

An Abstract Formalization of Gödel's Incompleteness Theorems

Andrei Popescu Dmitriy Traytel

February 6, 2026

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

1	Deduction with Two Provability Relations	3
1.1	From Deduction with One Provability Relation to Two	3
1.2	Factoring In Explicit Proofs	6
2	Abstract Encoding	8
3	Representability Assumptions	10
3.1	Representability of Negation	10
3.2	Representability of Self-Substitution	13
3.3	Representability of Self-Soft-Substitution	14
3.4	Clean Representability of the "Proof-of" Relation	15
4	Diagonalization	18
4.1	Alternative Diagonalization via Self-Substitution	18
4.2	Alternative Diagonalization via Soft Self-Substitution	20
5	The Hilbert-Bernays-Löb (HBL) Derivability Conditions	22
5.1	First Derivability Condition	22
5.2	Connections between Proof Representability, First Derivability Condition, and Its Converse	23
5.2.1	HBL1 from "proof-of" representability	23
5.2.2	Sufficient condition for the converse of HBL1	24
5.3	Second and Third Derivability Conditions	24
6	Gödel Formulas	26
7	Standard Models with Two Provability Relations	29
7.1	Proof recovery from <i>HBL1_iff</i>	31
8	Abstract Formulations of Gödel's First Incompleteness Theorem	38
8.1	Proof-Theoretic Versions of Gödel's First	38
8.1.1	The easy half	38
8.1.2	The hard half	38
8.2	Model-Theoretic Versions of Gödel's First	39
8.2.1	First model-theoretic version	39
8.2.2	Second model-theoretic version	40
8.3	Classical-Logic Versions of Gödel's First	42
8.3.1	First classical-logic version	42
8.3.2	Second classical-logic version	43
8.3.3	Third classical-logic version	44
9	Rosser Formulas	46

10 Abstract Formulations of Gödel-Rosser's First Incompleteness Theorem	50
10.1 Proof-Theoretic Versions	50
10.2 Model-Theoretic Versions	52
10.2.1 First model-theoretic version	52
10.2.2 Second model-theoretic version	54
11 Abstract Formulation of Gödel's Second Incompleteness Theorem	57
12 Jeroslow's Variant of Gödel's Second Incompleteness Theorem	59
12.1 Encodings and Derivability	59
12.1.1 Encoding of formulas	59
12.1.2 Encoding of computable functions	59
12.1.3 Term-encoding of computable functions	60
12.1.4 The first Hilbert-Bernays-Löb derivability condition	61
12.2 A Formalization of Jeroslow's Original Argument	61
12.2.1 Preliminaries	61
12.2.2 Jeroslow-style diagonalization	62
12.2.3 Jeroslow's Second Incompleteness Theorem	64
12.3 A Simplification of Jeroslow's Original Argument	67
12.3.1 Jeroslow-style term-based diagonalization	67
12.3.2 Term-based version of Jeroslow's Second Incompleteness Theorem	69
12.3.3 A variant of the Second Incompleteness Theorem	71
13 Löb Formulas	74
14 Löb's Theorem	76
15 Abstract Formulation of Tarski's Theorems	79
15.1 Non-Definability of Truth	79
15.2 Non-Expressiveness of Truth	80

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:  $\bigwedge \varphi. \varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{bprv } \varphi \implies \text{prv } \varphi$ 
begin

lemma bprv_prv':
  assumes  $\varphi: \varphi \in \text{fmla}$  and  $b: \text{bprv } \varphi$ 
  shows  $\text{prv } \varphi$ 
proof -
  obtain V where  $V: \text{Fvars } \varphi = V$  by blast
  have  $VV: V \subseteq \text{var}$  using  $\text{Fvars } V \varphi$  by blast
  have  $f: \text{finite } V$  using  $V \text{ finite\_Fvars}[OF \varphi]$  by auto
  thus  $?thesis$  using  $\varphi b V VV$ 
  proof (induction V arbitrary:  $\varphi$  rule: finite.induct)
    case (emptyI  $\varphi$ )
    then show  $?case$  by (simp add: bprv_prv)
```

```

next
  case (insertI W v  $\varphi$ )
  show ?case proof(cases v  $\in$  W)
    case True
    thus ?thesis
    using insertI.IH[OF < $\varphi \in$  fmla>] insertI.prem
    by (simp add: insert_absorb)
  next
  case False
  hence 1: Fvars (all v  $\varphi$ ) = W
    using insertI.prem by auto
  moreover have bprv (all v  $\varphi$ )
    using B.prv_all_gen insertI.prem by auto
  ultimately have prv (all v  $\varphi$ ) using insertI by auto
  thus ?thesis using allE_id insertI.prem by blast
qed
qed
qed

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  $\implies$  Fvars  $\varphi$  = {}  $\implies$  bprv  $\varphi \implies$  prv  $\varphi$ 

sublocale Deduct2_with_False < d_dwf: Deduct2
  by standard (fact bprv_prv)

context Deduct2_with_False begin

lemma consistent_B_consistent: consistent  $\implies$  B.consistent
  using B.consistent_def bprv_prv consistent_def by blast

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:  $\bigwedge \varphi. \varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{bprv } \varphi \implies \text{prv } \varphi$ 

sublocale Deduct2_with_False_Disj < dwf_dwfd: Deduct2_with_False
  by standard (fact bprv_prv)

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))
using assms Lq_num unfolding LLq_def by auto

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)))
using assms Lq_num2 unfolding LLq_def by auto

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: ∧ φ. φ ∈ fmla ⇒ Fvars φ = {} ⇒ prv φ ↔ (∃ prf ∈ proof. prfOf prf φ)

```

```

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  $\Rightarrow$  'trm ( $\langle \_ \rangle$ )  
assumes  
enc[simp,intro]:  $\bigwedge \varphi. \varphi \in \text{fmla} \implies \text{enc } \varphi \in \text{num}$   
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 ( $\langle \_ \rangle$ )  
and fls dsj  
and proof :: 'proof set and prfOf
```

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

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 ∈ fmla  
and  
  Fvars_N[simp]: Fvars N = {xx,yy}  
and  
  neg_implies_prv_N:  
  ∧ φ.  
  let NN = (λ t1 t2. psubst N [(t1,xx), (t2,yy)]) in  
  φ ∈ fmla ⟶ Fvars φ = {} ⟶ bprv (NN ⟨φ⟩ ⟨neg φ⟩)  
and  
  N_unique:  
  ∧ φ.  
  let NN = (λ t1 t2. psubst N [(t1,xx), (t2,yy)]) in
```

```

 $\varphi \in \text{fmla} \longrightarrow \text{Fvars } \varphi = \{\} \longrightarrow$ 
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 $NN \equiv \lambda t1 t2. \text{psubst } N [(t1,xx), (t2,yy)]$

lemma NN_def2: $t1 \in \text{trm} \implies t2 \in \text{trm} \implies yy \notin \text{FvarsT } t1 \implies$
 $NN t1 t2 = \text{subst} (\text{subst } N t1 xx) t2 yy$
unfolding NN_def **by** (rule psubst_eq_rawpsubst2[simplified]) auto

lemma NN_neg:

$\varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{bprv} (NN \langle \varphi \rangle \langle \text{neg } \varphi \rangle)$
using neg_implies_prv_N **unfolding** Let_def NN_def **by** meson

lemma NN_unique:

assumes $\varphi \in \text{fmla}$ $\text{Fvars } \varphi = \{\}$
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')))))
using assms N_unique **unfolding** Let_def NN_def **by** meson

lemma NN[simp,intro]:

$t1 \in \text{trm} \implies t2 \in \text{trm} \implies NN t1 t2 \in \text{fmla}$

unfolding NN_def **by** auto

lemma Fvars_NN[simp]: $t1 \in \text{trm} \implies t2 \in \text{trm} \implies yy \notin \text{FvarsT } t1 \implies$

$\text{Fvars} (NN t1 t2) = \text{FvarsT } t1 \cup \text{FvarsT } t2$

by (auto simp add: NN_def2 subst2_fresh_switch)

lemma [simp]:

$m \in \text{num} \implies n \in \text{num} \implies \text{subst} (NN m (Var yy)) n yy = NN m n$
 $m \in \text{num} \implies n \in \text{num} \implies \text{subst} (NN m (Var yy')) n yy = NN m (Var yy')$
 $m \in \text{num} \implies \text{subst} (NN m (Var yy')) (Var yy) yy' = NN m (Var yy)$
 $n \in \text{num} \implies \text{subst} (NN (Var xx) (Var yy)) n xx = NN n (Var yy)$
 $n \in \text{num} \implies \text{subst} (NN (Var xx) (Var xx')) n xx = NN n (Var xx')$
 $m \in \text{num} \implies n \in \text{num} \implies \text{subst} (NN m (Var xx')) n zz = NN m (Var xx')$
 $n \in \text{num} \implies \text{subst} (NN n (Var yy)) (Var xx') yy = NN n (Var xx')$
 $m \in \text{num} \implies n \in \text{num} \implies \text{subst} (NN m (Var xx')) n xx' = NN m n$
by (auto simp add: NN_def2 subst2_fresh_switch)

lemma NN_unique2:

assumes [simp]: $\varphi \in \text{fmla}$ $\text{Fvars } \varphi = \{\}$

shows

bprv (all yy
 (imp (NN $\langle \varphi \rangle$ (Var yy))
 (eql $\langle \text{neg } \varphi \rangle$ (Var yy))))

proof–

have 1: **bprv** (NN $\langle \varphi \rangle$ $\langle \text{neg } \varphi \rangle$)

using NN_neg[OF assms] .

have 2: **bprv** (all yy' (
 imp (cnj (NN $\langle \varphi \rangle$ $\langle \text{neg } \varphi \rangle$)
 (NN $\langle \varphi \rangle$ (Var yy'))))

