *paramss* (*extend S i*)›
using *wo-rel.worecZSL-succ*[*OF wo-rel-r adm-woL-extendL 2*(*1*)] *extendS-def extend-def*
by (*auto split*: *if-splits*)
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› ‹x ∈ paramss (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 ≤o |UNIV − paramss S|›
  shows ‹consistent (extend S a)›
  using assms(1)
proof (induct a 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 ?paramss = ‹λk. paramss ({k} ∪ witness k (extend S k))›
  let ?X = ‹⋃k ∈ underS r i. ?paramss k›
  have ‹|?X| <o r›
  proof (cases ‹finite (underS r i)›)
    case True
    then have ‹finite ?X›
      by (simp add: finite-params finite-witness-params)
    then show ?thesis
      using Cinfinite-r assms(2) unfolding cinfinite-def by (simp add: finite-ordLess-infinite)
  next
    case False
    moreover have ‹∀k. finite (?paramss k)›
      by (simp add: finite-params finite-witness-params)
    then have ‹∀k. |?paramss k| <o |underS r i|›
      using False by simp
    ultimately have ‹|?X| ≤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 − paramss S|›
    using assms(2) ordLess-ordLeq-trans by blast
```

```
  moreover have ‹infinite (UNIV − paramss S)›
    using assms(2) Cinfinite-r unfolding cinfinite-def by (metis Field-card-of ordLeq-finite-Field)
  ultimately have ‹|UNIV − paramss S − ?X| =o |UNIV − paramss S|›
    using card-of-Un-diff-infinite by blast
  moreover from this have ‹infinite (UNIV − paramss S − ?X)›
    using ‹infinite (UNIV − paramss S)› card-of-ordIso-finite by blast
  moreover have ‹⋀a. a ∈ paramss (extend S i) ⟹ a ∈ paramss S ∨ a ∈ ?X›
    using params-origin by simp
  then have ‹paramss (extend S i) ⊆ paramss S ∪ ?X›
    by fast
  ultimately have ‹infinite (UNIV − paramss (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 simp
  next
  case (3 i)
  show ?case
  proof (rule ccontr)
    assume ‹¬ consistent (extend S i)›
    then obtain S' where S': ‹finite S'› ‹S' ⊆ (⋃a ∈ underS r i. extend S a)› ‹¬ 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 as where as: ‹S' ⊆ (⋃a ∈ as. extend S a)› ‹as ⊆ underS r i› ‹finite as›
      by (metis finite-subset-Union finite-subset-image)
    moreover have ‹as ≠ {}›
      using S'(3) assms calculation(1) consistent-hereditary by auto
    ultimately obtain j where ‹∀a ∈ as. a ∈ under r j› ‹j ∈ underS r i›
      using wo-rel.finite-underS-bound wo-rel-r as by (meson subset-iff)
    then have ‹∀a ∈ as. extend S a ⊆ extend S j›
      using extend-under by fast
    then have ‹S' ⊆ extend S j›
      using S' as(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 − paramss S|›
  shows ‹consistent (Extend S)›
  unfolding Extend-def
proof (rule ccontr)
  assume ‹¬ consistent (⋃a ∈ Field r. extend S a)›
  then obtain S' where ‹finite S'› ‹S' ⊆ (⋃a ∈ Field r. extend S a)› ‹¬ consistent S'›
    using inconsistent-finite by metis
  then obtain b where ‹S' ⊆ (⋃a ∈ under r b. extend S a)› ‹b ∈ 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 b›
    using extend-under by fast
  moreover have ‹consistent (extend S b)›
    using assms consistent-extend ‹b ∈ Field r› by blast
  ultimately show False
    using ‹¬ consistent S'› consistent-hereditary by blast
```

qed

lemma *Extend-bound*: ‹$a \in Field\ r \Longrightarrow extend\ S\ a \subseteq Extend\ S$›
unfolding *Extend-def* **by** *blast*

1.3.3 Maximality

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

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

1.3.4 Witnessing

definition *witnessed'* :: ‹'*a set* ⇒ *bool*› **where**
‹$witnessed'\ S \equiv \forall p \in Field\ r.\ p \in S \longrightarrow (\exists S'.\ witness\ p\ S' \subseteq S)$›

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

ultimately show ‹∃ *a*. *witness p a* ⊆ *Extend S*›
by *fast*
qed

end

1.4 Locales for Universe Well-Order

locale *MCS-Witness-UNIV* = *MCS-Witness consistent witness params*
for *consistent* :: ‹'*a set* ⇒ *bool*›
and *witness* :: ‹'*a* ⇒ '*a set* ⇒ '*a set*›
and *params* :: ‹'*a* ⇒ '*i set*› +
assumes *infinite-UNIV*: ‹*infinite* (*UNIV* :: '*a set*)›

sublocale *MCS-Witness-UNIV* ⊆ *MCS-Lim-Ord consistent witness params* ‹|*UNIV*|›
proof
show ‹*Cinfinite* |*UNIV* :: '*a set*|›
unfolding *cinfinite-def* **using** *infinite-UNIV* **by** *simp*
qed

context *MCS-Witness-UNIV* **begin**

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*

lemma *witnessed-witnessed'*: ‹*witnessed S* ⟷ *witnessed' S*›
unfolding *witnessed-def witnessed'-def* **by** *auto*

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

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

end

locale *MCS-No-Witness-UNIV* = *MCS-No-Witness consistent*
for *consistent* :: ‹'*a set* ⇒ *bool*› +
assumes *infinite-UNIV'* [*simp*]: ‹*infinite* (*UNIV* :: '*a set*)›

sublocale *MCS-No-Witness-UNIV* ⊆ *MCS-Witness-UNIV consistent* ‹λ- -. {}› ‹λ-. {}›
proof qed *simp*

context *MCS-No-Witness-UNIV*
begin

theorem *MCS-Extend'*:
assumes ‹*consistent S*›

shows ‹*MCS* (*Extend S*)›
unfolding *witnessed-def* **using** *assms consistent-Extend maximal-Extend*
by (*metis Diff-empty UN-constant card-of-UNIV empty-subsetI*)

end

1.5 Truth Lemma

locale *Truth-Base* =
fixes *semics* :: ‹'*model* ⇒ ('*model* ⇒ '*fm* ⇒ *bool*) ⇒ '*fm* ⇒ *bool*› (‹(- ⟦-⟧ -)› [55, 0, 55] 55)
and *semantics* :: ‹'*model* ⇒ '*fm* ⇒ *bool*› (**infix** ‹⊨› 50)
and $\mathcal{M}$ :: ‹'*a set* ⇒ '*model set*›
and $\mathcal{R}$ :: ‹'*a set* ⇒ '*model* ⇒ '*fm* ⇒ *bool*›
assumes *semics-semantics*: ‹$M \models p \longleftrightarrow M \ [\![(\models)]\!] \ p$›
begin

abbreviation *saturated* :: ‹'*a set* ⇒ *bool*› **where**
‹*saturated S* ≡ ∀ *p*. ∀ $M \in \mathcal{M}(S)$. $M \ [\![\mathcal{R}(S)]\!] \ p \longleftrightarrow \mathcal{R}(S) \ M \ p$›

end

locale *Truth-Witness* = *Truth-Base semics semantics* $\mathcal{M}$ $\mathcal{R}$ + *MCS-Witness consistent witness params*
for *semics* :: ‹'*model* ⇒ ('*model* ⇒ '*fm* ⇒ *bool*) ⇒ '*fm* ⇒ *bool*› (‹(- ⟦-⟧ -)› [55, 0, 55] 55)
and *semantics* :: ‹'*model* ⇒ '*fm* ⇒ *bool*› (**infix** ‹⊨› 50)
and $\mathcal{M}$ :: ‹'*a set* ⇒ '*model set*›
and $\mathcal{R}$ :: ‹'*a set* ⇒ '*model* ⇒ '*fm* ⇒ *bool*›
and *consistent* :: ‹'*a set* ⇒ *bool*›
and *witness* :: ‹'*a* ⇒ '*a set* ⇒ '*a set*›
and *params* :: ‹'*a* ⇒ '*i set*› +
assumes *saturated-semantics*: ‹$\bigwedge S \ M \ p$. *saturated S* ⟹ $M \in \mathcal{M}(S)$ ⟹ $M \models p \longleftrightarrow \mathcal{R}(S) \ M \ p$›
and *MCS-saturated*: ‹$\bigwedge S$. *MCS S* ⟹ *saturated S*›
begin

theorem *truth-lemma*:
assumes ‹*MCS S*› ‹$M \in \mathcal{M}(S)$›
shows ‹$M \models p \longleftrightarrow \mathcal{R}(S) \ M \ p$›
using *saturated-semantics MCS-saturated assms* **by** *blast*

end

locale *Truth-No-Witness* = *Truth-Witness semics semantics* $\mathcal{M}$ $\mathcal{R}$ *consistent* ‹λ- -. {}› ‹λ-. {}›
for *semics* :: ‹'*model* ⇒ ('*model* ⇒ '*fm* ⇒ *bool*) ⇒ '*fm* ⇒ *bool*›
and *semantics* :: ‹'*model* ⇒ '*fm* ⇒ *bool*›
and $\mathcal{M}$ :: ‹'*a set* ⇒ '*model set*›
and $\mathcal{R}$ :: ‹'*a set* ⇒ '*model* ⇒ '*fm* ⇒ *bool*›
and *consistent* :: ‹'*a set* ⇒ *bool*›

end

Chapter 2

Derivations

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

lemma *split-finite-sets*:
  assumes ‹*finite A*› ‹*finite B*›
    and ‹$A \subseteq B \cup S$›
  shows ‹$\exists B'\ C.$ *finite* $C \wedge A = B' \cup C \wedge B' = A \cap B \wedge C \subseteq S$›
  using *assms subset-UnE* **by** *auto*

lemma *split-list*:
  assumes ‹*set A* $\subseteq$ *set B* $\cup$ *S*›
  shows ‹$\exists B'\ C.$ *set* $(B' @ C) =$ *set A* $\wedge$ *set* $B' =$ *set A* $\cap$ *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*)

2.1 Derivations

locale *Derivations* =
  fixes *derive* :: ‹*'fm list* ⇒ *'fm* ⇒ *bool*› (**infix** ‹⊢› *50*)
  assumes *derive-assm* [*simp*]: ‹$\bigwedge A\ p.\ p \in$ *set A* $\Longrightarrow A \vdash p$›
    and *derive-set*: ‹$\bigwedge A\ B\ r.\ A \vdash r \Longrightarrow$ *set A* = *set B* $\Longrightarrow B \vdash r$›
begin

theorem *derive-split*:
  assumes ‹*set A* $\subseteq$ *set B* $\cup$ *X*› ‹$A \vdash p$›
  shows ‹$\exists B'\ C.$ *set* $B' =$ *set A* $\cap$ *set B* $\wedge$ *set C* $\subseteq X \wedge B' @ C \vdash p$›
  using *assms derive-set split-list*[**where** *A*=*A* **and** *B*=*B*] **by** *metis*

corollary *derive-split1*:
  assumes ‹*set A* $\subseteq \{q\} \cup X$› ‹$A \vdash p$› ‹$q \in$ *set A*›
  shows ‹$\exists C.$ *set* $C \subseteq X \wedge q \# C \vdash p$›
  using *assms derive-split*[**where** *A*=*A* **and** *X*=*X* **and** *B*=‹[*q*]› **and** *p*=*p*] *derive-set*[**where** *B*=‹*q* # -›]
  by *auto*

end

2.2 MCSs and Explosion

locale *Derivations-MCS* = *MCS-Base consistent* + *Derivations derive*
  for *consistent* :: ‹*'fm set* ⇒ *bool*›

and *derive* :: ‹*'fm list ⇒ 'fm ⇒ bool*› (**infix** ‹⊢› *50*) +
assumes *consistent-underivable*: ‹⋀*S*. *consistent S* ⟷ (∀ *A*. *set A* ⊆ *S* ⟶ (∃ *q*. ¬ *A* ⊢ *q*))›
begin

theorem *MCS-explode*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹*p* ∉ *S* ⟷ (∃ *A*. *set A* ⊆ *S* ∧ (∀ *q*. *p* # *A* ⊢ *q*))›
proof *safe*
assume ‹*p* ∉ *S*›
then obtain *B* **where** *B*: ‹*set B* ⊆ {*p*} ∪ *S*› ‹*p* ∈ *set B*› ‹∀ *q*. *B* ⊢ *q*›
using *assms* **unfolding** *consistent-underivable maximal-def* **by** *blast*
moreover have ‹*set* (*p* # *removeAll p B*) = *set B*›
using *B*(*2*) **by** *auto*
ultimately have ‹∀ *q*. *p* # *removeAll p B* ⊢ *q*›
using *derive-set* **by** *metis*
then show ‹∃ *A*. *set A* ⊆ *S* ∧ (∀ *q*. *p* # *A* ⊢ *q*)›
using *B*(*1*) **by** (*metis Diff-subset-conv set-removeAll*)
next
fix *A*
assume ‹*set A* ⊆ *S*› ‹∀ *q*. *p* # *A* ⊢ *q*› ‹*p* ∈ *S*›
then show *False*
using *assms* **unfolding** *consistent-underivable*
by (*metis* (*no-types*, *lifting*) *insert-subsetI list.simps*(*15*))
qed

end

2.3 MCSs and Derivability

locale *Derivations-Cut-MCS* = *Derivations-MCS consistent derive*
for *consistent* :: ‹*'fm set ⇒ bool*›
and *derive* :: ‹*'fm list ⇒ 'fm ⇒ bool*› (**infix** ‹⊢› *50*) +
assumes *derive-cut*: ‹⋀*A B p q*. *A* ⊢ *p* ⟹ *p* # *B* ⊢ *q* ⟹ *A* @ *B* ⊢ *q*›
begin

theorem *MCS-derive*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹*p* ∈ *S* ⟷ (∃ *A*. *set A* ⊆ *S* ∧ *A* ⊢ *p*)›
proof *safe*
assume ‹*p* ∈ *S*›
then show ‹∃ *A*. *set A* ⊆ *S* ∧ *A* ⊢ *p*›
using *derive-assm* **by** (*metis List.set-insert empty-set empty-subsetI insert-subset singletonI*)
next
fix *A*
assume *A*: ‹*set A* ⊆ *S*› ‹*A* ⊢ *p*›

have *bot*: ‹∀ *A*. *set A* ⊆ *S* ⟶ (∃ *q*. ¬ *A* ⊢ *q*)›
using *assms*(*1*) **unfolding** *consistent-underivable* **by** *blast*

have ‹*consistent* ({*p*} ∪ *S*)›
unfolding *consistent-underivable*
proof *safe*
fix *B*
assume *B*: ‹*set B* ⊆ {*p*} ∪ *S*›
show ‹∃ *q*. ¬ *B* ⊢ *q*›

