An Abstract Formalization of Gödel's Incompleteness Theorems

Andrei Popescu Dmitriy Traytel

March 17, 2025

Abstract

We present an abstract formalization of Gödel's incompleteness theorems. We analyze sufficient conditions for the theorems' applicability to a partially specified logic. Our abstract perspective enables a comparison between alternative approaches from the literature. These include Rosser's variation of the first theorem, Jeroslow's variation of the second theorem, and the Swierczkowski–Paulson semantics-based approach. This AFP entry is the main entry point to the results described in our CADE-27 paper [1].

As part of our abstract formalization's validation, we instantiate our locales twice in the separate AFP entries Goedel_HFSet_Semantic and Goedel_HFSet_Semanticless.

Contents

Chapter 1

Deduction with Two Provability Relations

We work with two provability relations: provability *prv* and basic provability *bprv*.

1.1 From Deduction with One Provability Relation to Two

locale *Deduct2* =
Deduct
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv
+
B: *Deduct*
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  bprv
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *num*
and *eql cnj imp all exi*
and *prv bprv*
+
assumes *bprv_prv*: $\bigwedge\varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow bprv\ \varphi \Longrightarrow prv\ \varphi$
begin

lemma *bprv_prv'*:
  assumes φ: $\varphi \in fmla$ **and** *b*: *bprv* φ
  shows *prv* φ
⟨*proof*⟩

end — context *Deduct2*

locale *Deduct2_with_False* =
Deduct_with_False
  var trm fmla Var FvarsT substT Fvars subst

eql cnj imp all exi
fls
num
prv
+
B: *Deduct_with_False*
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
num
bprv
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *num*
and *prv bprv*
+
assumes *bprv_prv*: $\bigwedge\varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow bprv\ \varphi \Longrightarrow prv\ \varphi$

sublocale *Deduct2_with_False* < *d_dwf*: *Deduct2*
⟨*proof*⟩

context *Deduct2_with_False* **begin**

lemma *consistent_B_consistent*: *consistent* ⟹ *B.consistent*
⟨*proof*⟩

end — context *Deduct2_with_False*

locale *Deduct2_with_False_Disj* =
Deduct_with_False_Disj
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
prv
+
B: *Deduct_with_False_Disj*
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
bprv
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *prv bprv*
+
assumes *bprv_prv*: $\bigwedge\varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow bprv\ \varphi \Longrightarrow prv\ \varphi$

sublocale *Deduct2_with_False_Disj* < *dwf_dwfd*: *Deduct2_with_False*
⟨*proof*⟩

locale *Deduct2_with_PseudoOrder* =
Deduct2_with_False_Disj
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
prv bprv
+
Syntax_PseudoOrder
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
Lq
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *prv bprv*
and *Lq*
+
assumes

Lq_num:
let LLq = (λ *t1 t2*. *psubst Lq* [(*t1*,*zz*), (*t2*,*yy*)]) *in*
∀ φ ∈ *fmla*. ∀ *q* ∈ *num*. *Fvars* φ = {*zz*} ∧ (∀ *p* ∈ *num*. *bprv* (*subst* φ *p zz*))
⟶ *prv* (*all zz* (*imp* (*LLq* (*Var zz*) *q*) φ))
and

Lq_num2:
let LLq = (λ *t1 t2*. *psubst Lq* [(*t1*,*zz*), (*t2*,*yy*)]) *in*
∀ *p* ∈ *num*. ∃ *P* ⊆ *num*. *finite P* ∧ *prv* (*dsj* (*sdsj* {*eql* (*Var yy*) *r* | *r*. *r* ∈ *P*}) (*LLq p* (*Var yy*)))
begin

lemma *LLq_num*:
assumes φ ∈ *fmla q* ∈ *num Fvars* φ = {*zz*} ∀ *p* ∈ *num*. *bprv* (*subst* φ *p zz*)
shows *prv* (*all zz* (*imp* (*LLq* (*Var zz*) *q*) φ))
⟨*proof*⟩

lemma *LLq_num2*:
assumes *p* ∈ *num*
shows ∃ *P* ⊆ *num*. *finite P* ∧ *prv* (*dsj* (*sdsj* {*eql* (*Var yy*) *r* | *r*. *r* ∈ *P*}) (*LLq p* (*Var yy*)))
⟨*proof*⟩

end — context *Deduct2_with_PseudoOrder*

1.2 Factoring In Explicit Proofs

locale *Deduct_with_Proofs* =
Deduct_with_False_Disj
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
  prv
for
var :: ′*var set* **and** *trm* :: ′*trm set* **and** *fmla* :: ′*fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *prv*
+
fixes
proof :: ′*proof set*
and
prfOf :: ′*proof* ⇒ ′*fmla* ⇒ *bool*
assumes
— Provability means there exists a proof (only needed for sentences):
prv_prfOf: $\bigwedge \varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow prv\ \varphi \longleftrightarrow (\exists\ prf \in proof.\ prfOf\ prf\ \varphi)$

locale *Deduct2_with_Proofs* =
Deduct2_with_False_Disj
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
  prv bprv
+
Deduct_with_Proofs
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
  prv
  proof prfOf
for
var :: ′*var set* **and** *trm* :: ′*trm set* **and** *fmla* :: ′*fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *prv bprv*
and *proof* :: ′*proof set* **and** *prfOf*

locale *Deduct2_with_Proofs_PseudoOrder* =
Deduct2_with_Proofs
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi

fls
dsj
num
prv bprv
proof prfOf
+
Deduct2_with_PseudoOrder
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
prv bprv
Lq
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *prv bprv*
and *proof* :: *'proof set* **and** *prfOf*
and *Lq*

Chapter 2

Abstract Encoding

Here we simply fix some unspecified encoding functions: encoding formulas and proofs as numerals.

locale *Encode* =
Syntax_with_Numerals
  var trm fmla Var FvarsT substT Fvars subst
  num
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *num*
+
fixes *enc* :: *'fmla* ⇒ *'trm* (‹⟨_⟩›)
assumes
enc[*simp*,*intro*!]: $\bigwedge \varphi.\ \varphi \in fmla \Longrightarrow enc\ \varphi \in num$
begin

end — context *Encode*

Explicit proofs (encoded as numbers), needed only for the harder half of Goedel's first, and for both half's of Rosser's version; not needed in Goedel's second.

locale *Encode_Proofs* =
Encode
  var trm fmla Var FvarsT substT Fvars subst
  num
  enc
+
Deduct2_with_Proofs
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
  prv bprv
  proof prfOf
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *num*
and *eql cnj imp all exi*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *fls dsj*
and *proof* :: *'proof set* **and** *prfOf*

+
fixes *encPf* :: $'proof \Rightarrow 'trm$
assumes
encPf[*simp*,*intro*!]: $\bigwedge pf.\ pf \in proof \Longrightarrow encPf\ pf \in num$

Chapter 3

Representability Assumptions

Here we make assumptions about various functions or relations being representable.

3.1 Representability of Negation

The negation function neg is assumed to be representable by a two-variable formula N.

locale *Repr_Neg* =
Deduct2_with_False
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  num
  prv bprv
+
Encode
  var trm fmla Var FvarsT substT Fvars subst
  num
  enc
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *num*
and *prv bprv*
and *enc* (‹⟨_⟩›)
+
fixes *N* :: *'fmla*
assumes
N[*simp*,*intro*!]: $N \in fmla$
and
Fvars_N[*simp*]: $Fvars\ N = \{xx,yy\}$
and
neg_implies_prv_N:
$\bigwedge \varphi.$
  $let\ NN = (\lambda\ t1\ t2.\ psubst\ N\ [(t1,xx),\ (t2,yy)])\ in$
  $\varphi \in fmla \longrightarrow Fvars\ \varphi = \{\} \longrightarrow bprv\ (NN\ \langle\varphi\rangle\ \langle neg\ \varphi\rangle)$
and
N_unique:
$\bigwedge \varphi.$
  $let\ NN = (\lambda\ t1\ t2.\ psubst\ N\ [(t1,xx),\ (t2,yy)])\ in$

$\varphi \in fmla \longrightarrow Fvars\ \varphi = \{\} \longrightarrow$
$bprv\ (all\ yy\ (all\ yy'$
$(imp\ (cnj\ (NN\ \langle\varphi\rangle\ (Var\ yy))\ (NN\ \langle\varphi\rangle\ (Var\ yy')))$
$(eql\ (Var\ yy)\ (Var\ yy')))))$
begin

NN is a notation for the predicate that takes terms and returns corresponding instances of N, obtained by substituting its free variables with these terms. This is very convenient for reasoning, and will be done for all the representing formulas we will consider.

definition *NN* **where** $NN \equiv \lambda\ t1\ t2.\ psubst\ N\ [(t1,xx),\ (t2,yy)]$

lemma *NN_def2*: $t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow yy \notin FvarsT\ t1 \Longrightarrow$
$NN\ t1\ t2 = subst\ (subst\ N\ t1\ xx)\ t2\ yy$
⟨*proof*⟩

lemma *NN_neg*:
$\varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow bprv\ (NN\ \langle\varphi\rangle\ \langle neg\ \varphi\rangle)$
⟨*proof*⟩

lemma *NN_unique*:
assumes $\varphi \in fmla\ Fvars\ \varphi = \{\}$
shows $bprv\ (all\ yy\ (all\ yy'$
$(imp\ (cnj\ (NN\ \langle\varphi\rangle\ (Var\ yy))\ (NN\ \langle\varphi\rangle\ (Var\ yy')))$
$(eql\ (Var\ yy)\ (Var\ yy')))))$
⟨*proof*⟩

lemma *NN*[*simp*,*intro*]:
$t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow NN\ t1\ t2 \in fmla$
⟨*proof*⟩

lemma *Fvars_NN*[*simp*]: $t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow yy \notin FvarsT\ t1 \Longrightarrow$
$Fvars\ (NN\ t1\ t2) = FvarsT\ t1 \cup FvarsT\ t2$
⟨*proof*⟩

lemma [*simp*]:
$m \in num \Longrightarrow n \in num \Longrightarrow subst\ (NN\ m\ (Var\ yy))\ n\ yy = NN\ m\ n$
$m \in num \Longrightarrow n \in num \Longrightarrow subst\ (NN\ m\ (Var\ yy'))\ n\ yy = NN\ m\ (Var\ yy')$
$m \in num \Longrightarrow subst\ (NN\ m\ (Var\ yy'))\ (Var\ yy)\ yy' = NN\ m\ (Var\ yy)$
$n \in num \Longrightarrow subst\ (NN\ (Var\ xx)\ (Var\ yy))\ n\ xx = NN\ n\ (Var\ yy)$
$n \in num \Longrightarrow subst\ (NN\ (Var\ xx)\ (Var\ xx'))\ n\ xx = NN\ n\ (Var\ xx')$
$m \in num \Longrightarrow n \in num \Longrightarrow subst\ (NN\ m\ (Var\ xx'))\ n\ zz = NN\ m\ (Var\ xx')$
$n \in num \Longrightarrow subst\ (NN\ n\ (Var\ yy))\ (Var\ xx')\ yy = NN\ n\ (Var\ xx')$
$m \in num \Longrightarrow n \in num \Longrightarrow subst\ (NN\ m\ (Var\ xx'))\ n\ xx' = NN\ m\ n$
⟨*proof*⟩

lemma *NN_unique2*:
assumes [*simp*]:$\varphi \in fmla\ Fvars\ \varphi = \{\}$
shows
$bprv\ (all\ yy$
$(imp\ (NN\ \langle\varphi\rangle\ (Var\ yy))$
$(eql\ \langle neg\ \varphi\rangle\ (Var\ yy))))$
⟨*proof*⟩

lemma *NN_neg_unique*:
assumes [*simp*]:$\varphi \in fmla\ Fvars\ \varphi = \{\}$
shows
$bprv\ (imp\ (NN\ \langle\varphi\rangle\ (Var\ yy))$

$(eql\ \langle neg\ \varphi\rangle\ (Var\ yy)))$ (**is** *bprv ?A*)
⟨*proof*⟩

lemma *NN_exi_cnj*:
assumes φ[*simp*]: $\varphi \in fmla$ *Fvars* $\varphi = \{\}$ **and** χ[*simp*]: $\chi \in fmla$
assumes *f*: *Fvars* $\chi = \{yy\}$
shows *bprv* $(eqv\ (subst\ \chi\ \langle neg\ \varphi\rangle\ yy)$
$(exi\ yy\ (cnj\ \chi\ (NN\ \langle\varphi\rangle\ (Var\ yy)))))$
(**is** *bprv* (*eqv ?A ?B*))
⟨*proof*⟩

end — context *Repr_Neg*

3.2 Representability of Self-Substitution

Self-substitution is the function that takes a formula φ and returns $\phi[\langle\phi\rangle/xx]$ (for the fixed variable xx). This is all that will be needed for the diagonalization lemma.

locale *Repr_SelfSubst* =
Encode
var trm fmla Var FvarsT substT Fvars subst
num
enc
+
Deduct2
var trm fmla Var FvarsT substT Fvars subst
num
eql cnj imp all exi
prv bprv
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *num*
and *eql cnj imp all exi*
and *prv bprv*
and *enc* (‹⟨_⟩›)
+
fixes *S* :: *'fmla*
assumes
S[*simp*,*intro*!]: $S \in fmla$
and
Fvars_S[*simp*]: *Fvars* $S = \{xx,yy\}$
and
subst_implies_prv_S:
$\bigwedge \varphi$.
let $SS = (\lambda\ t1\ t2.\ psubst\ S\ [(t1,xx),\ (t2,yy)])$ *in*
$\varphi \in fmla \longrightarrow Fvars\ \varphi = \{xx\} \longrightarrow$
$bprv\ (SS\ \langle\varphi\rangle\ \langle subst\ \varphi\ \langle\varphi\rangle\ xx\rangle)$
and
S_unique:
$\bigwedge \varphi$.
let $SS = (\lambda\ t1\ t2.\ psubst\ S\ [(t1,xx),\ (t2,yy)])$ *in*
$\varphi \in fmla \longrightarrow Fvars\ \varphi = \{xx\} \longrightarrow$
$bprv\ (all\ yy\ (all\ yy'$
$(imp\ (cnj\ (SS\ \langle\varphi\rangle\ (Var\ yy))\ (SS\ \langle\varphi\rangle\ (Var\ yy')))$
$(eql\ (Var\ yy)\ (Var\ yy')))))$
begin