```

      (eql ⟨neg φ⟩ (Var yy′)))
    using B.prv_allE[of yy, OF _ _ _ NN_unique, of φ ⟨neg φ⟩]
    by fastforce
  have 31: bprv (all yy′ (
    imp (NN ⟨φ⟩ ⟨neg φ⟩)
      (imp (NN ⟨φ⟩ (Var yy′))
        (eql ⟨neg φ⟩ (Var yy′))))))
    using B.prv_all_imp_cnj_rev[OF _ _ _ 2] by simp
  have 32: bprv (imp (NN ⟨φ⟩ ⟨neg φ⟩)
    (all yy′ (imp (NN ⟨φ⟩ (Var yy′))
      (eql ⟨neg φ⟩ (Var yy′))))))
    by (rule B.prv_all_imp[OF _ _ _ 31]) (auto simp: NN_def2)
  have 33: bprv (all yy′ (imp (NN ⟨φ⟩ (Var yy′))
    (eql ⟨neg φ⟩ (Var yy′))))
    by (rule B.prv_imp_mp [OF _ _ 32 1]) auto
  thus ?thesis using B.all_subst_rename_prv[OF _ _ _ 33, of yy] by simp
qed

```

lemma *NN_neg_unique*:

assumes [simp]: $\varphi \in \text{fmla}$ *Fvars* $\varphi = \{\}$

shows

$\text{bprv} (\text{imp} (\text{NN} \langle \varphi \rangle (\text{Var } yy))$
 $(\text{eql} \langle \text{neg } \varphi \rangle (\text{Var } yy)))$ (is $\text{bprv } ?A$)

proof –

have 0: $\text{bprv} (\text{all } yy ?A)$

using *NN_unique2*[of φ] **by** *simp*

show ?thesis **by** (rule *B.allE_id*[OF _ _ 0]) *auto*

qed

lemma *NN_exi_cnj*:

assumes $\varphi[\text{simp}]$: $\varphi \in \text{fmla}$ *Fvars* $\varphi = \{\}$ **and** $\chi[\text{simp}]$: $\chi \in \text{fmla}$

assumes *f*: *Fvars* $\chi = \{yy\}$

shows $\text{bprv} (\text{eqv} (\text{subst } \chi \langle \text{neg } \varphi \rangle yy)$
 $(\text{exi } yy (\text{cnj } \chi (\text{NN} \langle \varphi \rangle (\text{Var } yy))))))$

(is $\text{bprv} (\text{eqv } ?A ?B)$)

proof(*intro B.prv_eqvI*)

have *yy*: $yy \in \text{var}$ **by** *simp*

let ?*N* = $\text{NN} \langle \varphi \rangle (\text{Var } yy)$

have $\text{bprv} (\text{imp} (\text{subst } \chi \langle \text{neg } \varphi \rangle yy) ((\text{subst} (\text{cnj } \chi ?N) \langle \text{neg } \varphi \rangle yy)))$ **using** *NN_neg*[OF φ]

by (*simp add*: *B.prv_imp_cnj B.prv_imp_refl B.prv_imp_triv*)

thus $\text{bprv} (\text{imp } ?A ?B)$

by (*elim B.prv_prv_imp_trans*[rotated 3], *intro B.prv_exi_inst*) *auto*

next

have 00: $\text{bprv} (\text{imp} (\text{eql} \langle \text{neg } \varphi \rangle (\text{Var } yy)) (\text{imp } \chi (\text{subst } \chi \langle \text{neg } \varphi \rangle yy)))$

by (rule *B.prv_eql_subst_trm_id_rev*) *auto*

have 11: $\text{bprv} (\text{imp} (\text{NN} \langle \varphi \rangle (\text{Var } yy)) (\text{imp } \chi (\text{subst } \chi \langle \text{neg } \varphi \rangle yy)))$

using 00 *NN_neg_unique*[OF φ]

using *NN num Var Variable* $\varphi \chi$ *eql imp subst B.prv_prv_imp_trans*

by (*metis* (*no_types*, *lifting*) *enc in_num neg*)

hence $\text{bprv} (\text{imp} (\text{cnj } \chi (\text{NN} \langle \varphi \rangle (\text{Var } yy))) (\text{subst } \chi \langle \text{neg } \varphi \rangle yy))$

by (*simp add*: 11 *B.prv_cnj_imp_monoR2 B.prv_imp_com*)

hence 1: $\text{bprv} (\text{all } yy (\text{imp} (\text{cnj } \chi (\text{NN} \langle \varphi \rangle (\text{Var } yy))) (\text{subst } \chi \langle \text{neg } \varphi \rangle yy)))$

by (*simp add*: *B.prv_all_gen*)

have 2: *Fvars* ($\text{subst } \chi \langle \text{neg } \varphi \rangle yy$) = $\{\}$ **using** *f* **by** *simp*

show $\text{bprv} (\text{imp } ?B ?A)$ **using** 1 2

by (*simp add*: *B.prv_exi_imp*)

qed *auto*

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 ( $\langle\_\rangle$ )
  +
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 λ *t1 t2.* *psubst S* [(*t1,xx*), (*t2,yy*)]

lemma *SS_def2*: *t1* \in *trm* \implies *t2* \in *trm* \implies
yy \notin *FvarsT t1* \implies
SS t1 t2 = *subst* (*subst S t1 xx*) *t2 yy*
unfolding *SS_def* **by** (*rule psubst_eq_rawpsubst2[simplified]*) *auto*

lemma *subst_implies_prv_SS*:
 $\varphi \in$ fmla \implies *Fvars* φ = {*xx*} \implies *bprv* (*SS* $\langle\varphi\rangle$ \langle *subst* φ $\langle\varphi\rangle$ *xx* \rangle)
using *subst_implies_prv_S* **unfolding** *Let_def SS_def* **by** *meson*

lemma *SS_unique*:
 $\varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{xx\} \implies$
 $\text{bprv } (\text{all } yy \ (\text{all } yy' \ (\text{imp } (\text{cnj } (\text{SS } \langle \varphi \rangle (\text{Var } yy)) (\text{SS } \langle \varphi \rangle (\text{Var } yy')))$
 $\quad (\text{eql } (\text{Var } yy) (\text{Var } yy')))))$
using *S_unique unfolding Let_def SS_def by meson*

lemma *SS[simp,intro]*:
 $t1 \in \text{trm} \implies t2 \in \text{trm} \implies \text{SS } t1 \ t2 \in \text{fmla}$
unfolding *SS_def* **by** *auto*

lemma *Fvars_SS[simp]*: $t1 \in \text{trm} \implies t2 \in \text{trm} \implies yy \notin \text{FvarsT } t1 \implies$
 $\text{Fvars } (\text{SS } t1 \ t2) = \text{FvarsT } t1 \cup \text{FvarsT } t2$
by (*auto simp add: SS_def2 subst2_fresh_switch*)

lemma [*simp*]:
 $m \in \text{num} \implies p \in \text{num} \implies \text{subst } (\text{SS } m \ (\text{Var } yy)) \ p \ yy = \text{SS } m \ p$
 $m \in \text{num} \implies \text{subst } (\text{SS } m \ (\text{Var } yy')) \ (\text{Var } yy) \ yy' = \text{SS } m \ (\text{Var } yy)$
 $m \in \text{num} \implies p \in \text{num} \implies \text{subst } (\text{SS } m \ (\text{Var } yy')) \ p \ yy' = \text{SS } m \ p$
 $m \in \text{num} \implies p \in \text{num} \implies \text{subst } (\text{SS } m \ (\text{Var } yy')) \ p \ yy = \text{SS } m \ (\text{Var } yy')$
 $m \in \text{num} \implies \text{subst } (\text{SS } (\text{Var } xx) \ (\text{Var } yy)) \ m \ xx = \text{SS } m \ (\text{Var } yy)$
by (*auto simp add: SS_def2 subst_comp_num Let_def*)

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 $\text{exi } xx \ (\text{cnj } (\text{eql } (\text{Var } xx) (\text{enc } \varphi)) \ \varphi)$. Representability of self-soft-substitution will be an alternative assumption in the diagonalization lemma.

locale *Repr_SelfSoftSubst* =
Encode
 $\text{var } \text{trm } \text{fmla } \text{Var } \text{FvarsT } \text{substT } \text{Fvars } \text{subst}$
 num
 enc
+
Deduct2
 $\text{var } \text{trm } \text{fmla } \text{Var } \text{FvarsT } \text{substT } \text{Fvars } \text{subst}$
 num
 $\text{eql } \text{cnj } \text{imp } \text{all } \text{exi}$
 $\text{prv } \text{bprv}$
for
 $\text{var } :: \text{'var set and trm } :: \text{'trm set and fmla } :: \text{'fmla set}$
and $\text{Var } \text{FvarsT } \text{substT } \text{Fvars } \text{subst}$
and num
and $\text{eql } \text{cnj } \text{imp } \text{all } \text{exi}$
and $\text{prv } \text{bprv}$
and $\text{enc } (\langle _ \rangle)$
+
fixes $S :: \text{'fmla}$
assumes
 $S[\text{simp,intro!}]: S \in \text{fmla}$
and
 $\text{Fvars } S[\text{simp}]: \text{Fvars } S = \{xx,yy\}$
and
 $\text{softSubst_implies_prv } S:$

$\wedge \varphi.$
 let $SS = (\lambda t1 t2. psubst S [(t1,xx), (t2,yy)])$ in
 $\varphi \in fmla \longrightarrow Fvars \varphi = \{xx\} \longrightarrow$
 $bprv (SS \langle \varphi \rangle \langle softSubst \varphi \langle \varphi \rangle xx \rangle)$

and

$S_unique:$

$\wedge \varphi.$
 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 $SS_def2: t1 \in trm \implies t2 \in trm \implies$
 $yy \notin FvarsT t1 \implies$
 $SS t1 t2 = subst (subst S t1 xx) t2 yy$
unfolding SS_def **by** (rule $psubst_eq_rawpsubst2[simplified]$) *auto*

lemma $softSubst_implies_prv_SS:$

$\varphi \in fmla \implies Fvars \varphi = \{xx\} \implies bprv (SS \langle \varphi \rangle \langle softSubst \varphi \langle \varphi \rangle xx \rangle)$
using $softSubst_implies_prv_S$ **unfolding** $Let_def SS_def$ **by** *meson*

lemma $SS_unique:$

$\varphi \in fmla \implies Fvars \varphi = \{xx\} \implies$
 $bprv (all yy (all yy'$
 $(imp (cnj (SS \langle \varphi \rangle (Var yy)) (SS \langle \varphi \rangle (Var yy')))$
 $(eql (Var yy) (Var yy'))))))$

using S_unique **unfolding** $Let_def SS_def$ **by** *meson*

lemma $SS[simp,intro]:$

$t1 \in trm \implies t2 \in trm \implies SS t1 t2 \in fmla$
unfolding SS_def **by** *auto*

lemma $Fvars_SS[simp]: t1 \in trm \implies t2 \in trm \implies yy \notin FvarsT t1 \implies$
 $Fvars (SS t1 t2) = FvarsT t1 \cup FvarsT t2$
by (*auto simp add: SS_def2 subst2_fresh_switch*)

lemma $[simp]:$

$m \in num \implies p \in num \implies subst (SS m (Var yy)) p yy = SS m p$
 $m \in num \implies subst (SS m (Var yy')) (Var yy) yy' = SS m (Var yy)$
 $m \in num \implies p \in num \implies subst (SS m (Var yy')) p yy' = SS m p$
 $m \in num \implies p \in num \implies subst (SS m (Var yy')) p yy = SS m (Var yy')$
 $m \in num \implies subst (SS (Var xx) (Var yy)) m xx = SS m (Var yy)$
by (*auto simp add: SS_def2 subst_comp_num Let_def*)

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:
  ∧ p φ. let PPf = (λ t1 t2. psubst Pf [(t1,yy), (t2,xx)]) in
  p ∈ num ∧ φ ∈ fmla ∧ Fvars φ = {} → p ∉ encPf ' proof → bprv (neg (PPf p ⟨φ⟩))
begin

```

PPf is the instantiation combinator of Pf:

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

lemma prfOf_PPf:
assumes prf ∈ proof φ ∈ fmla Fvars φ = {} prfOf prf φ
shows bprv (PPf (encPf prf) ⟨φ⟩)
using assms prfOf_Pf **unfolding** PPf_def **by** auto

lemma *not_prfOf_PPf*:

assumes $prf \in proof \ \varphi \in fmla \ Fvars \ \varphi = \{\} \ \neg \ prfOf \ prf \ \varphi$

shows $bprv \ (neg \ (PPf \ (encPf \ prf) \ \langle\varphi\rangle))$

using *assms not_prfOf_Pf unfolding PPf_def by auto*

lemma *Clean_PPf_encPf*:

assumes $\varphi \in fmla \ Fvars \ \varphi = \{\}$ **and** $p \in num \ p \notin encPf \ 'proof$

shows $bprv \ (neg \ (PPf \ p \ \langle\varphi\rangle))$

using *assms Clean_Pf_encPf unfolding PPf_def by auto*

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

unfolding *PPf_def by auto*

lemma *PPf_def2*: $t1 \in trm \implies t2 \in trm \implies xx \notin FvarsT \ t1 \implies$

$PPf \ t1 \ t2 = subst \ (subst \ Pf \ t1 \ yy) \ t2 \ xx$

unfolding *PPf_def by (rule psubst_eq_rawpsubst2[simplified]) auto*

lemma *Fvars_PPf[simp]*:

$t1 \in trm \implies t2 \in trm \implies xx \notin FvarsT \ t1 \implies$

$Fvars \ (PPf \ t1 \ t2) = FvarsT \ t1 \cup FvarsT \ t2$

by *(auto simp add: PPf_def2 subst2_fresh_switch)*

lemma *[simp]*:

$n \in num \implies subst \ (PPf \ (Var \ yy) \ (Var \ xx)) \ n \ xx = PPf \ (Var \ yy) \ n$

$m \in num \implies n \in num \implies subst \ (PPf \ (Var \ yy) \ m) \ n \ yy = PPf \ n \ m$

$n \in num \implies subst \ (PPf \ (Var \ yy) \ (Var \ xx)) \ n \ yy = PPf \ n \ (Var \ xx)$

$m \in num \implies n \in num \implies subst \ (PPf \ m \ (Var \ xx)) \ n \ xx = PPf \ m \ n$

$m \in num \implies subst \ (PPf \ (Var \ zz) \ (Var \ xx')) \ m \ zz = PPf \ m \ (Var \ xx')$

$m \in num \implies n \in num \implies subst \ (PPf \ m \ (Var \ xx')) \ n \ xx' = PPf \ m \ n$

$n \in num \implies subst \ (PPf \ (Var \ zz) \ (Var \ xx')) \ n \ xx' = PPf \ (Var \ zz) \ n$

$m \in num \implies n \in num \implies subst \ (PPf \ (Var \ zz) \ n) \ m \ zz = PPf \ m \ n$

by *(auto simp add: PPf_def2 subst2_fresh_switch)*

lemma *B_consistent_prfOf_iff_PPf*:

$B.consistent \implies prf \in proof \implies \varphi \in fmla \implies Fvars \ \varphi = \{\} \implies prfOf \ prf \ \varphi \longleftrightarrow bprv \ (PPf \ (encPf \ prf) \ \langle\varphi\rangle)$

unfolding *B.consistent_def3 using not_prfOf_PPf[of prf φ] prfOf_PPf[of prf φ] by force*

lemma *consistent_prfOf_iff_PPf*:

$consistent \implies prf \in proof \implies \varphi \in fmla \implies Fvars \ \varphi = \{\} \implies prfOf \ prf \ \varphi \longleftrightarrow bprv \ (PPf \ (encPf \ prf) \ \langle\varphi\rangle)$

using *B_consistent_prfOf_iff_PPf[OF dwf_dwfd.consistent_B_consistent]* .

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 $\varphi[\text{simp, intro!}]$: $\varphi \in \text{fmla } F\text{vars } \varphi = \{xx\}$
shows $\exists \psi. \psi \in \text{fmla} \wedge F\text{vars } \psi = \{\}$ $\wedge \text{bprv } (\text{eqv } \psi (\text{subst } \varphi \langle \psi \rangle xx))$

proof –

let $?phi = \lambda t. \text{subst } \varphi t xx$
define χ **where** $\chi \equiv \text{exi } yy (\text{cnj } (?phi (Var yy)) (SS (Var xx) (Var yy)))$
have $\chi[\text{simp, intro!}]$: $\chi \in \text{fmla}$ **unfolding** χ_def **by** *auto*
let $?chi = \lambda t. \text{subst } \chi t xx$
define ψ **where** $\psi \equiv ?chi \langle \chi \rangle$
have $\psi[\text{simp, intro!}]$: $\psi \in \text{fmla}$ **unfolding** ψ_def **by** *auto*
have $f\chi[\text{simp}]$: $F\text{vars } \chi = \{xx\}$ **unfolding** χ_def **by** *auto*
hence $F\text{vars } \psi$: $F\text{vars } \psi = \{\}$ **unfolding** ψ_def **by** *auto*
have 1: $\text{bprv } (SS \langle \chi \rangle \langle \psi \rangle)$
using *subst_implies_prv_SS[OF χ]* **unfolding** ψ_def **by** *simp*
have 2: $\text{bprv } (\text{all } yy' (\text{imp } (\text{cnj } (SS \langle \chi \rangle \langle \psi \rangle) (SS \langle \chi \rangle (Var yy')))) (\text{eql } \langle \psi \rangle (Var yy'))))$
using $F\text{vars } \psi$ *B.prv_allE[OF _ _ _ SS_unique, of $\chi \langle \psi \rangle$]*
by *fastforce*
have 31: $\text{bprv } (\text{all } yy' (\text{imp } (SS \langle \chi \rangle \langle \psi \rangle) (\text{imp } (SS \langle \chi \rangle (Var yy')) (\text{eql } \langle \psi \rangle (Var yy')))))$
using *B.prv_all_imp_cnj_rev[OF _ _ _ _ 2]* **by** *simp*
have 32: $\text{bprv } (\text{imp } (SS \langle \chi \rangle \langle \psi \rangle) (\text{all } yy' (\text{imp } (SS \langle \chi \rangle (Var yy')) (\text{eql } \langle \psi \rangle (Var yy')))))$

```

    by (intro B.prv_all_imp[OF _ _ _ 31]) (auto simp: SS_def2)
  have 33: bprv (all yy' (imp (SS ⟨χ⟩ (Var yy'))
    (eq1 ⟨ψ⟩ (Var yy'))))
    by (rule B.prv_imp_mp [OF _ _ 32 1]) auto
  have 3: bprv (all yy (imp (SS ⟨χ⟩ (Var yy))
    (eq1 ⟨ψ⟩ (Var yy))))
    using B.all_subst_rename_prv[OF _ _ _ 33, of yy]
    by fastforce
  have 41: bprv (imp (?phi ⟨ψ⟩)
    (cnj (?phi ⟨ψ⟩)
    (SS ⟨χ⟩ ⟨ψ⟩)))
    by (auto intro: in_num B.prv_imp_cnj B.prv_imp_refl B.prv_imp_triv[OF _ _ 1])
  have [simp]: subst (subst φ ⟨ψ⟩ xx) ⟨ψ⟩ yy = subst φ ⟨ψ⟩ xx
    by (intro subst_notIn) auto
  have [simp]: subst (subst φ (Var yy) xx) ⟨ψ⟩ yy = subst φ ⟨ψ⟩ xx
    by (intro subst_subst) auto
  have 42: bprv (ex1 yy (imp (?phi ⟨ψ⟩)
    (cnj (?phi (Var yy))
    (SS ⟨χ⟩ (Var yy)))))
    using 41 by (intro B.prv_ex1[OF _ _ ⟨ψ⟩]) auto
  have 4: bprv (imp (?phi ⟨ψ⟩) (?chi ⟨χ⟩))
    using B.prv_imp_ex1[OF _ _ _ 42, simplified]
    by (subst χ_def) (auto simp add: subst_comp_num)
  have 50: bprv (all yy (
    (imp (eq1 ⟨ψ⟩ (Var yy))
    (imp (?phi (Var yy))
    (?phi ⟨ψ⟩))))))
    using B.prv_all_eq1[OF yy xx φ ⟨ψ⟩ Var yy] by simp
  have 51: bprv (all yy (
    (imp (SS ⟨χ⟩ (Var yy))
    (imp (?phi (Var yy))
    (?phi ⟨ψ⟩)))))) using B.prv_all_imp_trans[OF _ _ _ 3 50] by simp
  have 52: bprv (all yy (
    (imp (cnj (?phi (Var yy))
    (SS ⟨χ⟩ (Var yy))
    (?phi ⟨ψ⟩)))))) using B.prv_all_imp_cnj[OF _ _ _ 51] by simp
  have 5: bprv (imp (?chi ⟨χ⟩) (?phi ⟨ψ⟩))
    using B.prv_ex1_imp[OF _ _ _ 52, simplified]
    by (subst χ_def) (simp add: subst_comp_num)
  have 6: bprv (eqv (?chi ⟨χ⟩) (?phi ⟨ψ⟩))
    using B.prv_cnjI[OF _ _ 5 4] unfolding eqv_def by simp
  have 7: bprv (eqv ψ (?phi ⟨ψ⟩)) using 6 unfolding ψ_def .
  show ?thesis using ψ 7 Fvars_ψ by blast
qed