```
    proof (rule ccontr)
      assume ∗: ‹∄q. ¬ B ⊢ q›
      then have ‹∀q. B ⊢ q›
        by blast
      show False
      proof (cases ‹p ∈ set B›)
        case True
        then have ‹set (p # removeAll p B) = set B›
          by auto
        then have ‹∀q. p # removeAll p B ⊢ q›
          using ‹∀q. B ⊢ q› derive-set by blast
        then have ‹∀q. A @ removeAll p B ⊢ q›
          using A(2) derive-cut by blast
        moreover have ‹set (A @ removeAll p B) ⊆ S›
          using A(1) B by auto
        ultimately show False
          using bot by blast
      next
        case False
        then show False
          using ∗ B bot by auto
      qed
    qed
  qed
  then show ‹p ∈ S›
    using assms unfolding maximal-def by auto
qed

end
```

2.4 Proof Rules

```
locale Derivations-Bot = Derivations-Cut-MCS consistent derive
  for consistent :: ‹'fm set ⇒ bool›
    and derive :: ‹'fm list ⇒ 'fm ⇒ bool› (infix ‹⊢› 50) +
  fixes bot :: 'fm (‹⊥›)
  assumes botE: ‹⋀A p. A ⊢ ⊥ ⟹ A ⊢ p›
begin

corollary MCS-botE [elim]:
  assumes ‹consistent S› ‹maximal S›
    and ‹⊥ ∈ S›
  shows ‹p ∈ S›
  using assms botE MCS-derive by blast

corollary MCS-bot [simp]:
  assumes ‹consistent S› ‹maximal S›
  shows ‹⊥ ∉ S›
  using assms botE MCS-derive consistent-underivable by blast

end

locale Derivations-Top = Derivations-Cut-MCS +
  fixes top (‹⊤›)
  assumes topI: ‹⋀A. A ⊢ ⊤›
```

begin

corollary *MCS-topI* [*simp*]:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹$\top \in S$›
using *assms topI MCS-derive* **by** (*metis empty-set empty-subsetI*)

end

locale *Derivations-Not* = *Derivations-Bot consistent derive bot*
for *consistent* :: ‹*'fm set* ⇒ *bool*›
and *derive* :: ‹*'fm list* ⇒ *'fm* ⇒ *bool*› (**infix** ‹⊢› *50*)
and *bot* :: *'fm* (‹⊥›) +
fixes *not* :: ‹*'fm* ⇒ *'fm*› (‹¬›)
assumes
notI: ‹$\bigwedge A\ p.\ p \# A \vdash \bot \Longrightarrow A \vdash \neg p$› **and**
notE: ‹$\bigwedge A\ p.\ A \vdash p \Longrightarrow A \vdash \neg p \Longrightarrow A \vdash \bot$›
begin

corollary *MCS-not-xor*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹$p \in S \longleftrightarrow \neg p \notin S$›
proof *safe*
assume ‹$p \in S$› ‹$\neg p \in S$›
then have ‹*set* [p, $\neg p$] ⊆ S›
by *simp*
moreover have ‹[p, $\neg p$] ⊢ ⊥›
using *notE derive-assm* **by** (*meson list.set-intros*(*1*) *list.set-intros*(*2*))
ultimately have ‹⊥ ∈ S›
using *assms MCS-derive* **by** *blast*
then show *False*
using *assms MCS-bot* **by** *blast*
next
assume ∗: ‹$\neg p \notin S$›
show ‹$p \in S$›
proof (*rule ccontr*)
assume ‹$p \notin S$›
then obtain A **where** A: ‹*set* $A \subseteq S$› ‹$\forall q.\ p \# A \vdash q$›
using *assms MCS-explode* **by** *blast*
then have ‹$p \# A \vdash \bot$›
by *fast*
then have ‹$A \vdash \neg p$›
using *notI* **by** *blast*
then have ‹$\neg p \in S$›
using $A(1)$ *assms MCS-derive* **by** *blast*
then show *False*
using ∗ **by** *blast*
qed
qed

corollary *MCS-not-both*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹$p \notin S \vee \neg p \notin S$›
using *assms MCS-not-xor* **by** *blast*

corollary *MCS-not-neither*:

assumes ‹*consistent S*› ‹*maximal S*›
shows ‹$p \in S \vee \neg\, p \in S$›
using *assms MCS-not-xor* **by** *blast*

end

locale *Derivations-Con* = *Derivations-Cut-MCS consistent derive*
for *consistent* :: ‹*'fm set* ⇒ *bool*›
and *derive* :: ‹*'fm list* ⇒ *'fm* ⇒ *bool*› (**infix** ‹⊢› *50*) +
fixes *con* :: ‹*'fm* ⇒ *'fm* ⇒ *'fm*› (‹- ∧ -›)
assumes
conI: ‹$\bigwedge A\ p\ q.\ A \vdash p \Longrightarrow A \vdash q \Longrightarrow A \vdash (p \wedge q)$› **and**
conE: ‹$\bigwedge A\ p\ q\ r.\ A \vdash (p \wedge q) \Longrightarrow p\ \#\ q\ \#\ A \vdash r \Longrightarrow A \vdash r$›
begin

corollary *MCS-conI* [*intro*]:
assumes ‹*consistent S*› ‹*maximal S*›
and ‹$p \in S$› ‹$q \in S$›
shows ‹$(p \wedge q) \in S$›
using *assms MCS-derive derive-assm conI*
by (*metis* (*mono-tags*, *lifting*) *insert-subset list.set-intros*(*1*) *list.simps*(*15*) *set-subset-Cons*)

corollary *MCS-conE* [*dest*]:
assumes ‹*consistent S*› ‹*maximal S*›
and ‹$(p \wedge q) \in S$›
shows ‹$p \in S \wedge q \in S$›
proof −
have ‹$p\ \#\ q\ \#\ A \vdash p$› ‹$p\ \#\ q\ \#\ A \vdash q$› **for** *A*
using *derive-assm* **by** *simp-all*
then show *?thesis*
using *assms MCS-derive conE* **by** *blast*
qed

corollary *MCS-con*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹$(p \wedge q) \in S \longleftrightarrow p \in S \wedge q \in S$›
using *assms MCS-conI MCS-conE* **by** *blast*

end

locale *Derivations-Dis* = *Derivations-Cut-MCS consistent derive*
for *consistent* :: ‹*'fm set* ⇒ *bool*›
and *derive* :: ‹*'fm list* ⇒ *'fm* ⇒ *bool*› (**infix** ‹⊢› *50*) +
fixes *dis* :: ‹*'fm* ⇒ *'fm* ⇒ *'fm*› (‹- ∨ -›)
assumes
disI1: ‹$\bigwedge A\ p\ q.\ A \vdash p \Longrightarrow A \vdash (p \vee q)$› **and**
disI2: ‹$\bigwedge A\ p\ q.\ A \vdash q \Longrightarrow A \vdash (p \vee q)$› **and**
disE: ‹$\bigwedge A\ p\ q\ r.\ A \vdash (p \vee q) \Longrightarrow p\ \#\ A \vdash r \Longrightarrow q\ \#\ A \vdash r \Longrightarrow A \vdash r$›
begin

corollary *MCS-disI1* [*intro*]:
assumes ‹*consistent S*› ‹*maximal S*›
and ‹$p \in S$›
shows ‹$(p \vee q) \in S$›
using *assms disI1 MCS-derive* **by** *auto*

corollary *MCS-disI2* [*intro*]:
assumes ‹*consistent S*› ‹*maximal S*›
and ‹$q \in S$›
shows ‹$(p \vee q) \in S$›
using *assms disI2 MCS-derive* **by** *auto*

corollary *MCS-disE* [*elim*]:
assumes ‹*consistent S*› ‹*maximal S*›
and ‹$(p \vee q) \in S$›
shows ‹$p \in S \vee q \in S$›
proof (*rule ccontr*)
have *bot*: ‹$\forall A.\ set\ A \subseteq S \longrightarrow (\exists q.\ \neg\ A \vdash q)$›
using *assms*(*1*) **unfolding** *consistent-underivable* **by** *blast*

assume ‹$\neg\ (p \in S \vee q \in S)$›
then obtain *P Q* **where**
P: ‹$set\ P \subseteq S$› ‹$\forall r.\ p\ \#\ P \vdash r$› **and**
Q: ‹$set\ Q \subseteq S$› ‹$\forall r.\ q\ \#\ Q \vdash r$›
using *assms MCS-explode* **by** *auto*

have ‹$p\ \#\ (p \vee q)\ \#\ Q \vdash p$›
by *simp*
then have ‹$p\ \#\ (p \vee q)\ \#\ Q\ @\ P \vdash r$› **for** *r*
using *P derive-cut*[*of - p*] **by** (*metis append-Cons*)
then have ‹$p\ \#\ (p \vee q)\ \#\ P\ @\ Q \vdash r$› **for** *r*
using *derive-set*[**where** *B*=‹$p\ \#\ (p \vee q)\ \#\ P\ @\ Q$›] **by** *fastforce*
moreover have ‹$q\ \#\ (p \vee q)\ \#\ P \vdash q$›
by *simp*
then have ‹$q\ \#\ (p \vee q)\ \#\ P\ @\ Q \vdash r$› **for** *r*
using *Q derive-cut*[*of - q*] **by** (*metis append-Cons*)
moreover have ‹$(p \vee q)\ \#\ P\ @\ Q \vdash (p \vee q)$›
by *simp*
ultimately have ‹$(p \vee q)\ \#\ P\ @\ Q \vdash r$› **for** *r*
using *disE* **by** *blast*
moreover have ‹$set\ ((p \vee q)\ \#\ P\ @\ Q) \subseteq S$›
using *assms*(*3*) *P Q* **by** *simp*
ultimately show *False*
using *assms*(*1*) **unfolding** *consistent-underivable* **by** *blast*
qed

corollary *MCS-dis*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹$(p \vee q) \in S \longleftrightarrow p \in S \vee q \in S$›
using *assms MCS-disI1 MCS-disI2 MCS-disE* **by** *blast*

end

locale *Derivations-Imp* = *Derivations-Cut-MCS consistent derive*
for *consistent* :: ‹*'fm set* ⇒ *bool*›
and *derive* :: ‹*'fm list* ⇒ *'fm* ⇒ *bool*› (**infix** ‹⊢› *50*) +
fixes *imp* :: ‹*'fm* ⇒ *'fm* ⇒ *'fm*› (‹- → -›)
assumes
impI: ‹$\bigwedge A\ p\ q.\ p\ \#\ A \vdash q \Longrightarrow A \vdash (p \rightarrow q)$› **and**
impE: ‹$\bigwedge A\ p\ q.\ A \vdash p \Longrightarrow A \vdash (p \rightarrow q) \Longrightarrow A \vdash q$›
begin

corollary *MCS-impI* [*intro*]:
assumes ‹*consistent S*› ‹*maximal S*›
and ‹$p \in S \longrightarrow q \in S$›
shows ‹$(p \rightarrow q) \in S$›
using *assms impI derive-assm MCS-derive MCS-explode*
by (*metis insert-subset list.simps*(15) *subsetI*)

corollary *MCS-impE* [*dest*]:
assumes ‹*consistent S*› ‹*maximal S*›
and ‹$(p \rightarrow q) \in S$› ‹$p \in S$›
shows ‹$q \in S$›
using *assms*(3−4) *impE MCS-derive*[*OF assms*(1−2)] *derive-assm*
by (*metis insert-subset list.set-intros*(1) *list.simps*(15) *set-subset-Cons*)

corollary *MCS-imp*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹$(p \rightarrow q) \in S \longleftrightarrow (p \in S \longrightarrow q \in S)$›
using *assms MCS-impI MCS-impE* **by** *blast*

end

locale *Derivations-Exi* = *MCS-Witness consistent witness params* + *Derivations-Cut-MCS consistent derive*
for *consistent* :: ‹*'fm set* ⇒ *bool*›
and *witness params*
and *derive* :: ‹*'fm list* ⇒ *'fm* ⇒ *bool*› (**infix** ‹⊢› *50*) +
fixes *exi* :: ‹*'fm* ⇒ *'fm*› (‹∃›)
and *inst* :: ‹*'t* ⇒ *'fm* ⇒ *'fm*› (‹⟨-⟩›)
assumes
exi-witness: ‹$\bigwedge S\ S'\ p.$ *MCS S* ⟹ *witness* (∃ *p*) $S' \subseteq S \Longrightarrow \exists\, t.\ \langle t\rangle p \in S$› **and**
exiI: ‹$\bigwedge A\ p\ t.\ A \vdash \langle t\rangle p \Longrightarrow A \vdash \exists\, p$›
begin

corollary *MCS-exiI* [*intro*]:
assumes ‹*consistent S*› ‹*maximal S*›
and ‹$\langle t\rangle p \in S$›
shows ‹$\exists\, p \in S$›
using *assms MCS-derive exiI* **by** *blast*

corollary *MCS-exiE* [*dest*]:
assumes ‹*consistent S*› ‹*maximal S*› ‹*witnessed S*›
and ‹$\exists\, p \in S$›
shows ‹$\exists\, t.\ \langle t\rangle p \in S$›
using *assms exi-witness* **unfolding** *witnessed-def* **by** *blast*

corollary *MCS-exi*:
assumes ‹*consistent S*› ‹*maximal S*› ‹*witnessed S*›
shows ‹$\exists\, p \in S \longleftrightarrow (\exists\, t.\ \langle t\rangle p \in S)$›
using *assms MCS-exiI MCS-exiE* **by** *blast*

end

locale *Derivations-Uni* = *MCS-Witness consistent witness params* + *Derivations-Not consistent derive bot not*
for *consistent* :: ‹*'fm set* ⇒ *bool*›
and *witness params*

and *derive* :: ‹*'fm list* ⇒ *'fm* ⇒ *bool*› (**infix** ‹⊢› *50*)
and *bot* :: *'fm* (‹⊥›)
and *not* :: ‹*'fm* ⇒ *'fm*› (‹¬›) +
fixes *uni* :: ‹*'fm* ⇒ *'fm*› (‹∀›)
and *inst* :: ‹*'t* ⇒ *'fm* ⇒ *'fm*› (‹⟨-⟩›)
assumes
uni-witness: ‹⋀*S S' p*. *MCS S* ⟹ *witness* (¬ (∀ *p*)) *S'* ⊆ *S* ⟹ ∃ *t*. ¬ (⟨*t*⟩*p*) ∈ *S*› **and**
uniE: ‹⋀*A p t*. *A* ⊢ ∀ *p* ⟹ *A* ⊢ ⟨*t*⟩*p*›
begin

corollary *MCS-uniE* [*dest*]:
assumes ‹*consistent S*› ‹*maximal S*›
and ‹∀ *p* ∈ *S*›
shows ‹⟨*t*⟩*p* ∈ *S*›
using *assms MCS-derive uniE* **by** *blast*

corollary *MCS-uniI* [*intro*]:
assumes ‹*consistent S*› ‹*maximal S*› ‹*witnessed S*›
and ‹∀ *t*. ⟨*t*⟩*p* ∈ *S*›
shows ‹∀ *p* ∈ *S*›
proof (*rule ccontr*)
assume ‹∀ *p* ∉ *S*›
then have ‹¬ (∀ *p*) ∈ *S*›
using *assms MCS-not-xor* **by** *blast*
then have ‹∃ *t*. ¬ (⟨*t*⟩*p*) ∈ *S*›
using *assms uni-witness* **unfolding** *witnessed-def* **by** *blast*
then show *False*
using *assms MCS-not-xor* **by** *blast*
qed

corollary *MCS-uni*:
assumes ‹*consistent S*› ‹*maximal S*› ‹*witnessed S*›
shows ‹∀ *p* ∈ *S* ⟷ (∀ *t*. ⟨*t*⟩*p* ∈ *S*)›
using *assms MCS-uniI MCS-uniE* **by** *blast*

end

end

Chapter 3

Refutations

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

lemma *split-finite-sets*:
  assumes ‹*finite A*› ‹*finite B*›
    and ‹$A \subseteq B \cup S$›
  shows ‹$\exists\, B'\ C.$ *finite* $C \wedge A = B' \cup C \wedge B' = A \cap B \wedge C \subseteq S$›
  using *assms subset-UnE* **by** *auto*

lemma *split-list*:
  assumes ‹*set A* $\subseteq$ *set B* $\cup$ *S*›
  shows ‹$\exists\, B'\ C.$ *set* $(B' @ C) =$ *set A* $\wedge$ *set* $B' =$ *set A* $\cap$ *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*)

3.1 Rearranging Refutations

locale *Refutations* =
  fixes *refute* :: ‹'*fm list* ⇒ *bool*›
  assumes *refute-set*: ‹$\bigwedge A\ B.$ *refute A* $\Longrightarrow$ *set A* = *set B* $\Longrightarrow$ *refute B*›
begin

theorem *refute-split*:
  assumes ‹*set A* $\subseteq$ *set B* $\cup$ *X*› ‹*refute A*›
  shows ‹$\exists\, B'\ C.$ *set* $B' =$ *set A* $\cap$ *set B* $\wedge$ *set* $C \subseteq X \wedge$ *refute* $(B' @ C)$›
  using *assms refute-set split-list*[**where** *A*=*A* **and** *B*=*B*] **by** *metis*

corollary *refute-split1*:
  assumes ‹*set A* $\subseteq \{q\} \cup X$› ‹*refute A*› ‹$q \in$ *set A*›
  shows ‹$\exists\, C.$ *set* $C \subseteq X \wedge$ *refute* $(q \# C)$›
  using *assms refute-split*[**where** *A*=*A* **and** *X*=*X* **and** *B*=‹$[q]$›] *refute-set* **by** *auto*

end

3.2 MCSs and Refutability

locale *Refutations-MCS* = *MCS-Base* + *Refutations* +
  assumes *consistent-refute*: ‹$\bigwedge S.$ *consistent S* $\longleftrightarrow$ $(\forall\, A.$ *set A* $\subseteq S \longrightarrow \neg$ *refute A*$)$›
begin

theorem *MCS-refute*:

assumes ‹*consistent S*› ‹*maximal S*›
shows ‹$p \notin S \longleftrightarrow (\exists A.\ set\ A \subseteq S \wedge refute\ (p\ \#\ A))$›
proof *safe*
assume ‹$p \notin S$›
then obtain *B* **where** *B*: ‹$set\ B \subseteq \{p\} \cup S$› ‹$p \in set\ B$› ‹*refute B*›
using *assms* **unfolding** *consistent-refute maximal-def* **by** *blast*
moreover have ‹$set\ (p\ \#\ removeAll\ p\ B) = set\ B$›
using *B*(*2*) **by** *auto*
ultimately have ‹*refute* $(p\ \#\ removeAll\ p\ B)$›
using *refute-set* **by** *metis*
then show ‹$\exists A.\ set\ A \subseteq S \wedge refute\ (p\ \#\ A)$›
using *B*(*1*) **by** (*metis Diff-subset-conv set-removeAll*)
next
fix *A*
assume ‹$set\ A \subseteq S$› ‹*refute* $(p\ \#\ A)$› ‹$p \in S$›
then show *False*
using *assms* **unfolding** *consistent-refute*
by (*metis* (*no-types*, *lifting*) *insert-subsetI list.simps*(*15*))
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*› (**infix** ‹⊨$_T$› *50*) **where**
‹*I* ⊨$_T$ •*P* ⟷ *I P*›
| ‹*I* ⊨$_T$ ¬ *p* ⟷ ¬ *I* ⊨$_T$ *p*›
| ‹*I* ⊨$_T$ *p* ⟶ *q* ⟷ *I* ⊨$_T$ *p* ⟶ *I* ⊨$_T$ *q*›

4.3 Calculus

inductive *Calculus* :: ‹*'p fm list* ⇒ *bool*› (‹⊢$_T$ -› [*50*] *50*) **where**
Axiom [*simp*]: ‹⊢$_T$ •*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*›
| *Weak*: ‹⊢$_T$ *A* ⟹ *set A* ⊆ *set B* ⟹ ⊢$_T$ *B*›

lemma *Weak2*:
assumes ‹⊢$_T$ *p* # *A*› ‹⊢$_T$ *q* # *B*›
shows ‹⊢$_T$ *p* # *A* @ *B* ∧ ⊢$_T$ *q* # *A* @ *B*›
using *assms Weak*[**where** *A*=‹- # -› **and** *B*=‹- # *A* @ *B*›] **by** *fastforce*

4.4 Soundness

theorem *soundness*: ‹⊢$_T$ *A* ⟹ ∃ *p* ∈ *set A*. ¬ *I* ⊨$_T$ *p*›
by (*induct A rule*: *Calculus.induct*) *auto*

corollary *soundness'*: ‹$\vdash_T [\neg\ p] \Longrightarrow I \models_T p$›
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* ≡ ∀ *A*. *set A* ⊆ *S* ⟶ ¬ ⊢$_T$ *A*›

interpretation *MCS-No-Witness-UNIV consistent*
proof
show ‹*infinite* (*UNIV* :: *'p fm set*)›
using *infinite-UNIV-size*[*of* ‹λ*p*. *p* ⟶ *p*›] **by** *simp*
qed (*auto simp*: *consistent-def*)

interpretation *Refutations-MCS consistent Calculus*
proof
fix *A B* :: ‹*'p fm list*›
assume ‹⊢$_T$ *A*› ‹*set A* = *set B*›
then show ‹⊢$_T$ *B*›
using *Weak* **by** *blast*
next
fix *S* :: ‹*'p fm set*›
show ‹*consistent S* ⟷ (∀ *A*. *set A* ⊆ *S* ⟶ ¬ ⊢$_T$ *A*)›
unfolding *consistent-def* **..**
qed

4.6 Truth Lemma

abbreviation (*input*) *canonical* :: ‹*'p fm set* ⇒ *'p model*› (‹$\mathcal{M}_T$›) **where**
‹$\mathcal{M}_T(S) \equiv \lambda P.\ \cdot P \in S$›

locale *Hintikka* =
fixes *H* :: ‹*'a fm set*›
assumes *AxiomH*: ‹$\bigwedge P.\ \cdot P \in H \Longrightarrow \neg\ \cdot P \in H \Longrightarrow$ *False*›
and *NegIH*: ‹$\bigwedge p.\ \neg\ \neg\ p \in H \Longrightarrow p \in H$›
and *ImpPH*: ‹$\bigwedge p\ q.\ p \longrightarrow q \in H \Longrightarrow \neg\ p \in H \vee q \in H$›
and *ImpNH*: ‹$\bigwedge p\ q.\ \neg\ (p \longrightarrow q) \in H \Longrightarrow p \in H \wedge \neg\ q \in H$›

lemma *Hintikka-model*:
assumes ‹*Hintikka H*›
shows ‹$(p \in H \longrightarrow \mathcal{M}_T(H) \models_T p) \wedge (\neg\ p \in H \longrightarrow \neg\ \mathcal{M}_T(H) \models_T p)$›
using *assms* **by** (*induct p*) (*unfold Hintikka-def semantics.simps*; *blast*)+

lemma *MCS-Hintikka*:
assumes ‹*MCS H*›
shows ‹*Hintikka H*›
proof
fix *P*
assume ‹$\cdot P \in H$› ‹$\neg\ \cdot P \in H$›
then have ‹*set* [$\cdot P$, $\neg\ \cdot P$] ⊆ *H*›
by *simp*