SS is the instantiation combinator of S:

definition *SS* **where** $SS \equiv \lambda\ t1\ t2.\ psubst\ S\ [(t1,xx),\ (t2,yy)]$

lemma *SS_def2*: $t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow$
$yy \notin FvarsT\ t1 \Longrightarrow$
$SS\ t1\ t2 = subst\ (subst\ S\ t1\ xx)\ t2\ yy$
⟨*proof*⟩

lemma *subst_implies_prv_SS*:
$\varphi \in fmla \Longrightarrow Fvars\ \varphi = \{xx\} \Longrightarrow bprv\ (SS\ \langle\varphi\rangle\ \langle subst\ \varphi\ \langle\varphi\rangle\ xx\rangle)$
⟨*proof*⟩

lemma *SS_unique*:
$\varphi \in fmla \Longrightarrow Fvars\ \varphi = \{xx\} \Longrightarrow$
$bprv\ (all\ yy\ (all\ yy'$
$(imp\ (cnj\ (SS\ \langle\varphi\rangle\ (Var\ yy))\ (SS\ \langle\varphi\rangle\ (Var\ yy')))$
$(eql\ (Var\ yy)\ (Var\ yy')))))$
⟨*proof*⟩

lemma *SS*[*simp*,*intro*]:
$t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow SS\ t1\ t2 \in fmla$
⟨*proof*⟩

lemma *Fvars_SS*[*simp*]: $t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow yy \notin FvarsT\ t1 \Longrightarrow$
$Fvars\ (SS\ t1\ t2) = FvarsT\ t1 \cup FvarsT\ t2$
⟨*proof*⟩

lemma [*simp*]:
$m \in num \Longrightarrow p \in num \Longrightarrow subst\ (SS\ m\ (Var\ yy))\ p\ yy = SS\ m\ p$
$m \in num \Longrightarrow subst\ (SS\ m\ (Var\ yy'))\ (Var\ yy)\ yy' = SS\ m\ (Var\ yy)$
$m \in num \Longrightarrow p \in num \Longrightarrow subst\ (SS\ m\ (Var\ yy'))\ p\ yy' = SS\ m\ p$
$m \in num \Longrightarrow p \in num \Longrightarrow subst\ (SS\ m\ (Var\ yy'))\ p\ yy = SS\ m\ (Var\ yy')$
$m \in num \Longrightarrow subst\ (SS\ (Var\ xx)\ (Var\ yy))\ m\ xx = SS\ m\ (Var\ yy)$
⟨*proof*⟩

end — context *Repr_SelfSubst*

3.3 Representability of Self-Soft-Substitution

The soft substitution function performs substitution logically instead of syntactically. In particular, its "self" version sends φ to $exi\ xx\ (cnj\ (eql\ (Var\ xx)\ (enc\ \varphi))\ \varphi)$. Representability of self-soft-substitution will be an alternative assumption in the diagonalization lemma.

locale *Repr_SelfSoftSubst* =
Encode
var trm fmla Var FvarsT substT Fvars subst
num
enc
\+
Deduct2
var trm fmla Var FvarsT substT Fvars subst
num
eql cnj imp all exi
prv bprv
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*

and *num*
and *eql cnj imp all exi*
and *prv bprv*
and *enc* (‹⟨_⟩›)
+
fixes *S* :: *′fmla*
assumes
S[*simp*,*intro*!]: *S* ∈ *fmla*
and
Fvars_S[*simp*]: *Fvars S* = {*xx*,*yy*}
and
softSubst_implies_prv_S:
⋀ *φ*.
  let SS = (λ *t1 t2*. *psubst S* [(*t1*,*xx*), (*t2*,*yy*)]) *in*
  φ ∈ *fmla* ⟶ *Fvars φ* = {*xx*} ⟶
  bprv (*SS* ⟨*φ*⟩ ⟨*softSubst φ* ⟨*φ*⟩ *xx*⟩)
and
S_unique:
⋀ *φ*.
  let SS = (λ *t1 t2*. *psubst S* [(*t1*,*xx*), (*t2*,*yy*)]) *in*
  φ ∈ *fmla* ⟶ *Fvars φ* = {*xx*} ⟶
  bprv (*all yy* (*all yy′*
     (*imp* (*cnj* (*SS* ⟨*φ*⟩ (*Var yy*)) (*SS* ⟨*φ*⟩ (*Var yy′*)))
        (*eql* (*Var yy*) (*Var yy′*)))))
begin

SS is the instantiation combinator of S:

definition *SS* **where** *SS* ≡ λ *t1 t2*. *psubst S* [(*t1*,*xx*), (*t2*,*yy*)]

lemma *SS_def2*: *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹
 yy ∉ *FvarsT t1* ⟹
 SS t1 t2 = *subst* (*subst S t1 xx*) *t2 yy*
  ⟨*proof*⟩

lemma *softSubst_implies_prv_SS*:
φ ∈ *fmla* ⟹ *Fvars φ* = {*xx*} ⟹ *bprv* (*SS* ⟨*φ*⟩ ⟨*softSubst φ* ⟨*φ*⟩ *xx*⟩)
⟨*proof*⟩

lemma *SS_unique*:
φ ∈ *fmla* ⟹ *Fvars φ* = {*xx*} ⟹
 bprv (*all yy* (*all yy′*
     (*imp* (*cnj* (*SS* ⟨*φ*⟩ (*Var yy*)) (*SS* ⟨*φ*⟩ (*Var yy′*)))
        (*eql* (*Var yy*) (*Var yy′*)))))
⟨*proof*⟩

lemma *SS*[*simp*,*intro*]:
t1 ∈ *trm* ⟹ *t2* ∈ *trm* ⟹ *SS t1 t2* ∈ *fmla*
⟨*proof*⟩

lemma *Fvars_SS*[*simp*]: *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹ *yy* ∉ *FvarsT t1* ⟹
Fvars (*SS t1 t2*) = *FvarsT t1* ∪ *FvarsT t2*
⟨*proof*⟩

lemma [*simp*]:
m ∈ *num* ⟹ *p* ∈ *num* ⟹ *subst* (*SS m* (*Var yy*)) *p yy* = *SS m p*
m ∈ *num* ⟹ *subst* (*SS m* (*Var yy′*)) (*Var yy*) *yy′* = *SS m* (*Var yy*)
m ∈ *num* ⟹ *p* ∈ *num* ⟹ *subst* (*SS m* (*Var yy′*)) *p yy′* = *SS m p*
m ∈ *num* ⟹ *p* ∈ *num* ⟹ *subst* (*SS m* (*Var yy′*)) *p yy* = *SS m* (*Var yy′*)

m ∈ *num* ⟹ *subst* (*SS* (*Var xx*) (*Var yy*)) *m xx* = *SS m* (*Var yy*)
⟨*proof*⟩

end — context *Repr_SelfSoftSubst*

3.4 Clean Representability of the "Proof-of" Relation

For the proof-of relation, we must assume a stronger version of representability, namely clean representability on the first argument, which is dedicated to encoding the proof component. The property asks that the representation predicate is provably false on numerals that do not encode proofs; it would hold trivially for surjective proof encodings.

Cleanness is not a standard concept in the literature – we have introduced it in our CADE 2019 paper [1].

locale *CleanRepr_Proofs* =
Encode_Proofs
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv bprv
  enc
  fls
  dsj
  proof prfOf
  encPf
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *num*
and *eql cnj imp all exi*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *fls dsj*
and *proof* :: *'proof set* **and** *prfOf*
and *encPf*
+
fixes *Pf* :: *'fmla*
assumes
Pf[*simp*,*intro*!]: *Pf* ∈ *fmla*
and
Fvars_Pf[*simp*]: *Fvars Pf* = {*yy*,*xx*}
and
prfOf_Pf:
⋀ *prf φ*.
  let PPf = (λ *t1 t2*. *psubst Pf* [(*t1*,*yy*), (*t2*,*xx*)]) *in*
  (*prf* ∈ *proof* ∧ *φ* ∈ *fmla* ∧ *Fvars φ* = {} ⟶
   prfOf prf φ
   ⟶
   bprv (*PPf* (*encPf prf*) ⟨*φ*⟩))
and
not_prfOf_Pf:
⋀ *prf φ*.
  let PPf = (λ *t1 t2*. *psubst Pf* [(*t1*,*yy*), (*t2*,*xx*)]) *in*
  (*prf* ∈ *proof* ∧ *φ* ∈ *fmla* ∧ *Fvars φ* = {} ⟶
   ¬ *prfOf prf φ*
   ⟶
   bprv (*neg* (*PPf* (*encPf prf*) ⟨*φ*⟩)))

and
Clean_Pf_encPf:
$\bigwedge p\ \varphi.\ let\ PPf = (\lambda\ t1\ t2.\ psubst\ Pf\ [(t1,yy),\ (t2,xx)])\ in$
$p \in num \wedge \varphi \in fmla \wedge Fvars\ \varphi = \{\} \longrightarrow p \notin encPf\ `\ proof \longrightarrow bprv\ (neg\ (PPf\ p\ \langle\varphi\rangle))$
begin

PPf is the instantiation combinator of Pf:

definition *PPf* **where** $PPf \equiv \lambda\ t1\ t2.\ psubst\ Pf\ [(t1,yy),\ (t2,xx)]$

lemma *prfOf_PPf*:
assumes $prf \in proof\ \varphi \in fmla\ Fvars\ \varphi = \{\}\ prfOf\ prf\ \varphi$
shows $bprv\ (PPf\ (encPf\ prf)\ \langle\varphi\rangle)$
⟨*proof*⟩

lemma *not_prfOf_PPf*:
assumes $prf \in proof\ \varphi \in fmla\ Fvars\ \varphi = \{\}\ \neg\ prfOf\ prf\ \varphi$
shows $bprv\ (neg\ (PPf\ (encPf\ prf)\ \langle\varphi\rangle))$
⟨*proof*⟩

lemma *Clean_PPf_encPf*:
assumes $\varphi \in fmla\ Fvars\ \varphi = \{\}$ **and** $p \in num\ p \notin encPf\ `\ proof$
shows $bprv\ (neg\ (PPf\ p\ \langle\varphi\rangle))$
⟨*proof*⟩

lemma *PPf*[*simp*,*intro*!]: $t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow xx \notin FvarsT\ t1 \Longrightarrow PPf\ t1\ t2 \in fmla$
⟨*proof*⟩

lemma *PPf_def2*: $t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow xx \notin FvarsT\ t1 \Longrightarrow$
$PPf\ t1\ t2 = subst\ (subst\ Pf\ t1\ yy)\ t2\ xx$
⟨*proof*⟩

lemma *Fvars_PPf*[*simp*]:
$t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow xx \notin FvarsT\ t1 \Longrightarrow$
$Fvars\ (PPf\ t1\ t2) = FvarsT\ t1 \cup FvarsT\ t2$
⟨*proof*⟩

lemma [*simp*]:
$n \in num \Longrightarrow subst\ (PPf\ (Var\ yy)\ (Var\ xx))\ n\ xx = PPf\ (Var\ yy)\ n$
$m \in num \Longrightarrow n \in num \Longrightarrow subst\ (PPf\ (Var\ yy)\ m)\ n\ yy = PPf\ n\ m$
$n \in num \Longrightarrow subst\ (PPf\ (Var\ yy)\ (Var\ xx))\ n\ yy = PPf\ n\ (Var\ xx)$
$m \in num \Longrightarrow n \in num \Longrightarrow subst\ (PPf\ m\ (Var\ xx))\ n\ xx = PPf\ m\ n$
$m \in num \Longrightarrow subst\ (PPf\ (Var\ zz)\ (Var\ xx'))\ m\ zz = PPf\ m\ (Var\ xx')$
$m \in num \Longrightarrow n \in num \Longrightarrow subst\ (PPf\ m\ (Var\ xx'))\ n\ xx' = PPf\ m\ n$
$n \in num \Longrightarrow subst\ (PPf\ (Var\ zz)\ (Var\ xx'))\ n\ xx' = PPf\ (Var\ zz)\ n$
$m \in num \Longrightarrow n \in num \Longrightarrow subst\ (PPf\ (Var\ zz)\ n)\ m\ zz = PPf\ m\ n$
⟨*proof*⟩

lemma *B_consistent_prfOf_iff_PPf*:
$B.consistent \Longrightarrow prf \in proof \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \longrightarrow prfOf\ prf\ \varphi \longleftrightarrow bprv\ (PPf\ (encPf\ prf)\ \langle\varphi\rangle)$
⟨*proof*⟩

lemma *consistent_prfOf_iff_PPf*:
$consistent \Longrightarrow prf \in proof \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \longrightarrow prfOf\ prf\ \varphi \longleftrightarrow bprv\ (PPf\ (encPf\ prf)\ \langle\varphi\rangle)$
⟨*proof*⟩

end — context *CleanRepr_Proofs*

Chapter 4

Diagonalization

This theory proves abstract versions of the diagonalization lemma, with both hard and soft substitution.