```

Making this existential into a function.

definition *diag* :: 'fmla \Rightarrow 'fmla **where**

diag φ \equiv SOME ψ. ψ \in fmla \wedge Fvars ψ = {} \wedge bprv (eqv ψ (subst φ ⟨ψ⟩ xx))

theorem *diag_everything*:

assumes φ \in fmla **and** Fvars φ = {xx}

shows *diag* φ \in fmla \wedge Fvars (*diag* φ) = {} \wedge bprv (eqv (*diag* φ) (subst φ ⟨*diag* φ⟩ xx))

unfolding *diag_def* **using** someI_ex[OF diagonalization[OF assms]] .

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!*]: $\varphi \in \text{fm}la \text{ Fvars } \varphi = \{xx\}$
shows $\exists \psi. \psi \in \text{fm}la \wedge \text{Fvars } \psi = \{\} \wedge \text{bprv } (\text{eqv } \psi (\text{subst } \varphi \langle \psi \rangle xx))$

proof—

let $?phi = \lambda t. \text{subst } \varphi t xx$
define χ **where** $\chi \equiv \text{exi } yy (\text{cnj } (?phi (Var yy)) (SS (Var xx) (Var yy)))$
have χ [*simp,intro!*]: $\chi \in \text{fm}la$ **unfolding** χ_def **by** *auto*
let $?chi = \lambda t. \text{softSubst } \chi t xx$
define ψ **where** $\psi \equiv ?chi \langle \chi \rangle$
have ψ [*simp,intro!*]: $\psi \in \text{fm}la$ **unfolding** ψ_def **by** *auto*
have $f\chi$ [*simp*]: $\text{Fvars } \chi = \{xx\}$ **unfolding** χ_def **by** *auto*
hence $\text{Fvars } \psi$: $\text{Fvars } \psi = \{\}$ **unfolding** ψ_def **by** *auto*
have 1: $\text{bprv } (SS \langle \chi \rangle \langle \psi \rangle)$
using *softSubst_implies_prv_SS[OF χ]* **unfolding** ψ_def **by** *simp*
have 2: $\text{bprv } (\text{all } yy' (\text{imp } (\text{cnj } (SS \langle \chi \rangle \langle \psi \rangle) (SS \langle \chi \rangle (Var yy')) (eql \langle \psi \rangle (Var yy')))))$
using $\text{Fvars } \psi$ *B.prv_allE[OF _ _ _ SS_unique, of $\chi \langle \psi \rangle$]*
by *fastforce*
have 31: $\text{bprv } (\text{all } yy' (\text{imp } (SS \langle \chi \rangle \langle \psi \rangle) (\text{imp } (SS \langle \chi \rangle (Var yy')) (eql \langle \psi \rangle (Var yy')))))$
using *B.prv_all_imp_cnj_rev[OF _ _ _ _ 2]* **by** *simp*
have 32: $\text{bprv } (\text{imp } (SS \langle \chi \rangle \langle \psi \rangle) (\text{all } yy' (\text{imp } (SS \langle \chi \rangle (Var yy')) (eql \langle \psi \rangle (Var yy')))))$
by (*intro B.prv_all_imp[OF _ _ _ 31]*) (*auto simp: SS_def2*)
have 33: $\text{bprv } (\text{all } yy' (\text{imp } (SS \langle \chi \rangle (Var yy')) (eql \langle \psi \rangle (Var yy'))))$
by (*rule B.prv_imp_mp [OF _ _ 32 1]*) *auto*
have 3: $\text{bprv } (\text{all } yy (\text{imp } (SS \langle \chi \rangle (Var yy)) (eql \langle \psi \rangle (Var yy))))$
using *B.all_subst_rename_prv[OF _ _ _ 33, of yy]*
by *fastforce*
have 41: $\text{bprv } (\text{imp } (?phi \langle \psi \rangle) (\text{cnj } (?phi \langle \psi \rangle) (SS \langle \chi \rangle \langle \psi \rangle)))$
by (*auto intro: in_num B.prv_imp_cnj B.prv_imp_refl B.prv_imp_triv[OF _ _ 1]*)
have [*simp*]: $\text{subst } (\text{subst } \varphi \langle \psi \rangle xx) \langle \psi \rangle yy = \text{subst } \varphi \langle \psi \rangle xx$
by (*intro subst_notIn*) *auto*
have [*simp*]: $\text{subst } (\text{subst } \varphi (Var yy) xx) \langle \psi \rangle yy = \text{subst } \varphi \langle \psi \rangle xx$
by (*intro subst_subst*) *auto*
have 42: $\text{bprv } (\text{exi } yy (\text{imp } (?phi \langle \psi \rangle) (\text{cnj } (?phi (Var yy)) (SS \langle \chi \rangle (Var yy))))$
using 41 **by** (*intro B.prv_exiI[of _ _ $\langle \psi \rangle$]*) *auto*
have 4: $\text{bprv } (\text{imp } (?phi \langle \psi \rangle) (\text{subst } \chi \langle \chi \rangle xx))$
using *B.prv_imp_exi[OF _ _ _ 42,simplified]*
by (*subst χ_def*) (*auto simp add: subst_comp_num*)

```

moreover have bprv (imp (subst  $\chi$   $\langle\chi\rangle$   $xx$ ) (?chi  $\langle\chi\rangle$ )) by (rule B.prv_subst_imp_softSubst) auto
ultimately have 4: bprv (imp (?phi  $\langle\psi\rangle$ ) (?chi  $\langle\chi\rangle$ ))
  by (rule B.prv_prv_imp_trans[rotated -2]) auto
have 50: bprv (all yy (
  (imp (eqI  $\langle\psi\rangle$  (Var yy))
  (imp (?phi (Var yy))
  (?phi  $\langle\psi\rangle$ ))))
  using B.prv_all_eqI[of yy  $xx$   $\varphi$   $\langle\psi\rangle$  Var yy] by simp
have 51: bprv (all yy (
  (imp (SS  $\langle\chi\rangle$  (Var yy))
  (imp (?phi (Var yy))
  (?phi  $\langle\psi\rangle$ )))) using B.prv_all_imp_trans[OF _ _ _ 3 50] by simp
have 52: bprv (all yy (
  (imp (cnj (?phi (Var yy))
  (SS  $\langle\chi\rangle$  (Var yy))
  (?phi  $\langle\psi\rangle$ )))) using B.prv_all_imp_cnj[OF _ _ _ 51] by simp
have bprv (imp (?chi  $\langle\chi\rangle$ ) (subst  $\chi$   $\langle\chi\rangle$   $xx$ )) by (rule B.prv_softSubst_imp_subst) auto
moreover have bprv (imp (subst  $\chi$   $\langle\chi\rangle$   $xx$ ) (?phi  $\langle\psi\rangle$ ))
  using B.prv_exi_imp[OF _ _ _ 52, simplified]
  by (subst  $\chi$ _def) (simp add: subst_comp_num)
ultimately have 5: bprv (imp (?chi  $\langle\chi\rangle$ ) (?phi  $\langle\psi\rangle$ ))
  by (rule B.prv_prv_imp_trans[rotated -2]) auto
have 6: bprv (eqv (?chi  $\langle\chi\rangle$ ) (?phi  $\langle\psi\rangle$ ))
  using B.prv_cnjI[OF _ _ 5 4] unfolding eqv_def by simp
have 7: bprv (eqv  $\psi$  (?phi  $\langle\psi\rangle$ )) using 6 unfolding  $\psi$ _def .
show ?thesis using  $\psi$  7 Fvars_ $\psi$  by blast
qed

```

Making this existential into a function.

definition diag :: 'fmla \Rightarrow 'fmla **where**

diag $\varphi \equiv$ SOME ψ . $\psi \in$ 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))

unfolding diag_def **using** someI_ex[OF diagonalization[OF assms]] .

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 ∈ fmla
and
  Fvars_PP[simp]: Fvars P = {xx}
and
  HBL1:  $\bigwedge \varphi. \varphi \in fmla \implies Fvars \varphi = \{\} \implies prv \varphi \implies bprv (subst P \langle \varphi \rangle xx)$ 
begin

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

lemma PP[simp]:  $\bigwedge t. t \in trm \implies PP t \in fmla$ 
  unfolding PP_def by auto

lemma Fvars_PP[simp]:  $\bigwedge t. t \in trm \implies Fvars (PP t) = FvarsT t$ 
  unfolding PP_def by auto
```

lemma *[simp]*:
 $n \in \text{num} \implies \text{subst } (PP \text{ (Var } yy)) \ n \ yy = PP \ n$
 $n \in \text{num} \implies \text{subst } (PP \text{ (Var } xx)) \ n \ xx = PP \ n$
unfolding *PP_def* **by** *auto*

lemma *HBL1_PP*:
 $\varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{prv } \varphi \implies \text{bprv } (PP \ \langle \varphi \rangle)$
using *HBL1* **unfolding** *PP_def* **by** *auto*

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 $P :: 'fmla \text{ where } P \equiv \text{exi } yy \ (PPf \text{ (Var } yy)) \text{ (Var } xx)$

lemma $P[\text{simp}, \text{intro!}]$: $P \in \text{fmla}$ **and** $\text{Fvars_}P[\text{simp}]$: $\text{Fvars } P = \{xx\}$
unfolding P_def **by** $(\text{auto } \text{simp}: PPf_def2)$

We infer HBL1 from Pf representing prv :

lemma *HBL1*:
assumes $\varphi \in \text{fmla}$ $\text{Fvars } \varphi = \{\}$ **and** $p\varphi$: $\text{prv } \varphi$
shows $\text{bprv } (\text{subst } P \ \langle \varphi \rangle \ xx)$
proof —
obtain prf **where** pf : $\text{prf} \in \text{proof}$ **and** $\text{prfOf } \text{prf } \varphi$ **using** $\text{prv_prfOf } \text{assms}$ **by** *auto*
hence 0 : $\text{bprv } (PPf \text{ (encPf } \text{prf})) \text{ (enc } \varphi)$
using $\text{prfOf_}PPf \ \varphi$ **by** *auto*
have 1 : $\text{subst } (\text{subst } Pf \text{ (encPf } \text{prf}) \ yy) \ \langle \varphi \rangle \ xx = \text{subst } (\text{subst } Pf \ \langle \varphi \rangle \ xx) \ (\text{substT } (\text{encPf } \text{prf}) \ \langle \varphi \rangle \ xx)$
 yy
using $\text{assms } \text{pf}$ **by** $(\text{intro } \text{subst_compose_diff})$ *auto*
show $?thesis$ **using** 0 **unfolding** P_def **using** assms
by $(\text{auto } \text{simp}: PPf_def2 \ 1 \ \text{pf } \text{intro!}: B.\text{prv_exiI}[\text{of } _ _ \text{encPf } \text{prf}])$
qed

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 $\text{not_prv_prv_neg_}PPf$:
assumes $[\text{simp}]$: $\varphi \in \text{fmla}$ $\text{Fvars } \varphi = \{\}$ **and** p : $\neg \text{prv } \varphi$ **and** $n[\text{simp}]$: $n \in \text{num}$
shows $\text{bprv } (\text{neg } (PPf \ n \ \langle \varphi \rangle))$
proof —
have $\forall \text{prf} \in \text{proof}. \neg \text{prfOf } \text{prf } \varphi$ **using** $\text{prv_prfOf } p$ **by** *auto*
hence $\forall \text{prf} \in \text{proof}. \text{bprv } (\text{neg } (PPf \text{ (encPf } \text{prf}) \ \langle \varphi \rangle))$ **using** $\text{not_prfOf_}PPf$ **by** *auto*
thus $?thesis$ **using** $\text{not_prfOf_}PPf$ **using** $\text{Clean_}PPf_encPf$ **by** $(\text{cases } n \in \text{encPf } ' \text{proof})$ *auto*
qed

```

lemma consistent_not_prv_not_prv_PPf:
assumes c: consistent
and 0[simp]:  $\varphi \in \text{fmla } F\text{vars } \varphi = \{\} \neg \text{prv } \varphi \ n \in \text{num}$ 
shows  $\neg \text{bprv } (\text{PPf } n \ \langle \varphi \rangle)$ 
  using not_prv_prv_neg_PPf[OF 0] c[THEN dwf_dwfd.consistent_B_consistent] unfolding B.consistent_def3
by auto

```

end — context *CleanRepr_Proofs*

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

```

sublocale CleanRepr_Proofs < wrepr: HBL1
  where P = P
  using HBL1 by unfold_locales auto

```

5.2.2 Sufficient condition for the converse of HBL1

```

context CleanRepr_Proofs
begin

```

```

lemma PP_PPf:
assumes  $\varphi \in \text{fmla}$ 
shows  $\text{wrepr.PP } \langle \varphi \rangle = \text{exi } yy \ (\text{PPf } (\text{Var } yy) \ \langle \varphi \rangle)$ 
  unfolding wrepr.PP_def using assms
  by (auto simp: PP_def2 P_def)

```

The converse of HBL1 condition follows from (the standard notion of) ω -consistency for *bprv* and strong representability of proofs:

```

lemma omegaconsistentStd1_HBL1_rev:
assumes oc: B.omegaconsistentStd1
and  $\varphi$ [simp]:  $\varphi \in \text{fmla } F\text{vars } \varphi = \{\}$ 
and iPP:  $\text{bprv } (\text{wrepr.PP } \langle \varphi \rangle)$ 
shows  $\text{prv } \varphi$ 
proof—
  have 0:  $\text{bprv } (\text{exi } yy \ (\text{PPf } (\text{Var } yy) \ \langle \varphi \rangle))$  using iPP by (simp add: PP_PPf)
  {assume  $\neg \text{prv } \varphi$ 
   hence  $\forall n \in \text{num}. \text{bprv } (\text{neg } (\text{PPf } n \ \langle \varphi \rangle))$  using not_prv_prv_neg_PPf by auto
   hence  $\neg \text{bprv } (\text{exi } yy \ (\text{PPf } (\text{Var } yy) \ \langle \varphi \rangle))$ 
   using oc unfolding B.omegaconsistentStd1_def using  $\varphi$  by auto
   hence False using 0 by simp
  }
  thus ?thesis by auto
qed

```

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:  $\bigwedge \varphi \chi. \varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{Fvars } \chi = \{\} \implies$ 
    bprv (imp (cnj (PP ⟨φ⟩) (PP ⟨imp φ χ⟩))
      (PP ⟨χ⟩))
and
  HBL3:  $\bigwedge \varphi. \varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{bprv (imp (PP ⟨φ⟩) (PP ⟨PP ⟨φ⟩⟩))}$ 
begin

```

The implicational form of HBL2:

```

lemma HBL2_imp:
   $\bigwedge \varphi \chi. \varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{Fvars } \chi = \{\} \implies$ 
  bprv (imp (PP ⟨imp φ χ⟩) (imp (PP ⟨φ⟩) (PP ⟨χ⟩)))
using HBL2 by (simp add: B.prv_cnj_imp B.prv_imp_com)

```

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

```

lemma HBL2_imp2:
assumes  $\varphi \in \text{fmla}$  and  $\chi \in \text{fmla}$   $\text{Fvars } \varphi = \{\}$   $\text{Fvars } \chi = \{\}$ 
assumes bprv (PP ⟨imp φ χ⟩)
shows bprv (imp (PP ⟨φ⟩) (PP ⟨χ⟩))
using assms by (meson HBL2_imp PP enc imp num B.prv_imp_mp subsetCE)

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 $\varphi G :: 'fmla$ **where** $\varphi G \equiv \text{diag } (\text{neg } P)$

lemma $\varphi G[\text{simp, intro!}]$: $\varphi G \in \text{fmla}$
and

```

Fvars_φG[simp]: Fvars φG = {}
unfolding φG_def PP_def by auto

```

```

lemma bprv_φG_eqv:
bprv (eqv φG (neg (PP ⟨φG⟩)))
unfolding φG_def PP_def using bprv_diag_eqv[of neg P] by simp

```

```

lemma prv_φG_eqv:
prv (eqv φG (neg (PP ⟨φG⟩)))
using bprv_prv[OF _ _ bprv_φG_eqv, simplified] .

```

```

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 by standard

```

```

context Goedel_Form_Proofs
begin

```

```

lemma bprv_φG_eqv_not_exi_PPf:
bprv (eqv φG (neg (exi yy (PPf (Var yy) ⟨φG⟩))))
proof –
  have P: P = exi yy Pf using P_def by (simp add: PPf_def2)
  hence subst P ⟨φG⟩ xx = subst (exi yy Pf) ⟨φG⟩ xx by auto

```

hence $\text{subst } P \langle \varphi G \rangle xx = \text{exi } yy (\text{subst } Pf \langle \varphi G \rangle xx)$ **by** *simp*
thus *?thesis* **using** *bprv_φG_eqv* **by** (*simp add: wrepr.PP_def PPf_def2*)
qed

lemma *prv_φG_eqv_not_exi_PPf*:
 $\text{prv } (\text{eqv } \varphi G (\text{neg } (\text{exi } yy (\text{PPf } (\text{Var } yy) \langle \varphi G \rangle))))$
using *bprv_prv[OF _ _ bprv_φG_eqv_not_exi_PPf, simplified]* .

lemma *bprv_φG_eqv_all_not_PPf*:
 $\text{bprv } (\text{eqv } \varphi G (\text{all } yy (\text{neg } (\text{PPf } (\text{Var } yy) \langle \varphi G \rangle))))$
by (*rule B.prv_eqv_trans[OF _ _ _ bprv_φG_eqv_not_exi_PPf B.prv_neg_exi_eqv_all_neg]*) *auto*

lemma *prv_φG_eqv_all_not_PPf*:
 $\text{prv } (\text{eqv } \varphi G (\text{all } yy (\text{neg } (\text{PPf } (\text{Var } yy) \langle \varphi G \rangle))))$
using *bprv_prv[OF _ _ bprv_φG_eqv_all_not_PPf, simplified]* .

lemma *bprv_eqv_all_not_PPf_imp_φG*:
 $\text{bprv } (\text{imp } (\text{all } yy (\text{neg } (\text{PPf } (\text{Var } yy) \langle \varphi G \rangle))) \varphi G)$
using *bprv_φG_eqv_all_not_PPf* **by** (*auto intro: B.prv_imp_eqvER*)

lemma *prv_eqv_all_not_PPf_imp_φG*:
 $\text{prv } (\text{imp } (\text{all } yy (\text{neg } (\text{PPf } (\text{Var } yy) \langle \varphi G \rangle))) \varphi G)$
using *bprv_prv[OF _ _ bprv_eqv_all_not_PPf_imp_φG, simplified]* .

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 ∈ proof ⇒ φ ∈ fmla ⇒ Fvars φ = {} ⇒
  bprv (PPf (encPf prf) ⟨φ⟩) ∨ bprv (neg (PPf (encPf prf) ⟨φ⟩))
using not_prfOf_PPf prfOf_PPf by blast

```

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

```

lemma isTrue_PPf_Implies_bprv_PPf:
prf ∈ proof ⇒ φ ∈ fmla ⇒ Fvars φ = {} ⇒
  isTrue (PPf (encPf prf) ⟨φ⟩) ⇒ bprv (PPf (encPf prf) ⟨φ⟩)
by (metis FvarsT_num Fvars_PPf Fvars_fls PPf
  Un_empty empty_iff enc encPf fls in_num isTrue_prv_PPf_prf_or_neg
  neg_def B.not_isTrue_fls B.prv_imp_implies_isTrue)

```

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

```

lemma isTrue_implies_bprv_PPf:
assumes [simp]: n ∈ num φ ∈ fmla Fvars φ = {}
and iT: isTrue (PPf n ⟨φ⟩)
shows bprv (PPf n ⟨φ⟩)
proof(cases n ∈ encPf ‘ proof)
  case True
  thus ?thesis
  using iT isTrue_PPf_Implies_bprv_PPf by auto
next
  case False
  hence bprv (neg (PPf n ⟨φ⟩)) by (simp add: Clean_PPf_encPf)
  hence isTrue (neg (PPf n ⟨φ⟩)) by (intro B.sound_isTrue) auto
  hence False using iT by (intro B.isTrue_neg_excl) auto
  thus ?thesis by auto
qed

```

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

```

lemma isTrue_iff_bprv_PPf:
∧ n φ. n ∈ num ⇒ φ ∈ fmla ⇒ Fvars φ = {} ⇒ isTrue (PPf n ⟨φ⟩) ↔ bprv (PPf n ⟨φ⟩)
using isTrue_implies_bprv_PPf
using exists_no_Fvars B.not_isTrue_fls B.sound_isTrue by auto

```

Truth of the provability representation implies provability (TIP):

```

lemma TIP:
assumes φ[simp]: φ ∈ fmla Fvars φ = {}
and iPP: isTrue (wrepr.PP ⟨φ⟩)
shows prv φ
proof –
  have isTrue (exi yy (PPf (Var yy) ⟨φ⟩)) using iPP unfolding PP_PPf[OF φ(1)] .
  from B.isTrue_exi[OF _ _ _ this]
  obtain n where n[simp]: n ∈ num and isTrue (PPf n ⟨φ⟩) by auto
  hence pP: bprv (PPf n ⟨φ⟩) using isTrue_implies_bprv_PPf by auto
  hence ¬ bprv (neg (PPf n ⟨φ⟩))
  using B.prv_neg_excl[of PPf n ⟨φ⟩] by auto
  then obtain prf where prf[simp]: prf ∈ proof and nn: n = encPf prf
  using assms n Clean_PPf_encPf φ(1) by blast
  have prfOf prf φ using pP unfolding nn using prfOf_iff_PPf by auto
  thus ?thesis using prv_prfOf by auto
qed

```

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*: $\forall \varphi. \varphi \in \text{fm}la \wedge \text{Fvars } \varphi = \{\} \wedge \text{isTrue } (\text{wrepr.PP } \langle \varphi \rangle) \longrightarrow \text{prv } \varphi$

shows $\forall \varphi. \varphi \in \text{fm}la \wedge \text{Fvars } \varphi = \{\} \wedge \text{bprv } (\text{wrepr.PP } \langle \varphi \rangle) \longrightarrow \text{prv } \varphi$

using *B.sound_isTrue*[of *wrepr.PP* $\langle _ \rangle$] *TIP* **by** *auto*

end — context *Minimal_Truth_Soundness_Proof_Repr*

7.1 Proof recovery from *HBL1_iff*

locale *Minimal_Truth_Soundness_HBL1iff_Cmpl_Pf* =
HBL1

var trm fm}la Var FvarsT substT Fvars subst
num
eql cnj imp all exi
prv bprv
enc
P

+

B : *Minimal_Truth_Soundness*

var trm fm}la Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
bprv
isTrue

+

Deduct_with_False_Disj

var trm fm}la Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
prv

for

var :: '*var set and trm* :: '*trm set and fm}la* :: '*fm}la set*

and *Var FvarsT substT Fvars subst*

and *eql cnj imp all exi*

and *fls*

and *dsj*

and *num*

and *enc* ($\langle \langle _ \rangle \rangle$)

and *prv bprv*

and *P*

and *isTrue*

+

fixes *Pf* :: '*fm}la*

assumes

— *Pf* is a formula with free variables *xx yy*:

Pf[*simp,intro!*]: *Pf* \in *fm}la*

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:

$\varphi \in \text{fm}la \implies \text{Fvars } \varphi = \{\} \implies$
 $\text{let } PPf = (\lambda t1 t2. \text{psubst } Pf [(t1, yy), (t2, xx)]) \text{ in}$
 $\text{bprv } (\text{eqv } (\text{subst } P \langle \varphi \rangle xx) (\text{exi } yy (PPf (Var yy) \langle \varphi \rangle)))$
assumes
 — We assume both *HBL1* and *HBL1_rev*, i.e., an iff version:
HBL1_iff: $\bigwedge \varphi. \varphi \in \text{fm}la \implies \text{Fvars } \varphi = \{\} \implies \text{bprv } (PP \langle \varphi \rangle) \longleftrightarrow \text{prv } \varphi$
and
Compl_Pf:
 $\bigwedge n \varphi. n \in \text{num} \implies \varphi \in \text{fm}la \implies \text{Fvars } \varphi = \{\} \implies$
 $\text{let } PPf = (\lambda t1 t2. \text{psubst } Pf [(t1, yy), (t2, xx)]) \text{ in}$
 $\text{isTrue } (PPf n \langle \varphi \rangle) \longrightarrow \text{bprv } (PPf n \langle \varphi \rangle)$
begin

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

lemma *PP_def*: $PP t = \text{subst } P t xx$ **using** *PP_def* **by** *auto*

lemma *PP_PPf_eqv*:

$\varphi \in \text{fm}la \implies \text{Fvars } \varphi = \{\} \implies \text{bprv } (\text{eqv } (PP \langle \varphi \rangle) (\text{exi } yy (PPf (Var yy) \langle \varphi \rangle)))$
using *PP_def* *PPf_def* *P_Pf* **by** *auto*

lemma *PPf[simp,intro!]*: $t1 \in \text{trm} \implies t2 \in \text{trm} \implies xx \notin \text{FvarsT } t1 \implies PPf t1 t2 \in \text{fm}la$
unfolding *PPf_def* **by** *auto*

lemma *PPf_def2*: $t1 \in \text{trm} \implies t2 \in \text{trm} \implies xx \notin \text{FvarsT } t1 \implies$
 $PPf t1 t2 = \text{subst } (\text{subst } Pf t1 yy) t2 xx$
unfolding *PPf_def* **by** (*rule* *psubst_eq_rawpsubst2[simplified]*) *auto*

lemma *Fvars_PPf[simp]*:

$t1 \in \text{trm} \implies t2 \in \text{trm} \implies xx \notin \text{FvarsT } t1 \implies \text{Fvars } (PPf t1 t2) = \text{FvarsT } t1 \cup \text{FvarsT } t2$
by (*auto* *simp* *add*: *PPf_def2* *subst2_fresh_switch*)

lemma [*simp*]:

$n \in \text{num} \implies \text{subst } (PPf (Var yy) (Var xx)) n xx = PPf (Var yy) n$
 $m \in \text{num} \implies n \in \text{num} \implies \text{subst } (PPf (Var yy) m) n yy = PPf n m$
 $n \in \text{num} \implies \text{subst } (PPf (Var yy) (Var xx)) n yy = PPf n (Var xx)$
 $m \in \text{num} \implies n \in \text{num} \implies \text{subst } (PPf m (Var xx)) n xx = PPf m n$
 $m \in \text{num} \implies \text{subst } (PPf (Var zz) (Var xx')) m zz = PPf m (Var xx')$
 $m \in \text{num} \implies n \in \text{num} \implies \text{subst } (PPf m (Var xx')) n xx' = PPf m n$
 $n \in \text{num} \implies \text{subst } (PPf (Var zz) (Var xx')) n xx' = PPf (Var zz) n$
 $m \in \text{num} \implies n \in \text{num} \implies \text{subst } (PPf (Var zz) n) m zz = PPf m n$
by (*auto* *simp*: *PPf_def2* *subst2_fresh_switch*)

lemma *PP_PPf*:

assumes $\varphi \in \text{fm}la$ $\text{Fvars } \varphi = \{\}$ **shows** $\text{bprv } (PP \langle \varphi \rangle) \longleftrightarrow \text{bprv } (\text{exi } yy (PPf (Var yy) \langle \varphi \rangle))$
using *assms* *PP_PPf_eqv[OF assms]* *B.prv_eqv_sym[OF _ _ PP_PPf_eqv[OF assms]]*
by (*auto* *intro!*: *B.prv_eqv_prv*[*of* *PP* $\langle \varphi \rangle$] *exi* *yy* (*PPf* (*Var* *yy*) $\langle \varphi \rangle$)]
B.prv_eqv_prv[*of* *exi* *yy* (*PPf* (*Var* *yy*) $\langle \varphi \rangle$) *PP* $\langle \varphi \rangle$])

lemma *isTrue_implies_bprv_PPf*:

$\bigwedge n \varphi. n \in \text{num} \implies \varphi \in \text{fm}la \implies \text{Fvars } \varphi = \{\} \implies$
 $\text{isTrue } (PPf n \langle \varphi \rangle) \implies \text{bprv } (PPf n \langle \varphi \rangle)$
using *Compl_Pf* **by**(*simp* *add*: *PPf_def*)

lemma *isTrue_iff_bprv_PPf*:
 $\bigwedge n \varphi. n \in \text{num} \implies \varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{isTrue } (\text{PPf } n \langle \varphi \rangle) \longleftrightarrow \text{bprv } (\text{PPf } n \langle \varphi \rangle)$
using *isTrue_implies_bprv_PPf*
using *exists_no_Fvars B.not_isTrue_fls B.sound_isTrue* **by** *auto*

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* $\varphi \equiv \text{bprv } (\text{PPf } n \langle \varphi \rangle)$
definition *encPf* :: 'trm \Rightarrow 'trm **where** [*simp*]: *encPf* $\equiv \text{id}$

lemma *prv_exi_PPf_iff_isTrue*:
assumes [*simp*]: $\varphi \in \text{fmla}$ $\text{Fvars } \varphi = \{\}$
shows $\text{bprv } (\text{exi } yy \ (\text{PPf } (\text{Var } yy) \langle \varphi \rangle)) \longleftrightarrow \text{isTrue } (\text{exi } yy \ (\text{PPf } (\text{Var } yy) \langle \varphi \rangle))$ (**is** ?L \longleftrightarrow ?R)
proof
assume ?L **thus** ?R **by** (*intro B.sound_isTrue*) *auto*
next
assume ?R
obtain *n* **where** *n*[*simp*]: $n \in \text{num}$ **and** *i*: $\text{isTrue } (\text{PPf } n \langle \varphi \rangle)$ **using** *B.isTrue_exi[OF _ _ _ <?R>]*
by *auto*
hence $\text{bprv } (\text{PPf } n \langle \varphi \rangle)$ **by** (*auto simp: isTrue_iff_bprv_PPf*)
thus ?L **by** (*intro B.prv_exi[of _ _ n]*) *auto*
qed

lemma *isTrue_exi_iff*:
assumes [*simp*]: $\varphi \in \text{fmla}$ $\text{Fvars } \varphi = \{\}$
shows $\text{isTrue } (\text{exi } yy \ (\text{PPf } (\text{Var } yy) \langle \varphi \rangle)) \longleftrightarrow (\exists n \in \text{num}. \text{isTrue } (\text{PPf } n \langle \varphi \rangle))$ (**is** ?L \longleftrightarrow ?R)
proof
assume ?L **thus** ?R **using** *B.isTrue_exi[OF _ _ _ <?L>]* **by** *auto*
next
assume ?R
then obtain *n* **where** *n*[*simp*]: $n \in \text{num}$ **and** *i*: $\text{isTrue } (\text{PPf } n \langle \varphi \rangle)$ **by** *auto*
hence $\text{bprv } (\text{PPf } n \langle \varphi \rangle)$ **by** (*auto simp: isTrue_iff_bprv_PPf*)
hence $\text{bprv } (\text{exi } yy \ (\text{PPf } (\text{Var } yy) \langle \varphi \rangle))$ **by** (*intro B.prv_exi[of _ _ n]*) *auto*
thus ?L **by** (*intro B.sound_isTrue*) *auto*
qed

lemma *prv_prfOf*:
assumes $\varphi \in \text{fmla}$ $\text{Fvars } \varphi = \{\}$
shows $\text{prv } \varphi \longleftrightarrow (\exists n \in \text{num}. \text{prfOf } n \varphi)$
proof –
have $\text{prv } \varphi \longleftrightarrow \text{bprv } (\text{PPf } n \langle \varphi \rangle)$ **using** *HBL1_iff[OF assms]* **by** *simp*
also have $\dots \longleftrightarrow \text{bprv } (\text{exi } yy \ (\text{PPf } (\text{Var } yy) \langle \varphi \rangle))$ **unfolding** *PP_PPf[OF assms]* ..
also have $\dots \longleftrightarrow \text{isTrue } (\text{exi } yy \ (\text{PPf } (\text{Var } yy) \langle \varphi \rangle))$ **using** *prv_exi_PPf_iff_isTrue[OF assms]* .
also have $\dots \longleftrightarrow (\exists n \in \text{num}. \text{isTrue } (\text{PPf } n \langle \varphi \rangle))$ **using** *isTrue_exi_iff[OF assms]* .
also have $\dots \longleftrightarrow (\exists n \in \text{num}. \text{bprv } (\text{PPf } n \langle \varphi \rangle))$ **using** *isTrue_iff_bprv_PPf[OF _ assms]* **by** *auto*
also have $\dots \longleftrightarrow (\exists n \in \text{num}. \text{prfOf } n \varphi)$ **unfolding** *prfOf_def* ..
finally show ?thesis .
qed

lemma *prfOf_prv_Pf*:
assumes $n \in \text{num}$ **and** $\varphi \in \text{fmla}$ $\text{Fvars } \varphi = \{\}$ **and** $\text{prfOf } n \varphi$
shows $\text{bprv } (\text{psubst } Pf \ [(n, yy), (\langle \varphi \rangle, xx)])$
using *assms unfolding prfOf_def* **by** (*auto simp: PPf_def2 psubst_eq_rawpsubst2*)

lemma *isTrue_exi_iff_PP*:
assumes [simp]: $\varphi \in \text{fmla } F\text{vars } \varphi = \{\}$
shows $\text{isTrue } (PP \langle \varphi \rangle) \longleftrightarrow (\exists n \in \text{num. } \text{isTrue } (PPf \ n \langle \varphi \rangle))$
proof –
have $\text{bprv } (\text{eqv } (PP \langle \varphi \rangle) (\text{exi } yy \ (PPf \ (\text{Var } yy) \langle \varphi \rangle)))$
using *PP_PPf_eqv* **by** *auto*
hence $\text{bprv } (\text{imp } (PP \langle \varphi \rangle) (\text{exi } yy \ (PPf \ (\text{Var } yy) \langle \varphi \rangle)))$
and $\text{bprv } (\text{imp } (\text{exi } yy \ (PPf \ (\text{Var } yy) \langle \varphi \rangle)) (PP \langle \varphi \rangle))$
by (*simp_all add: B.prv_imp_eqvEL B.prv_imp_eqvER*)
thus *?thesis* **using** *isTrue_exi_iff[OF assms, symmetric]*
by (*intro iffI*) (*rule B.prv_imp_implies_isTrue; simp*) +
qed

lemma *bprv_compl_isTrue_PP_enc*:
assumes *1*: $\varphi \in \text{fmla } F\text{vars } \varphi = \{\}$ **and** *2*: $\text{isTrue } (PP \langle \varphi \rangle)$
shows $\text{bprv } (PP \langle \varphi \rangle)$
proof –
obtain *n* **where** *nn*: $n \in \text{num}$ **and** *i*: $\text{isTrue } (PPf \ n \langle \varphi \rangle)$
using *2* **unfolding** *isTrue_exi_iff_PP[OF 1]* ..
hence $\text{bprv } (PPf \ n \langle \varphi \rangle)$
using *i* **using** *nn* *assms* *isTrue_iff_bprv_PPf* **by** *blast*
hence $\text{bprv } (\text{exi } yy \ (PPf \ (\text{Var } yy) \langle \varphi \rangle))$
using *2* *assms* *isTrue_exi_iff* *isTrue_exi_iff_PP* *prv_exi_PPf_iff_isTrue* **by** *auto*
thus *?thesis* **using** *PP_PPf 1* **by** *blast*
qed

lemma *TIP*:
assumes *1*: $\varphi \in \text{fmla } F\text{vars } \varphi = \{\}$ **and** *2*: $\text{isTrue } (PP \langle \varphi \rangle)$
shows $\text{prv } \varphi$
using *bprv_compl_isTrue_PP_enc[OF assms]* *HBL1_iff* *assms* **by** *blast*

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:
 $\bigwedge n \varphi. n \in \text{num} \implies \varphi \in \text{fmla} \implies F\text{vars } \varphi = \{\} \implies$
 $\text{let } PPf = (\lambda t1 \ t2. \text{psubst } Pf \ [(t1,yy), (t2,xx)]) \text{ in}$
 $\text{isTrue } (B.\text{neg } (PPf \ n \langle \varphi \rangle)) \longrightarrow \text{bprv } (B.\text{neg } (PPf \ n \langle \varphi \rangle))$
begin

lemma *isTrue_implies_prv_neg_PPf*:
 $\bigwedge n \varphi. n \in \text{num} \implies \varphi \in \text{fmla} \implies F\text{vars } \varphi = \{\} \implies$
 $\text{isTrue } (B.\text{neg } (PPf \ n \langle \varphi \rangle)) \implies \text{bprv } (B.\text{neg } (PPf \ n \langle \varphi \rangle))$
using *Compl_NegPf* **by** (*simp add: PPf_def*)

lemma *isTrue_iff_prv_neg_PPf*:
 $\bigwedge n \varphi. n \in \text{num} \implies \varphi \in \text{fmla} \implies F\text{vars } \varphi = \{\} \implies \text{isTrue } (B.\text{neg } (PPf \ n \langle \varphi \rangle)) \longleftrightarrow \text{bprv } (B.\text{neg } (PPf \ n \langle \varphi \rangle))$
using *isTrue_implies_prv_neg_PPf*
using *exists_no_Fvars B.not_isTrue_fls B.sound_isTrue* **by** *auto*

lemma *prv_PPf_decide*:
assumes [simp]: $n \in \text{num } \varphi \in \text{fmla } F\text{vars } \varphi = \{\}$

```

and np:  $\neg$  bprv (PPf n  $\langle\varphi\rangle$ )
shows bprv (B.neg (PPf n  $\langle\varphi\rangle$ ))
proof -
  have  $\neg$  isTrue (PPf n  $\langle\varphi\rangle$ ) using assms by (auto simp: isTrue_iff_bprv_PPf)
  hence isTrue (B.neg (PPf n  $\langle\varphi\rangle$ )) using B.isTrue_neg[of PPf n  $\langle\varphi\rangle$ ] by auto
  thus ?thesis by (auto simp: isTrue_iff_prv_neg_PPf)
qed

lemma not_prfOf_prv_neg_Pf:
assumes n $\varphi$ : 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)]))
  using assms prv_PPf_decide[OF n $\varphi$ ] by (auto simp: prfOf_def PPf_def2 psubst_eq_rawpsubst2)

end — context Minimal_Truth_Soundness_HBL1iff_Cmpl_Pf_Cmpl_NegPf

sublocale Minimal_Truth_Soundness_HBL1iff_Cmpl_Pf_Cmpl_NegPf <
  repr: CleanRepr_Proofs

  where proof = proof and prfOf = prfOf and encPf = encPf
  by standard (auto simp: bprv_prv_prv_prfOf prfOf_prv_Pf not_prfOf_prv_neg_Pf)

sublocale Minimal_Truth_Soundness_HBL1iff_Cmpl_Pf_Cmpl_NegPf <
  min_truth: Minimal_Truth_Soundness_Proof_Repr
where proof = proof and prfOf = prfOf and encPf = encPf
  by standard

locale Minimal_Truth_Soundness_HBL1iff_prv_Cmpl_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 conj 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
  using assms by (auto intro!: HBL1_rev_prv bprv_prv[OF _ _ bp])

```

```

lemma HBL1_iff: φ ∈ fmla ⇒ Fvars φ = {} ⇒ bprv (PP ⟨φ⟩) ↔ prv φ
  using HBL1 HBL1_rev unfolding PP_def by auto

```

```

lemma HBL1_iff_prv: φ ∈ fmla ⇒ Fvars φ = {} ⇒ prv (PP ⟨φ⟩) ↔ prv φ
  by (intro iffI bprv_prv[OF _ _ HBL1_PP], elim HBL1_rev_prv[rotated -1]) auto

```

```

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
  using P_Pf HBL1_rev HBL1_PP Compl_Pf
  by unfold_locales auto

```

```

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*: $\varphi \in \text{fm}la \implies \text{Fvars } \varphi = \{\} \implies \text{prv } (B.\text{neg } (B.\text{neg } (PP \langle \varphi \rangle))) \implies \text{prv } (PP \langle \varphi \rangle)$
using *classical_P unfolding PP_def by auto*

end — context *Minimal_Truth_Soundness_HBL1iff_prv_Cmpl_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  $\varphi G$  shows prv fls
proof -
  have prv (neg (PP ( $\varphi G$ ))) using prv_eqv_prv[OF _ _ assms prv  $\varphi G$  eqv] by auto
  moreover
  {have bprv (PP ( $\varphi G$ )) using HBL1[OF  $\varphi G$  Fvars  $\varphi G$  assms] unfolding PP_def .
   from bprv_prv[OF _ _ this, simplified] have prv (PP ( $\varphi G$ )) .
  }
  ultimately show ?thesis using PP prv_neg_fls by (meson  $\varphi G$  enc in_num)
qed
```

... then the more standard contrapositive formulation:

```
corollary goedel_first_theEasyHalf:
  consistent  $\implies \neg$  prv  $\varphi G$ 
using goedel_first_theEasyHalf_pos unfolding consistent_def by auto

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:  $\omega$ consistent
  shows  $\neg$  prv (neg  $\varphi G$ )
proof
  assume pn: prv (neg  $\varphi G$ )
  hence pnn: prv (neg (neg (wrepr.PP ( $\varphi G$ ))))
    using prv_eqv_imp_transi num wrepr.PP  $\varphi G$  fls neg neg_def prv  $\varphi G$  eqv prv_eqv_sym

```

```

  by (metis (full_types) enc in_num)
note c =  $\omega$ consistent_implies_consistent[OF oc]
have np:  $\neg$  prv  $\varphi G$  using pn c unfolding consistent_def3 by blast
have  $\forall p \in \text{num}. \text{bprv} (\text{neg} (\text{PPf } p \langle \varphi G \rangle))$  using not_prv_prv_neg_PPf[OF __ np] by auto
hence 0:  $\forall p \in \text{num}. \text{prv} (\text{neg} (\text{PPf } p \langle \varphi G \rangle))$  using not_prv_prv_neg_PPf[OF __ np]
  by (fastforce intro: bprv_prv)
have  $\neg$  prv (neg (neg (exi yy (PPf (Var yy)  $\langle \varphi G \rangle$ )))) using 0 oc unfolding  $\omega$ consistent_def by auto
hence  $\neg$  prv (neg (neg (wrepr.PP  $\langle \varphi G \rangle$ )))
  unfolding wrepr.PP_def by (subst P_def) (simp add: PPf_def2)
thus False using pnn by auto
qed