```
  moreover have ‹⊢T [·P, ¬ ·P]›
    by simp
  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 Weak2[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 Weak[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 Weak 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 ‹MCS H› ‹p ∈ H›
 shows ‹𝓜_T(H) ⊨_T p›
 using Hintikka-model MCS-Hintikka assms by blast
```

4.7 Completeness

```
theorem strong-completeness:
 assumes ‹∀ M. (∀ q ∈ X. M ⊨_T q) ⟶ M ⊨_T p›
 shows ‹∃ A. set A ⊆ X ∧ ⊢_T ¬ p # A›
proof (rule ccontr)
 assume ‹∄ A. set A ⊆ X ∧ ⊢_T ¬ p # A›
 then have ∗: ‹∀ A. set A ⊆ {¬ p} ∪ X ⟶ ¬ ⊢_T A›
   using refute-split1 by (metis Weak insert-is-Un set-subset-Cons subset-insert)

 let ?S = ‹{¬ p} ∪ X›
 let ?H = ‹Extend ?S›

 have ‹consistent ?S›
   unfolding consistent-def using ∗ by blast
 then have ‹MCS ?H›
   using MCS-Extend′ by blast
 then have ‹p ∈ ?H ⟶ 𝓜_T(?H) ⊨_T p› for p
   using truth-lemma by blast
 then have ‹p ∈ ?S ⟶ 𝓜_T(?H) ⊨_T p› for p
   using Extend-subset by blast
 then have ‹𝓜_T(?H) ⊨_T ¬ p› ‹∀ q ∈ X. 𝓜_T(?H) ⊨_T q›
   by blast+
 moreover from this have ‹𝓜_T(?H) ⊨_T p›
   using assms(1) by blast
 ultimately show False
   by simp
qed

abbreviation valid :: ‹'p fm ⇒ bool› where
 ‹valid p ≡ ∀ M. M ⊨_T p›

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

theorem main: ‹valid p ⟷ ⊢_T [¬ 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*› (**infix** ‹⊨$_S$› *50*) **where**
  ‹- ⊨$_S$ ⊥ ⟷ *False*›
| ‹*I* ⊨$_S$ •*P* ⟷ *I P*›
| ‹*I* ⊨$_S$ *p* ⟶ *q* ⟷ *I* ⊨$_S$ *p* ⟶ *I* ⊨$_S$ *q*›

5.3 Calculus

inductive *Calculus* :: ‹*'p fm list* ⇒ *'p fm list* ⇒ *bool*› (**infix** ‹⊢$_S$› *50*) **where**
  Axiom [*simp*]: ‹*p* # *A* ⊢$_S$ *p* # *B*›
| *FlsL* [*simp*]: ‹⊥ # *A* ⊢$_S$ *B*›
| *FlsR* [*elim*]: ‹*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*: ‹*A* ⊢$_S$ [*p*] ⟹ *p* # *A* ⊢$_S$ *B* ⟹ *A* ⊢$_S$ *B*›
| *WeakL*: ‹*A* ⊢$_S$ *B* ⟹ *set A* ⊆ *set A'* ⟹ *A'* ⊢$_S$ *B*›
| *WeakR*: ‹*A* ⊢$_S$ *B* ⟹ *set B* ⊆ *set B'* ⟹ *A* ⊢$_S$ *B'*›

lemma *Boole*: ‹¬ *p* # *A* ⊢$_S$ [] ⟹ *A* ⊢$_S$ [*p*]›
  by (*meson Axiom Cut ImpL ImpR WeakR set-subset-Cons*)

5.4 Soundness

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

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

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

5.5 Maximal Consistent Sets

definition *consistent* :: ‹$'p\ \mathit{fm}\ \mathit{set} \Rightarrow \mathit{bool}$› **where**
‹$\mathit{consistent}\ S \equiv \forall\, A.\ \mathit{set}\ A \subseteq S \longrightarrow \neg\ A \vdash_S [\bot]$›

interpretation *MCS-No-Witness-UNIV consistent*
proof
show ‹$\mathit{infinite}\ (\mathit{UNIV} :: 'p\ \mathit{fm}\ \mathit{set})$›
using *infinite-UNIV-size*[*of* ‹$\lambda p.\ p \longrightarrow p$›] **by** *simp*
qed (*auto simp*: *consistent-def*)

interpretation *Derivations-Cut-MCS consistent* ‹$\lambda A\ p.\ A \vdash_S [p]$›
proof
fix *A B* **and** *p* :: ‹$'p\ \mathit{fm}$›
assume ‹$A \vdash_S [p]$› ‹$\mathit{set}\ A = \mathit{set}\ B$›
then show ‹$B \vdash_S [p]$›
using *WeakL* **by** *blast*
next
fix *S* :: ‹$'p\ \mathit{fm}\ \mathit{set}$›
show ‹$\mathit{consistent}\ S \longleftrightarrow (\forall\, A.\ \mathit{set}\ A \subseteq S \longrightarrow (\exists\, q.\ \neg\ A \vdash_S [q]))$›
unfolding *consistent-def* **using** *Cut FlsL* **by** *blast*
next
fix *A* **and** *p* :: ‹$'p\ \mathit{fm}$›
assume ‹$p \in \mathit{set}\ A$›
then show ‹$A \vdash_S [p]$›
by (*metis Axiom WeakL set-ConsD subsetI*)
next
fix *A B* **and** *p q* :: ‹$'p\ \mathit{fm}$›
assume ‹$A \vdash_S [p]$› ‹$p\ \#\ B \vdash_S [q]$›
then have ‹$A\ @\ B \vdash_S [p]$› ‹$p\ \#\ A\ @\ B \vdash_S [q]$›
by (*fastforce intro*: *WeakL*)+
then show ‹$A\ @\ B \vdash_S [q]$›
using *Cut* **by** *blast*
qed

interpretation *Derivations-Bot consistent* ‹$\lambda A\ p.\ A \vdash_S [p]$› ‹$\bot$›
proof
show ‹$\bigwedge A\ r.\ A \vdash_S [\bot] \Longrightarrow A \vdash_S [r]$›
using *Cut FlsL* **by** *blast*
qed

interpretation *Derivations-Imp consistent* ‹$\lambda A\ p.\ A \vdash_S [p]$› ‹$\lambda p\ q.\ p \longrightarrow q$›
proof
show ‹$\bigwedge A\ p\ q.\ A \vdash_S [p] \Longrightarrow A \vdash_S [p \longrightarrow q] \Longrightarrow A \vdash_S [q]$›

by (*meson Axiom Cut ImpL*)
qed *fast+*

5.6 Truth Lemma

abbreviation *canonical* :: ‹*'p fm set* ⇒ *'p model*› (‹$\mathcal{M}_S$›) **where**
‹$\mathcal{M}_S(S) \equiv \lambda P.\ \cdot P \in S$›

fun *semics* :: ‹*'p model* ⇒ (*'p model* ⇒ *'p fm* ⇒ *bool*) ⇒ *'p fm* ⇒ *bool*›
(‹- ⟦-⟧$_S$ -› [*55*, *0*, *55*] *55*) **where**
‹- ⟦-⟧$_S$ ⊥ ⟷ *False*›
| ‹*I* ⟦-⟧$_S$ $\cdot P$ ⟷ *I P*›
| ‹*I* ⟦$\mathcal{R}$⟧$_S$ *p* ⟶ *q* ⟷ $\mathcal{R}$ *I p* ⟶ $\mathcal{R}$ *I q*›

fun *rel* :: ‹*'p fm set* ⇒ *'p model* ⇒ *'p fm* ⇒ *bool*› (‹$\mathcal{R}_S$›) **where**
‹$\mathcal{R}_S(S)$ - *p* ⟷ *p* ∈ *S*›

theorem *saturated-model*:
assumes ‹$\bigwedge p.\ \forall M{\in}\{\mathcal{M}_S(S)\}.\ M$ ⟦$\mathcal{R}_S(S)$⟧$_S$ *p* = $\mathcal{R}_S(S)$ *M p*› ‹*M* ∈ {$\mathcal{M}_S(S)$}›
shows ‹$\mathcal{R}_S(S)$ *M p* ⟷ *M* ⊨$_S$ *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*(*1*)[*of x*] *assms*(*2*) **by** (*cases x*) *simp-all*
qed

theorem *saturated-MCS*:
assumes ‹*MCS S*› ‹*M* ∈ {$\mathcal{M}_S(S)$}›
shows ‹*M* ⟦$\mathcal{R}_S(S)$⟧$_S$ *p* ⟷ $\mathcal{R}_S(S)$ *M p*›
using *assms* **by** (*cases p*) *auto*

interpretation *Truth-No-Witness semics semantics* ‹λ*S*. {$\mathcal{M}_S(S)$}› *rel consistent*
proof
fix *p* **and** *M* :: ‹*'p model*›
show ‹(*M* ⊨$_S$ *p*) = *M* ⟦(⊨$_S$)⟧$_S$ *p*›
by (*induct p*) *auto*
qed (*use saturated-model saturated-MCS* **in** *blast*)+

5.7 Completeness

theorem *strong-completeness*:
assumes ‹∀ *M*. (∀ *q* ∈ *X*. *M* ⊨$_S$ *q*) ⟶ *M* ⊨$_S$ *p*›
shows ‹∃ *A*. *set A* ⊆ *X* ∧ *A* ⊢$_S$ [*p*]›
proof (*rule ccontr*)
assume ‹∄ *A*. *set A* ⊆ *X* ∧ *A* ⊢$_S$ [*p*]›
then have ∗: ‹∀ *A*. *set A* ⊆ {¬ *p*} ∪ *X* ⟶ ¬ *A* ⊢$_S$ [⊥]›
using *derive-split1 botE Boole FlsR* **by** (*metis* (*full-types*) *insert-is-Un subset-insert-iff*)