4.1 Alternative Diagonalization via Self-Substitution

Assuming representability of the diagonal instance of the substitution function, we prove the standard diagonalization lemma. More precisely, we show that it applies to any logic that – embeds intuitionistic first-order logic over numerals – has a countable number of formulas – has formula self-substitution representable

context *Repr_SelfSubst*
begin

theorem *diagonalization*:
  assumes φ[*simp*,*intro*!]: $\varphi \in$ *fmla Fvars* $\varphi = \{xx\}$
  shows $\exists\ \psi.\ \psi \in$ *fmla* $\wedge$ *Fvars* $\psi = \{\} \wedge$ *bprv* (*eqv* ψ (*subst* φ $\langle\psi\rangle$ *xx*))
⟨*proof*⟩

Making this existential into a function.

definition *diag* :: *′fmla* ⇒ *′fmla* **where**
  diag $\varphi \equiv$ *SOME* $\psi.\ \psi \in$ *fmla* $\wedge$ *Fvars* $\psi = \{\} \wedge$ *bprv* (*eqv* ψ (*subst* φ $\langle\psi\rangle$ *xx*))

theorem *diag_everything*:
  assumes $\varphi \in$ *fmla* **and** *Fvars* $\varphi = \{xx\}$
  shows *diag* $\varphi \in$ *fmla* $\wedge$ *Fvars* (*diag* φ) $= \{\} \wedge$ *bprv* (*eqv* (*diag* φ) (*subst* φ ⟨*diag* φ⟩ *xx*))
  ⟨*proof*⟩

lemmas *diag*[*simp*] = *diag_everything*[*THEN conjunct1*]
lemmas *Fvars_diag*[*simp*] = *diag_everything*[*THEN conjunct2*, *THEN conjunct1*]
lemmas *bprv_diag_eqv* = *diag_everything*[*THEN conjunct2*, *THEN conjunct2*]

end — context *Repr_SelfSubst*

4.2 Alternative Diagonalization via Soft Self-Substitution

context *Repr_SelfSoftSubst*
begin

theorem *diagonalization*:
  assumes φ[*simp*,*intro*!]: $\varphi \in$ *fmla Fvars* $\varphi = \{xx\}$

shows $\exists\ \psi.\ \psi \in \mathit{fmla} \wedge \mathit{Fvars}\ \psi = \{\} \wedge \mathit{bprv}\ (\mathit{eqv}\ \psi\ (\mathit{subst}\ \varphi\ \langle\psi\rangle\ \mathit{xx}))$
$\langle \mathit{proof} \rangle$

Making this existential into a function.

definition $\mathit{diag} :: {}'\mathit{fmla} \Rightarrow {}'\mathit{fmla}$ **where**
$\mathit{diag}\ \varphi \equiv \mathit{SOME}\ \psi.\ \psi \in \mathit{fmla} \wedge \mathit{Fvars}\ \psi = \{\} \wedge \mathit{bprv}\ (\mathit{eqv}\ \psi\ (\mathit{subst}\ \varphi\ \langle\psi\rangle\ \mathit{xx}))$

theorem *diag_everything*:
assumes $\varphi \in \mathit{fmla}$ **and** $\mathit{Fvars}\ \varphi = \{\mathit{xx}\}$
shows $\mathit{diag}\ \varphi \in \mathit{fmla} \wedge \mathit{Fvars}\ (\mathit{diag}\ \varphi) = \{\} \wedge \mathit{bprv}\ (\mathit{eqv}\ (\mathit{diag}\ \varphi)\ (\mathit{subst}\ \varphi\ \langle \mathit{diag}\ \varphi\rangle\ \mathit{xx}))$
$\langle \mathit{proof} \rangle$

lemmas *diag*[*simp*] = *diag_everything*[*THEN conjunct1*]
lemmas *Fvars_diag*[*simp*] = *diag_everything*[*THEN conjunct2*, *THEN conjunct1*]
lemmas *prv_diag_eqv* = *diag_everything*[*THEN conjunct2*, *THEN conjunct2*]

end — context *Repr_SelfSoftSubst*

Chapter 5

The Hilbert-Bernays-Löb (HBL) Derivability Conditions

5.1 First Derivability Condition

locale *HBL1* =
Encode
  var trm fmla Var FvarsT substT Fvars subst
  num
  enc
+
Deduct2
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv bprv
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *num*
and *eql cnj imp all exi*
and *prv bprv*
and *enc* (‹⟨_⟩›)
+

fixes *P* :: *'fmla*
assumes
P[*intro*!,*simp*]: $P \in fmla$
and
Fvars_P[*simp*]: $Fvars\ P = \{xx\}$
and
HBL1: $\bigwedge\varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow prv\ \varphi \Longrightarrow bprv\ (subst\ P\ \langle\varphi\rangle\ xx)$
begin

definition *PP* **where** $PP \equiv \lambda t.\ subst\ P\ t\ xx$

lemma *PP*[*simp*]: $\bigwedge t.\ t \in trm \Longrightarrow PP\ t \in fmla$
  ⟨*proof*⟩

lemma *Fvars_PP*[*simp*]: $\bigwedge t.\ t \in trm \Longrightarrow Fvars\ (PP\ t) = FvarsT\ t$
  ⟨*proof*⟩

lemma [*simp*]:
$n \in num \Longrightarrow subst\ (PP\ (Var\ yy))\ n\ yy = PP\ n$
$n \in num \Longrightarrow subst\ (PP\ (Var\ xx))\ n\ xx = PP\ n$
⟨*proof*⟩

lemma *HBL1_PP*:
$\varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow prv\ \varphi \Longrightarrow bprv\ (PP\ \langle\varphi\rangle)$
⟨*proof*⟩

end — context *HBL1*

5.2 Connections between Proof Representability, First Derivability Condition, and Its Converse

context *CleanRepr_Proofs*
begin

Defining *P*, the internal notion of provability, from *Pf* (in its predicate form *PPf*), the internal notion of "proof-of". NB: In the technical sense of the term "represents", we have that *Pf* represents *pprv*, whereas *P* will not represent *prv*, but satisfy a weaker condition (weaker than weak representability), namely HBL1.

5.2.1 HBL1 from "proof-of" representability

definition P :: *'fmla* **where** $P \equiv exi\ yy\ (PPf\ (Var\ yy)\ (Var\ xx))$

lemma *P*[*simp*,*intro*!]: $P \in fmla$ **and** *Fvars_P*[*simp*]: $Fvars\ P = \{xx\}$
⟨*proof*⟩

We infer HBL1 from Pf representing prv:

lemma *HBL1*:
assumes φ: $\varphi \in fmla\ Fvars\ \varphi = \{\}$ **and** $p\varphi$: $prv\ \varphi$
shows $bprv\ (subst\ P\ \langle\varphi\rangle\ xx)$
⟨*proof*⟩

This is used in several places, including for the hard half of Gödel's First and the truth of Gödel formulas (and also for the Rosser variants of these).

lemma *not_prv_prv_neg_PPf*:
assumes [*simp*]: $\varphi \in fmla\ Fvars\ \varphi = \{\}$ **and** *p*: $\neg\ prv\ \varphi$ **and** *n*[*simp*]: $n \in num$
shows $bprv\ (neg\ (PPf\ n\ \langle\varphi\rangle))$
⟨*proof*⟩

lemma *consistent_not_prv_not_prv_PPf*:
assumes *c*: *consistent*
and *0*[*simp*]: $\varphi \in fmla\ Fvars\ \varphi = \{\}\ \neg\ prv\ \varphi\ n \in num$
shows $\neg\ bprv\ (PPf\ n\ \langle\varphi\rangle)$
⟨*proof*⟩

end — context *CleanRepr_Proofs*

The inference of HBL1 from "proof-of" representability, in locale form:

sublocale *CleanRepr_Proofs* < *wrepr*: *HBL1*
where $P = P$
⟨*proof*⟩

5.2.2 Sufficient condition for the converse of HBL1

context *CleanRepr_Proofs*
begin

lemma *PP_PPf*:
assumes $\varphi \in$ *fmla*
shows *wrepr.PP* $\langle\varphi\rangle$ = *exi yy* (*PPf* (*Var yy*) $\langle\varphi\rangle$)
⟨*proof*⟩

The converse of HLB1 condition follows from (the standard notion of) ω-consistency for *bprv* and strong representability of proofs:

lemma *ωconsistentStd1_HBL1_rev*:
assumes *oc*: *B.ωconsistentStd1*
and φ[*simp*]: $\varphi \in$ *fmla* *Fvars* φ = {}
and *iPP*: *bprv* (*wrepr.PP* $\langle\varphi\rangle$)
shows *prv* φ
⟨*proof*⟩

end — context *CleanRepr_Proofs*

5.3 Second and Third Derivability Conditions

These are only needed for Gödel's Second.

locale *HBL1_2_3* =
HBL1
var trm fmla Var FvarsT substT Fvars subst
num
eql cnj imp all exi
prv bprv
enc
P
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *num*
and *eql cnj imp all exi*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *P*
\+
assumes
HBL2: $\bigwedge\varphi\ \chi.\ \varphi \in$ *fmla* $\Longrightarrow \chi \in$ *fmla* $\Longrightarrow$ *Fvars* φ = {} $\Longrightarrow$ *Fvars* χ = {} $\Longrightarrow$
bprv (*imp* (*cnj* (*PP* $\langle\varphi\rangle$) (*PP* ⟨*imp* $\varphi\ \chi$⟩))
(*PP* $\langle\chi\rangle$))
and
HBL3: $\bigwedge\varphi.\ \varphi \in$ *fmla* $\Longrightarrow$ *Fvars* φ = {} $\Longrightarrow$ *bprv* (*imp* (*PP* $\langle\varphi\rangle$) (*PP* ⟨*PP* $\langle\varphi\rangle$⟩))
begin

The implicational form of HBL2:

lemma *HBL2_imp*:
$\bigwedge\varphi\ \chi.\ \varphi \in$ *fmla* $\Longrightarrow \chi \in$ *fmla* $\Longrightarrow$ *Fvars* φ = {} $\Longrightarrow$ *Fvars* χ = {} $\Longrightarrow$
bprv (*imp* (*PP* ⟨*imp* $\varphi\ \chi$⟩) (*imp* (*PP* $\langle\varphi\rangle$) (*PP* $\langle\chi\rangle$)))
⟨*proof*⟩

... and its weaker, "detached" version:

lemma *HBL2_imp2*:

assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla* *Fvars* $\varphi = \{\}$ *Fvars* $\chi = \{\}$
assumes *bprv* (*PP* ⟨*imp* φ χ⟩)
shows *bprv* (*imp* (*PP* ⟨φ⟩) (*PP* ⟨χ⟩))
⟨*proof*⟩

end — context *HBL1_2_3*

Chapter 6

Gödel Formulas

Gödel formulas are defined by diagonalizing the negation of the provability predicate.

locale *Goedel_Form* =
— Assuming the *fls* (False) connective gives us negation:
Deduct2_with_False
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  num
  prv bprv
+
Repr_SelfSubst
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv bprv
  enc
  S
+
HBL1
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv bprv
  enc
  P
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var num FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *S*
and *P*
begin

The Gödel formula. NB, we speak of "the" Gödel formula because the diagonalization function makes a choice.

definition φG :: *'fmla* **where** $\varphi G \equiv$ *diag* (*neg P*)

lemma φG[*simp*,*intro*!]: $\varphi G \in$ *fmla*
and

Fvars_φG[*simp*]: *Fvars φG* = {}
⟨*proof*⟩

lemma *bprv_φG_eqv*:
bprv (*eqv φG* (*neg* (*PP* ⟨*φG*⟩)))
⟨*proof*⟩

lemma *prv_φG_eqv*:
prv (*eqv φG* (*neg* (*PP* ⟨*φG*⟩)))
⟨*proof*⟩

end — context *Goedel_Form*

Adding cleanly representable proofs to the assumptions behind Gödel formulas:

locale *Goedel_Form_Proofs* =
Repr_SelfSubst
var trm fmla Var FvarsT substT Fvars subst
num
eql cnj imp all exi
prv bprv
enc
S
\+
CleanRepr_Proofs
var trm fmla Var FvarsT substT Fvars subst
num
eql cnj imp all exi
prv bprv
enc
fls
dsj
proof prfOf
encPf
Pf
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst num*
and *eql cnj imp all exi*
and *fls*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *S*
and *dsj*
and *proof* :: *'proof set* **and** *prfOf encPf*
and *Pf*

... and extending the sublocale relationship *CleanRepr_Proofs* < *HBL1*:

sublocale *Goedel_Form_Proofs* < *Goedel_Form* **where** *P* = *P* ⟨*proof*⟩

context *Goedel_Form_Proofs*
begin

lemma *bprv_φG_eqv_not_exi_PPf*:
bprv (*eqv φG* (*neg* (*exi yy* (*PPf* (*Var yy*) ⟨*φG*⟩))))
⟨*proof*⟩

lemma *prv_φG_eqv_not_exi_PPf*:

prv (*eqv* φG (*neg* (*exi* *yy* (*PPf* (*Var* *yy*) $\langle\varphi G\rangle$)))))
$\langle proof \rangle$

lemma *bprv_φG_eqv_all_not_PPf*:
bprv (*eqv* φG (*all* *yy* (*neg* (*PPf* (*Var* *yy*) $\langle\varphi G\rangle$)))))
$\langle proof \rangle$

lemma *prv_φG_eqv_all_not_PPf*:
prv (*eqv* φG (*all* *yy* (*neg* (*PPf* (*Var* *yy*) $\langle\varphi G\rangle$)))))
$\langle proof \rangle$

lemma *bprv_eqv_all_not_PPf_imp_φG*:
bprv (*imp* (*all* *yy* (*neg* (*PPf* (*Var* *yy*) $\langle\varphi G\rangle$)))) φG)
$\langle proof \rangle$

lemma *prv_eqv_all_not_PPf_imp_φG*:
prv (*imp* (*all* *yy* (*neg* (*PPf* (*Var* *yy*) $\langle\varphi G\rangle$)))) φG)
$\langle proof \rangle$

end — context *Goedel_Form_Proofs*

Chapter 7

Standard Models with Two Provability Relations

```
locale Minimal_Truth_Soundness_Proof_Repr =
CleanRepr_Proofs
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv bprv
  enc
  fls
  dsj
  proof prfOf
  encPf
  Pf
+ — The label "B" stands for "basic", as a reminder that soundness refers to the basic provability relation:
B: Minimal_Truth_Soundness
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
  bprv
  isTrue
for
var :: 'var set and trm :: 'trm set and fmla :: 'fmla set
and Var FvarsT substT Fvars subst
and eql cnj imp all exi
and fls
and dsj
and num
and prv bprv
and isTrue
and enc (‹⟨_⟩›)
and proof :: 'proof set and prfOf
and encPf Pf
begin
```

lemmas *prfOf_iff_PPf* = *B_consistent_prfOf_iff_PPf*[*OF B.consistent*]

The provability predicate is decided by basic provability on encodings:

lemma *isTrue_prv_PPf_prf_or_neg*:

$prf \in proof \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow$
$\quad bprv\ (PPf\ (encPf\ prf)\ \langle\varphi\rangle) \vee bprv\ (neg\ (PPf\ (encPf\ prf)\ \langle\varphi\rangle))$
⟨*proof*⟩