```

```

theorem goedel_first:
assumes  $\omega$ consistent
shows  $\neg$  prv  $\varphi G \wedge \neg$  prv (neg  $\varphi G$ )
  using assms goedel_first_theEasyHalf goedel_first_theHardHalf  $\omega$ consistent_implies_consistent by
blast

```

```

theorem goedel_first_ex:
assumes  $\omega$ consistent
shows  $\exists \varphi. \varphi \in \text{fmla} \wedge \neg$  prv  $\varphi \wedge \neg$  prv (neg  $\varphi$ )
  using assms goedel_first by (intro exI[of _  $\varphi G$ ]) blast

```

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 ( $\langle\langle\_ \rangle\rangle$ )
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 -
  have  $\forall n \in \text{num. } bprv (neg (PPf\ n\ \langle\varphi G\rangle))$ 
  using not_prv_prv_neg_PPf[OF _ _ goedel_first_theEasyHalf[OF assms]] by auto
  hence  $\forall n \in \text{num. } isTrue (neg (PPf\ n\ \langle\varphi G\rangle))$  by (auto intro: sound_isTrue)
  hence isTrue (all yy (neg (PPf (Var yy)  $\langle\varphi G\rangle$ ))) by (auto intro: isTrue_all)
  moreover have isTrue (imp (all yy (neg (PPf (Var yy)  $\langle\varphi G\rangle$ )))  $\varphi G$ )
  using bprv_eqv_all_not_PPf_imp_φG by (auto intro!: sound_isTrue)
  ultimately show ?thesis by (rule isTrue_imp[rotated -2]) auto
qed

```

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

```

theorem goedel_first_strong:
 $\omega\text{consistent} \implies \neg prv\ \varphi G \wedge \neg prv (neg\ \varphi G) \wedge isTrue\ \varphi G$ 
  using goedel_first isTrue_φG ωconsistent_implies_consistent by blast

```

```

theorem goedel_first_strong_ex:
 $\omega\text{consistent} \implies \exists \varphi. \varphi \in \text{fmla} \wedge \neg prv\ \varphi \wedge \neg prv (neg\ \varphi) \wedge isTrue\ \varphi$ 
  using goedel_first_strong by (intro exI[of _ φG]) blast

```

end — context *Goedel_Form_Proofs_Minimal_Truth*

8.2.2 Second model-theoretic version

```

locale Goedel_Form_Minimal_Truth_Soundness_HBL1iff_Cmpl_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_Cmpl_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
by standard

```

```

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:  $\bigwedge \varphi. \varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{let } PP = (\lambda t. \text{subst } P \ t \ xx) \text{ in}$ 
  prv (neg (neg (PP ⟨ϕ⟩)))  $\implies$  prv (PP ⟨ϕ⟩)

```

and

```

HBL1_rev_prv:  $\bigwedge \varphi. \varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{prv } (PP \langle \varphi \rangle) \implies \text{prv } \varphi$ 

```

begin

lemma HBL1_rev:

```

assumes f:  $\varphi \in \text{fmla}$  and fv:  $\text{Fvars } \varphi = \{\}$  and bp:  $\text{bprv } (PP \langle \varphi \rangle)$ 

```

```

shows prv  $\varphi$ 

```

```

using assms by (auto intro!: HBL1_rev_prv bprv_prv[OF __ bp])

```

```

lemma classical_PP_prv:  $\varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{prv } (\text{neg } (\text{neg } (PP \langle \varphi \rangle))) \implies \text{prv } (PP \langle \varphi \rangle)$ 

```

```

using classical_P_prv unfolding PP_def by auto

```

```

lemma HBL1_iff:  $\varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{bprv } (PP \langle \varphi \rangle) \longleftrightarrow \text{prv } \varphi$ 

```

```

using HBL1 HBL1_rev unfolding PP_def by auto

```

```

lemma HBL1_iff_prv:  $\varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{prv } (PP \langle \varphi \rangle) \longleftrightarrow \text{prv } \varphi$ 

```

```

by (meson HBL1_PP HBL1_rev_prv PP d_dwf.bprv_prv' enc in_num)

lemma goedel_first_theHardHalf_pos:
assumes prv (neg  $\varphi G$ ) shows prv fls
proof -
  have prv (neg (neg (PP  $\langle\varphi G\rangle$ )))
  using assms neg_def prv_ $\varphi G$ _equiv prv_equiv_imp_transi_rev by fastforce
  hence prv (PP  $\langle\varphi G\rangle$ ) using classical_PP_prv by auto
  hence prv  $\varphi G$  using Fvars_ $\varphi G$  HBL1_rev_prv by blast
  thus ?thesis using assms prv_neg_fls by blast
qed

corollary goedel_first_theHardHalf:
consistent  $\implies \neg prv$  (neg  $\varphi G$ )
using goedel_first_theHardHalf_pos unfolding consistent_def by auto

theorem goedel_first_classic:
assumes consistent
shows  $\neg prv$   $\varphi G \wedge \neg prv$  (neg  $\varphi G$ )
using assms goedel_first_theEasyHalf goedel_first_theHardHalf by blast

theorem goedel_first_classic_ex:
assumes consistent
shows  $\exists \varphi. \varphi \in fmla \wedge \neg prv$   $\varphi \wedge \neg prv$  (neg  $\varphi$ )
using assms goedel_first_classic by (intro exI[of _  $\varphi G$ ]) blast

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 dsj 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 ( $\langle\_\rangle$ )
and S
and P
and isTrue

```

```

+
assumes
— Truth of  $\varphi$  implies provability (TIP) of (the internal representation of)  $\varphi$ 
TIP:  $\bigwedge \varphi. \varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies$ 
  let  $PP = (\lambda t. \text{subst } P \ t \ xx)$  in
   $\text{isTrue } (PP \ \langle \varphi \rangle) \implies \text{prv } \varphi$ 
begin

lemma TIP_PP:  $\varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{isTrue } (PP \ \langle \varphi \rangle) \implies \text{prv } \varphi$ 
  using TIP_unfolding PP_def by auto

```

```

theorem isTrue_φG:
  assumes consistent
  shows isTrue φG
proof –
  {assume  $\neg \text{isTrue } \varphi G$ 
  hence 1:  $\text{isTrue } (\text{neg } \varphi G)$  using isTrue_neg by fastforce
  have bprv ( $\text{imp } (\text{neg } \varphi G) (\text{neg } (\text{neg } (PP \ \langle \varphi G \rangle)))$ )
  by ( $\text{auto simp add: bprv_φG_eqv B.prv_imp_eqvER B.prv_imp_neg_rev}$ )
  from prv_imp_implies_isTrue[OF _ _ _ _ this 1, simplified]
  have  $\text{isTrue } (\text{neg } (\text{neg } (PP \ \langle \varphi G \rangle)))$  .
  from isTrue_neg_neg[OF _ _ this, simplified] have  $\text{isTrue } (PP \ \langle \varphi G \rangle)$  .
  hence prv φG using assms TIP_PP by auto
  hence False using goedel_first_classic assms by auto
  }
  thus ?thesis by auto
qed

```

```

theorem goedel_first_classic_strong: consistent  $\implies \neg \text{prv } \varphi G \wedge \neg \text{prv } (\text{neg } \varphi G) \wedge \text{isTrue } \varphi G$ 
  using goedel_first_classic isTrue_φG by simp

```

```

theorem goedel_first_classic_strong_ex:
  consistent  $\implies \exists \varphi. \varphi \in \text{fmla} \wedge \neg \text{prv } \varphi \wedge \neg \text{prv } (\text{neg } \varphi) \wedge \text{isTrue } \varphi$ 
  using goedel_first_classic_strong by (intro exI[of _ φG]) blast

```

```

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 (standard, goal_cases classical rev_rpv TIPf)
  case (classical  $\varphi$ )
  then show ?case using HBL1_iff_classical_P by (simp add: mts_prv_mts.PP_deff)
next
  case (rev_rpv  $\varphi$ )
  then show ?case using HBL1_iff_prv_PP_def by simp
next
  case (TIPf  $\varphi$ )
  then show ?case using classical_P by (simp add: SS_def PP_def mts_prv_mts.TIP)
qed

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 ( $\langle\langle\_ \rangle\rangle$ )
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 ( $\langle\langle\_ \rangle\rangle$ )
  and N
  and S

```

and *encPf Pf*

context *Rosser_Form_Proofs*
begin

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

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

lemma $R[\text{simp,intro}]$: $R \in \text{fmla}$ **unfolding** *R_def* **by** *auto*

lemma *RR_def2*:
 $t1 \in \text{trm} \implies t2 \in \text{trm} \implies xx \notin \text{FvarsT } t1 \implies RR \ t1 \ t2 = \text{subst } (\text{subst } R \ t1 \ yy) \ t2 \ xx$
unfolding *RR_def* **by** (*rule psubst_eq_rawpsubst2[simplified]*) *auto*

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

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

lemma $\text{Fvars}_R[\text{simp}]$: $\text{Fvars } R = \{xx,yy\}$ **unfolding** *R_def* **by** *auto*

lemma $[\text{simp}]$: $\text{Fvars } (RR \ (\text{Var } yy) (\text{Var } xx)) = \{yy,xx\}$ **by** (*auto simp: RR_def2*)

lemma $P'[\text{simp,intro}]$: $P' \in \text{fmla}$ **unfolding** *P'_def* **by** (*auto simp: PPf_def2 RR_def2*)

lemma $\text{Fvars}_{P'}[\text{simp}]$: $\text{Fvars } P' = \{xx\}$ **unfolding** *P'_def* **by** (*auto simp: PPf_def2 RR_def2*)

lemma $PP'[\text{simp,intro}]$: $t \in \text{trm} \implies PP' \ t \in \text{fmla}$
unfolding *PP'_def* **by** *auto*

lemma $RR[\text{simp,intro}]$: $t1 \in \text{trm} \implies t2 \in \text{trm} \implies RR \ t1 \ t2 \in \text{fmla}$
by (*auto simp: RR_def*)

lemma *RR_simps[simp]*:
 $n \in \text{num} \implies \text{subst } (RR \ (\text{Var } yy) (\text{Var } xx)) \ n \ xx = RR \ (\text{Var } yy) \ n$
 $m \in \text{num} \implies n \in \text{num} \implies \text{subst } (RR \ (\text{Var } yy) \ m) \ n \ yy = RR \ n \ m$
by (*simp add: RR_def2 subst2_fresh_switch*)

The Rosser modification of the Gödel formula

definition $\varphi R :: \text{'fmla where } \varphi R \equiv \text{diag } (\text{neg } P')$

lemma $\varphi R[\text{simp,intro}]$: $\varphi R \in \text{fmla}$ **and** $\text{Fvars}_{\varphi R}[\text{simp}]$: $\text{Fvars } \varphi R = \{\}$
unfolding φR_def *wrepr.PP_def* **by** *auto*

lemma *bprv_φR_eqv*:
 $\text{bprv } (\text{eqv } \varphi R \ (\text{neg } (\text{PP}' \ \langle \varphi R \rangle)))$
unfolding φR_def *PP'_def* **using** *bprv_diag_eqv[of neg P']* **by** *simp*

lemma *bprv_imp_φR*:
 $\text{bprv } (\text{imp } (\text{neg } (\text{PP}' \ \langle \varphi R \rangle)) \ \varphi R)$
by (*rule B.prv_imp_eqvER*) (*auto intro: bprv_φR_eqv*)

lemma *prv_φR_eqv*:
 $\text{prv } (\text{eqv } \varphi R \ (\text{neg } (\text{PP}' \ \langle \varphi R \rangle)))$
using *dwf_dwfd.d dwf.bprv_prv'[OF _ bprv_φR_eqv, simplified]* .

```
lemma prv_imp_φR:  
  prv (imp (neg (PP' (φR)))) φR  
  by (rule prv_imp_eqvER) (auto intro: prv_φR_eqv)
```

```
end — context Rosser_Form
```

```
sublocale Rosser_Form_Proofs < Rosser_Form where  $P = P$   
  by standard
```

```
sublocale Rosser_Form_Proofs < Goedel_Form where  $P = P$   
  by standard
```

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]:  $\varphi \in \text{fmla } F\text{vars } \varphi = \{\}$ 
  shows
    bprv (imp (NN  $\langle \varphi \rangle$  (Var xx'))
      (eql  $\langle \text{neg } \varphi \rangle$  (Var xx')))
  using B.prv_subst[of yy _ Var xx', OF _ _ _ NN_neg_unique[OF assms]] by fastforce
```

```
lemma NN_imp_xx':
  assumes [simp]:  $\varphi \in \text{fmla } F\text{vars } \varphi = \{\}$   $\chi \in \text{fmla}$ 
  shows bprv (imp (subst  $\chi$   $\langle \text{neg } \varphi \rangle$  xx')
    (all xx' (imp (NN  $\langle \varphi \rangle$  (Var xx'))  $\chi$ )))
proof-
  have 2: bprv (imp (eql  $\langle \text{neg } \varphi \rangle$  (Var xx')) (imp (subst  $\chi$   $\langle \text{neg } \varphi \rangle$  xx')  $\chi$ ))
    using B.prv_eql_subst_trm[of xx'  $\chi$   $\langle \text{neg } \varphi \rangle$  Var xx', simplified] .
  have 1: bprv (imp (subst  $\chi$   $\langle \text{neg } \varphi \rangle$  xx') (imp (eql  $\langle \text{neg } \varphi \rangle$  (Var xx'))  $\chi$ ))
    by (simp add: 2 B.prv_imp_com)
  have 0: bprv (imp (subst  $\chi$   $\langle \text{neg } \varphi \rangle$  xx') (imp (NN  $\langle \varphi \rangle$  (Var xx'))  $\chi$ ))
    using 1
    by (elim B.prv_prv_imp_trans[rotated 3])
    (auto simp add: B.prv_imp_com B.prv_imp_monoR NN_neg_unique_xx')
  show ?thesis by (rule B.prv_all_imp_gen) (auto simp: 0)
qed
```

```
lemma goedel_rosser_first_theEasyHalf:
  assumes c: consistent
  shows  $\neg \text{prv } \varphi R$ 
proof
  assume 0:  $\text{prv } \varphi R$ 
```