let *?X* = ‹{¬ *p*} ∪ *X*›
let *?S* = ‹*Extend ?X*›

have ‹*consistent ?X*›

```
    unfolding consistent-def using * .
  then have ‹MCS ?S›
    using MCS-Extend' by blast
  then have ‹p ∈ ?S ⟷ 𝓜S ?S ⊨S p› for p
    using truth-lemma by fastforce
  then have ‹p ∈ ?X ⟶ 𝓜S ?S ⊨S p› for p
    using Extend-subset by blast
  then have ‹𝓜S ?S ⊨S ¬ p› ‹∀ q ∈ X. 𝓜S ?S ⊨S q›
    by blast+
  moreover from this have ‹𝓜S ?S ⊨S p›
    using assms(1) by blast
  ultimately show False
    by simp
qed

abbreviation valid :: ‹'p fm ⇒ bool› where
  ‹valid p ≡ ∀ M. M ⊨S 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* (‹$\bot$›)
  | *Pro* $'p$ (‹•›)
  | *Imp* ‹$('i, 'p)$ *fm*› ‹$('i, 'p)$ *fm*› (**infixr** ‹$\longrightarrow$› *55*)
  | *Box* $'i$ ‹$('i, 'p)$ *fm*› (‹$\Box$›)

abbreviation *Neg* (‹$\neg$ -› [*70*] *70*) **where**
  ‹$\neg\ p \equiv p \longrightarrow \bot$›

6.2 Semantics

datatype $('i, 'p, 'w)$ *model* =
  Model (*W*: ‹$'w$ *set*›) (*R*: ‹$'i \Rightarrow 'w \Rightarrow 'w$ *set*›) (*V*: ‹$'w \Rightarrow 'p \Rightarrow$ *bool*›)

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

fun *semantics* :: ‹$('i, 'p, 'w)$ *ctx* $\Rightarrow$ $('i, 'p)$ *fm* $\Rightarrow$ *bool*› (**infix** ‹$\models_\Box$› *50*) **where**
  ‹- $\models_\Box \bot \longleftrightarrow$ *False*›
| ‹$(M, w) \models_\Box \bullet P \longleftrightarrow V\ M\ w\ P$›
| ‹$(M, w) \models_\Box p \longrightarrow q \longleftrightarrow (M, w) \models_\Box p \longrightarrow (M, w) \models_\Box q$›
| ‹$(M, w) \models_\Box \Box\ i\ p \longleftrightarrow (\forall v \in W\ M \cap R\ M\ i\ w.\ (M, v) \models_\Box p)$›

6.3 Calculus

primrec *eval* :: ‹$('p \Rightarrow$ *bool*$) \Rightarrow (('i, 'p)$ *fm* $\Rightarrow$ *bool*$) \Rightarrow ('i, 'p)$ *fm* $\Rightarrow$ *bool*› **where**
  ‹*eval* - - $\bot$ = *False*›
| ‹*eval* *g* - $(\bullet P) = g\ P$›
| ‹*eval* *g* *h* $(p \longrightarrow q) =$ (*eval* *g* *h* *p* $\longrightarrow$ *eval* *g* *h* *q*)›
| ‹*eval* - *h* $(\Box\ i\ p) = h\ (\Box\ i\ p)$›

abbreviation ‹*tautology* $p \equiv \forall g\ h.$ *eval* *g* *h* *p*›

inductive *Calculus* :: ‹$('i, 'p)$ *fm* $\Rightarrow$ *bool*› (‹$\vdash_\Box$ -› [*50*] *50*) **where**
  A1: ‹*tautology* $p \Longrightarrow \vdash_\Box p$›
| *A2*: ‹$\vdash_\Box \Box\ i\ (p \longrightarrow q) \longrightarrow \Box\ i\ p \longrightarrow \Box\ i\ q$›

| *R1*: ‹$\vdash_\Box p \Longrightarrow \vdash_\Box p \longrightarrow q \Longrightarrow \vdash_\Box q$›
| *R2*: ‹$\vdash_\Box p \Longrightarrow \vdash_\Box \Box\, i\, p$›

primrec *imply* :: ‹$('i, 'p)\ fm\ list \Rightarrow ('i, 'p)\ fm \Rightarrow ('i, 'p)\ fm$› (**infixr** ‹$\rightsquigarrow$› *56*) **where**
‹$([] \rightsquigarrow p) = p$›
| ‹$(q \# A \rightsquigarrow p) = (q \longrightarrow A \rightsquigarrow p)$›

abbreviation *Calculus-assms* (**infix** ‹$\vdash_\Box$› *50*) **where**
‹$A \vdash_\Box p \equiv \vdash_\Box A \rightsquigarrow p$›

6.4 Soundness

lemma *eval-semantics*: ‹$eval\ (g\ w)\ (\lambda q.\ (Model\ Ws\ r\ g,\ w) \models_\Box q)\ p = ((Model\ Ws\ r\ g,\ w) \models_\Box p)$›
by (*induct p*) *simp-all*

lemma *tautology*:
assumes ‹*tautology p*›
shows ‹$(M,\ w) \models_\Box p$›
proof −
from *assms* **have** ‹$eval\ (g\ w)\ (\lambda q.\ (Model\ Ws\ r\ g,\ w) \models_\Box q)\ p$› **for** *Ws g r*
by *simp*
then have ‹$(Model\ Ws\ r\ g,\ w) \models_\Box p$› **for** *Ws g r*
using *eval-semantics* **by** *fast*
then show ‹$(M,\ w) \models_\Box p$›
by (*metis model.exhaust*)
qed

theorem *soundness*: ‹$\vdash_\Box p \Longrightarrow w \in W\ M \Longrightarrow (M,\ w) \models_\Box p$›
by (*induct p arbitrary*: *w rule*: *Calculus.induct*) (*auto simp*: *tautology*)

6.5 Admissible rules

lemma *K-imply-head*: ‹$p \# A \vdash_\Box p$›
proof −
have ‹$tautology\ (p \# A \rightsquigarrow p)$›
by (*induct A*) *simp-all*
then show *?thesis*
using *A1* **by** *blast*
qed

lemma *K-imply-Cons*:
assumes ‹$A \vdash_\Box q$›
shows ‹$p \# A \vdash_\Box q$›
using *assms* **by** (*auto simp*: *A1 intro*: *R1*)

lemma *K-right-mp*:
assumes ‹$A \vdash_\Box p$› ‹$A \vdash_\Box p \longrightarrow q$›
shows ‹$A \vdash_\Box q$›
proof −
have ‹$tautology\ (A \rightsquigarrow p \longrightarrow A \rightsquigarrow (p \longrightarrow q) \longrightarrow A \rightsquigarrow q)$›
by (*induct A*) *simp-all*
with *A1* **have** ‹$\vdash_\Box A \rightsquigarrow p \longrightarrow A \rightsquigarrow (p \longrightarrow q) \longrightarrow A \rightsquigarrow q$› **.**
then show *?thesis*
using *assms R1* **by** *blast*
qed

lemma *deduct1*: ‹$A \vdash_\Box p \longrightarrow q \Longrightarrow p \# A \vdash_\Box q$›
by (*meson K-right-mp K-imply-Cons K-imply-head*)

lemma *imply-append* [*iff*]: ‹$(A @ B \rightsquigarrow r) = (A \rightsquigarrow B \rightsquigarrow r)$›
by (*induct A*) *simp-all*

lemma *imply-swap-append*: ‹$A @ B \vdash_\Box r \Longrightarrow B @ A \vdash_\Box r$›
proof (*induct B arbitrary*: *A*)
case *Cons*
then show *?case*
by (*metis deduct1 imply.simps*(*2*) *imply-append*)
qed *simp*

lemma *K-ImpI*: ‹$p \# A \vdash_\Box q \Longrightarrow A \vdash_\Box p \longrightarrow q$›
by (*metis imply.simps imply-append imply-swap-append*)

lemma *imply-mem* [*simp*]: ‹$p \in set\ A \Longrightarrow A \vdash_\Box p$›
using *K-imply-head K-imply-Cons* **by** (*induct A*) *fastforce+*

lemma *add-imply* [*simp*]: ‹$\vdash_\Box q \Longrightarrow A \vdash_\Box q$›
using *K-imply-head R1* **by** *auto*

lemma *K-imply-weaken*: ‹$A \vdash_\Box q \Longrightarrow set\ A \subseteq set\ A' \Longrightarrow A' \vdash_\Box q$›
by (*induct A arbitrary*: *q*) (*simp*, *metis K-right-mp K-ImpI imply-mem insert-subset list.set*(*2*))

lemma *K-Boole*:
assumes ‹$(\neg p) \# A \vdash_\Box \bot$›
shows ‹$A \vdash_\Box p$›
proof −
have ‹$A \vdash_\Box \neg \neg p$›
using *assms K-ImpI* **by** *blast*
moreover have ‹*tautology* $(A \rightsquigarrow \neg \neg p \longrightarrow A \rightsquigarrow p)$›
by (*induct A*) *simp-all*
then have ‹$\vdash_\Box (A \rightsquigarrow \neg \neg p \longrightarrow A \rightsquigarrow p)$›
using *A1* **by** *blast*
ultimately show *?thesis*
using *R1* **by** *blast*
qed

lemma *K-distrib-K-imp*:
assumes ‹$\vdash_\Box \Box\ i\ (A \rightsquigarrow q)$›
shows ‹*map* $(\Box\ i)\ A \vdash_\Box \Box\ i\ q$›
proof −
have ‹$\vdash_\Box \Box\ i\ (A \rightsquigarrow q) \longrightarrow map\ (\Box\ i)\ A \rightsquigarrow \Box\ i\ q$›
proof (*induct A*)
case *Nil*
then show *?case*
by (*simp add*: *A1*)
next
case (*Cons a A*)
have ‹$\vdash_\Box \Box\ i\ (a \# A \rightsquigarrow q) \longrightarrow \Box\ i\ a \longrightarrow \Box\ i\ (A \rightsquigarrow q)$›
by (*simp add*: *A2*)
moreover have
‹$\vdash_\Box ((\Box\ i\ (a \# A \rightsquigarrow q) \longrightarrow \Box\ i\ a \longrightarrow \Box\ i\ (A \rightsquigarrow q)) \longrightarrow$
$(\Box\ i\ (A \rightsquigarrow q) \longrightarrow map\ (\Box\ i)\ A \rightsquigarrow \Box\ i\ q) \longrightarrow$

```
      (□ i (a # A ⇝ q) ⟶ □ i a ⟶ map (□ i) A ⇝ □ i q))›
    by (simp add: A1)
  ultimately have ‹⊢□ □ i (a # A ⇝ q) ⟶ □ i a ⟶ map (□ i) A ⇝ □ i q›
    using Cons R1 by blast
  then show ?case
    by simp
 qed
 then show ?thesis
  using assms R1 by blast
qed
```

6.6 Maximal Consistent Sets

```
definition consistent :: ‹('i, 'p) fm set ⇒ bool› where
 ‹consistent S ≡ ∀ A. set A ⊆ S ⟶ ¬ A ⊢□ ⊥›

interpretation MCS-No-Witness-UNIV consistent
proof
 show ‹infinite (UNIV :: ('i, 'p) fm set)›
  using infinite-UNIV-size[of ‹λp. p ⟶ p›] by simp
qed (auto simp: consistent-def)

interpretation Derivations-Cut-MCS consistent Calculus-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
next
 fix S :: ‹('i, 'p) fm set›
 show ‹consistent S = (∀ A. set A ⊆ S ⟶ (∃ q. ¬ A ⊢□ q))›
  unfolding consistent-def using K-Boole K-imply-Cons by blast
next
 fix A B and p q :: ‹('i, 'p) fm›
 assume ‹A ⊢□ p› ‹p # B ⊢□ q›
 then show ‹A @ B ⊢□ q›
  by (metis K-right-mp add-imply imply.simps(2) imply-append)
qed simp

interpretation Derivations-Bot consistent Calculus-assms ‹⊥›
proof
 show ‹⋀A r. A ⊢□ ⊥ ⟹ A ⊢□ r›
  using K-Boole K-imply-Cons by blast
qed

interpretation Derivations-Imp consistent Calculus-assms ‹λp q. p ⟶ q›
proof
 show ‹⋀A p q. p # A ⊢□ q ⟹ A ⊢□ p ⟶ q›
  using K-ImpI by blast
 show ‹⋀A p q. A ⊢□ p ⟹ A ⊢□ p ⟶ q ⟹ A ⊢□ q›
  using K-right-mp by blast
qed

theorem deriv-in-maximal:
 assumes ‹consistent S› ‹maximal S› ‹⊢□ p›
```

shows ‹$p \in S$›
using *assms MCS-derive* **by** *fastforce*

6.7 Truth Lemma

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

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

abbreviation *canonical* :: ‹('*i*, '*p*) *fm set* ⇒ ('*i*, '*p*, ('*i*, '*p*) *fm set*) *ctx*› (‹$\mathcal{M}_\Box$›) **where**
‹$\mathcal{M}_\Box(S)$ ≡ (*Model* {*S*. *MCS S*} *reach* (λ*S P*. ·*P* ∈ *S*), *S*)›

fun *semics* ::
‹('*i*, '*p*, '*w*) *ctx* ⇒ (('*i*, '*p*, '*w*) *ctx* ⇒ ('*i*, '*p*) *fm* ⇒ *bool*) ⇒ ('*i*, '*p*) *fm* ⇒ *bool*›
(‹(- ⟦-⟧□ -)› [*55*, *0*, *55*] *55*) **where**
‹- ⟦-⟧□ ⊥ ⟷ *False*›
| ‹(*M*, *w*) ⟦-⟧□ ·*P* ⟷ *V M w P*›
| ‹(*M*, *w*) ⟦$\mathcal{R}$⟧□ *p* ⟶ *q* ⟷ $\mathcal{R}$ (*M*, *w*) *p* ⟶ $\mathcal{R}$ (*M*, *w*) *q*›
| ‹(*M*, *w*) ⟦$\mathcal{R}$⟧□ □ *i p* ⟷ (∀ *v* ∈ *W M* ∩ *R M i w*. $\mathcal{R}$ (*M*, *v*) *p*)›

fun *rel* :: ‹('*i*, '*p*) *fm set* ⇒ ('*i*, '*p*, ('*i*, '*p*) *fm set*) *ctx* ⇒ ('*i*, '*p*) *fm* ⇒ *bool*› (‹$\mathcal{R}_\Box$›) **where**
‹$\mathcal{R}_\Box$(-) (-, *w*) *p* ⟷ *p* ∈ *w*›

theorem *saturated-model*:
fixes *S* :: ‹('*i*, '*p*) *fm set*›
assumes ‹⋀(*S* :: ('*i*, '*p*) *fm set*) *p*. *MCS S* ⟹ $\mathcal{M}_\Box(S)$ ⟦$\mathcal{R}_\Box(S')$⟧□ *p* ⟷ *p* ∈ *S*›
shows ‹*MCS S* ⟹ $\mathcal{M}_\Box(S)$ ⊨□ *p* ⟷ *p* ∈ *S*›
proof (*induct p arbitrary*: *S rule*: *wf-induct*[**where** *r*=‹*measure size*›])
case *1*
then show *?case* **..**
next
case (*2 x*)
then show *?case*
using *assms*[*of S x*] **by** (*cases x*) *auto*
qed

theorem *saturated-MCS*:
assumes ‹*MCS S*›
shows ‹$\mathcal{M}_\Box(S)$ ⟦$\mathcal{R}_\Box(S')$⟧□ *p* ⟷ $\mathcal{R}_\Box(S')$ ($\mathcal{M}_\Box(S)$) *p*›
proof (*cases p*)
case *Fls*
have ‹⊥ ∉ *S*›
using *assms MCS-derive* **unfolding** *consistent-def* **by** *blast*
then show *?thesis*
using *Fls* **by** *simp*
next
case (*Imp p q*)
then show *?thesis*
using *assms* **by** *auto*
next
case (*Box i p*)
have ‹(∀ *S'* ∈ *reach i S*. *p* ∈ *S'*) ⟷ □ *i p* ∈ *S*›
proof

```
    assume ‹□ i p ∈ S›
    then show ‹∀ S' ∈ reach i S. p ∈ S'›
      by auto
  next
    assume ∗: ‹∀ S' ∈ reach i S. p ∈ S'›
    have ‹¬ consistent ({¬ p} ∪ known S i)›
    proof
      assume ‹consistent ({¬ p} ∪ known S i)›
      then obtain S' where S': ‹{¬ p} ∪ known S i ⊆ S'› ‹MCS S'›
        using ‹MCS S› MCS-Extend' Extend-subset by metis
      then show False
        using ∗ MCS-impE MCS-bot by force
    qed
    then obtain A where A: ‹¬ p # A ⊢□ ⊥› ‹set A ⊆ known S i›
      unfolding consistent-def using derive-split1 K-imply-Cons
      by (metis (no-types, lifting) insert-is-Un subset-insert)
    then have ‹⊢□ A ⇝ p›
      using K-Boole by blast
    then have ‹⊢□ □ i (A ⇝ p)›
      using R2 by fast
    then have ‹map (□ i) A ⊢□ □ i p›
      using K-distrib-K-imp by fast
    then have ‹(map (□ i) A ⇝ □ i p) ∈ S›
      using deriv-in-maximal ‹MCS S› by blast
    then show ‹□ i p ∈ S›
      using A(2)
    proof (induct A)
      case (Cons a L)
      then have ‹□ i a ∈ S›
        by auto
      then have ‹(map (□ i) L ⇝ □ i p) ∈ S›
        using Cons(2) ‹MCS S› MCS-impE by auto
      then show ?case
        using Cons by simp
    qed simp
  qed
  then show ?thesis
    using Box by auto
qed simp