... hence that predicate is complete w.r.t. truth:

lemma *isTrue_PPf_Implies_bprv_PPf*:
$prf \in proof \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow$
$\quad isTrue\ (PPf\ (encPf\ prf)\ \langle\varphi\rangle) \Longrightarrow bprv\ (PPf\ (encPf\ prf)\ \langle\varphi\rangle)$
⟨*proof*⟩

... and thanks cleanness we can replace encoded proofs with arbitrary numerals in the completeness property:

lemma *isTrue_implies_bprv_PPf*:
assumes [*simp*]: $n \in num\ \varphi \in fmla\ Fvars\ \varphi = \{\}$
and *iT*: $isTrue\ (PPf\ n\ \langle\varphi\rangle)$
shows $bprv\ (PPf\ n\ \langle\varphi\rangle)$
⟨*proof*⟩

... in fact, by *Minimal_Truth_Soundness* we even have an iff:

lemma *isTrue_iff_bprv_PPf*:
$\bigwedge n\ \varphi.\ n \in num \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow isTrue\ (PPf\ n\ \langle\varphi\rangle) \longleftrightarrow bprv\ (PPf\ n\ \langle\varphi\rangle)$
⟨*proof*⟩

Truth of the provability representation implies provability (TIP):

lemma *TIP*:
assumes φ[*simp*]: $\varphi \in fmla\ Fvars\ \varphi = \{\}$
and *iPP*: $isTrue\ (wrepr.PP\ \langle\varphi\rangle)$
shows $prv\ \varphi$
⟨*proof*⟩

The reverse HBL1 – now without the ω-consistency assumption which holds here thanks to our truth-in-standard-model assumption:

lemmas *HBL1_rev* = *ωconsistentStd1_HBL1_rev*[*OF B.ωconsistentStd1*]

Note that the above would also follow by *Minimal_Truth_Soundness* from TIP:

lemma *TIP_implies_HBL1_rev*:
assumes *TIP*: $\forall\varphi.\ \varphi \in fmla \wedge Fvars\ \varphi = \{\} \wedge isTrue\ (wrepr.PP\ \langle\varphi\rangle) \longrightarrow prv\ \varphi$
shows $\forall\varphi.\ \varphi \in fmla \wedge Fvars\ \varphi = \{\} \wedge bprv\ (wrepr.PP\ \langle\varphi\rangle) \longrightarrow prv\ \varphi$
⟨*proof*⟩

end — context *Minimal_Truth_Soundness_Proof_Repr*

7.1 Proof recovery from *HBL1_iff*

locale *Minimal_Truth_Soundness_HBL1iff_Compl_Pf* =
HBL1
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv bprv
  enc
  P
+
B : *Minimal_Truth_Soundness*
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi

fls
dsj
num
bprv
isTrue
+
Deduct_with_False_Disj
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
prv
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *enc* (‹⟨_⟩›)
and *prv bprv*
and *P*
and *isTrue*
+
fixes *Pf* :: *'fmla*
assumes
— *Pf* is a formula with free variables *xx yy*:
Pf[*simp*,*intro*!]: *Pf* ∈ *fmla*
and
Fvars_Pf[*simp*]: *Fvars Pf* = {*yy*,*xx*}
and
— *P* relates to *Pf* internally (inside basic provability) just like a *prv* and a *prfOf* would relate—via an existential:
P_Pf:
φ ∈ *fmla* ⟹ *Fvars* φ = {} ⟹
let PPf = (λ *t1 t2*. *psubst Pf* [(*t1*,*yy*), (*t2*,*xx*)]) *in*
bprv (*eqv* (*subst P* ⟨φ⟩ *xx*) (*exi yy* (*PPf* (*Var yy*) ⟨φ⟩)))
assumes
— We assume both *HBL1* and *HBL1_rev*, i.e., an iff version:
HBL1_iff: ⋀ φ. φ ∈ *fmla* ⟹ *Fvars* φ = {} ⟹ *bprv* (*PP* ⟨φ⟩) ⟷ *prv* φ
and
Compl_Pf:
⋀ *n* φ. *n* ∈ *num* ⟹ φ ∈ *fmla* ⟹ *Fvars* φ = {} ⟹
let PPf = (λ *t1 t2*. *psubst Pf* [(*t1*,*yy*), (*t2*,*xx*)]) *in*
isTrue (*PPf n* ⟨φ⟩) ⟶ *bprv* (*PPf n* ⟨φ⟩)
begin

definition *PPf* **where** *PPf* ≡ λ *t1 t2*. *psubst Pf* [(*t1*,*yy*), (*t2*,*xx*)]

lemma *PP_deff*: *PP t* = *subst P t xx* ⟨*proof*⟩

lemma *PP_PPf_eqv*:
φ ∈ *fmla* ⟹ *Fvars* φ = {} ⟹ *bprv* (*eqv* (*PP* ⟨φ⟩) (*exi yy* (*PPf* (*Var yy*) ⟨φ⟩)))
⟨*proof*⟩

lemma *PPf*[*simp*,*intro*!]: $t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow xx \notin FvarsT\ t1 \Longrightarrow PPf\ t1\ t2 \in fmla$
⟨*proof*⟩

lemma *PPf_def2*: $t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow xx \notin FvarsT\ t1 \Longrightarrow$
$PPf\ t1\ t2 = subst\ (subst\ Pf\ t1\ yy)\ t2\ xx$
⟨*proof*⟩

lemma *Fvars_PPf*[*simp*]:
$t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow xx \notin FvarsT\ t1 \Longrightarrow Fvars\ (PPf\ t1\ t2) = FvarsT\ t1 \cup FvarsT\ t2$
⟨*proof*⟩

lemma [*simp*]:
$n \in num \Longrightarrow subst\ (PPf\ (Var\ yy)\ (Var\ xx))\ n\ xx = PPf\ (Var\ yy)\ n$
$m \in num \Longrightarrow n \in num \Longrightarrow subst\ (PPf\ (Var\ yy)\ m)\ n\ yy = PPf\ n\ m$
$n \in num \Longrightarrow subst\ (PPf\ (Var\ yy)\ (Var\ xx))\ n\ yy = PPf\ n\ (Var\ xx)$
$m \in num \Longrightarrow n \in num \Longrightarrow subst\ (PPf\ m\ (Var\ xx))\ n\ xx = PPf\ m\ n$
$m \in num \Longrightarrow subst\ (PPf\ (Var\ zz)\ (Var\ xx'))\ m\ zz = PPf\ m\ (Var\ xx')$
$m \in num \Longrightarrow n \in num \Longrightarrow subst\ (PPf\ m\ (Var\ xx'))\ n\ xx' = PPf\ m\ n$
$n \in num \Longrightarrow subst\ (PPf\ (Var\ zz)\ (Var\ xx'))\ n\ xx' = PPf\ (Var\ zz)\ n$
$m \in num \Longrightarrow n \in num \Longrightarrow subst\ (PPf\ (Var\ zz)\ n)\ m\ zz = PPf\ m\ n$
⟨*proof*⟩

lemma *PP_PPf*:
assumes $\varphi \in fmla\ Fvars\ \varphi = \{\}$ **shows** $bprv\ (PP\ \langle\varphi\rangle) \longleftrightarrow bprv\ (exi\ yy\ (PPf\ (Var\ yy)\ \langle\varphi\rangle))$
⟨*proof*⟩

lemma *isTrue_implies_bprv_PPf*:
$\bigwedge n\ \varphi.\ n \in num \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow$
$isTrue\ (PPf\ n\ \langle\varphi\rangle) \Longrightarrow bprv\ (PPf\ n\ \langle\varphi\rangle)$
⟨*proof*⟩

lemma *isTrue_iff_bprv_PPf*:
$\bigwedge n\ \varphi.\ n \in num \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow isTrue\ (PPf\ n\ \langle\varphi\rangle) \longleftrightarrow bprv\ (PPf\ n\ \langle\varphi\rangle)$
⟨*proof*⟩

Preparing to instantiate this "proof recovery" alternative into our mainstream locale hierarchy, which assumes proofs. We define the "missing" proofs to be numerals, we encode them as the identity, and we "copy" *prfOf* from the corresponding predicate one-level-up, *PPf*:

definition *proof* :: *'trm set* **where** [*simp*]: $proof = num$
definition *prfOf* :: $'trm \Rightarrow 'fmla \Rightarrow bool$ **where**
$prfOf\ n\ \varphi \equiv bprv\ (PPf\ n\ \langle\varphi\rangle)$
definition *encPf* :: $'trm \Rightarrow 'trm$ **where** [*simp*]: $encPf \equiv id$

lemma *prv_exi_PPf_iff_isTrue*:
assumes [*simp*]: $\varphi \in fmla\ Fvars\ \varphi = \{\}$
shows $bprv\ (exi\ yy\ (PPf\ (Var\ yy)\ \langle\varphi\rangle)) \longleftrightarrow isTrue\ (exi\ yy\ (PPf\ (Var\ yy)\ \langle\varphi\rangle))$ (**is** $?L \longleftrightarrow ?R$)
⟨*proof*⟩

lemma *isTrue_exi_iff*:
assumes [*simp*]: $\varphi \in fmla\ Fvars\ \varphi = \{\}$
shows $isTrue\ (exi\ yy\ (PPf\ (Var\ yy)\ \langle\varphi\rangle)) \longleftrightarrow (\exists\, n{\in}num.\ isTrue\ (PPf\ n\ \langle\varphi\rangle))$ (**is** $?L \longleftrightarrow ?R$)
⟨*proof*⟩

lemma *prv_prfOf*:
assumes $\varphi \in fmla\ Fvars\ \varphi = \{\}$

shows *prv* $\varphi \longleftrightarrow (\exists\, n{\in}num.\ prfOf\ n\ \varphi)$
⟨*proof*⟩

lemma *prfOf_prv_Pf*:
assumes $n \in num$ **and** $\varphi \in fmla$ $Fvars\ \varphi = \{\}$ **and** $prfOf\ n\ \varphi$
shows *bprv* $(psubst\ Pf\ [(n,\ yy),\ (\langle\varphi\rangle,\ xx)])$
⟨*proof*⟩

lemma *isTrue_exi_iff_PP*:
assumes [*simp*]: $\varphi \in fmla$ $Fvars\ \varphi = \{\}$
shows *isTrue* $(PP\ \langle\varphi\rangle) \longleftrightarrow (\exists\, n{\in}num.\ isTrue\ (PPf\ n\ \langle\varphi\rangle))$
⟨*proof*⟩

lemma *bprv_compl_isTrue_PP_enc*:
assumes *1*: $\varphi \in fmla$ $Fvars\ \varphi = \{\}$ **and** *2*: *isTrue* $(PP\ \langle\varphi\rangle)$
shows *bprv* $(PP\ \langle\varphi\rangle)$
⟨*proof*⟩

lemma *TIP*:
assumes *1*: $\varphi \in fmla$ $Fvars\ \varphi = \{\}$ **and** *2*: *isTrue* $(PP\ \langle\varphi\rangle)$
shows *prv* φ
⟨*proof*⟩

end — context *Minimal_Truth_Soundness_HBL1iff_Compl_Pf*

locale *Minimal_Truth_Soundness_HBL1iff_Compl_Pf_Compl_NegPf* =
Minimal_Truth_Soundness_HBL1iff_Compl_Pf
+
assumes
Compl_NegPf:
$\bigwedge n\ \varphi.\ n \in num \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow$
let $PPf = (\lambda\ t1\ t2.\ psubst\ Pf\ [(t1,yy),\ (t2,xx)])$ *in*
isTrue $(B.neg\ (PPf\ n\ \langle\varphi\rangle)) \longrightarrow bprv\ (B.neg\ (PPf\ n\ \langle\varphi\rangle))$
begin

lemma *isTrue_implies_prv_neg_PPf*:
$\bigwedge n\ \varphi.\ n \in num \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow$
isTrue $(B.neg\ (PPf\ n\ \langle\varphi\rangle)) \Longrightarrow bprv\ (B.neg\ (PPf\ n\ \langle\varphi\rangle))$
⟨*proof*⟩

lemma *isTrue_iff_prv_neg_PPf*:
$\bigwedge n\ \varphi.\ n \in num \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow isTrue\ (B.neg\ (PPf\ n\ \langle\varphi\rangle)) \longleftrightarrow bprv\ (B.neg\ (PPf\ n\ \langle\varphi\rangle))$
⟨*proof*⟩

lemma *prv_PPf_decide*:
assumes [*simp*]: $n \in num$ $\varphi \in fmla$ $Fvars\ \varphi = \{\}$
and *np*: $\neg\ bprv\ (PPf\ n\ \langle\varphi\rangle)$
shows *bprv* $(B.neg\ (PPf\ n\ \langle\varphi\rangle))$
⟨*proof*⟩

lemma *not_prfOf_prv_neg_Pf*:
assumes *nφ*: $n \in num$ $\varphi \in fmla$ $Fvars\ \varphi = \{\}$ **and** $\neg\ prfOf\ n\ \varphi$
shows *bprv* $(B.neg\ (psubst\ Pf\ [(n,\ yy),\ (\langle\varphi\rangle,\ xx)]))$
⟨*proof*⟩

end — context *Minimal_Truth_Soundness_HBL1iff_Compl_Pf_Compl_NegPf*

sublocale *Minimal_Truth_Soundness_HBL1iff_Compl_Pf_Compl_NegPf* <
repr: *CleanRepr_Proofs*

where *proof* = *proof* **and** *prfOf* = *prfOf* **and** *encPf* = *encPf*
⟨*proof*⟩

sublocale *Minimal_Truth_Soundness_HBL1iff_Compl_Pf_Compl_NegPf* <
min_truth: *Minimal_Truth_Soundness_Proof_Repr*
where *proof* = *proof* **and** *prfOf* = *prfOf* **and** *encPf* = *encPf*
⟨*proof*⟩

locale *Minimal_Truth_Soundness_HBL1iff_prv_Compl_Pf* =
HBL1
var trm fmla Var FvarsT substT Fvars subst
num
eql cnj imp all exi
prv bprv
enc
P
+
B: *Minimal_Truth_Soundness*
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
bprv
isTrue
+
Deduct_with_False_Disj
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
prv
for
var :: *′var set* **and** *trm* :: *′trm set* **and** *fmla* :: *′fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *enc* (‹⟨_⟩›)
and *prv bprv*
and *P*
and *isTrue*
+
fixes *Pf* :: *′fmla*
assumes