then obtain prf **where** $[simp]: prf \in proof$ **and** $prfOf\ prf\ \varphi R$ **using** prv_prfOf **by** $auto$
hence $00: bprv\ (PPf\ (encPf\ prf)\ \langle\varphi R\rangle)$ **using** $prfOf_PPf$ **by** $auto$
from $dwf_dwfd.d_dwf.bprv_prv'[OF_00, simplified]$ **have** $b00: prv\ (PPf\ (encPf\ prf)\ \langle\varphi R\rangle)$.
have $\neg\ prv\ (neg\ \varphi R)$ **using** $0\ c$ **unfolding** $consistent_def3$ **by** $auto$
hence $\forall\ prf \in proof.$ $\neg\ prfOf\ prf\ (neg\ \varphi R)$ **using** $00\ prv_prfOf$ **by** $auto$
hence $bprv\ (neg\ (PPf\ p\ \langle neg\ \varphi R\rangle))$ **if** $p \in num$ **for** p **using** $not_prfOf_PPf\ Clean_PPf_encPf$ **that**
by $(cases\ p \in encPf\ 'proof)\ auto$
hence $1: prv\ (all\ zz\ (imp\ (LLq\ (Var\ zz)\ (encPf\ prf))\ (neg\ (PPf\ (Var\ zz)\ \langle neg\ \varphi R\rangle))))$

by $(intro\ LLq_num)\ auto$
have $11: prv\ (RR\ (encPf\ prf)\ \langle\varphi R\rangle)$
using $NN_imp_xx'[of\ \varphi R\ neg\ (PPf\ (Var\ zz)\ (Var\ xx')), simplified]$
by $(auto\ simp\ add: RR_def2\ R_def$
 $intro!: prv_all_congW[OF_ _ _ _ 1]\ prv_imp_monoL[OF_ _ _ _ dwf_dwfd.d_dwf.bprv_prv'])$
have $3: prv\ (cnj\ (PPf\ (encPf\ prf)\ \langle\varphi R\rangle)\ (RR\ (encPf\ prf)\ \langle\varphi R\rangle))$
by $(rule\ prv_cnjI[OF_ _ _ b00\ 11])\ auto$
have $prv\ ((PP'\ \langle\varphi R\rangle))$ **unfolding** $PP'_def\ P'_def$
using 3 **by** $(auto\ intro: prv_exiI[of\ _ _ encPf\ prf])$
moreover **have** $prv\ (neg\ (PP'\ \langle\varphi R\rangle))$
using $prv_eqv_prv[OF_ _ _ 0\ prv_var_eqv]$ **by** $auto$
ultimately show $False$ **using** c **unfolding** $consistent_def3$ **by** $auto$
qed

lemma $goedel_rosser_first_theHardHalf:$

assumes $c: consistent$

shows $\neg\ prv\ (neg\ \varphi R)$

proof

assume $0: prv\ (neg\ \varphi R)$

then obtain prf **where** $[simp,intro!]: prf \in proof$ **and** $pr: prfOf\ prf\ (neg\ \varphi R)$ **using** prv_prfOf **by** $auto$

define p **where** $p: p = encPf\ prf$

have $[simp,intro!]: p \in num$ **unfolding** p **by** $auto$

have $11: bprv\ (PPf\ p\ \langle neg\ \varphi R\rangle)$ **using** $pr\ prfOf_PPf$ **unfolding** p **by** $auto$

have $1: bprv\ (NN\ \langle\varphi R\rangle\ \langle neg\ \varphi R\rangle)$ **using** NN_neg **by** $simp$

have $\neg\ prv\ \varphi R$ **using** $0\ c$ **unfolding** $consistent_def3$ **by** $auto$

from $not_prv_prv_neg_PPf[OF_ _ _ this]$

have $b2: \forall\ r \in num. bprv\ (neg\ (PPf\ r\ \langle\varphi R\rangle))$ **by** $auto$

hence $2: \forall\ r \in num. prv\ (neg\ (PPf\ r\ \langle\varphi R\rangle))$

by $(auto\ intro: dwf_dwfd.d_dwf.bprv_prv')$

obtain P **where** $P[simp,intro!]: P \subseteq num$ **finite** P

and $3: prv\ (dsj\ (sdsj\ \{eql\ (Var\ yy)\ r\ |r. r \in P\})\ (LLq\ p\ (Var\ yy)))$

using LLq_num2 **by** $auto$

have $prv\ (imp\ (cnj\ (PPf\ (Var\ yy)\ \langle\varphi R\rangle)\ (RR\ (Var\ yy)\ \langle\varphi R\rangle))\ fls)$

proof $(rule\ prv_dsj_cases[OF_ _ _ 3])$

{fix r **assume** $r: r \in P$ **hence** $rn[simp]: r \in num$ **using** $P(1)$ **by** $blast$

have $prv\ (imp\ (cnj\ (PPf\ r\ \langle\varphi R\rangle)\ (RR\ r\ \langle\varphi R\rangle))\ fls)$

using 2 **unfolding** neg_def

by $(metis\ FvarsT_num\ PPf\ RR\ rn\ \varphi R\ all_not_in_conv\ cnj\ enc\ fls\ imp\ in_num\ prv_imp_cnj3L\ prv_imp_mp)$

hence $prv\ (imp\ (eql\ (Var\ yy)\ r)$

$(imp\ (cnj\ (PPf\ (Var\ yy)\ \langle\varphi R\rangle)\ (RR\ (Var\ yy)\ \langle\varphi R\rangle))\ fls))$

using $prv_eql_subst_trm_id[of\ yy\ cnj\ (PPf\ (Var\ yy)\ \langle\varphi R\rangle)\ (RR\ (Var\ yy)\ \langle\varphi R\rangle)\ r, simplified]$

unfolding $neg_def[symmetric]$

by $(intro\ prv_neg_imp_imp_trans)\ auto$

```

}
thus prv (imp (sdsj {eql (Var yy) r | r. r ∈ P})
  (imp (cnj (PPf (Var yy) ⟨φR⟩) (RR (Var yy) ⟨φR⟩)) fls))
  using Var P(1) eql by (intro prv_sdsj_imp) (auto 0 0 simp: set_rev_mp)
next
let ?φ = all xx' (imp (NN ⟨φR⟩ (Var xx^)) (neg (PPf p (Var xx^))))
have bprv (neg ?φ)
using 1 11 by (intro B.prv_imp_neg_allWI[where t = ⟨neg φR⟩]) (auto intro: B.prv_prv_neg_imp_neg)
hence prv (neg ?φ) by (auto intro: dwf_dwfd.d_dwf.bprv_prv')
hence 00: prv (imp (LLq p (Var yy))
  (imp (imp (LLq p (Var yy)) ?φ) fls))
  unfolding neg_def[symmetric] by (intro prv_imp_neg_imp_neg_imp) auto
have prv (imp (LLq p (Var yy))
  (imp (RR (Var yy) ⟨φR⟩) fls))
  unfolding neg_def[symmetric] using 00[folded neg_def]
  by (auto simp add: RR_def2 R_def intro!: prv_imp_neg_allI[where t = p])
thus prv (imp (LLq p (Var yy))
  (imp (cnj (PPf (Var yy) ⟨φR⟩) (RR (Var yy) ⟨φR⟩)) fls))
  unfolding neg_def[symmetric] by (intro prv_imp_neg_imp_cnjR) auto
qed(auto, insert Var P(1) eql, simp_all add: set_rev_mp)
hence prv (neg (exi yy (cnj (PPf (Var yy) ⟨φR⟩) (RR (Var yy) ⟨φR⟩))))
  unfolding neg_def[symmetric] by (intro prv_neg_neg_exi) auto
hence prv (neg (PP' ⟨φR⟩)) unfolding PP'_def P'_def by simp
hence prv φR using prv_φR_eqv by (meson PP' φR enc in_num neg prv_eqv_prv_rev)
with ⟨¬ prv φR⟩ show False using c unfolding consistent_def3 by auto
qed

```

theorem goedel_rosser_first:

```

assumes consistent
shows ¬ prv φR ∧ ¬ prv (neg φR)
using assms goedel_rosser_first_theEasyHalf goedel_rosser_first_theHardHalf by blast

```

theorem goedel_rosser_first_ex:

```

assumes consistent
shows ∃ φ. φ ∈ fmla ∧ ¬ prv φ ∧ ¬ prv (neg φ)
using assms goedel_rosser_first by (intro exI[of _ φR]) blast

```

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' ⟨φR⟩) = {} unfolding PP'_def
  by (subst Fvars_subst) auto

lemma Fvars_RR'[simp]: Fvars (RR (Var yy) ⟨φR⟩) = {yy}
  unfolding RR_def
  by (subst Fvars_psubst) (fastforce intro!: exI[of _ {yy}])+

lemma isTrue_PPf_implies_φR:
assumes isTrue (all yy (neg (PPf (Var yy) ⟨φR⟩)))
(is isTrue ?H)
shows isTrue φR
proof-
  define F where F ≡ RR (Var yy) ⟨φR⟩
  have [simp]: F ∈ fmla Fvars F = {yy}
    unfolding F_def by auto
  have [simp]: exi yy (PPf (Var yy) ⟨φR⟩) ∈ fmla
    unfolding PPf_def by auto

  have 1: bprv
    (imp (all yy (neg (PPf (Var yy) ⟨φR⟩)))
      (neg (exi yy (PPf (Var yy) ⟨φR⟩))))
  (is bprv (imp (all yy (neg ?G)) (neg (exi yy ?G))))
  using B.prv_all_neg_imp_neg_exi[of yy ?G] by auto
  have 2: bprv (imp (neg (exi yy ?G)) (neg (exi yy (cnj ?G F))))
    by (auto intro!: B.prv_imp_neg_rev B.prv_exi_cong B.prv_imp_cnjL)
  have bprv (imp (all yy (neg ?G)) (neg (exi yy (cnj ?G F))))
    using B.prv_prv_imp_trans[OF _ _ _ 1 2] by simp
  hence bprv (imp ?H (neg (PP' ⟨φR⟩)))
    unfolding PP'_def P'_def
    by (simp add: F_def)
  from B.prv_prv_imp_trans[OF _ _ _ this bprv_imp_φR]
  have bprv (imp ?H φR) by auto
  from prv_imp_implies_isTrue[OF _ _ _ this assms, simplified]
  show ?thesis .

```

qed

theorem *isTrue_φR*:

assumes *consistent*

shows *isTrue φR*

proof –

have $\forall n \in \text{num. } \text{bprv } (\text{neg } (\text{PPf } n \langle \varphi R \rangle))$

using *not_prv_prv_neg_PPf[OF _ _ goedel_rosser_first_theEasyHalf[OF assms]]*

by *auto*

have $\forall n \in \text{num. } \text{isTrue } (\text{neg } (\text{PPf } n \langle \varphi R \rangle))$ **by** (*auto intro: sound_isTrue*)

hence *isTrue (all yy (neg (PPf (Var yy) ⟨φR⟩)))* **by** (*auto intro: isTrue_all*)

thus *?thesis* **using** *isTrue_PPf_implies_φR* **by** *auto*

qed

theorem *goedel_rosser_first_strong*: *consistent* $\implies \neg \text{prv } \varphi R \wedge \neg \text{prv } (\text{neg } \varphi R) \wedge \text{isTrue } \varphi R$

using *isTrue_φR goedel_rosser_first* **by** *blast*

theorem *goedel_rosser_first_strong_ex*:

consistent $\implies \exists \varphi. \varphi \in \text{fmla} \wedge \neg \text{prv } \varphi \wedge \neg \text{prv } (\text{neg } \varphi) \wedge \text{isTrue } \varphi$

using *goedel_rosser_first_strong* **by** (*intro exI[of _ φR]*) *blast*

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** $bprv = bprv$
by *standard*

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-
  have 0: prv (imp φG (neg (PP ⟨φG⟩)))
  using prv_φG_eqv by (intro prv_imp_eqvEL) auto
  have 1: bprv (PP ⟨imp φG (neg (PP ⟨φG⟩))⟩)
  using HBL1_PP[OF _ 0] by simp
  have 2: bprv (imp (PP ⟨φG⟩) (PP ⟨neg (PP ⟨φG⟩))⟩)
  using HBL2_imp2[OF _ _ 1] by simp
```

```

have 3: bprv (imp (PP  $\langle \varphi G \rangle$ ) (PP  $\langle PP \langle \varphi G \rangle \rangle$ ))
  using HBL3[OF  $\varphi G$ ] by simp
have 23: bprv (imp (PP  $\langle \varphi G \rangle$ )
  (cnj (PP (PP  $\langle \varphi G \rangle$ ))
  (PP  $\langle neg \langle PP \langle \varphi G \rangle \rangle \rangle$ )))
using B.prv_imp_cnj[OF _ _ _ 3 2] by simp
have 4: bprv (imp (cnj (PP  $\langle PP \langle \varphi G \rangle \rangle$ )
  (PP  $\langle neg \langle PP \langle \varphi G \rangle \rangle \rangle$ ))
  (PP  $\langle fls \rangle$ ))
  using HBL2[of PP  $\langle \varphi G \rangle$  fls] unfolding neg_def[symmetric] by simp
show ?thesis using B.prv_prv_imp_trans[OF _ _ _ 23 4] by simp
qed

```

First the "direct", positive formulation:

```

lemma goedel_second_pos:
assumes prv (neg (PP  $\langle fls \rangle$ ))
shows prv fls
proof –
  note PG = bprv_prv[OF _ _ P_G, simplified]
  have prv (neg (PP  $\langle \varphi G \rangle$ ))
  using PG assms unfolding neg_def by (rule prv_prv_imp_trans[rotated 3]) auto
  hence prv  $\varphi G$  using prv_φG_eqv by (rule prv_eqv_prv_rev[rotated 2]) auto
  thus ?thesis
  — The only part of Goedel's first theorem that is needed:
  using goedel_first_theEasyHalf_pos by simp
qed

```

Then the more standard, counterpositive formulation:

```

theorem goedel_second:
consistent  $\implies \neg prv$  (neg (PP  $\langle fls \rangle$ ))
using goedel_second_pos unfolding consistent_def by auto