interpretation Truth-No-Witness semics semantics ‹λ-. {M□(S) |S. MCS S}› rel consistent
proof
  fix p and M :: ‹('i, 'p, ('i, 'p) fm set) ctx›
  show ‹(M ⊨□ p) = M ⟦semantics⟧□ p›
    by (cases M, induct p) simp-all
next
  fix p and S :: ‹('i, 'p) fm set› and M :: ‹('i, 'p, ('i, 'p) fm set) ctx›
  assume ‹∀ p. ∀ M∈{M□(S) |S. MCS S}. M ⟦R□(S)⟧□ p ⟷ R□(S) M p› ‹M ∈ {M□(S) |S. MCS
S}›
  then show ‹M ⊨□ p ⟷ R□(S) M p›
    using saturated-model[of S - p] by auto
next
  fix S :: ‹('i, 'p) fm set› and M :: ‹('i, 'p, ('i, 'p) fm set) ctx›
  assume ‹MCS S›
  then show ‹∀ p. ∀ M∈{M□(S) |S. MCS S}. M ⟦R□(S)⟧□ p ⟷ R□(S) M p›
    using saturated-MCS by blast
```

qed

lemma *Truth-lemma*:
assumes ‹*MCS S*›
shows ‹$\mathcal{M}_\Box(S) \models_\Box p \longleftrightarrow p \in S$›
using *assms truth-lemma* **by** *fastforce*

6.8 Completeness

theorem *strong-completeness*:
assumes ‹$\forall\, M :: ('i, 'p, ('i, 'p)\ fm\ set)\ model.\ \forall\, w \in W\ M.$
$(\forall\, q \in X.\ (M, w) \models_\Box q) \longrightarrow (M, w) \models_\Box p$›
shows ‹$\exists\, A.\ set\ A \subseteq X \wedge A \vdash_\Box p$›
proof (*rule ccontr*)
assume ‹$\nexists\, A.\ set\ A \subseteq X \wedge A \vdash_\Box p$›
then have $*$: ‹$\forall\, A.\ set\ A \subseteq \{\neg\ p\} \cup X \longrightarrow \neg\ A \vdash_\Box \bot$›
using *K-Boole botE* **by** (*metis derive-split1 insert-is-Un subset-insert*)

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

have ‹*consistent ?X*›
using $*$ **unfolding** *consistent-def* **.**
then have ‹*MCS ?S*›
using *MCS-Extend′* **by** *blast*
moreover have ‹$\neg\ p \in$ *?S*› ‹$X \subseteq$ *?S*›
using *Extend-subset* **by** *fast+*
ultimately have ‹$\mathcal{M}_\Box$ *?S* $\models_\Box (\neg\ p)$› ‹$\forall\, q \in X.\ \mathcal{M}_\Box$ *?S* $\models_\Box q$›
using *assms Truth-lemma* **by** *fast+*
then have ‹$\mathcal{M}_\Box$ *?S* $\models_\Box p$›
using *assms* ‹*MCS ?S*› **by** *simp*
then show *False*
using ‹$\mathcal{M}_\Box$ *?S* $\models_\Box (\neg\ p)$› **by** *simp*
qed

abbreviation *valid* :: ‹$('i, 'p)\ fm \Rightarrow bool$› **where**
‹$valid\ p \equiv \forall\, (M :: ('i, 'p, ('i, 'p)\ fm\ set)\ model).\ \forall\, w \in W\ M.\ (M, w) \models_\Box p$›

corollary *completeness*: ‹$valid\ p \Longrightarrow \vdash_\Box p$›
using *strong-completeness*[**where** X=‹$\{\}$›] **by** *simp*

theorem *main*: ‹$valid\ p \longleftrightarrow \vdash_\Box p$›
using *soundness completeness* **by** *meson*

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*› (‹@›)
| *All* ‹(*'i*, *'p*) *fm*› (‹**A**›)

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

abbreviation *Con* (**infixr** ‹∧› *35*) **where** ‹*p* ∧ *q* ≡ ¬ (*p* ⟶ ¬ *q*)›

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

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

7.2 Semantics

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

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

fun *semantics* :: ‹(*'i*, *'p*, *'w*) *ctx* ⇒ (*'i*, *'p*) *fm* ⇒ *bool*› (**infix** ‹⊨@› *50*) **where**
‹(*M*, *g*, *w*) ⊨@ ⊥ ⟷ *False*›

| ‹(M, -, w) ⊨@ •P ⟷ V M w P›
| ‹(-, g, w) ⊨@ •i ⟷ w = g i›
| ‹(M, g, w) ⊨@ p ⟶ q ⟷ (M, g, w) ⊨@ p ⟶ (M, g, w) ⊨@ q›
| ‹(M, g, w) ⊨@ ◇ p ⟷ (∃ v ∈ W M ∩ R M w. (M, g, v) ⊨@ p)›
| ‹(M, g, -) ⊨@ @i p ⟷ (M, g, g i) ⊨@ p›
| ‹(M, g, -) ⊨@ **A** p ⟷ (∀ v ∈ W M. (M, g, v) ⊨@ p)›

lemma *semantics-fresh*: ‹i ∉ nominals-fm p ⟹ (M, g, w) ⊨@ p ⟷ (M, g(i := v), w) ⊨@ p›
by (*induct p arbitrary*: *w*) *auto*

lemma *semantics-fresh-lbd*:
‹k ∉ nominals-lbd (i, p) ⟹ (M, g, w) ⊨@ p ⟷ (M, g(k := v), w) ⊨@ p›
by (*induct p arbitrary*: *w*) *auto*

7.3 Calculus

inductive *Calculus* :: ‹('i, 'p) lbd list ⇒ ('i, 'p) lbd ⇒ bool› (**infix** ‹⊢@› 50) **where**
Assm [*simp*]: ‹(i, p) ∈ set A ⟹ A ⊢@ (i, p)›
| *Ref* [*simp*]: ‹A ⊢@ (i, •i)›
| *Nom* [*dest*]: ‹A ⊢@ (i, •k) ⟹ A ⊢@ (i, p) ⟹ A ⊢@ (k, p)›
| *FlsE* [*elim*]: ‹A ⊢@ (i, ⊥) ⟹ A ⊢@ (k, p)›
| *ImpI* [*intro*]: ‹(i, p) # A ⊢@ (i, q) ⟹ A ⊢@ (i, p ⟶ q)›
| *ImpE* [*dest*]: ‹A ⊢@ (i, p ⟶ q) ⟹ A ⊢@ (i, p) ⟹ A ⊢@ (i, q)›
| *SatI* [*intro*]: ‹A ⊢@ (i, p) ⟹ A ⊢@ (k, @i p)›
| *SatE* [*dest*]: ‹A ⊢@ (i, @k p) ⟹ A ⊢@ (k, p)›
| *DiaI* [*intro*]: ‹A ⊢@ (i, ◇ (•k)) ⟹ A ⊢@ (k, p) ⟹ A ⊢@ (i, ◇ p)›
| *DiaE* [*elim*]: ‹A ⊢@ (i, ◇ p) ⟹ k ∉ nominals ({(i, p), (j, q)} ∪ set A) ⟹
(k, p) # (i, ◇ (•k)) # A ⊢@ (j, q) ⟹ A ⊢@ (j, q)›
| *AllI* [*intro*]: ‹A ⊢@ (k, p) ⟹ k ∉ nominals ({(i, p)} ∪ set A) ⟹ A ⊢@ (i, **A** p)›
| *AllE* [*dest*]: ‹A ⊢@ (i, **A** p) ⟹ A ⊢@ (k, p)›
| *Clas*: ‹(i, p ⟶ q) # A ⊢@ (i, p) ⟹ A ⊢@ (i, p)›
| *Cut*: ‹A ⊢@ (k, q) ⟹ (k, q) # B ⊢@ (i, p) ⟹ A @ B ⊢@ (i, p)›

7.4 Soundness

theorem *soundness*: ‹A ⊢@ (i, p) ⟹ *list-all* (λ(i, p). (M, g, g i) ⊨@ p) A ⟹ range g ⊆ W M ⟹
(M, g, g i) ⊨@ p›
proof (*induct* ‹(i, p)› *arbitrary*: *i p g rule*: *Calculus.induct*)
case (*Nom A i k p*)
then show *?case*
by (*metis semantics.simps*(*3*))
next
case (*DiaE A i p k j q*)
then have ‹(M, g, g i) ⊨@ ◇ p›
by *blast*
then obtain *v* **where** *v*: ‹v ∈ W M ∩ R M (g i)› ‹(M, g, v) ⊨@ p›
by *auto*
let *?g* = ‹g(k := v)›
have ‹(M, *?g*, *?g* k) ⊨@ p› ‹(M, *?g*, *?g* i) ⊨@ ◇ (•k)›
using *v fun-upd-same DiaE*(*3*) *semantics-fresh-lbd* **by** *fastforce+*
moreover have ‹*list-all* (λ(i, p). (M, *?g*, *?g* i) ⊨@ p) A›
using *DiaE.prems*(*1*) *DiaE.hyps*(*3*) *semantics-fresh-lbd* **by** (*fastforce simp*: *list-all-iff*)
ultimately have ‹*list-all* (λ(i, p). (M, *?g*, *?g* i) ⊨@ p) ((k, p) # (i, ◇ (•k)) # A)›
by *simp*
moreover have ‹range *?g* ⊆ W M›

```
    using DiaE.prems v by auto
  ultimately have ‹(M, ?g, ?g j) ⊨@ q›
    using DiaE.hyps by blast
  then show ?case
    using DiaE.hyps(3) semantics-fresh-lbd by fastforce
next
  case (AllI A k p i)
  {
    fix v
    assume ‹v ∈ W M›
    let ?g = ‹g(k := v)›
    have ‹∀ v. list-all (λ(i, p). (M, ?g, ?g i) ⊨@ p) A›
      using AllI.prems(1) AllI.hyps(3) semantics-fresh-lbd by (fastforce simp: list-all-iff)
    moreover have ‹range ?g ⊆ W M›
      using AllI.prems ‹v ∈ W M› by auto
    ultimately have ‹(M, ?g, ?g k) ⊨@ p›
      using AllI.hyps by fast
  }
  then have ‹∀ v ∈ W M. (M, g(k := v), v) ⊨@ p›
    by simp
  then have ‹∀ v ∈ W M. (M, g, v) ⊨@ p›
    using AllI.hyps(3) semantics-fresh-lbd by fast
  then show ?case
    by simp
next
  case (AllE A i p k)
  then show ?case
    by fastforce
qed (auto simp: list-all-iff)

corollary soundness':
  assumes ‹[] ⊢@ (i, p)› ‹i ∉ nominals-fm p›
    and ‹range g ⊆ W M› ‹w ∈ W M›
  shows ‹(M, g, w) ⊨@ p›
proof −
  let ?g = ‹g(i := w)›
  have ‹range ?g ⊆ W M›
    using assms(3−4) by auto
  then have ‹(M, ?g, ?g i) ⊨@ p›
    using assms(1, 4) soundness by (metis list-all-simps(2))
  then have ‹(M, ?g, w) ⊨@ p›
    by simp
  then show ?thesis
    using assms(2) semantics-fresh by fast
qed

corollary ‹¬ ([] ⊢@ (i, ⊥))›
  by (metis list.pred-inject(1) model.sel(1) semantics.simps(1) soundness subset-refl)
```

7.5 Admissible Rules