Pf[*simp*,*intro*!]: *Pf* ∈ *fmla*
and
Fvars_Pf[*simp*]: *Fvars Pf* = {*yy*,*xx*}
and

P_Pf:
φ ∈ *fmla* ⟹
let PPf = (λ *t1 t2*. *psubst Pf* [(*t1*,*yy*), (*t2*,*xx*)]) *in*
bprv (*eqv* (*subst P* ⟨*φ*⟩ *xx*) (*exi yy* (*PPf* (*Var yy*) ⟨*φ*⟩)))
assumes

HBL1_rev_prv: ⋀ *φ*. *φ* ∈ *fmla* ⟹ *Fvars φ* = {} ⟹ *prv* (*PP* ⟨*φ*⟩) ⟹ *prv φ*
and
Compl_Pf:
⋀ *n φ*. *n* ∈ *num* ⟹ *φ* ∈ *fmla* ⟹ *Fvars φ* = {} ⟹
let PPf = (λ *t1 t2*. *psubst Pf* [(*t1*,*yy*), (*t2*,*xx*)]) *in*
isTrue (*PPf n* ⟨*φ*⟩) ⟶ *bprv* (*PPf n* ⟨*φ*⟩)
begin

lemma *HBL1_rev*:
assumes *f*: *φ* ∈ *fmla* **and** *fv*: *Fvars φ* = {} **and** *bp*: *bprv* (*PP* ⟨*φ*⟩)
shows *prv φ*
⟨*proof*⟩

lemma *HBL1_iff*: *φ* ∈ *fmla* ⟹ *Fvars φ* = {} ⟹ *bprv* (*PP* ⟨*φ*⟩) ⟷ *prv φ*
⟨*proof*⟩

lemma *HBL1_iff_prv*: *φ* ∈ *fmla* ⟹ *Fvars φ* = {} ⟹ *prv* (*PP* ⟨*φ*⟩) ⟷ *prv φ*
⟨*proof*⟩

end — context *Minimal_Truth_Soundness_HBL1iff_prv_Compl_Pf*

sublocale *Minimal_Truth_Soundness_HBL1iff_prv_Compl_Pf* <
mts_prv_mts: *Minimal_Truth_Soundness_HBL1iff_Compl_Pf* **where** *Pf* = *Pf*
⟨*proof*⟩

locale *Minimal_Truth_Soundness_HBL1iff_prv_Compl_Pf_Classical* =
Minimal_Truth_Soundness_HBL1iff_prv_Compl_Pf
+
assumes
— NB: we don't really need to assume classical reasoning (double negation) all throughout, but only for the provability predicate:
classical_P: ⋀ *φ*. *φ* ∈ *fmla* ⟹ *Fvars φ* = {} ⟹ *let PP* = (λ*t*. *subst P t xx*) *in*
prv (*B.neg* (*B.neg* (*PP* ⟨*φ*⟩))) ⟹ *prv* (*PP* ⟨*φ*⟩)
begin

lemma *classical_PP*: *φ* ∈ *fmla* ⟹ *Fvars φ* = {} ⟹ *prv* (*B.neg* (*B.neg* (*PP* ⟨*φ*⟩))) ⟹ *prv* (*PP* ⟨*φ*⟩)
⟨*proof*⟩

end — context *Minimal_Truth_Soundness_HBL1iff_prv_Compl_Pf_Classical*

Chapter 8

Abstract Formulations of Gödel's First Incompleteness Theorem

8.1 Proof-Theoretic Versions of Gödel's First

context *Goedel_Form*
begin

8.1.1 The easy half

First the "direct", positive formulation:

lemma *goedel_first_theEasyHalf_pos*:
assumes *prv* φG **shows** *prv fls*
⟨*proof*⟩

... then the more standard contrapositive formulation:

corollary *goedel_first_theEasyHalf*:
consistent $\Longrightarrow \neg$ *prv* φG
⟨*proof*⟩

end — context *Goedel_Form*

8.1.2 The hard half

The hard half needs explicit proofs:

context *Goedel_Form_Proofs* **begin**

lemma *goedel_first_theHardHalf*:
assumes *oc*: ω*consistent*
shows $\neg$ *prv* (*neg* φG)
⟨*proof*⟩

theorem *goedel_first*:
assumes ω*consistent*
shows $\neg$ *prv* $\varphi G \wedge \neg$ *prv* (*neg* φG)
⟨*proof*⟩

theorem *goedel_first_ex*:
assumes ω*consistent*
shows $\exists\ \varphi.\ \varphi \in$ *fmla* $\wedge \neg$ *prv* $\varphi \wedge \neg$ *prv* (*neg* φ)
⟨*proof*⟩

end — context *Goedel_Form_Proofs*

8.2 Model-Theoretic Versions of Gödel's First

The model-theoretic twist is that of additionally proving the truth of Gödel sentences.

8.2.1 First model-theoretic version

locale *Goedel_Form_Proofs_Minimal_Truth* =
Goedel_Form_Proofs
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  fls
  prv bprv
  enc
  S
  dsj
  proof prfOf encPf
  Pf
+
Minimal_Truth_Soundness
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
  bprv
  isTrue
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *S*
and *proof* :: *'proof set* **and** *prfOf encPf*
and *Pf*
and *isTrue*
begin

Recall that "consistent" and "ω-consistent" refer to *prv*, not to *bprv*.

theorem *isTrue_φG*:
  assumes *consistent*
  shows *isTrue* φG
⟨*proof*⟩

The "strong" form of Gödel's First (also asserting the truth of the Gödel sentences):

theorem *goedel_first_strong*:
$\omega consistent \Longrightarrow \neg\ prv\ \varphi G \wedge \neg\ prv\ (neg\ \varphi G) \wedge isTrue\ \varphi G$
  ⟨*proof*⟩

theorem *goedel_first_strong_ex*:
ωconsistent ⟹ ∃ *φ*. *φ* ∈ *fmla* ∧ ¬ *prv φ* ∧ ¬ *prv* (*neg φ*) ∧ *isTrue φ*
⟨*proof*⟩

end — context *Goedel_Form_Proofs_Minimal_Truth*

8.2.2 Second model-theoretic version

locale *Goedel_Form_Minimal_Truth_Soundness_HBL1iff_Compl_Pf* =
Goedel_Form
var trm fmla Var num
FvarsT substT Fvars subst
eql cnj imp all exi
fls
prv bprv
enc
S
P
+
Minimal_Truth_Soundness_HBL1iff_Compl_Pf
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
enc
prv bprv
P
isTrue
Pf
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *S*
and *isTrue*
and *P*
and *Pf*

locale *Goedel_Form_Minimal_Truth_Soundness_HBL1iff_Compl_Pf_Compl_NegPf* =
Goedel_Form_Minimal_Truth_Soundness_HBL1iff_Compl_Pf
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
prv bprv
enc
S
isTrue
P

Pf
+
Minimal_Truth_Soundness_HBL1iff_Compl_Pf_Compl_NegPf
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
enc
prv bprv
P
isTrue
Pf
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *S*
and *isTrue*
and *P*
and *Pf*
+
assumes *prv_ωconsistent*: *ωconsistent*

sublocale
Goedel_Form_Minimal_Truth_Soundness_HBL1iff_Compl_Pf_Compl_NegPf <
recover_proofs: *Goedel_Form_Proofs_Minimal_Truth*
where *prfOf* = *prfOf* **and** *proof* = *proof* **and** *encPf* = *encPf*
and *prv* = *prv* **and** *bprv* = *bprv*
⟨*proof*⟩

context *Goedel_Form_Minimal_Truth_Soundness_HBL1iff_Compl_Pf_Compl_NegPf* **begin**
thm *recover_proofs.goedel_first_strong*

end

8.3 Classical-Logic Versions of Gödel's First

8.3.1 First classical-logic version

locale *Goedel_Form_Classical_HBL1_rev_prv* =
Goedel_Form
var trm fmla Var num FvarsT substT Fvars subst
eql cnj imp all exi
fls
prv bprv
enc
S
P
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*

and *Var num FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *S*
and *P*
+
assumes
— NB: we don't really need to assume classical reasoning (double negation) for all formulas, but only for the provability predicate:
classical_P_prv: $\bigwedge \varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow let\ PP = (\lambda t.\ subst\ P\ t\ xx)\ in$
$prv\ (neg\ (neg\ (PP\ \langle\varphi\rangle))) \Longrightarrow prv\ (PP\ \langle\varphi\rangle)$
and
HBL1_rev_prv: $\bigwedge \varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow prv\ (PP\ \langle\varphi\rangle) \Longrightarrow prv\ \varphi$
begin

lemma *HBL1_rev*:
assumes *f*: $\varphi \in fmla$ **and** *fv*: $Fvars\ \varphi = \{\}$ **and** *bp*: $bprv\ (PP\ \langle\varphi\rangle)$
shows $prv\ \varphi$
⟨*proof*⟩

lemma *classical_PP_prv*: $\varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow prv\ (neg\ (neg\ (PP\ \langle\varphi\rangle))) \Longrightarrow prv\ (PP\ \langle\varphi\rangle)$
⟨*proof*⟩

lemma *HBL1_iff*: $\varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow bprv\ (PP\ \langle\varphi\rangle) \longleftrightarrow prv\ \varphi$
⟨*proof*⟩

lemma *HBL1_iff_prv*: $\varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow prv\ (PP\ \langle\varphi\rangle) \longleftrightarrow prv\ \varphi$
⟨*proof*⟩

lemma *goedel_first_theHardHalf_pos*:
assumes $prv\ (neg\ \varphi G)$ **shows** *prv fls*
⟨*proof*⟩

corollary *goedel_first_theHardHalf*:
$consistent \Longrightarrow \neg\ prv\ (neg\ \varphi G)$
⟨*proof*⟩

theorem *goedel_first_classic*:
assumes *consistent*
shows $\neg\ prv\ \varphi G \wedge \neg\ prv\ (neg\ \varphi G)$
⟨*proof*⟩

theorem *goedel_first_classic_ex*:
assumes *consistent*
shows $\exists\ \varphi.\ \varphi \in fmla \wedge \neg\ prv\ \varphi \wedge \neg\ prv\ (neg\ \varphi)$
⟨*proof*⟩

end — context *Goedel_Form_Classical_HBL1_rev_prv*

8.3.2 Second classical-logic version

locale *Goedel_Form_Classical_HBL1_rev_prv_Minimal_Truth_Soundness_TIP* =
Goedel_Form_Classical_HBL1_rev_prv
var trm fmla Var num FvarsT substT Fvars subst
eql cnj imp all exi
fls

prv bprv
enc
S
P
+
Minimal_Truth_Soundness
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
bprv
isTrue
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var num FvarsT substT Fvars subst*
and *eql cnj dsj imp all exi*
and *fls*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *S*
and *P*
and *isTrue*
+
assumes
— Truth of φ implies provability (TIP) of (the internal representation of) φ
TIP: $\bigwedge \varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow$
$let\ PP = (\lambda t.\ subst\ P\ t\ xx)\ in$
$isTrue\ (PP\ \langle\varphi\rangle) \Longrightarrow prv\ \varphi$
begin

lemma *TIP_PP*: $\varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow isTrue\ (PP\ \langle\varphi\rangle) \Longrightarrow prv\ \varphi$
⟨*proof*⟩

theorem *isTrue_φG*:
assumes *consistent*
shows *isTrue φG*
⟨*proof*⟩

theorem *goedel_first_classic_strong*: $consistent \Longrightarrow \neg\ prv\ \varphi G \wedge \neg\ prv\ (neg\ \varphi G) \wedge isTrue\ \varphi G$
⟨*proof*⟩

theorem *goedel_first_classic_strong_ex*:
$consistent \Longrightarrow \exists\ \varphi.\ \varphi \in fmla \wedge \neg\ prv\ \varphi \wedge \neg\ prv\ (neg\ \varphi) \wedge isTrue\ \varphi$
⟨*proof*⟩

end — context *Goedel_Form_Classical_HBL1_rev_prv_Minimal_Truth_Soundness_TIP*

8.3.3 Third classical-logic version

locale *Goedel_Form_Minimal_Truth_Soundness_HBL1iff_prv_Compl_Pf_Classical* =
Goedel_Form
var trm fmla Var num FvarsT substT Fvars subst
eql cnj imp all exi
fls
prv bprv
enc
S

P
+
Minimal_Truth_Soundness_HBL1iff_prv_Compl_Pf_Classical
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
enc
prv bprv
P
isTrue
Pf
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *S*
and *isTrue*
and *P*
and *Pf*

sublocale *Goedel_Form_Minimal_Truth_Soundness_HBL1iff_prv_Compl_Pf_Classical* <
recover_proofs: *Goedel_Form_Classical_HBL1_rev_prv_Minimal_Truth_Soundness_TIP* **where** *prv* = *prv* **and** *bprv* = *bprv*
⟨*proof*⟩

context *Goedel_Form_Minimal_Truth_Soundness_HBL1iff_prv_Compl_Pf_Classical* **begin**
thm *recover_proofs.goedel_first_classic_strong*
end — context *Goedel_Form_Minimal_Truth_Soundness_HBL1iff_prv_Compl_Pf_Classical*

Chapter 9

Rosser Formulas

The Rosser formula is a modification of the Gödel formula that is undecidable assuming consistency only (not ω-consistency).