```
lemma Assm-head [simp]: ‹(p, i) # A ⊢@ (p, i)›
  by auto

lemma SatE':
```

assumes ‹$(k,\ q)\ \#\ A \vdash_@ (i,\ p)$›
shows ‹$(j,\ @k\ q)\ \#\ A \vdash_@ (i,\ p)$›
proof −
have ‹$[(j,\ @k\ q)] \vdash_@ (k,\ q)$›
by (*meson Assm-head SatE*)
then show *?thesis*
using *assms* **by** (*auto dest*: *Cut*)
qed

lemma *ImpI′*:
assumes ‹$(k,\ q)\ \#\ A \vdash_@ (i,\ p)$›
shows ‹$A \vdash_@ (i,\ (@k\ q) \longrightarrow p)$›
using *assms SatE′* **by** *fast*

lemma *Weak′*: ‹$A \vdash_@ (i,\ p) \Longrightarrow A\ @\ B \vdash_@ (i,\ p)$›
by (*simp add*: *Cut*)

lemma *Weaken*: ‹$A \vdash_@ (i,\ p) \Longrightarrow set\ A \subseteq set\ B \Longrightarrow B \vdash_@ (i,\ p)$›
proof (*induct A arbitrary*: *p*)
case *Nil*
then show *?case*
by (*metis Weak′ append-Nil*)
next
case (*Cons kq A*)
then show *?case*
proof (*cases kq*)
case (*Pair k q*)
then have ‹$B \vdash_@ (i,\ @k\ q \longrightarrow p)$›
using *Cons* **by** (*simp add*: *ImpI′*)
then show *?thesis*
using *Pair Cons*(*3*) **by** *fastforce*
qed
qed

lemma *Weak*: ‹$A \vdash_@ (i,\ p) \Longrightarrow (k,\ q)\ \#\ A \vdash_@ (i,\ p)$›
using *Weaken*[**where** *A*=*A* **and** *B*=‹$(k,\ q)\ \#\ A$›] **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*

interpretation *Derivations Calculus*
proof
fix *A* **and** *p* :: ‹(′*i*, ′*p*) *lbd*›
show ‹$p \in set\ A \Longrightarrow A \vdash_@(p)$›
by (*cases p*) *simp*
next
fix *A B* **and** *p* :: ‹(′*i*, ′*p*) *lbd*›
assume ‹$A \vdash_@(p)$› ‹$set\ A = set\ B$›
then show ‹$B \vdash_@(p)$›
by (*cases p*) (*simp add*: *Weaken*)
qed

7.6 Maximal Consistent Sets

definition *consistent* :: ‹('i, 'p) *lbd set* ⇒ *bool*› **where**
‹*consistent S* ≡ ∀ *A a*. *set A* ⊆ *S* ⟶ ¬ *A* ⊢@ (*a*, ⊥)›

lemma *consistent-add-diamond-witness*:
assumes ‹*consistent S*› ‹(*i*, ◇ *p*) ∈ *S*› ‹*k* ∉ *nominals S*›
shows ‹*consistent* ({(*k*, *p*), (*i*, ◇ (•*k*))} ∪ *S*)›
unfolding *consistent-def*
proof *safe*
fix *A a*
assume *A*: ‹*set A* ⊆ {(*k*, *p*), (*i*, ◇ (•*k*))} ∪ *S*› ‹*A* ⊢@ (*a*, ⊥)›
then obtain *A'* *a B* **where** ‹*set A'* ⊆ *S*› ‹*B* @ *A'* ⊢@ (*a*, ⊥)› ‹*set B* = {(*k*, *p*), (*i*, ◇ (•*k*))} ∩ *set A*›
using *assms derive-split*[**where** *p*=‹(*a*, ⊥)› **and** *X*=*S* **and** *B*=‹[(*k*, *p*), (*i*, ◇ (•*k*))]›]
by (*metis Int-commute empty-set list.simps*(*15*))
then have ‹(*k*, *p*) # (*i*, ◇ (•*k*)) # *A'* ⊢@ (*a*, ⊥)›
by (*auto intro*: *Weaken*)
then have ‹(*k*, *p*) # (*i*, ◇ (•*k*)) # *A'* ⊢@ (*i*, ⊥)›
by *fast*
then have ‹(*k*, *p*) # (*i*, ◇ (•*k*)) # (*i*, ◇ *p*) # *A'* ⊢@ (*i*, ⊥)›
by (*fastforce intro*: *Weaken*)
moreover have ‹*k* ∉ *nominals* ({(*i*, *p*), (*i*, ⊥)} ∪ *set* ((*i*, ◇ *p*) # *A'*))›
using ‹*set A'* ⊆ *S*› *assms*(*2*−*3*) **by** *auto*
moreover have ‹(*i*, ◇ *p*) # *A'* ⊢@ (*i*, ◇ *p*)›
by *auto*
ultimately have ‹(*i*, ◇ *p*) # *A'* ⊢@ (*i*, ⊥)›
by *fast*
moreover have ‹*set* ((*i*, ◇ *p*) # *A'*) ⊆ *S*›
using ‹*set A'* ⊆ *S*› *assms*(*2*) **by** *simp*
ultimately show *False*
using *assms*(*1*) **unfolding** *consistent-def* **by** *blast*
qed

lemma *consistent-add-global-witness*:
assumes ‹*consistent S*› ‹(*i*, ¬ **A** *p*) ∈ *S*› ‹*k* ∉ *nominals S*›
shows ‹*consistent* ({(*k*, ¬ *p*)} ∪ *S*)›
unfolding *consistent-def*
proof *safe*
fix *A a*
assume ‹*set A* ⊆ {(*k*, ¬ *p*)} ∪ *S* › ‹*A* ⊢@ (*a*, ⊥)›
then obtain *A'* **where** ‹*set A'* ⊆ *S*› ‹(*k*, ¬ *p*) # *A'* ⊢@ (*a*, ⊥)›
using *assms derive-split1* **by** (*metis consistent-def insert-is-Un subset-insert*)
then have ‹(*k*, ¬ *p*) # *A'* ⊢@ (*k*, ⊥)›
by *fast*
then have ‹*A'* ⊢@ (*k*, *p*)›
by (*meson Boole*)
moreover have ‹*k* ∉ *nominals* ({(*i*, *p*), (*i*, ⊥)} ∪ *set* ((*i*, **A** *p*) # *A'*))›
using ‹*set A'* ⊆ *S*› *assms*(*2*−*3*) **by** *auto*
ultimately have ‹*A'* ⊢@ (*i*, **A** *p*)›
by *fastforce*
then have ‹(*i*, ¬ **A** *p*) # *A'* ⊢@ (*i*, ⊥)›
by (*meson Assm-head ImpE Weak*)
moreover have ‹*set* ((*i*, ¬ **A** *p*) # *A'*) ⊆ *S*›
using ‹*set A'* ⊆ *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*, ◇ (•*k*))})›
| ‹*witness* (*i*, ¬ **A** *p*) *S* = (*let k* = *SOME k*. *k* ∉ *nominals* ({(*i*, *p*)} ∪ *S*) *in* {(*k*, ¬ *p*)})›
| ‹*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*, ◇ (•*k*))}›
    ‹*k* ∉ *nominals* ({(*i*, ◇ *p*)} ∪ *S*)›
    by (*simp add*: *Let-def*)
  then show *?case*
    using *1*(*1*) *consistent-add-diamond-witness*[**where** *S*=‹{(*i*, ◇ *p*)} ∪ *S*›] **by** *simp*
next
  case (*2 i p S*)
  have ‹*infinite* (*UNIV* − *nominals* ({(*i*, *p*)} ∪ *S*))›
    using *2*(*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*, ¬ **A** *p*) *S* = {(*k*, ¬ *p*)}› ‹*k* ∉ *nominals* ({(*i*, ¬ **A** *p*)} ∪ *S*)›
    by (*simp add*: *Let-def*)
  then show *?case*
    using *2*(*1*) *consistent-add-global-witness*[**where** *S*=‹{(*i*, ¬ **A** *p*)} ∪ *S*›] **by** *auto*
qed (*auto simp*: *assms*)

interpretation *MCS-Witness-UNIV consistent witness nominals-lbd*
proof
  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*: *Let-def*)
next
  fix *ip* **and** *S* :: ‹('*i*, '*p*) *lbd set*›
  assume ‹*consistent* ({*ip*} ∪ *S*)› ‹*infinite* (*UNIV* − *nominals S*)›
  then show ‹*consistent* ({*ip*} ∪ *S* ∪ *witness ip S*)›
    using *consistent-witness'* **by** (*cases ip*) (*simp add*: *sup-commute*)
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 (*auto simp*: *consistent-def*)

lemma *witnessed-diamond*: ‹*witnessed S* $\Longrightarrow$ $(i, \Diamond\ p) \in S \Longrightarrow \exists k.\ (i, \Diamond\ (\cdot k)) \in S \wedge (k, p) \in S$›
unfolding *witnessed-def* **by** (*metis insert-subset witness.simps*(*1*))

lemma *witnessed-global*: ‹*witnessed S* $\Longrightarrow$ $(i, \neg\ \mathbf{A}\ p) \in S \Longrightarrow \exists k.\ (k, \neg\ p) \in S$›
unfolding *witnessed-def* **by** (*metis insert-subset witness.simps*(*2*))

interpretation *Derivations-Cut-MCS consistent Calculus*
proof
show ‹$\bigwedge S$. *consistent S* = $(\forall A.\ \mathit{set}\ A \subseteq S \longrightarrow (\exists q.\ \neg\ A \vdash_{@} (q)))$›
unfolding *consistent-def* **using** *FlsE* **by** *fast*
next
fix *A B* **and** *p q* :: ‹('*i*, '*p*) *lbd*›
assume ‹$A \vdash_{@} (p)$› ‹$p\ \#\ B \vdash_{@} (q)$›
then show ‹$A\ @\ B \vdash_{@} (q)$›
by (*cases p*, *cases q*) (*meson Cut*)
qed

interpretation *Derivations-Bot consistent Calculus* ‹$(i, \bot)$›
proof qed *auto*

interpretation *Derivations-Not consistent Calculus* ‹$(i, \bot)$› ‹$\lambda(i, p).\ (i, \neg\ p)$›
proof qed *auto*

lemma *MCS-impE'*: ‹*consistent S* $\Longrightarrow$ *maximal S* $\Longrightarrow$ $(i, p \longrightarrow q) \in S \Longrightarrow (i, p) \in S \longrightarrow (i, q) \in S$›
by (*metis MCS-derive deduct1 insert-subset list.simps*(*15*))

interpretation *Derivations-Uni consistent witness nominals-lbd Calculus* ‹$(i, \bot)$› ‹$\lambda(i, p).\ (i, \neg\ p)$›
‹$\lambda(i, p).\ (i, \mathbf{A}\ p)$› ‹$\lambda k\ (i, p).\ (k, p)$›
proof
have ‹$\bigwedge S\ S'\ i\ p.$ *MCS S* $\Longrightarrow$ *witness* $(i, \neg\ \mathbf{A}\ p)\ S' \subseteq S \Longrightarrow \exists k.\ (k, \neg\ p) \in S$›
by *auto*
then show ‹$\bigwedge S\ S'\ p.$ *MCS S* $\Longrightarrow$
witness (*case case p of* $(i, p) \Rightarrow (i, \mathbf{A}\ p)$ *of* $(i, p) \Rightarrow (i, \neg\ p))\ S' \subseteq S \Longrightarrow$
$\exists t.$ (*case case p of* $(i, p) \Rightarrow (t, p)$ *of* $(i, p) \Rightarrow (i, \neg\ p)) \in S$›
by *fast*
next
have ‹$\bigwedge A\ i\ p\ k.\ A \vdash_{@} (i, \mathbf{A}\ p) \Longrightarrow A \vdash_{@} (k, p)$›
..
then show ‹$\bigwedge A\ p\ t.\ A \vdash_{@}$ (*case p of* $(i, p) \Rightarrow (i, \mathbf{A}\ p)) \Longrightarrow A \vdash_{@}$ (*case p of* $(i, p) \Rightarrow (t, p))$›
by *fast*
qed

lemma *conE1* [*elim*]: ‹$A \vdash_{@} (i, p \wedge q) \Longrightarrow A \vdash_{@} (i, p)$›
by (*meson Clas FlsE deduct1*)

lemma *conE2* [*elim*]: ‹$A \vdash_{@} (i, p \wedge q) \Longrightarrow A \vdash_{@} (i, q)$›
by (*meson Assm-head Boole ImpE ImpI Weak*)

lemma *conI* [*intro*]: ‹$A \vdash_{@} (i, p) \Longrightarrow A \vdash_{@} (i, q) \Longrightarrow A \vdash_{@} (i, p \wedge q)$›
by (*meson Assm-head ImpE ImpI Weak*)

lemma *MCS-con*:
assumes ‹*MCS S*›
shows ‹$(i, p \wedge q) \in S \longleftrightarrow (i, p) \in S \wedge (i, q) \in S$›
using *assms MCS-derive conE1 conE2*

by (*metis Boole MCS-explode MCS-impE′ derive-assm list.set-intros*(*1*))

interpretation *Derivations-Exi consistent witness nominals-lbd Calculus*
‹$\lambda(i,\ p).\ (i,\ \Diamond\ p)$› ‹$\lambda k\ (i,\ p).\ (i,\ @k\ p \wedge \Diamond(\cdot k))$›
proof
have ‹$\bigwedge S\ S'\ i\ p.\ MCS\ S \Longrightarrow witness\ (i,\ \Diamond\ p)\ S' \subseteq S \Longrightarrow \exists k.\ (i,\ @\ k\ p \wedge \Diamond\ (\cdot\ k)) \in S$›
unfolding *witness.simps* **using** *MCS-con* **by** (*metis MCS-derive SatI insert-subset*)
then show ‹$\bigwedge S\ S'\ p.\ MCS\ S \Longrightarrow witness\ (case\ p\ of\ (i,\ p) \Rightarrow (i,\ \Diamond\ p))\ S' \subseteq S \Longrightarrow \exists t.\ (case\ p\ of\ (i,\ p) \Rightarrow (i,\ @\ t\ p \wedge \Diamond\ (\cdot\ t))) \in S$›
by *fast*
next
have ‹$\bigwedge A\ i\ k\ p.\ A \vdash_@ (i,\ @\ k\ p \wedge \Diamond\ (\cdot\ k)) \Longrightarrow A \vdash_@ (i,\ \Diamond\ p)$›
by (*metis DiaI SatE conE1 conE2*)
then show ‹$\bigwedge A\ p\ t.\ A \vdash_@ (case\ p\ of\ (i,\ p) \Rightarrow (i,\ @\ t\ p \wedge \Diamond\ (\cdot\ t))) \Longrightarrow A \vdash_@ (case\ p\ of\ (i,\ p) \Rightarrow (i,\ \Diamond\ p))$›
by *fast*
qed

corollary *MCS-uni′*:
assumes ‹*MCS S*› ‹*witnessed S*›
shows ‹$(i,\ \mathbf{A}\ p) \in S \longleftrightarrow (\forall k.\ (k,\ p) \in S)$›
using *assms MCS-uni* **by** *fastforce*

corollary *MCS-exi′*:
assumes ‹*MCS S*› ‹*witnessed S*›
shows ‹$(i,\ \Diamond\ p) \in S \longleftrightarrow (\exists k.\ (i,\ @k\ p \wedge \Diamond\ (\cdot k)) \in S)$›
using *assms MCS-exi* **by** *fastforce*

7.7 Nominals

lemma *MCS-Nom-refl*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹$(i,\ \cdot i) \in S$›
using *assms Ref* **by** (*metis MCS-derive MCS-explode*)

lemma *MCS-Nom-sym*:
assumes ‹*consistent S*› ‹*maximal S*› ‹$(i,\ \cdot k) \in S$›
shows ‹$(k,\ \cdot i) \in S$›
using *assms Nom Ref* **by** (*metis MCS-derive*)

lemma *MCS-Nom-trans*:
assumes ‹*consistent S*› ‹*maximal S*› ‹$(i,\ \cdot j) \in S$› ‹$(j,\ \cdot k) \in S$›
shows ‹$(i,\ \cdot k) \in S$›
proof −
have ‹$[(i,\ \cdot j),\ (j,\ \cdot k)] \vdash_@ (i,\ \cdot j)$› ‹$[(i,\ \cdot j),\ (j,\ \cdot k)] \vdash_@ (j,\ \cdot k)$›
by *simp-all*
then have ‹$[(i,\ \cdot j),\ (j,\ \cdot k)] \vdash_@ (i,\ \cdot k)$›
using *Nom Ref* **by** *metis*
then show *?thesis*
using *assms MCS-derive*
by (*metis bot.extremum insert-subset list.set*(*1*) *list.simps*(*15*))
qed

7.8 Truth Lemma

fun *semics* :: ‹('i, 'p, 'w) *ctx* ⇒ (('i, 'p, 'w) *ctx* ⇒ ('i, 'p) *fm* ⇒ *bool*) ⇒ ('i, 'p) *fm* ⇒ *bool*›
(‹(- ⟦-⟧@ -)› [55, 0, 55] 55) **where**
‹- ⟦-⟧@ ⊥ ⟷ *False*›
| ‹(*M*, -, *w*) ⟦-⟧@ ·*P* ⟷ *V M w P*›
| ‹(-, *g*, *w*) ⟦-⟧@ ·*i* ⟷ *w* = *g i*›
| ‹(*M*, *g*, *w*) ⟦*R*⟧@(*p*) ⟶ *q* ⟷ *R* (*M*, *g*, *w*) *p* ⟶ *R* (*M*, *g*, *w*) *q*›
| ‹(*M*, *g*, *w*) ⟦*R*⟧@ ◇ *p* ⟷ (∃ *v* ∈ *W M* ∩ *R M w*. *R* (*M*, *g*, *v*) *p*)›
| ‹(*M*, *g*, -) ⟦*R*⟧@ @*i p* ⟷ *R* (*M*, *g*, *g i*) *p*›
| ‹(*M*, *g*, -) ⟦*R*⟧@ **A** *p* ⟷ (∀ *v* ∈ *W M*. *R* (*M*, *g*, *v*) *p*)›

fun *rel* :: ‹('i, 'p) *lbd set* ⇒ ('i, 'p, 'i) *ctx* ⇒ ('i, 'p) *fm* ⇒ *bool*› (‹*R*@›) **where**
‹*R*@(*S*) (-, -, *i*) *p* ⟷ (*i*, *p*) ∈ *S*›

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

lemma *equiv-nom-reflp*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹*reflp* (*equiv-nom S*)›
unfolding *equiv-nom-def reflp-def* **using** *assms MCS-Nom-refl* **by** *fast*

lemma *equiv-nom-symp*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹*symp* (*equiv-nom S*)›
unfolding *equiv-nom-def symp-def* **using** *assms MCS-Nom-sym* **by** *fast*

lemma *equiv-nom-transp*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹*transp* (*equiv-nom S*)›
unfolding *equiv-nom-def transp-def* **using** *assms MCS-Nom-trans* **by** *fast*

lemma *equiv-nom-equivp*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹*equivp* (*equiv-nom S*)›
using *assms* **by** (*simp add*: *equivpI equiv-nom-reflp equiv-nom-symp equiv-nom-transp*)

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

lemma *equiv-nom-ne*:
assumes ‹*consistent S*› ‹*maximal S*›
shows ‹{*k*. *equiv-nom S i k*} ≠ {}›
unfolding *equiv-nom-def* **using** *assms MCS-Nom-refl* **by** *fast*

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

lemma *equiv-nom-Nom*:
assumes ‹*consistent S*› ‹*maximal S*› ‹*equiv-nom S i k*› ‹(*i*, *p*) ∈ *S*›
shows ‹(*k*, *p*) ∈ *S*›
proof −

have ‹$[(i, \cdot k), (i, p)] \vdash_@ (k, p)$›
by (*meson Assm-head Nom Weak*)
then show *?thesis*
using *assms MCS-derive* **unfolding** *equiv-nom-def* **by** *force*
qed

definition *reach* :: ‹('*i*, '*p*) *lbd set* ⇒ '*i* ⇒ '*i set*› **where**
‹*reach S i* ≡ $\{[k]_S \mid k.\ (i, \Diamond(\cdot k)) \in S\}$›

primrec *canonical* :: ‹('*i*, '*p*) *lbd set* × '*i* ⇒ ('*i*, '*p*, '*i*) *ctx*› (‹$\mathcal{M}_@$›) **where**
‹$\mathcal{M}_@(S, i)$ = (*Model* $\{[k]_S \mid k.\ True\}$ (*reach S*) ($\lambda i\ P.\ (i, \cdot P) \in S$), $\lambda i.\ [i]_S$, $[i]_S$)›

theorem *saturated-model*:
assumes ‹$\bigwedge p\ i.\ \mathcal{M}_@(S, i)\ [\![\mathcal{R}_@(S)]\!]_@(p) \longleftrightarrow \mathcal{R}_@(S)\ (\mathcal{M}_@(S, i))\ p$› ‹$M \in \{\mathcal{M}_@(S, i) \mid i.\ True\}$›
shows ‹$\mathcal{R}_@(S)\ (\mathcal{M}_@(S, i))\ p \longleftrightarrow \mathcal{M}_@(S, i) \models_@ p$›
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*(*1*)[*of i x*] *assms*(*2*)
by (*cases x*) (*auto simp*: *reach-def*)
qed

lemma *reach-assign*: ‹*reach S* $([i]_S) \subseteq \{[k]_S \mid k.\ True\}$›
unfolding *reach-def assign-def* **by** *blast*

theorem *saturated-MCS*:
assumes ‹*MCS S*›
shows ‹$\mathcal{M}_@(S, i)\ [\![\mathcal{R}_@(S)]\!]_@(p) \longleftrightarrow \mathcal{R}_@(S)\ (\mathcal{M}_@(S, i))\ p$›
proof (*cases p*)
case (*Nom k*)
have ‹$([i]_S = [k]_S) \longleftrightarrow ([i]_S, \cdot k) \in S$›
using *assms equiv-nom-equivp equiv-nom-assign* **by** (*metis assign-def equivp-def equiv-nom-def*)
then show *?thesis*
using *Nom* **by** *simp*
next
case (*Imp p q*)
have ‹$(([i]_S, p) \in S \longrightarrow ([i]_S, q) \in S) \longleftrightarrow ([i]_S, p \longrightarrow q) \in S$›
using *assms MCS-derive MCS-explode MCS-impE′* **by** (*metis ImpI Weaken set-subset-Cons*)
then show *?thesis*
using *Imp* **by** *simp*
next
case (*Dia p*)
have ‹$([i]_S, \Diamond p) \in S \longleftrightarrow (\exists k.\ ([i]_S, @k\ p \wedge \Diamond(\cdot k)) \in S)$›
using *assms MCS-exi* **by** *fastforce*

moreover have ‹$([i]_S, @k\ p \wedge \Diamond(\cdot k)) \in S \longleftrightarrow ([i]_S, @k\ p) \in S \wedge ([i]_S, \Diamond(\cdot k)) \in S$› **for** *k*
using *assms MCS-con* **by** *fast*
moreover have ‹$([i]_S, @k\ p) \in S \longleftrightarrow (k, p) \in S$› **for** *k*
using *assms* **by** (*meson MCS-derive SatE SatI*)
moreover have ‹$(k, p) \in S \longleftrightarrow ([k]_S, p) \in S$› **for** *k*
using *assms* **by** (*meson MCS-Nom-refl equiv-nom-Nom equiv-nom-assign equiv-nom-def*)
moreover have ‹$([i]_S, \Diamond(\cdot k)) \in S \Longrightarrow [k]_S \in$ *reach S* $([i]_S)$› **for** *k*
unfolding *reach-def* **by** *blast*

ultimately have ‹$([i]_S,\ \Diamond\ p) \in S \longleftrightarrow (\exists\, k \in \mathit{reach}\ S\ ([i]_S).\ (k,\ p) \in S)$›
unfolding *reach-def* **by** *blast*
then show *?thesis*
using *Dia reach-assign* **by** *fastforce*
next
case (*Sat k p*)
have ‹$([k]_S,\ p) \in S \longleftrightarrow ([i]_S,\ @k\ p) \in S$›
by (*metis SatE SatI assms MCS-derive equiv-nom-Nom equiv-nom-assign equiv-nom-symp sympD*)
then show *?thesis*
using *Sat* **by** *simp*
next
case (*All p*)
have ‹$([i]_S,\ \mathbf{A}\ p) \in S \longleftrightarrow (\forall\, k.\ (k,\ p) \in S)$›
using *assms MCS-uni* **by** *fastforce*
then have ‹$([i]_S,\ \mathbf{A}\ p) \in S \longleftrightarrow (\forall\, k.\ ([k]_S,\ p) \in S)$›
by (*meson MCS-Nom-sym assms equiv-nom-Nom equiv-nom-assign equiv-nom-def*)
then show *?thesis*
using *All* **by** *auto*
qed (*use assms* **in** *auto*)

interpretation *Truth-Witness semics semantics* ‹$\lambda S.\ \{\mathcal{M}_@(S,\ i)\ |i.\ True\}$› *rel consistent witness nominals-lbd*
proof
fix *p* **and** *M* :: ‹$('i,\ 'p,\ 'w)\ ctx$›
show ‹$(M \models_@ p) = M\ [\![semantics]\!]_@(p)$›
by (*cases M*, *induct p*) *auto*
next
fix *p M* **and** *S* :: ‹$('i,\ 'p)\ lbd\ set$›
assume ‹$\forall\, p.\ \forall\, M{\in}\{\mathcal{M}_@\ (S,\ i)\ |i.\ True\}.\ M\ [\![\mathcal{R}_@(S)]\!]_@\ p \longleftrightarrow \mathcal{R}_@(S)\ M\ p$› ‹$M \in \{\mathcal{M}_@(S,\ i)\ |i.\ True\}$›
then show ‹$M \models_@ p \longleftrightarrow \mathcal{R}_@(S)\ M\ p$›
using *saturated-model* **by** *fast*
next
fix *S* :: ‹$('i,\ 'p)\ lbd\ set$›
assume ‹*MCS S*›
then show ‹$\forall\, p.\ \forall\, M{\in}\{\mathcal{M}_@(S,\ i)\ |i.\ True\}.\ M\ [\![\mathcal{R}_@(S)]\!]_@\ p \longleftrightarrow \mathcal{R}_@(S)\ M\ p$›
using *saturated-MCS* **by** *fastforce*
qed

lemma *Truth-lemma*:
assumes ‹*MCS S*›
shows ‹$\mathcal{M}_@(S,\ i) \models_@ p \longleftrightarrow (i,\ p) \in S$›
proof −
have ‹$\mathcal{M}_@(S,\ i) \models_@ p \longleftrightarrow ([i]_S,\ p) \in S$›
using *assms truth-lemma* **by** *fastforce*
then show *?thesis*
using *assms* **by** (*meson MCS-Nom-sym equiv-nom-Nom equiv-nom-assign equiv-nom-def*)
qed

7.9 Cardinalities

datatype *marker* = *FlsM* | *ImpM* | *DiaM* | *SatM* | *AllM*

type-synonym $('i,\ 'p)\ enc$ = ‹$('i + 'p) + marker \times nat$›

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*)›
abbreviation ‹*GLO* ≡ *Inr* (*AllM*, *0*)›

primrec *encode* :: ‹('*i*, '*p*) *fm* ⇒ ('*i*, '*p*) *enc list*› **where**
  ‹*encode* ⊥ = [*FLS*]›
| ‹*encode* (•*P*) = [*PRO P*]›
| ‹*encode* (•*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*›
| ‹*encode* (**A** *p*) = *GLO* # *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 P*)
  then show *?case*
    by (*cases q*) *auto*
next
  case (*Nom i*)
  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 i p*)
  then show *?case*
    by (*cases q*) *auto*
next
  case (*All p*)
  then show *?case*
    by (*cases q*) *auto*
qed

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*, *AllM*}› **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-params-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*. ∀ *w* ∈ *W M*. *range g* ⊆ *W M* ⟶
(∀ (*k*, *q*) ∈ *X*. (*M*, *g*, *g k*) ⊨@ *q*) ⟶ (*M*, *g*, *w*) ⊨@ *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* ⊆ {(*i*, ¬ *p*)} ∪ *X* ⟶ ¬ *A* ⊢@ (*a*, ⊥)›
using *Boole FlsE* **by** (*metis derive-split1 insert-is-Un subset-insert*)

let *?X* = ‹{(*i*, ¬ *p*)} ∪ *X*›
let *?S* = ‹*Extend ?X*›

have ‹*consistent ?X*›
unfolding *consistent-def* **using** ∗ **by** *blast*
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 ‹$|UNIV :: 'i\ set| +c |UNIV :: 'p\ set| \leq o |UNIV - (nominals\ X \cup nominals\text{-}lbd\ (i, \neg p))|$›
by (*metis Set-Diff-Un*)
then have ‹$|UNIV :: 'i\ set| +c |UNIV :: 'p\ set| \leq o |UNIV - nominals\ ?X|$›
by (*metis UN-insert insert-is-Un sup-commute*)
then have ‹$|UNIV :: ('i, 'p)\ lbd\ set| \leq o |UNIV - nominals\ ?X|$›
using *assms card-of-lbd ordLeq-transitive* **by** *blast*
ultimately have ‹*MCS ?S*›
using *MCS-Extend* **by** *fast*
then have ‹$\mathcal{M}_@(?S, i) \models_@ p \longleftrightarrow (i, p) \in ?S$› **for** *i p*
using *Truth-lemma* **by** *fast*
then have ‹$(i, p) \in ?X \Longrightarrow \mathcal{M}_@(?S, i) \models_@ p$› **for** *i p*
using *Extend-subset* **by** *blast*
then have ‹$\mathcal{M}_@(?S, i) \models_@ \neg p$› ‹$\forall (k, q) \in X.\ \mathcal{M}_@(?S, k) \models_@ q$›
by *blast+*
moreover from *this* **have** ‹$\mathcal{M}_@(?S, i) \models_@ p$›
using *assms*(*1*) **by** *force*
ultimately show *False*
by *simp*
qed

abbreviation *valid* :: ‹$('i, 'p)\ fm \Rightarrow bool$› **where**
‹$valid\ p \equiv \forall (M :: ('i, 'p)\ model).\ g.\ \forall w \in W\ M.\ range\ g \subseteq W\ M \longrightarrow (M, g, w) \models_@ p$›

theorem *completeness*:
fixes *p* :: ‹$('i, 'p)\ fm$›
assumes ‹*valid p*› ‹$infinite\ (UNIV :: 'i\ set)$› ‹$|UNIV :: 'p\ set| \leq o |UNIV :: 'i\ set|$›
shows ‹$[] \vdash_@ (i, p)$›
proof −
have ‹$|UNIV :: 'i\ set| +c |UNIV :: 'p\ set| \leq o |UNIV :: 'i\ set|$›
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 ‹$[] \vdash_@ (i, p)$›
using *assms completeness*[*of p*] **by** *simp*

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

corollary *main′*:
fixes *p* :: ‹$('i, 'i)\ fm$›
assumes ‹$i \notin nominals\text{-}fm\ p$› ‹$infinite\ (UNIV :: 'i\ set)$›
shows ‹$valid\ p \longleftrightarrow [] \vdash_@ (i, p)$›
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*)
  | *Exi* ‹(*'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*› (**infix** ‹⊨∃› *50*) **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* [*simp*]: ‹sub-tm # t = t›
  by (*induct t*) (*auto cong*: *map-cong*)

lemma *reduce-Var* [*simp*]: ‹(# 0 ⨟ λn. # (Suc n)) = #›
proof (*rule ext*)
  fix *n*
  show ‹(# 0 ⨟ λn. # (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*)
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*› (**infix** ‹⊢∃› *50*) **where**
Assm [*simp*]: ‹*p* ∈ *set A* ⟹ *A* ⊢∃ *p*›
| *FlsE* [*elim*]: ‹*A* ⊢∃ ⊥ ⟹ *A* ⊢∃ *p*›
| *ImpI* [*intro*]: ‹*p* # *A* ⊢∃ *q* ⟹ *A* ⊢∃ *p* ⟶ *q*›
| *ImpE* [*dest*]: ‹*A* ⊢∃ *p* ⟶ *q* ⟹ *A* ⊢∃ *p* ⟹ *A* ⊢∃ *q*›
| *ExiI* [*intro*]: ‹*A* ⊢∃ ⟨*t*⟩*p* ⟹ *A* ⊢∃ ∃ *p*›
| *ExiE* [*elim*]: ‹*A* ⊢∃ ∃ *p* ⟹ *a* ∉ *params* (*set* (*p* # *q* # *A*)) ⟹ ⟨⋆*a*⟩*p* # *A* ⊢∃ *q* ⟹ *A* ⊢∃ *q*›
| *Clas*: ‹(*p* ⟶ *q*) # *A* ⊢∃ *p* ⟹ *A* ⊢∃ *p*›

8.4.1 Weakening

abbreviation ‹*psub f* ≡ *map-fm f id*›

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

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

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

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

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

lemma *map-tm-upd* [*simp*]: ‹*a* ∉ *params-tm t* ⟹ *map-tm* (*f*(*a* := *b*)) *t* = *map-tm f t*›
by (*induct t*) *auto*

lemma *psub-upd* [*simp*]: ‹$a \notin$ *params-fm* $p \Longrightarrow$ *psub* $(f(a := b))$ p = *psub* f p›
by (*induct* p) *auto*

class *inf-univ* =
fixes *itself* :: ‹$'a$ *itself*›
assumes *infinite-UNIV*: ‹*infinite* (*UNIV* :: $'a$ *set*)›

lemma *Calculus-psub*:
fixes f :: ‹$'f \Rightarrow 'g$ :: *inf-univ*›
shows ‹$A \vdash_\exists p \Longrightarrow$ *map* (*psub* f) $A \vdash_\exists$ *psub* f p›
proof (*induct* A p *arbitrary*: f *pred*: *Calculus*)
case (*Assm* p A)
then show *?case*
by *simp*
next
case (*FlsE* A p)
then show *?case*
by *force*
next
case (*ImpI* p A q)
then show *?case*
by *auto*
next
case (*ImpE* A p q)
then show *?case*
by *auto*
next
case (*ExiI* A t p)
then show *?case*
by (*metis Calculus.ExiI fm.simps*(*27*) *psub-inst-single*)
next
case (*ExiE* A p a q)
let *?params* = ‹*params'* $(p \# q \# A)$›
have ‹*finite ?params*›
by *simp*
then obtain b **where** b: ‹$b \notin \{f\ a\} \cup f$ ‘ *?params*›
using *ex-new-if-finite infinite-UNIV*
by (*metis finite.emptyI finite.insertI finite-UnI finite-imageI*)

define g **where** ‹$g \equiv f(a := b)$›

have ‹$a \notin$ *params'* $(p \# q \# A)$›
using *ExiE* **by** *simp*
then have b': ‹$b \notin$ *params'* (*map* (*psub* g) $(p \# q \# A)$)›
unfolding *g-def* **using** b *ExiE*(*3*) **by** (*auto simp*: *fm.set-map*(*1*))

have ‹*map* (*psub* g) $A \vdash_\exists$ *psub* g $(\exists\, p)$›
using *ExiE* **by** *blast*
then have ‹*map* (*psub* g) $A \vdash_\exists \exists\,$(*psub* g p)›
by *simp*
moreover have ‹*map* (*psub* g) $(\langle\star a\rangle p \# A) \vdash_\exists$ *psub* g q›
using *ExiE* **by** *blast*
then have ‹$\langle\star b\rangle$(*psub* g p) # *map* (*psub* g) $A \vdash_\exists$ *psub* g q›
unfolding *g-def* **by** *simp*
ultimately have ‹*map* (*psub* g) $A \vdash_\exists$ *psub* g q›

```
    using b' by fastforce
  moreover have ‹psub g q = psub f q› ‹map (psub g) A = map (psub f) A›
    unfolding g-def using ExiE.hyps(3) by simp-all
  ultimately show ?case
    by metis
next
  case (Clas p q A)
  then show ?case
    using Calculus.Clas by auto
qed

lemma Weaken:
  fixes p :: ‹('f :: inf-univ, 'p) fm›
  shows ‹A ⊢∃ p ⟹ set A ⊆ set B ⟹ B ⊢∃ p›
proof (induct A p arbitrary: B pred: Calculus)
  case (Assm p A)
  then show ?case
    by auto
next
  case (FlsE A p)
  then show ?case
    using Calculus.FlsE by blast
next
  case (ImpI p A q)
  then show ?case
    by (simp add: Calculus.ImpI subset-code(1))
next
  case (ImpE A p q)
  then show ?case
    by blast
next
  case (ExiI A t p)
  then show ?case
    by blast
next
  case (ExiE A p a q)
  let ?params = ‹params' (p # q # B)›
  have ‹finite ?params›
    by simp
  then obtain b where b: ‹b ∉ ?params›
    using ex-new-if-finite infinite-UNIV by blast
  then have b': ‹b ∉ params' A›
    using ExiE by auto

  define f where ‹f ≡ id(a := b, b := a)›
  let ?B = ‹map (psub f) B›

  have f: ‹∀ p ∈ set A. psub f p = p›
    using ExiE(3) b' by (simp add: fm.map-id f-def)
  then have ‹set A ⊆ set ?B›
    using ExiE.prems by force
  then have ‹?B ⊢∃ ∃ p›
    using ExiE.hyps by blast

  moreover have ‹⟨⋆a⟩p ∈ set (⟨⋆a⟩p # ?B)›
    using ExiE(3) b by (auto simp: fm.map-id0)
```

then have ‹*set* $(\langle\star\ a\rangle\ p\ \#\ A) \subseteq$ *set* $(\langle\star a\rangle p\ \#\ ?B)$›
using ‹*set* $A \subseteq$ *set* *?B*› **by** *auto*
then have ‹$\langle\star a\rangle p\ \#\ ?B \vdash_\exists\ q$›
using *ExiE*(*5*) **by** *blast*

moreover have ‹$a \notin$ *params'* $(p\ \#\ q\ \#\ ?B)$›
using *ExiE*(*3*) *b* **by** (*simp add*: *fm.set-map*(*1*) *image-iff f-def*)

ultimately have ‹$?B \vdash_\exists\ q$›
by *fast*
then have ‹*map* (*psub f*) *?B* $\vdash_\exists$ *psub f q*›
using *Calculus-psub* **by** *blast*
moreover have ‹*psub f q* = *q*›
using *ExiE.hyps*(*3*) *b fm.map-id* **unfolding** *f-def* **by** *auto*
moreover have ‹*f o f* = *id*›
unfolding *f-def* **by** *auto*
then have ‹*psub f o psub f* = *id*›
by (*auto simp*: *fm.map-comp fm.map-id0*)
then have ‹*map* (*psub f*) *?B* = *B*›
unfolding *map-map* **by** (*metis list.map-id*)
ultimately show *?case*
by *simp*
next
case (*Clas p q A*)
then show *?case*
using *Calculus.Clas*
by (*metis insert-mono list.simps*(*15*))
qed

8.5 Soundness

theorem *soundness*: ‹$A \vdash_\exists\ p \Longrightarrow \forall\, q \in$ *set* $A.\ (E,\ F,\ G) \models_\exists\ q \Longrightarrow (E,\ F,\ G) \models_\exists\ p$›
proof (*induct p arbitrary*: *F pred*: *Calculus*)
case (*ExiE A p a q*)
then obtain *x* **where** ‹$(x \fatsemi E,\ F,\ G) \models_\exists\ p$›
by *fastforce*
then have ‹$(E,\ F(a := \lambda\text{-}.\ x),\ G) \models_\exists\ \langle\star a\rangle p$›
using *ExiE*(*3*) **by** *simp*
moreover have ‹$\forall\, q \in$ *set* $A.\ (E,\ F(a := \lambda\text{-}.\ x),\ G) \models_\exists\ q$›
using *ExiE*(*3*, *6*) **by** *simp*
ultimately have ‹$(E,\ F(a := \lambda\text{-}.\ x),\ G) \models_\exists\ q$›
using *ExiE*(*5*) **by** *simp*
then show *?case*
using *ExiE*(*3*) **by** *simp*
qed *auto*

corollary *soundness'*: ‹$[] \vdash_\exists\ p \Longrightarrow M \models_\exists\ p$›
using *soundness* **by** (*cases M*) *fastforce*

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

8.6 Admissible Rules

lemma *Assm-head*: ‹$p\ \#\ A \vdash_\exists\ p$›

by *auto*

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

corollary *Weak*:
fixes p :: ‹$('f$ :: *inf-univ*, $'p)$ *fm*›
shows ‹$A \vdash_\exists\ p \Longrightarrow q\ \#\ A \vdash_\exists\ p$›
using *Weaken*[**where** B=‹$q\ \#\ A$›] **by** *auto*

lemma *deduct1*:
fixes p :: ‹$('f$ :: *inf-univ*, $'p)$ *fm*›
shows ‹$A \vdash_\exists\ p \longrightarrow q \Longrightarrow p\ \#\ A \vdash_\exists\ q$›
using *Assm-head Weak* **by** *blast*

lemma *Weak'*:
fixes p :: ‹$('f$ :: *inf-univ*, $'p)$ *fm*›
shows ‹$A \vdash_\exists\ p \Longrightarrow B\ @\ A \vdash_\exists\ p$›
by (*induct B*) (*simp-all add*: *Weak*)

interpretation *Derivations* ‹*Calculus* :: ($'f$ :: *inf-univ*, $'p$) *fm list* ⇒ -›
proof
show ‹$\bigwedge A\ p.\ p \in set\ A \Longrightarrow A \vdash_\exists\ p$›
by *simp*
next
show ‹$\bigwedge A\ B.\ A \vdash_\exists\ r \Longrightarrow set\ A = set\ B \Longrightarrow B \vdash_\exists\ r$› **for** r :: ‹($'f$, $'p$) *fm*›
using *Weaken* **by** *blast*
qed

8.7 Maximal Consistent Sets

definition *consistent* :: ‹($'f$, $'p$) *fm set* ⇒ *bool*› **where**
‹*consistent* $S \equiv \forall A.\ set\ A \subseteq S \longrightarrow \neg\ A \vdash_\exists \bot$›

fun *witness* :: ‹($'f$, $'p$) *fm* ⇒ ($'f$, $'p$) *fm set* ⇒ ($'f$, $'p$) *fm set*› **where**
‹*witness* $(\exists\ p)\ S$ = (*let* a = *SOME* $a.\ a \notin params\ (\{p\} \cup S)$ *in* $\{\langle\star a\rangle p\}$)›
| ‹*witness* - - = {}›

lemma *consistent-add-witness*:
fixes p :: ‹$('f$ :: *inf-univ*, $'p)$ *fm*›
assumes ‹*consistent* S› ‹$\exists\ p \in S$› ‹$a \notin params\ S$›
shows ‹*consistent* $(\{\langle\star a\rangle p\} \cup S)$›
unfolding *consistent-def*
proof *safe*
fix A
assume ‹$set\ A \subseteq \{\langle\star a\rangle p\} \cup S$› ‹$A \vdash_\exists \bot$›
then obtain A' **where** ‹$set\ A' \subseteq S$› ‹$(\langle\star a\rangle p)\ \#\ A' \vdash_\exists \bot$›
using *assms derive-split1* **by** (*metis consistent-def insert-is-Un subset-insert*)
then have ‹$\langle\star a\rangle p\ \#\ A' \vdash_\exists \bot$›
using *Boole* **by** *blast*
then have ‹$\langle\star a\rangle p\ \#\ \exists\ p\ \#\ A' \vdash_\exists \bot$›
using *Weak deduct1* **by** *blast*
moreover have ‹$a \notin params\text{-}fm\ p$› ‹$\forall p \in set\ (\exists\ p\ \#\ A').\ a \notin params\text{-}fm\ p$›
using ‹$set\ A' \subseteq S$› *assms*(*2*−*3*) **by** *auto*
then have ‹$a \notin params\ (\{p\} \cup \{\bot\} \cup set\ (\exists\ p\ \#\ A'))$›