locale *Rosser_Form* =
Deduct2_with_PseudoOrder
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
  prv bprv
  Lq
  +
Repr_Neg
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  num
  prv bprv
  enc
  N
  +
Repr_SelfSubst
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv bprv
  enc
  S
  +
HBL1
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv bprv
  enc
  P
  for
    var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
    and *Var FvarsT substT Fvars subst*
    and *num*
    and *eql cnj imp all exi*

and *fls*
and *prv bprv*
and *Lq*
and *dsj*
and *enc* (‹⟨_⟩›)
and *N S P*

locale *Rosser_Form_Proofs* =
Deduct2_with_PseudoOrder
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
prv bprv
Lq
+
Repr_Neg
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
num
prv bprv
enc
N
+
Repr_SelfSubst
var trm fmla Var FvarsT substT Fvars subst
num
eql cnj imp all exi
prv bprv
enc
S
+
CleanRepr_Proofs
var trm fmla Var FvarsT substT Fvars subst
num
eql cnj imp all exi
prv bprv
enc
fls
dsj
proof prfOf
encPf
Pf
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *num*
and *eql cnj imp all exi*
and *fls*
and *prv bprv*
and *Lq*
and *dsj* **and** *proof* :: *'proof set* **and** *prfOf*
and *enc* (‹⟨_⟩›)
and *N*
and *S*

and *encPf Pf*

context *Rosser_Form_Proofs*
begin

definition R **where** $R = all\ zz\ (imp\ (LLq\ (Var\ zz)\ (Var\ yy))$
$(all\ xx'\ (imp\ (NN\ (Var\ xx)\ (Var\ xx'))$
$(neg\ (PPf\ (Var\ zz)\ (Var\ xx'))))))$

definition RR **where** $RR\ t1\ t2 = psubst\ R\ [(t1,yy),\ (t2,xx)]$

lemma $R[simp,intro!]$: $R \in fmla$ $\langle proof \rangle$

lemma *RR_def2*:
$t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow xx \notin FvarsT\ t1 \Longrightarrow RR\ t1\ t2 = subst\ (subst\ R\ t1\ yy)\ t2\ xx$
$\langle proof \rangle$

definition P' **where** $P' = exi\ yy\ (cnj\ (PPf\ (Var\ yy)\ (Var\ xx))\ (RR\ (Var\ yy)\ (Var\ xx)))$

definition PP' **where** $PP'\ t = subst\ P'\ t\ xx$

lemma *Fvars_R*[*simp*]: $Fvars\ R = \{xx,yy\}$ $\langle proof \rangle$

lemma [*simp*]: $Fvars\ (RR\ (Var\ yy)\ (Var\ xx)) = \{yy,xx\}$ $\langle proof \rangle$

lemma $P'[simp,intro!]$: $P' \in fmla$ $\langle proof \rangle$

lemma *Fvars_P'*[*simp*]: $Fvars\ P' = \{xx\}$ $\langle proof \rangle$

lemma $PP'[simp,intro!]$: $t \in trm \Longrightarrow PP'\ t \in fmla$
$\langle proof \rangle$

lemma $RR[simp,intro]$: $t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow RR\ t1\ t2 \in fmla$
$\langle proof \rangle$

lemma *RR_simps*[*simp*]:
$n \in num \Longrightarrow subst\ (RR\ (Var\ yy)\ (Var\ xx))\ n\ xx = RR\ (Var\ yy)\ n$
$m \in num \Longrightarrow n \in num \Longrightarrow subst\ (RR\ (Var\ yy)\ m)\ n\ yy = RR\ n\ m$
$\langle proof \rangle$

The Rosser modification of the Gödel formula

definition φR :: *'fmla* **where** $\varphi R \equiv diag\ (neg\ P')$

lemma $\varphi R[simp,intro!]$: $\varphi R \in fmla$ **and** *Fvars_φR*[*simp*]: $Fvars\ \varphi R = \{\}$
$\langle proof \rangle$

lemma *bprv_φR_eqv*:
$bprv\ (eqv\ \varphi R\ (neg\ (PP'\ \langle \varphi R \rangle)))$
$\langle proof \rangle$

lemma *bprv_imp_φR*:
$bprv\ (imp\ (neg\ (PP'\ \langle \varphi R \rangle))\ \varphi R)$
$\langle proof \rangle$

lemma *prv_φR_eqv*:
$prv\ (eqv\ \varphi R\ (neg\ (PP'\ \langle \varphi R \rangle)))$
$\langle proof \rangle$

lemma *prv_imp_φR*:
prv (*imp* (*neg* (*PP′* ⟨*φR*⟩)) *φR*)
⟨*proof*⟩

end — context *Rosser_Form*

sublocale *Rosser_Form_Proofs* < *Rosser_Form* **where** *P* = *P*
⟨*proof*⟩

sublocale *Rosser_Form_Proofs* < *Goedel_Form* **where** *P* = *P*
⟨*proof*⟩

Chapter 10

Abstract Formulations of Gödel-Rosser's First Incompleteness Theorem

The development here is very similar to that of Gödel First Incompleteness Theorem. It lacks classical logical variants, since for them Rosser's trick does bring any extra value.

10.1 Proof-Theoretic Versions

context *Rosser_Form_Proofs*
begin

lemma *NN_neg_unique_xx'*:
  assumes [*simp*]:$\varphi \in$ *fmla Fvars* $\varphi = \{\}$
  shows
    bprv (*imp* (*NN* ⟨φ⟩ (*Var xx'*))
        (*eql* ⟨*neg* φ⟩ (*Var xx'*)))
  ⟨*proof*⟩

lemma *NN_imp_xx'*:
  assumes [*simp*]: $\varphi \in$ *fmla Fvars* $\varphi = \{\}$ $\chi \in$ *fmla*
  shows *bprv* (*imp* (*subst* χ ⟨*neg* φ⟩ *xx'*)
          (*all xx'* (*imp* (*NN* ⟨φ⟩ (*Var xx'*)) χ)))
⟨*proof*⟩

lemma *goedel_rosser_first_theEasyHalf*:
  assumes *c*: *consistent*
  shows ¬ *prv* φR
⟨*proof*⟩

lemma *goedel_rosser_first_theHardHalf*:
  assumes *c*: *consistent*
  shows ¬ *prv* (*neg* φR)
⟨*proof*⟩

theorem *goedel_rosser_first*:
  assumes *consistent*
  shows ¬ *prv* φR ∧ ¬ *prv* (*neg* φR)
  ⟨*proof*⟩

theorem *goedel_rosser_first_ex*:
  assumes *consistent*
  shows ∃ φ. φ ∈ *fmla* ∧ ¬ *prv* φ ∧ ¬ *prv* (*neg* φ)
  ⟨*proof*⟩

end — context *Rosser_Form*

10.2 Model-Theoretic Versions

10.2.1 First model-theoretic version

locale *Rosser_Form_Proofs_Minimal_Truth* =
Rosser_Form_Proofs
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  fls
  prv bprv
  Lq
  dsj
  proof prfOf
  enc
  N S
  encPf
  Pf
+
Minimal_Truth_Soundness
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
  bprv
  isTrue
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *Lq*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *N S P*
and *proof* :: *'proof set* **and** *prfOf encPf*
and *Pf*
and *isTrue*
begin

lemma *Fvars_PP'*[*simp*]: *Fvars* (*PP'* ⟨φ*R*⟩) = {} ⟨*proof*⟩

lemma *Fvars_RR'*[*simp*]: *Fvars* (*RR* (*Var yy*) ⟨φ*R*⟩) = {*yy*}
  ⟨*proof*⟩

lemma *isTrue_PPf_implies_φR*:
assumes *isTrue* (*all yy* (*neg* (*PPf* (*Var yy*) ⟨φ*R*⟩)))
(**is** *isTrue ?H*)

shows *isTrue* φR
⟨*proof*⟩

theorem *isTrue_φR*:
  assumes *consistent*
  shows *isTrue* φR
⟨*proof*⟩

theorem *goedel_rosser_first_strong*: *consistent* $\Longrightarrow \neg$ *prv* $\varphi R \wedge \neg$ *prv* (*neg* φR) $\wedge$ *isTrue* φR
  ⟨*proof*⟩

theorem *goedel_rosser_first_strong_ex*:
consistent $\Longrightarrow \exists\ \varphi.\ \varphi \in$ *fmla* $\wedge \neg$ *prv* $\varphi \wedge \neg$ *prv* (*neg* φ) $\wedge$ *isTrue* φ
  ⟨*proof*⟩

end — context *Rosser_Form_Proofs_Minimal_Truth*

10.2.2 Second model-theoretic version

context *Rosser_Form*
begin
print_context
end

locale *Rosser_Form_Minimal_Truth_Soundness_HBL1iff_Compl_Pf* =
Rosser_Form
  var trm fmla Var
  FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  fls
  prv bprv
  Lq
  dsj
  enc
  N
  S
  P
+
Minimal_Truth_Soundness_HBL1iff_Compl_Pf
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
  enc
  prv bprv
  P
  isTrue
  Pf
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*

and *prv bprv*
and *Lq*
and *enc* (‹⟨_⟩›)
and *N S*
and *isTrue*
and *P Pf*

locale *Rosser_Form_Minimal_Truth_Soundness_HBL1iff_Compl_Pf_Compl_NegPf* =
Rosser_Form_Minimal_Truth_Soundness_HBL1iff_Compl_Pf
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
  prv bprv
  Lq
  enc
  N S
  isTrue
  P
  Pf
+
M : *Minimal_Truth_Soundness_HBL1iff_Compl_Pf_Compl_NegPf*
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
  enc
  prv bprv
  N
  isTrue
  Pf
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *prv bprv*
and *Lq*
and *enc* (‹⟨_⟩›)
and *N S P*
and *isTrue*
and *Pf*

sublocale
  Rosser_Form_Minimal_Truth_Soundness_HBL1iff_Compl_Pf_Compl_NegPf <
  recover_proofs: *Rosser_Form_Proofs_Minimal_Truth*
  where *prfOf* = *prfOf* **and** *proof* = *proof* **and** *encPf* = *encPf*
  and *prv* = *prv* **and** *bprv* = *bprv*
  ⟨*proof*⟩

context *Rosser_Form_Minimal_Truth_Soundness_HBL1iff_Compl_Pf_Compl_NegPf*
begin

thm *recover_proofs.goedel_rosser_first_strong*
end

Chapter 11

Abstract Formulation of Gödel's Second Incompleteness Theorem

We assume all three derivability conditions, and assumptions behind Gödel formulas:

locale *Goedel_Second_Assumptions* =
HBL1_2_3
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv bprv
  enc
  P
+
Goedel_Form
  var trm fmla Var num FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  prv bprv
  enc
  S
  P
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *num*
and *eql cnj imp all exi*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *S*
and *P*
and *fls*
begin

lemma *P_G*:
bprv (*imp* (*PP* ⟨*φG*⟩) (*PP* ⟨*fls*⟩))
⟨*proof*⟩

First the "direct", positive formulation:

lemma *goedel_second_pos*:
assumes *prv* (*neg* (*PP* ⟨*fls*⟩))
shows *prv fls*
⟨*proof*⟩

Then the more standard, counterpositive formulation:

theorem *goedel_second*:
consistent $\Longrightarrow$ ¬ *prv* (*neg* (*PP* ⟨*fls*⟩))
⟨*proof*⟩

It is an immediate consequence of Gödel's Second HLB1, HLB2 that (assuming consistency) *prv* (*neg* (*PP* ⟨φ⟩)) holds for no sentence, be it provable or not. The theory is omniscient about what it can prove (thanks to HLB1), but completely ignorant about what it cannot prove.

corollary *not_prv_neg_PP*:
assumes *c*: *consistent* **and** [*simp*]: $\varphi \in$ *fmla Fvars* $\varphi = \{\}$
shows ¬ *prv* (*neg* (*PP* ⟨φ⟩))
⟨*proof*⟩

end — context *Goedel_Second_Assumptions*

Chapter 12

Jeroslow's Variant of Gödel's Second Incompleteness Theorem

12.1 Encodings and Derivability

Here we formalize some of the assumptions of Jeroslow's theorem: encoding, term-encoding and the First Derivability Condition.

12.1.1 Encoding of formulas

locale *Encode* =
Syntax_with_Numerals
  var trm fmla Var FvarsT substT Fvars subst
  num
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *num*
+
fixes

enc :: *'fmla* ⇒ *'trm* (‹⟨_⟩›)
assumes
enc[*simp*,*intro*!]: $\bigwedge \varphi.\ \varphi \in \mathit{fmla} \Longrightarrow \mathit{enc}\ \varphi \in \mathit{num}$
begin

end — context *Encode*

12.1.2 Encoding of computable functions

Jeroslow assumes the encodability of an abstract (unspecified) class of computable functions and the assumption that a particular function, *sub* φ for each formula φ, is in this collection. This is used to prove a different flavor of the diagonalization lemma (Jeroslow 1973). It turns out that only an encoding of unary computable functions is needed, so we only assume that.

locale *Encode_UComput* =
Encode
  var trm fmla Var FvarsT substT Fvars subst
  num
  enc
for

var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *num*
and *enc* (‹⟨_⟩›)
+
— Abstract (unspeficied) notion of unary "computable" function between numerals, which are encoded as numerals. They contain a special substitution-like function *sub* φ for each formula φ.
fixes *ucfunc* :: (*'trm* ⇒ *'trm*) *set*
  and *encF* :: (*'trm* ⇒ *'trm*) ⇒ *'trm*
  and *sub* :: *'fmla* ⇒ *'trm* ⇒ *'trm*
assumes
— NB: Due to the limitations of the type system, we define *ufunc* as a set of functions between terms, but we only care about their actions on numerals ... so we assume they send numerals to numerals:
ucfunc[*simp*,*intro*!]: $\bigwedge f\ n.\ f \in ucfunc \Longrightarrow n \in num \Longrightarrow f\ n \in num$
and
encF[*simp*,*intro*!]: $\bigwedge f.\ f \in ucfunc \Longrightarrow encF\ f \in num$
and
sub[*simp*]: $\bigwedge\varphi.\ \varphi \in fmla \Longrightarrow sub\ \varphi \in ucfunc$
and
— The function *sub* φ takes any encoding of a function *f* and returns the encoding of the formula obtained by substituting for *xx* the value of *f* applied to its own encoding:
sub_enc:
$\bigwedge \varphi\ f.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{xx\} \Longrightarrow f \in ucfunc \Longrightarrow$
  $sub\ \varphi\ (encF\ f) = enc\ (inst\ \varphi\ (f\ (encF\ f)))$

12.1.3 Term-encoding of computable functons

For handling the notion of term-representation (which we introduce later), we assume we are given a set *Ops* of term operators and their encodings as numerals. We additionally assume that the term operators behave well w.r.t. free variables and substitution.