```
    using calculation by simp
  moreover have ‹∃ p # A' ⊢∃ ∃ p›
    by simp
  ultimately have ‹∃ p # A' ⊢∃ ⊥›
    by fastforce
  moreover have ‹set (∃ p # A') ⊆ S›
    using ‹set A' ⊆ S› assms(2) by simp
  ultimately show False
    using assms(1) unfolding consistent-def by blast
qed

lemma consistent-witness':
  fixes p :: ‹('f :: inf-univ, 'p) fm›
  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 auto
qed (auto simp: assms)

interpretation MCS-Witness-UNIV consistent witness ‹params-fm :: ('f :: inf-univ, 'p) fm ⇒ -›
proof
  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 ({p} ∪ S ∪ witness p S)›
    using consistent-witness' by (simp add: sup-commute)
next
  show ‹infinite (UNIV :: ('f, 'p) fm set)›
    using infinite-UNIV-size[of ‹λp. p ⟶ p›] by simp
qed (auto simp: consistent-def)

interpretation Derivations-Cut-MCS consistent ‹Calculus :: ('f :: inf-univ, 'p) fm list ⇒ -›
proof
  show ‹⋀S. consistent S = (∀ A. set A ⊆ S ⟶ (∃ q. ¬ A ⊢∃ q))›
    unfolding consistent-def using FlsE by blast
next
  show ‹⋀A B p. A ⊢∃ p ⟹ p # B ⊢∃ q ⟹ A @ B ⊢∃ q› for q :: ‹('f, 'p) fm›
    by (metis ImpE ImpI Un-upper1 Weak' Weaken set-append)
qed

interpretation Derivations-Bot consistent Calculus ‹⊥ :: ('f :: inf-univ, 'p) fm›
proof
```

qed *fast*

interpretation *Derivations-Imp consistent Calculus* ‹$\lambda p\ q.\ p \longrightarrow q :: ('f :: \mathit{inf\text{-}univ}, 'p)\ \mathit{fm}$›
proof qed *fast+*

interpretation *Derivations-Exi consistent witness params-fm Calculus* ‹$\exists$› ‹$\lambda t\ p.\ \langle t\rangle p :: ('f :: \mathit{inf\text{-}univ}, 'p)\ \mathit{fm}$›
proof qed *auto*

8.8 Truth Lemma

abbreviation *canonical* :: ‹$('f, 'p)\ \mathit{fm}\ \mathit{set} \Rightarrow ('f\ \mathit{tm}, 'f, 'p)\ \mathit{model}$› (‹$\mathcal{M}_\exists$›) **where**
‹$\mathcal{M}_\exists\,(S) \equiv (\#, \cdot, \lambda P\ \mathit{ts}.\ \cdot P\ \mathit{ts} \in S)$›

fun *semics* ::
‹$('a, 'f, 'p)\ \mathit{model} \Rightarrow (('a, 'f, 'p)\ \mathit{model} \Rightarrow ('f, 'p)\ \mathit{fm} \Rightarrow \mathit{bool}) \Rightarrow ('f, 'p)\ \mathit{fm} \Rightarrow \mathit{bool}$›
($‹(- [\![-]\!]_\exists\ -)›$ [55, 0, 55] 55) **where**
‹$-\ [\![-]\!]_\exists\ \bot \longleftrightarrow \mathit{False}$›
| ‹$(E, F, G)\ [\![-]\!]_\exists\ \cdot P\ \mathit{ts} \longleftrightarrow G\ P\ (\mathit{map}\ (\!|(E, F)|\!)\ \mathit{ts})$›
| ‹$(E, F, G)\ [\![\mathcal{R}]\!]_\exists\ p \longrightarrow q \longleftrightarrow \mathcal{R}\ (E, F, G)\ p \longrightarrow \mathcal{R}\ (E, F, G)\ q$›
| ‹$(E, F, G)\ [\![\mathcal{R}]\!]_\exists\ \exists p \longleftrightarrow (\exists x.\ \mathcal{R}\ (x \fatsemi E, F, G)\ p)$›

fun *rel* :: ‹$('f, 'p)\ \mathit{fm}\ \mathit{set} \Rightarrow ('f\ \mathit{tm}, 'f, 'p)\ \mathit{model} \Rightarrow ('f, 'p)\ \mathit{fm} \Rightarrow \mathit{bool}$› (‹$\mathcal{R}_\exists$›) **where**
‹$\mathcal{R}_\exists\,(S)\ (E, -, -)\ p \longleftrightarrow \mathit{sub\text{-}fm}\ E\ p \in S$›

theorem *saturated-model*:
assumes ‹$\bigwedge p.\ \forall M \in \{\mathcal{M}_\exists\,(S)\}.\ M\ [\![\mathcal{R}_\exists\,(S)]\!]_\exists\ p \longleftrightarrow \mathcal{R}_\exists\,(S)\ M\ p$› ‹$M \in \{\mathcal{M}_\exists\,(S)\}$›
shows ‹$\mathcal{R}_\exists\,(S)\ M\ p \longleftrightarrow M \models_\exists p$›
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*(*1*)[*of x*] *assms*(*2*) **by** (*cases x*) *simp-all*
qed

theorem *saturated-MCS*:
fixes *p* :: ‹$('f :: \mathit{inf\text{-}univ}, 'p)\ \mathit{fm}$›
assumes ‹*MCS S*›
shows ‹$\mathcal{R}_\exists\,(S)\ (\mathcal{M}_\exists\,(S))\ p \longleftrightarrow \mathcal{M}_\exists\,(S)\ [\![\mathcal{R}_\exists\,(S)]\!]_\exists\ p$›
using *assms* **by** (*cases p*) (*auto cong*: *map-cong*)

interpretation *Truth-Witness semics semantics-fm* ‹$\lambda S.\ \{\mathcal{M}_\exists\,(S)\}$› *rel consistent witness*
‹*params-fm* :: $('f :: \mathit{inf\text{-}univ}, 'p)\ \mathit{fm} \Rightarrow$ -›
proof
fix *p* **and** *M* :: ‹$('f\ \mathit{tm}, 'f, 'p)\ \mathit{model}$›
show ‹$M \models_\exists p \longleftrightarrow M\ [\![(\models_\exists)]\!]_\exists\ p$›
by (*cases M*, *induct p*) *auto*
qed (*use saturated-model saturated-MCS* **in** *blast*)+

8.9 Cardinalities

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

type-synonym $('f, 'p)\ enc = ‹('f + 'p) + marker \times nat›$

abbreviation $‹FUNS\ f \equiv Inl\ (Inl\ f)›$
abbreviation $‹PRES\ p \equiv Inl\ (Inr\ p)›$

abbreviation $‹VAR\ n \equiv Inr\ (VarM,\ n)›$
abbreviation $‹FUN\ n \equiv Inr\ (FunM,\ n)›$
abbreviation $‹TM\ n \equiv Inr\ (TmM,\ n)›$

abbreviation $‹PRE\ n \equiv Inr\ (PreM,\ n)›$
abbreviation $‹FLS \equiv Inr\ (FlsM,\ 0)›$
abbreviation $‹IMP\ n \equiv Inr\ (FlsM,\ n)›$
abbreviation $‹EXI \equiv Inr\ (ExiM,\ 0)›$

primrec
  encode-tm :: $‹'f\ tm \Rightarrow ('f, 'p)\ enc\ list›$ **and**
  encode-tms :: $‹'f\ tm\ list \Rightarrow ('f, 'p)\ enc\ list›$ **where**
  $‹encode\text{-}tm\ (\#n) = [VAR\ n]›$
| $‹encode\text{-}tm\ (\cdot f\ ts) = FUN\ (length\ ts)\ \#\ FUNS\ f\ \#\ encode\text{-}tms\ ts›$
| $‹encode\text{-}tms\ [] = []›$
| $‹encode\text{-}tms\ (t\ \#\ ts) = TM\ (length\ (encode\text{-}tm\ t))\ \#\ encode\text{-}tm\ t\ @\ encode\text{-}tms\ ts›$

lemma *encode-tm-ne* [*simp*]: $‹encode\text{-}tm\ t \neq []›$
  by (*induct t*) *auto*

lemma *inj-encode-tm'*:
  $‹(encode\text{-}tm\ t :: ('f, 'p)\ enc\ list) = encode\text{-}tm\ s \Longrightarrow t = s›$
  $‹(encode\text{-}tms\ ts :: ('f, 'p)\ enc\ list) = encode\text{-}tms\ ss \Longrightarrow ts = ss›$
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*: $‹inj\ encode\text{-}tm›$
  unfolding *inj-def* **using** *inj-encode-tm'* **by** *blast*

primrec *encode-fm* :: $‹('f, 'p)\ fm \Rightarrow ('f, 'p)\ enc\ list›$ **where**
  $‹encode\text{-}fm\ \bot = [FLS]›$
| $‹encode\text{-}fm\ (\cdot P\ ts) = PRE\ (length\ ts)\ \#\ PRES\ P\ \#\ encode\text{-}tms\ ts›$
| $‹encode\text{-}fm\ (p \longrightarrow q) = IMP\ (length\ (encode\text{-}fm\ p))\ \#\ encode\text{-}fm\ p\ @\ encode\text{-}fm\ q›$
| $‹encode\text{-}fm\ (\exists\ p) = EXI\ \#\ encode\text{-}fm\ p›$

lemma *encode-fm-ne* [*simp*]: $‹encode\text{-}fm\ p \neq []›$

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 (*Exi 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* ∈ {*VarM*, *FunM*, *TmM*, *FlsM*, *PreM*, *ImpM*, *ExiM*}› **for** *p*
by (*cases p*) *auto*
then show *?thesis*
by (*meson ex-new-if-finite finite.emptyI finite-insert*)
qed

lemma *card-of-fm*:
‹|*UNIV* :: (*'f* :: *inf-univ*, *'p*) *fm set*| ≤*o* |*UNIV* :: *'f 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* :: (*'f* + *'p*) *set*)›
by (*simp add*: *inf-univ-class.infinite-UNIV*)
ultimately have ‹|*UNIV* :: (*'f*, *'p*) *enc list set*| ≤*o* |*UNIV* :: (*'f* + *'p*) *set*|›
using *card-of-params-marker-lists* **by** *blast*
moreover have ‹|*UNIV* :: (*'f*, *'p*) *fm set*| ≤*o* |*UNIV* :: (*'f*, *'p*) *enc list set*|›
using *card-of-ordLeq inj-encode-fm* **by** *blast*
ultimately have ‹|*UNIV* :: (*'f*, *'p*) *fm set*| ≤*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 ‹∀ *M* :: (*'f tm*, *'f* :: *inf-univ*, *'p*) *model*. (∀ *q* ∈ *X*. *M* ⊨∃ *q*) ⟶ *M* ⊨∃ *p*›
‹|*UNIV* :: *'f set*| +*c* |*UNIV* :: *'p set*| ≤*o* |*UNIV* − *params X*|›
shows ‹∃ *A*. *set A* ⊆ *X* ∧ *A* ⊢∃ *p*›

proof (*rule ccontr*)
  assume ‹$\nexists A.\ set\ A \subseteq X \wedge A \vdash_\exists p$›
  then have $*$: ‹$\forall A.\ set\ A \subseteq \{\neg\ p\} \cup X \longrightarrow \neg\ A \vdash_\exists \bot$›
    using *Boole FlsE* **by** (*metis derive-split1 insert-is-Un subset-insert*)

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

  have ‹*consistent ?X*›
    unfolding *consistent-def* **using** $*$ **by** *blast*
  moreover have ‹*infinite* (*UNIV* − *params X*)›
    using *assms*(*2*) *inf-univ-class.infinite-UNIV*
    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*(*2*) *finite-params-fm card-of-infinite-diff-finite*
    by (*metis ordIso-iff-ordLeq ordLeq-transitive*)
  then have ‹$|UNIV :: 'f\ set| +c |UNIV :: 'p\ set| \leq o |UNIV - (params\ X \cup params\text{-}fm\ (\neg\ p))|$›
    by (*metis Set-Diff-Un*)
  then have ‹$|UNIV :: 'f\ set| +c |UNIV :: 'p\ set| \leq o |UNIV - params\ ?X|$›
    by (*metis UN-insert insert-is-Un sup-commute*)
  then have ‹$|UNIV :: ('f, 'p)\ fm\ set| \leq o |UNIV - params\ ?X|$›
    using *assms card-of-fm ordLeq-transitive* **by** *blast*
  ultimately have ‹*MCS ?S*›
    using *MCS-Extend* **by** *fast*
  then have ‹$p \in ?S \longleftrightarrow \mathcal{M}_\exists(?S) \models_\exists p$› **for** *p*
    using *truth-lemma* **by** *fastforce*
  then have ‹$p \in ?X \longrightarrow \mathcal{M}_\exists(?S) \models_\exists p$› **for** *p*
    using *Extend-subset* **by** *blast*
  then have ‹$\mathcal{M}_\exists(?S) \models_\exists \neg\ p$› ‹$\forall q \in X.\ \mathcal{M}_\exists(?S) \models_\exists q$›
    by *blast*+
  moreover from *this* **have** ‹$\mathcal{M}_\exists(?S) \models_\exists p$›
    using *assms*(*1*) **by** *blast*
  ultimately show *False*
    by *simp*
qed

abbreviation *valid* :: ‹$('f, 'p)\ fm \Rightarrow bool$› **where**
  ‹$valid\ p \equiv \forall M :: ('f\ tm, -, -)\ model.\ M \models_\exists p$›

theorem *completeness*:
  fixes *p* :: ‹$('f :: inf\text{-}univ, 'p)\ fm$›
  assumes ‹*valid p*› ‹$|UNIV :: 'p\ set| \leq o |UNIV :: 'f\ set|$›
  shows ‹$[] \vdash_\exists p$›
proof −
  have ‹$|UNIV :: 'f\ set| +c |UNIV :: 'p\ set| \leq o |UNIV :: 'f\ set|$›
    using *assms*(*2*)
    by (*simp add*: *inf-univ-class.infinite-UNIV 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* :: ‹$('f :: inf\text{-}univ, 'f)\ fm$›
  assumes ‹*valid p*›
  shows ‹$[] \vdash_\exists p$›
  using *assms completeness*[*of p*] **by** *simp*

theorem *main*:
fixes $p :: \langle('f :: \mathit{inf\text{-}univ}, 'p)\ \mathit{fm}\rangle$
assumes $\langle|\mathit{UNIV} :: 'p\ \mathit{set}| \leq o\ |\mathit{UNIV} :: 'f\ \mathit{set}|\rangle$
shows $\langle \mathit{valid}\ p \longleftrightarrow [] \vdash_\exists\ p\rangle$
using *assms completeness soundness'* **by** *blast*

corollary *main'*:
fixes $p :: \langle('f :: \mathit{inf\text{-}univ}, 'f)\ \mathit{fm}\rangle$
shows $\langle \mathit{valid}\ p \longleftrightarrow [] \vdash_\exists\ p\rangle$
using *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.