locale *TermEncode* =
Syntax_with_Numerals
  var trm fmla Var FvarsT substT Fvars subst
  num
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *num*
+
fixes
Ops :: (*'trm* ⇒ *'trm*) *set*
and
enc :: (*'trm* ⇒ *'trm*) ⇒ *'trm* (‹⟨_⟩›)
assumes
Ops[*simp*,*intro*!]: $\bigwedge f\ t.\ f \in Ops \Longrightarrow t \in trm \Longrightarrow f\ t \in trm$
and
enc[*simp*,*intro*!]: $\bigwedge f.\ f \in Ops \Longrightarrow enc\ f \in num$
and
Ops_FvarsT[*simp*]: $\bigwedge f\ t.\ f \in Ops \Longrightarrow t \in trm \Longrightarrow FvarsT\ (f\ t) = FvarsT\ t$
and
Ops_substT[*simp*]: $\bigwedge f\ t.\ f \in Ops \Longrightarrow t \in trm \Longrightarrow t1 \in trm \Longrightarrow x \in var \Longrightarrow$
  $substT\ (f\ t)\ t1\ x = f\ (substT\ t\ t1\ x)$
begin

end — context *TermEncode*

12.1.4 The first Hilbert-Bernays-Löb derivability condition

locale *HBL1* =
Encode
  var trm fmla Var FvarsT substT Fvars subst
  num
  enc
+
Deduct
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *num*
and *eql cnj imp all exi*
and *prv bprv*
and *enc* (‹⟨_⟩›)
+
fixes *P* :: *'fmla*
assumes
P[*intro*!,*simp*]: $P \in fmla$
and
Fvars_P[*simp*]: $Fvars\ P = \{xx\}$
and
HBL1: $\bigwedge\varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow prv\ \varphi \Longrightarrow prv\ (subst\ P\ \langle\varphi\rangle\ xx)$
begin

Predicate version of the provability formula

definition *PP* **where** $PP \equiv \lambda t.\ subst\ P\ t\ xx$

lemma *PP*[*simp*]: $\bigwedge t.\ t \in trm \Longrightarrow PP\ t \in fmla$
  ⟨*proof*⟩

lemma *Fvars_PP*[*simp*]: $\bigwedge t.\ t \in trm \Longrightarrow Fvars\ (PP\ t) = FvarsT\ t$
  ⟨*proof*⟩

lemma [*simp*]:
$n \in num \Longrightarrow subst\ (PP\ (Var\ yy))\ n\ yy = PP\ n$
$n \in num \Longrightarrow subst\ (PP\ (Var\ xx))\ n\ xx = PP\ n$
  ⟨*proof*⟩

lemma *HBL1_PP*:
$\varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow prv\ \varphi \Longrightarrow prv\ (PP\ \langle\varphi\rangle)$
  ⟨*proof*⟩

end — context *HBL1*

12.2 A Formalization of Jeroslow's Original Argument

12.2.1 Preliminaries

The First Derivability Condition was stated using a formula with free variable *xx*, whereas the pseudo-term theory employs a different variable, inp. The distinction is of course immaterial, because we can perform a change of variable in the instantiation:

context *HBL1*
begin

Changing the variable (from *xx* to *inp*) in the provability predicate:

definition *Pinp* ≡ *subst P* (*Var inp*) *xx*
lemma *PP_Pinp*: *t* ∈ *trm* ⟹ *PP t* = *instInp Pinp t*
⟨*proof*⟩

A version of HBL1 that uses the *inp* variable:

lemma *HBL1_inp*:
φ ∈ *fmla* ⟹ *Fvars φ* = {} ⟹ *prv φ* ⟹ *prv* (*instInp Pinp* ⟨*φ*⟩)
⟨*proof*⟩

end — context *HBL1*

12.2.2 Jeroslow-style diagonalization

locale *Jeroslow_Diagonalization* =
Deduct_with_False_Disj_Rename
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
  prv
+
Encode
  var trm fmla Var FvarsT substT Fvars subst
  num
  enc
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *prv*
and *enc* (‹⟨_⟩›)
+
fixes *F* :: (*'trm* ⇒ *'trm*) *set*
  and *encF* :: (*'trm* ⇒ *'trm*) ⇒ *'fmla*
  and *N* :: *'trm* ⇒ *'trm*
  and *ssap* :: *'fmla* ⇒ *'trm* ⇒ *'trm*
assumes
— For the members *f* of *F*, we will only care about their action on numerals, and we assume that they send numerals to numerals.
F[*simp*,*intro*!]: ⋀ *f n*. *f* ∈ *F* ⟹ *n* ∈ *num* ⟹ *f n* ∈ *num*
and
encF[*simp*,*intro*!]: ⋀ *f*. *f* ∈ *F* ⟹ *encF f* ∈ *ptrm* (*Suc 0*)
and
N[*simp*,*intro*!]: *N* ∈ *F*
and
ssap[*simp*]: ⋀*φ*. *φ* ∈ *fmla* ⟹ *Fvars φ* = {*inp*} ⟹ *ssap φ* ∈ *F*
and
ReprF: ⋀*f n*. *f* ∈ *F* ⟹ *n* ∈ *num* ⟹ *prveqlPT* (*instInp* (*encF f*) *n*) (*f n*)
and
CapN: ⋀*φ*. *φ* ∈ *fmla* ⟹ *Fvars φ* = {} ⟹ *N* ⟨*φ*⟩ = ⟨*neg φ*⟩

and
CapSS: — We consider formulas ψ of one variable, called *inp*:
$\bigwedge \psi\ f.\ \psi \in fmla \Longrightarrow Fvars\ \psi = \{inp\} \Longrightarrow f \in F \Longrightarrow$
$\quad ssap\ \psi\ \langle encF\ f\rangle = \langle instInpP\ \psi\ 0\ (instInp\ (encF\ f)\ \langle encF\ f\rangle)\rangle$
begin

lemma *encF_fmla*[*simp*,*intro*!]: $\bigwedge f.\ f \in F \Longrightarrow encF\ f \in fmla$
⟨*proof*⟩

lemma *enc_trm*: $\varphi \in fmla \Longrightarrow \langle\varphi\rangle \in trm$
⟨*proof*⟩

definition τJ :: *'fmla* ⇒ *'fmla* **where**
$\tau J\ \psi \equiv instInp\ (encF\ (ssap\ \psi))\ (\langle encF\ (ssap\ \psi)\rangle)$

definition φJ :: *'fmla* ⇒ *'fmla* **where**
$\varphi J\ \psi \equiv instInpP\ \psi\ 0\ (\tau J\ \psi)$

lemma τJ[*simp*]:
assumes $\psi \in fmla$ **and** $Fvars\ \psi = \{inp\}$
shows $\tau J\ \psi \in ptrm\ 0$
⟨*proof*⟩

lemma *τJ_fmla*[*simp*]:
assumes $\psi \in fmla$ **and** $Fvars\ \psi = \{inp\}$
shows $\tau J\ \psi \in fmla$
⟨*proof*⟩

lemma *FvarsT_τJ*[*simp*]:
assumes $\psi \in fmla$ **and** $Fvars\ \psi = \{inp\}$
shows $Fvars\ (\tau J\ \psi) = \{out\}$
⟨*proof*⟩

lemma φJ[*simp*]:
assumes $\psi \in fmla$ **and** $Fvars\ \psi = \{inp\}$
shows $\varphi J\ \psi \in fmla$
⟨*proof*⟩

lemma *Fvars_φJ*[*simp*]:
assumes $\psi \in fmla$ **and** $Fvars\ \psi = \{inp\}$
shows $Fvars\ (\varphi J\ \psi) = \{\}$
⟨*proof*⟩

lemma *diagonalization*:
assumes ψ[*simp*]: $\psi \in fmla$ **and** [*simp*]: $Fvars\ \psi = \{inp\}$
shows $prveqlPT\ (\tau J\ \psi)\ \langle instInpP\ \psi\ 0\ (\tau J\ \psi)\rangle\ \wedge$
$\quad prv\ (eqv\ (\varphi J\ \psi)\ (instInp\ \psi\ \langle\varphi J\ \psi\rangle))$
⟨*proof*⟩

end — context *Jeroslow_Diagonalization*

12.2.3 Jeroslow's Second Incompleteness Theorem

We follow Jeroslow's pseudo-term-based development of the Second Incompleteness Theorem and point out the location in the proof that implicitly uses an unstated assumption: the fact that, for certain two provably equivalent formulas φ and φ', it is provable that the provability of the encoding of φ' implies the provability of the encoding of φ.

locale *Jeroslow_Godel_Second* =

Jeroslow_Diagonalization
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
prv
enc
F encF N ssap
+
HBL1
var trm fmla Var FvarsT substT Fvars subst
num
eql cnj imp all exi
prv prv
enc
P
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *prv*
and *enc* (‹⟨_⟩›)
and *P*
and *F encF N ssap*
+
assumes
SHBL3: $\bigwedge\tau.\ \tau \in ptrm\ 0 \Longrightarrow prv\ (imp\ (instInpP\ Pinp\ 0\ \tau)\ (instInp\ Pinp\ \langle instInpP\ Pinp\ 0\ \tau\rangle))$
begin

Consistency formula a la Jeroslow:

definition *jcons* :: *'fmla* **where**
$jcons \equiv all\ xx\ (neg\ (cnj\ (instInp\ Pinp\ (Var\ xx))$
$(instInpP\ Pinp\ 0\ (instInp\ (encF\ N)\ (Var\ (xx))))))$

lemma *prv_eql_subst_trm3*:
$x \in var \Longrightarrow \varphi \in fmla \Longrightarrow t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow$
$prv\ (eql\ t1\ t2) \Longrightarrow prv\ (subst\ \varphi\ t1\ x) \Longrightarrow prv\ (subst\ \varphi\ t2\ x)$
⟨*proof*⟩

lemma *Pinp*[*simp,intro*!]: $Pinp \in fmla$
and *Fvars_Pinp*[*simp*]: $Fvars\ Pinp = \{inp\}$
⟨*proof*⟩

lemma *ReprF_combineWith_CapN*:
assumes $\varphi \in fmla$ **and** $Fvars\ \varphi = \{\}$
shows $prveqlPT\ (instInp\ (encF\ N)\ \langle\varphi\rangle)\ \langle neg\ \varphi\rangle$
⟨*proof*⟩

theorem *jeroslow_godel_second*:
assumes *consistent*
— Assumption that is not stated by Jeroslow, but seems to be needed:
assumes *unstated*:
  let $\psi = instInpP\ Pinp\ (Suc\ 0)\ (encF\ N);$
    $\tau = \tau J\ \psi;$

$\varphi = instInpP\ (instInpP\ Pinp\ (Suc\ 0)\ (encF\ N))\ 0\ \tau;$
$\varphi' = instInpP\ Pinp\ 0\ (instInpP\ (encF\ N)\ 0\ \tau)$
in prv $(imp\ (instInp\ Pinp\ \langle\varphi'\rangle)\ (instInp\ Pinp\ \langle\varphi\rangle))$
shows $\neg$ *prv jcons*
⟨*proof*⟩

end — context *Jeroslow_Godel_Second*

12.3 A Simplification of Jeroslow's Original Argument

This is the simplified version of Jeroslow's Second Incompleteness Theorem reported in our CADE 2019 paper [1]. The simplification consists of replacing pseudo-terms with plain terms and representability with (what we call in the paper) term-representability. This simplified version does not incur the complications of the original.

12.3.1 Jeroslow-style term-based diagonalization

locale *Jeroslow_Diagonalization* =
Deduct_with_False
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  num
  prv
+
Encode
  var trm fmla Var FvarsT substT Fvars subst
  num
  enc
+
TermEncode
  var trm fmla Var FvarsT substT Fvars subst
  num
  Ops tenc
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *num*
and *prv*
and *enc* (‹⟨_⟩›)
and *Ops* **and** *tenc*
+
fixes F :: $('trm \Rightarrow 'trm)$ *set*
  and *encF* :: $('trm \Rightarrow 'trm) \Rightarrow ('trm \Rightarrow 'trm)$
  and N :: $'trm \Rightarrow 'trm$
  and *ssap* :: $'fmla \Rightarrow 'trm \Rightarrow 'trm$
assumes
F[*simp,intro*!]: $\bigwedge f\ n.\ f \in F \Longrightarrow n \in num \Longrightarrow f\ n \in num$
and
encF[*simp,intro*!]: $\bigwedge f.\ f \in F \Longrightarrow encF\ f \in Ops$
and
N[*simp,intro*!]: $N \in F$
and
ssap[*simp*]: $\bigwedge\varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{xx\} \Longrightarrow ssap\ \varphi \in F$

and
ReprF: $\bigwedge f\ n.\ f \in F \Longrightarrow n \in num \Longrightarrow prv\ (eql\ (encF\ f\ n)\ (f\ n))$
and
CapN: $\bigwedge\varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow N\ \langle\varphi\rangle = \langle neg\ \varphi\rangle$
and
CapSS:
$\bigwedge\ \psi\ f.\ \psi \in fmla \Longrightarrow Fvars\ \psi = \{xx\} \Longrightarrow f \in F \Longrightarrow$
$\quad ssap\ \psi\ (tenc\ (encF\ f)) = \langle inst\ \psi\ (encF\ f\ (tenc\ (encF\ f)))\rangle$
begin

definition $tJ :: {}'fmla \Rightarrow {}'trm$ **where**
$tJ\ \psi \equiv encF\ (ssap\ \psi)\ (tenc\ (encF\ (ssap\ \psi)))$

definition $\varphi J :: {}'fmla \Rightarrow {}'fmla$ **where**
$\varphi J\ \psi \equiv subst\ \psi\ (tJ\ \psi)\ xx$

lemma *tJ*[*simp*]:
assumes $\psi \in fmla$ **and** $Fvars\ \psi = \{xx\}$
shows $tJ\ \psi \in trm$
⟨*proof*⟩

lemma *FvarsT_tJ*[*simp*]:
assumes $\psi \in fmla$ **and** $Fvars\ \psi = \{xx\}$
shows $FvarsT\ (tJ\ \psi) = \{\}$
⟨*proof*⟩

lemma φJ[*simp*]:
assumes $\psi \in fmla$ **and** $Fvars\ \psi = \{xx\}$
shows $\varphi J\ \psi \in fmla$
⟨*proof*⟩

lemma *Fvars_*φJ[*simp*]:
assumes $\psi \in fmla$ **and** $Fvars\ \psi = \{xx\}$
shows $Fvars\ (\varphi J\ \psi) = \{\}$
⟨*proof*⟩

lemma *diagonalization*:
assumes $\psi \in fmla$ **and** $Fvars\ \psi = \{xx\}$
shows $prv\ (eql\ (tJ\ \psi)\ \langle inst\ \psi\ (tJ\ \psi)\rangle) \wedge$
$\quad prv\ (eqv\ (\varphi J\ \psi)\ (inst\ \psi\ \langle\varphi J\ \psi\rangle))$
⟨*proof*⟩

end — context *Jeroslow_Diagonalization*

12.3.2 Term-based version of Jeroslow's Second Incompleteness Theorem

locale *Jeroslow_Godel_Second* =
Jeroslow_Diagonalization
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
num
prv
enc
Ops tenc
F encF N ssap
+

HBL1
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv prv
  enc
  P
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *num*
and *prv*
and *enc* (‹⟨_⟩›)
and *Ops* **and** *tenc*
and *P*
and *F encF N ssap*
+
assumes
SHBL3: $\bigwedge t.\ t \in trm \Longrightarrow FvarsT\ t = \{\} \Longrightarrow prv\ (imp\ (PP\ t)\ (PP\ \langle PP\ t\rangle))$
begin

Consistency formula a la Jeroslow:

definition *jcons* :: *'fmla* **where**
$jcons \equiv all\ xx\ (neg\ (cnj\ (PP\ (Var\ xx))\ (PP\ (encF\ N\ (Var\ (xx))))))$

lemma *prv_eql_subst_trm3*:
$x \in var \Longrightarrow \varphi \in fmla \Longrightarrow t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow$
$prv\ (eql\ t1\ t2) \Longrightarrow prv\ (subst\ \varphi\ t1\ x) \Longrightarrow prv\ (subst\ \varphi\ t2\ x)$
⟨*proof*⟩

lemma *prv_eql_neg_encF_N*:
assumes $\varphi \in fmla$ **and** $Fvars\ \varphi = \{\}$
shows $prv\ (eql\ \langle neg\ \varphi\rangle\ (encF\ N\ \langle\varphi\rangle))$
  ⟨*proof*⟩

lemma *prv_imp_neg_encF_N_aux*:
assumes $\varphi \in fmla$ **and** $Fvars\ \varphi = \{\}$
shows $prv\ (imp\ (PP\ \langle neg\ \varphi\rangle)\ (PP\ (encF\ N\ \langle\varphi\rangle)))$
⟨*proof*⟩

lemma *prv_cnj_neg_encF_N_aux*:
assumes $\varphi \in fmla$ **and** $Fvars\ \varphi = \{\}\ \chi \in fmla\ Fvars\ \chi = \{\}$
and $prv\ (neg\ (cnj\ \chi\ (PP\ \langle neg\ \varphi\rangle)))$
shows$prv\ (neg\ (cnj\ \chi\ (PP\ (encF\ N\ \langle\varphi\rangle))))$
⟨*proof*⟩

theorem *jeroslow_godel_second*:
assumes *consistent*
shows ¬ *prv jcons*
⟨*proof*⟩

12.3.3 A variant of the Second Incompleteness Theorem

This variant (also discussed in our CADE 2019 paper [1]) strengthens the conclusion of the theorem to the standard formulation of "does not prove its own consistency" at the expense of two additional derivability-like conditions, HBL4 and WHBL2.

theorem *jeroslow_godel_second_standardCon*:
assumes *consistent*
and *HBL4*: $\bigwedge\varphi 1\ \varphi 2.\ \{\varphi 1,\varphi 2\} \subseteq fmla \Longrightarrow Fvars\ \varphi 1 = \{\} \Longrightarrow Fvars\ \varphi 2 = \{\} \Longrightarrow$
$prv\ (imp\ (cnj\ (PP\ \langle\varphi 1\rangle)\ (PP\ \langle\varphi 2\rangle))\ (PP\ \langle cnj\ \varphi 1\ \varphi 2\rangle))$
and *WHBL2*: $\bigwedge\varphi 1\ \varphi 2.\ \{\varphi 1,\varphi 2\} \subseteq fmla \Longrightarrow Fvars\ \varphi 1 = \{\} \Longrightarrow Fvars\ \varphi 2 = \{\} \Longrightarrow$
$prv\ (imp\ \varphi 1\ \varphi 2) \Longrightarrow prv\ (imp\ (PP\ \langle\varphi 1\rangle)\ (PP\ \langle\varphi 2\rangle))$
shows $\neg\ prv\ (neg\ (PP\ \langle fls\rangle))$
⟨*proof*⟩

Next we perform a formal analysis of some connection between the above theorems' hypotheses.

definition *noContr* :: *bool* **where**
$noContr \equiv \forall\ \varphi \in fmla.\ Fvars\ \varphi = \{\} \longrightarrow prv\ (neg\ (cnj\ (PP\ \langle\varphi\rangle)\ (PP\ \langle neg\ \varphi\rangle)))$

lemma *jcons_noContr*:
assumes *j*: *prv jcons*
shows *noContr*
⟨*proof*⟩

noContr is still stronger than the standard notion of proving own consistency:

lemma *noContr_implies_neg_PP_fls*:
assumes *noContr*
shows $prv\ (neg\ (PP\ \langle fls\rangle))$
⟨*proof*⟩

corollary *jcons_implies_neg_PP_fls*:
assumes *prv jcons*
shows $prv\ (neg\ (PP\ \langle fls\rangle))$
⟨*proof*⟩

However, unlike *jcons*, which seems to be quite a bit stronger, *noContr* is equivalent to the standard notion under a slightly stronger assumption than our WWHBL2, namely, a binary version of that:

lemma *neg_PP_fls_implies_noContr*:
assumes *WWHBL22*:
$\bigwedge\varphi\ \chi\ \psi.\ \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
$Fvars\ \varphi = \{\} \Longrightarrow Fvars\ \chi = \{\} \Longrightarrow Fvars\ \psi = \{\} \Longrightarrow$
$prv\ (imp\ \varphi\ (imp\ \chi\ \psi)) \Longrightarrow prv\ (imp\ (PP\ \langle\varphi\rangle)\ (imp\ (PP\ \langle\chi\rangle)\ (PP\ \langle\psi\rangle)))$
assumes *p*: $prv\ (neg\ (PP\ \langle fls\rangle))$
shows *noContr*
⟨*proof*⟩

end — context *Jeroslow_Godel_Second*

Chapter 13

Löb Formulas

The Löb formula, parameterized by a sentence φ, is defined by diagonalizing *imp P* φ.

locale *Loeb_Form* =
Deduct2
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv bprv
+
Repr_SelfSubst
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv bprv
  enc
  S
+
HBL1
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv bprv
  enc
  P
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var num FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *S*
and *P*
begin

The Löb formula associated to a formula φ:

definition φL :: *'fmla* ⇒ *'fmla* **where** $\varphi L\ \varphi \equiv$ *diag* (*imp P* φ)

lemma φL[*simp*,*intro*]: $\bigwedge\varphi.\ \varphi \in$ *fmla* $\Longrightarrow$ *Fvars* $\varphi = \{\} \Longrightarrow \varphi L\ \varphi \in$ *fmla*
and
*Fvars_*φL[*simp*]: $\varphi \in$ *fmla* $\Longrightarrow$ *Fvars* $\varphi = \{\} \Longrightarrow$ *Fvars* $(\varphi L\ \varphi) = \{\}$
  ⟨*proof*⟩

lemma *bprv_*φL*_eqv*:

$\varphi \in fmla \Longrightarrow Fvars\ \varphi\ = \{\} \Longrightarrow bprv\ (eqv\ (\varphi L\ \varphi)\ (imp\ (PP\ \langle\varphi L\ \varphi\rangle)\ \varphi))$
$\langle proof\rangle$

lemma *prv_φL_eqv*:
$\varphi \in fmla \Longrightarrow Fvars\ \varphi\ = \{\} \Longrightarrow prv\ (eqv\ (\varphi L\ \varphi)\ (imp\ (PP\ \langle\varphi L\ \varphi\rangle)\ \varphi))$
$\langle proof\rangle$

end — context *Loeb_Form*

Chapter 14

Löb's Theorem

We have set up the formalization of Gödel's first (easy half) and Gödel's second so that the following generalizations, leading to Löb's theorem, are trivial modifications of these, replacing negation with "implies φ" in all proofs.

locale *Loeb_Assumptions* =
HBL1_2_3
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
  prv bprv
  enc
  P
+
Loeb_Form
  var trm fmla Var num FvarsT substT Fvars subst
  eql cnj imp all exi
  prv bprv
  enc
  S
  P
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var num FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *S*
and *P*
begin

Generalization of *goedel_first_theEasyHalf_pos*, replacing *fls* with a sentence φ:

lemma *loeb_aux_prv*:
assumes φ[*simp*]: $\varphi \in$ *fmla Fvars* $\varphi = \{\}$ **and** *p*: *prv* (φL φ)
shows *prv* φ
⟨*proof*⟩

lemma *loeb_aux_bprv*:
assumes φ[*simp*]: $\varphi \in$ *fmla Fvars* $\varphi = \{\}$ **and** *p*: *bprv* (φL φ)
shows *bprv* φ
⟨*proof*⟩

Generalization of *P_G*, the main lemma used for Gödel's second:

lemma *P_L*:
assumes φ[*simp*]: $\varphi \in$ *fmla Fvars* $\varphi = \{\}$
shows *bprv* (*imp* (*PP* ⟨φ*L* φ⟩) (*PP* ⟨φ⟩))
⟨*proof*⟩

Löb's theorem generalizes the positive formulation Gödel's Second (*goedel_second*). In our two-provability-relation framework, we get two variants of Löb's theorem. A stronger variant, assuming *prv* and proving *bprv*, seems impossible.

theorem *loeb_bprv*:
assumes φ[*simp*]: $\varphi \in$ *fmla Fvars* $\varphi = \{\}$ **and** *p*: *bprv* (*imp* (*PP* ⟨φ⟩) φ)
shows *bprv* φ
⟨*proof*⟩

theorem *loeb_prv*:
assumes φ[*simp*]: $\varphi \in$ *fmla Fvars* $\varphi = \{\}$ **and** *p*: *prv* (*imp* (*PP* ⟨φ⟩) φ)
shows *prv* φ
⟨*proof*⟩

We could have of course inferred *goedel_first_theEasyHalf_pos* and *goedel_second* from these more general versions, but we leave the original arguments as they are more instructive.

end — context *Loeb_Assumptions*

Chapter 15

Abstract Formulation of Tarski's Theorems

We prove Tarski's proof-theoretic and semantic theorems about the non-definability and respectively non-expressiveness (in the standard model) of truth

15.1 Non-Definability of Truth

context *Goedel_Form*
begin

context
fixes T :: *'fmla*
assumes T[*simp*,*intro*!]: $T \in$ *fmla*
and *Fvars_T*[*simp*]: *Fvars* $T = \{xx\}$
and *prv_T*: $\bigwedge\varphi.\ \varphi \in$ *fmla* $\Longrightarrow$ *Fvars* $\varphi = \{\} \Longrightarrow$ *prv* (*eqv* (*subst* T $\langle\varphi\rangle$ *xx*) φ)
begin

definition φT :: *'fmla* **where** $\varphi T \equiv$ *diag* (*neg* T)

lemma φT[*simp*,*intro*!]: $\varphi T \in$ *fmla* **and**
*Fvars_*φT[*simp*]: *Fvars* $\varphi T = \{\}$
⟨*proof*⟩

lemma *bprv_*φT*_eqv*:
bprv (*eqv* φT (*neg* (*subst* T $\langle\varphi T\rangle$ *xx*)))
⟨*proof*⟩

lemma *prv_*φT*_eqv*:
prv (*eqv* φT (*neg* (*subst* T $\langle\varphi T\rangle$ *xx*)))
⟨*proof*⟩

lemma φT*_prv_fls*: *prv fls*
⟨*proof*⟩

end — context

theorem *Tarski_proof_theoretic*:
assumes $T \in$ *fmla* *Fvars* $T = \{xx\}$
and $\bigwedge\varphi.\ \varphi \in$ *fmla* $\Longrightarrow$ *Fvars* $\varphi = \{\} \Longrightarrow$ *prv* (*eqv* (*subst* T $\langle\varphi\rangle$ *xx*) φ)
shows ¬ *consistent*

⟨*proof*⟩

end — context *Goedel_Form*

15.2 Non-Expressiveness of Truth

This follows as a corollary of the syntactic version, after taking prv to be isTrue on sentences.Indeed, this is a virtue of our abstract treatment of provability: We don't work with a particular predicate, but with any predicate that is closed under some rules — which could as well be a semantic notion of truth (for sentences).

locale *Goedel_Form_prv_eq_isTrue* =
Goedel_Form
  var trm fmla Var num FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  prv bprv
  enc
  P
  S
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var num FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *prv bprv*
and *enc* (‹⟨_⟩›)
and *S*
and *P*
+
fixes *isTrue* :: *'fmla* ⇒ *bool*
assumes *prv_eq_isTrue*: $\bigwedge \varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow prv\ \varphi = isTrue\ \varphi$
begin

theorem *Tarski_semantic*:
assumes *0*: $T \in fmla\ Fvars\ T = \{xx\}$
and *1*: $\bigwedge\varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow isTrue\ (eqv\ (subst\ T\ \langle\varphi\rangle\ xx)\ \varphi)$
shows ¬ *consistent*
⟨*proof*⟩

NB: To instantiate the semantic version of Tarski's theorem for a truth predicate isTruth on sentences, one needs to extend it to a predicate "prv" on formulas and verify that "prv" satisfies the rules of intuitionistic logic.

end — context *Goedel_Form_prv_eq_isTrue*

Bibliography

[1] A. Popescu and D. Traytel. A formally verified abstract account of Gödel's incompleteness theorems. In P. Fontaine, editor, *CADE 27*, volume 11716 of *LNCS*, pages 442–461. Springer, 2019.