```

It is an immediate consequence of Gödel's Second HLB1, HLB2 that (assuming consistency) *prv* (*neg* (*PP* $\langle \varphi \rangle$)) 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  $\neg prv$  (neg (PP  $\langle \varphi \rangle$ ))
proof
  assume 0: prv (neg (PP  $\langle \varphi \rangle$ ))
  have prv (imp fls  $\varphi$ ) by simp
  hence bprv (PP  $\langle imp fls \varphi \rangle$ ) by (intro HBL1_PP) auto
  hence bprv (imp (PP  $\langle fls \rangle$ ) (PP  $\langle \varphi \rangle$ )) by (intro HBL2_imp2) auto
  hence bprv (imp (neg (PP  $\langle \varphi \rangle$ )) (neg (PP  $\langle fls \rangle$ ))) by (intro B.prv_imp_neg_rev) auto
  from prv_imp_mp[OF _ _ bprv_prv[OF _ _ this, simplified] 0, simplified]
  have prv (neg (PP  $\langle fls \rangle$ )) .
  thus False using goedel_second[OF c] by simp
qed

```

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  $\Rightarrow$  'trm ( $\langle \_ \rangle$ )  
assumes  
enc[simp,intro!]:  $\bigwedge \varphi. \varphi \in \text{fmla} \implies \text{enc } \varphi \in \text{num}$   
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 (unspecified) notion of unary "computable" function between numerals, which are encoded as
numerals. They contain a special substitution-like function sub  $\varphi$  for each formula  $\varphi$ .
fixes ufunc :: ('trm  $\Rightarrow$  'trm) set
  and encF :: ('trm  $\Rightarrow$  'trm)  $\Rightarrow$  'trm
  and sub :: 'fmla  $\Rightarrow$  'trm  $\Rightarrow$  '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:
ufunc[simp,intro!]:  $\bigwedge f n. f \in \text{ufunc} \implies n \in \text{num} \implies f n \in \text{num}$ 
and
encF[simp,intro!]:  $\bigwedge f. f \in \text{ufunc} \implies \text{encF } f \in \text{num}$ 
and
sub[simp!]:  $\bigwedge \varphi. \varphi \in \text{fmla} \implies \text{sub } \varphi \in \text{ufunc}$ 
and
— The function sub  $\varphi$  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:
 $\bigwedge \varphi f. \varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{xx\} \implies f \in \text{ufunc} \implies$ 
   $\text{sub } \varphi (\text{encF } f) = \text{enc } (\text{inst } \varphi (f (\text{encF } f)))$ 

```

12.1.3 Term-encoding of computable functions

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  $\Rightarrow$  'trm) set
and
enc :: ('trm  $\Rightarrow$  'trm)  $\Rightarrow$  'trm (⟨⟨_⟩⟩)
assumes
Ops[simp,intro!]:  $\bigwedge f t. f \in \text{Ops} \implies t \in \text{trm} \implies f t \in \text{trm}$ 
and
enc[simp,intro!]:  $\bigwedge f. f \in \text{Ops} \implies \text{enc } f \in \text{num}$ 
and
Ops_FvarsT[simp!]:  $\bigwedge f t. f \in \text{Ops} \implies t \in \text{trm} \implies \text{FvarsT } (f t) = \text{FvarsT } t$ 
and
Ops_substT[simp!]:  $\bigwedge f t. f \in \text{Ops} \implies t \in \text{trm} \implies t1 \in \text{trm} \implies x \in \text{var} \implies$ 
   $\text{substT } (f t) t1 x = f (\text{substT } t t1 x)$ 
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 ∈ fmla
and
  Fvars_P[simp]: Fvars P = {xx}
and
  HBL1:  $\bigwedge \varphi. \varphi \in fmla \implies Fvars \varphi = \{\} \implies prv \varphi \implies prv (subst P \langle \varphi \rangle xx)$ 
begin

```

Predicate version of the provability formula

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

lemma PP[*simp*]: $\bigwedge t. t \in trm \implies PP t \in fmla$
unfolding PP_def **by** auto

lemma Fvars_PP[*simp*]: $\bigwedge t. t \in trm \implies Fvars (PP t) = FvarsT t$
unfolding PP_def **by** auto

lemma [*simp*]:
 $n \in num \implies subst (PP (Var yy)) n yy = PP n$
 $n \in num \implies subst (PP (Var xx)) n xx = PP n$
unfolding PP_def **by** auto

lemma HBL1_PP:
 $\varphi \in fmla \implies Fvars \varphi = \{\} \implies prv \varphi \implies prv (PP \langle \varphi \rangle)$
using HBL1 **unfolding** PP_def **by** auto

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* \equiv *subst P (Var inp) xx*
lemma *PP_Pinp*: $t \in \text{trm} \implies PP\ t = \text{instInp Pinp } t$
unfolding *PP_def Pinp_def instInp_def* **by** *auto*

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

lemma *HBL1_inp*:
 $\varphi \in \text{fmLa} \implies \text{Fvars } \varphi = \{\} \implies \text{prv } \varphi \implies \text{prv } (\text{instInp Pinp } \langle \varphi \rangle)$
unfolding *Pinp_def instInp_def* **by** (*auto intro: HBL1*)

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* ($\langle _ \rangle$)
+
fixes *F* :: ('*trm* \Rightarrow '*trm*) *set*
and *encF* :: ('*trm* \Rightarrow '*trm*) \Rightarrow '*fmLa*
and *N* :: '*trm* \Rightarrow '*trm*
and *ssap* :: '*fmLa* \Rightarrow '*trm* \Rightarrow '*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!]: $\bigwedge f\ n. f \in F \implies n \in \text{num} \implies f\ n \in \text{num}$
and
encF[simp,intro!]: $\bigwedge f. f \in F \implies \text{encF } f \in \text{ptrm } (\text{Suc } 0)$
and
N[simp,intro!]: $N \in F$
and
ssap[simp]: $\bigwedge \varphi. \varphi \in \text{fmLa} \implies \text{Fvars } \varphi = \{\text{inp}\} \implies \text{ssap } \varphi \in F$
and
ReprF: $\bigwedge f\ n. f \in F \implies n \in \text{num} \implies \text{prveqlPT } (\text{instInp } (\text{encF } f)\ n)\ (f\ n)$
and
CapN: $\bigwedge \varphi. \varphi \in \text{fmLa} \implies \text{Fvars } \varphi = \{\} \implies N\ \langle \varphi \rangle = \langle \text{neg } \varphi \rangle$

and

CapSS: — We consider formulas ψ of one variable, called *inp*:

$\bigwedge \psi f. \psi \in \text{fmla} \implies \text{Fvars } \psi = \{\text{inp}\} \implies f \in F \implies$

$\text{ssap } \psi \langle \text{encF } f \rangle = \langle \text{instInpP } \psi \ 0 \ (\text{instInp } (\text{encF } f) \langle \text{encF } f \rangle) \rangle$

begin

lemma *encF_fm1a*[*simp,intro!*]: $\bigwedge f. f \in F \implies \text{encF } f \in \text{fmla}$

by *auto*

lemma *enc_trm*: $\varphi \in \text{fmla} \implies \langle \varphi \rangle \in \text{trm}$

by *auto*

definition $\tau J :: 'fmla \Rightarrow 'fmla$ **where**

$\tau J \psi \equiv \text{instInp } (\text{encF } (\text{ssap } \psi)) \ (\langle \text{encF } (\text{ssap } \psi) \rangle)$

definition $\varphi J :: 'fmla \Rightarrow 'fmla$ **where**

$\varphi J \psi \equiv \text{instInpP } \psi \ 0 \ (\tau J \psi)$

lemma τJ [*simp*]:

assumes $\psi \in \text{fmla}$ **and** $\text{Fvars } \psi = \{\text{inp}\}$

shows $\tau J \psi \in \text{ptrm } 0$

unfolding τJ_def **apply**(*rule instInp*)

using *assms* **by** *auto*

lemma τJ_fm1a [*simp*]:

assumes $\psi \in \text{fmla}$ **and** $\text{Fvars } \psi = \{\text{inp}\}$

shows $\tau J \psi \in \text{fmla}$

using τJ [*OF assms*] **unfolding** *ptrm_def* **by** *auto*

lemma $\text{FvarsT_}\tau J$ [*simp*]:

assumes $\psi \in \text{fmla}$ **and** $\text{Fvars } \psi = \{\text{inp}\}$

shows $\text{Fvars } (\tau J \psi) = \{\text{out}\}$

using τJ [*OF assms*] **unfolding** *ptrm_def* **by** *auto*

lemma φJ [*simp*]:

assumes $\psi \in \text{fmla}$ **and** $\text{Fvars } \psi = \{\text{inp}\}$

shows $\varphi J \psi \in \text{fmla}$

unfolding φJ_def **using** *assms* **by** (*intro instInpP_fm1a*) *auto*

lemma $\text{Fvars_}\varphi J$ [*simp*]:

assumes $\psi \in \text{fmla}$ **and** $\text{Fvars } \psi = \{\text{inp}\}$

shows $\text{Fvars } (\varphi J \psi) = \{\}$

using *assms* **unfolding** φJ_def **by** *auto*

lemma *diagonalization*:

assumes ψ [*simp*]: $\psi \in \text{fmla}$ **and** [*simp*]: $\text{Fvars } \psi = \{\text{inp}\}$

shows *prveqlPT* ($\tau J \psi$) $\langle \text{instInpP } \psi \ 0 \ (\tau J \psi) \rangle \wedge$

$\text{prv } (\text{eqv } (\varphi J \psi) \ (\text{instInp } \psi \ (\varphi J \psi)))$

proof

define *f* **where** $f \equiv \text{ssap } \psi$

have f [*simp*]: $f \in F$ **unfolding** f_def **using** *assms* **by** *auto*

have *ff*: $f \langle \text{encF } f \rangle = \langle \text{instInpP } \psi \ 0 \ (\tau J \psi) \rangle$

using *assms* **unfolding** f_def τJ_def **by** (*intro CapSS*) *auto*

show *prveqlPT* ($\tau J \psi$) $\langle \text{instInpP } \psi \ 0 \ (\tau J \psi) \rangle$

using *ReprF*[*OF f*, *of* $\langle \text{encF } f \rangle$]

unfolding τJ_def [*of* ψ , *unfolded f_def*[*symmetric*],*symmetric*] *ff*[*symmetric*]

by *auto*

```

from prveqlPT_prv_instInp_eqv_instInpP[OF  $\psi$ , of  $\tau J \psi$ , OF _ _ _ _ this,
      unfolded  $\varphi J\_def[symmetric]$ ]
show prv (eqv ( $\varphi J \psi$ ) (instInp  $\psi \langle \varphi J \psi \rangle$ ))
by auto
qed

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 ( $\langle \_ \rangle$ )
and P
and F encF N ssap
+
assumes
  SHBL3:  $\bigwedge \tau. \tau \in ptrm\ 0 \implies prv\ (imp\ (instInpP\ Pinp\ 0\ \tau)\ (instInp\ Pinp\ \langle instInpP\ Pinp\ 0\ \tau \rangle))$ 
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:
   $x \in var \implies \varphi \in fmla \implies t1 \in trm \implies t2 \in trm \implies$ 
   $prv\ (eql\ t1\ t2) \implies prv\ (subst\ \varphi\ t1\ x) \implies prv\ (subst\ \varphi\ t2\ x)$ 
using prv_eql_subst_trm2

```

by (meson subst prv_imp_mp)

lemma *Pinp[simp,intro!]*: $Pinp \in fmla$
and *Fvars_Pinp[simp]*: $Fvars\ Pinp = \{inp\}$
unfolding *Pinp_def* **by** *auto*

lemma *ReprF_combineWith_CapN*:
assumes $\varphi \in fmla$ **and** $Fvars\ \varphi = \{\}$
shows *prveqlPT* (instInp (encF N) $\langle\varphi\rangle$) $\langle neg\ \varphi\rangle$
using *assms* **unfolding** *CapN[symmetric, OF assms]* **by** (intro ReprF) *auto*

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

assume *: *prv jcons*

define ψ **where** $\psi \equiv instInpP\ Pinp\ (Suc\ 0)\ (encF\ N)$

define τ **where** $\tau \equiv \tau J\ \psi$

define φ **where** $\varphi \equiv \varphi J\ \psi$

have $\psi[simp,intro]$: $\psi \in fmla\ Fvars\ \psi = \{inp\}$

unfolding ψ_def **by** *auto*

have $\tau[simp,intro]$: $\tau \in ptrm\ 0\ \tau \in fmla\ Fvars\ \tau = \{out\}$

unfolding τ_def **by** *auto*

have $[simp]$: $\varphi \in fmla\ Fvars\ \varphi = \{\}$ **unfolding** φ_def **by** *auto*

define $eN\tau$ **where** $eN\tau \equiv instInpP\ (encF\ N)\ 0\ \tau$

have $eN\tau[simp]$: $eN\tau \in ptrm\ 0\ eN\tau \in fmla\ Fvars\ eN\tau = \{out\}$

unfolding $eN\tau_def$ **by** *auto*

define φ' **where** $\varphi' \equiv instInpP\ Pinp\ 0\ eN\tau$

have $[simp]$: $\varphi' \in fmla\ Fvars\ \varphi' = \{\}$ **unfolding** φ'_def **by** *auto*

have $\varphi\varphi'$: *prv* (imp $\varphi\ \varphi'$) **and** $\varphi'\varphi$: *prv* (imp $\varphi'\ \varphi$) **and** $\varphi e\varphi'$: *prv* (equiv $\varphi\ \varphi'$)

unfolding $\varphi_def\ \varphi J_def\ \varphi'_def\ eN\tau_def\ \tau_def[symmetric]$ **unfolding** ψ_def

using *prv_instInpP_compose[of Pinp encF N τ]* **by** *auto*

from *diagonalization[OF ψ]*

have *prveqlPT* $\tau\ \langle instInpP\ \psi\ 0\ \tau\rangle$ **and** **: *prv* (equiv $\varphi\ (instInp\ \psi\ \langle\varphi\rangle)$)

unfolding $\tau_def[symmetric]\ \varphi_def[symmetric]$ **by** *auto*

have **:1: *prv* (imp $\varphi\ (instInp\ \psi\ \langle\varphi\rangle)$) *prv* (imp (instInp $\psi\ \langle\varphi\rangle$) φ)

using *prv_imp_eqvEL[OF _ _ **]* *prv_imp_eqvER[OF _ _ **]* **by** *auto*

from *SHBL3[OF $eN\tau(1)$]*

have *prv* (imp (instInpP Pinp 0 $eN\tau$) (instInp Pinp $\langle instInpP\ Pinp\ 0\ eN\tau\rangle$)) .

hence *prv* (imp $\varphi'\ (instInp\ Pinp\ \langle\varphi'\rangle)$) **unfolding** φ'_def .

from *prv_prv_imp_trans[OF _ _ _ $\varphi\varphi'$ this]*

have 0: *prv* (imp $\varphi\ (instInp\ Pinp\ \langle\varphi'\rangle)$) **by** *auto*

note *unr = unstated[unfolded Let_def*

$\varphi_def[unfolded\ \varphi J_def\ \tau_def[symmetric],\ symmetric]\ \psi_def[symmetric]$

$\tau_def[symmetric]\ eN\tau_def[symmetric]\ \varphi'_def[symmetric]\ \varphi J_def]$

```

have 1: prv (imp  $\varphi$  (instInp Pinp  $\langle\varphi\rangle$ ))
apply(nrule r: nprv_prvI)
apply(nrule r: nprv_impI)
apply(nrule r: nprv_addLemmaE[OF unr])
apply(nrule r: nprv_addImpLemmaE[OF 0])
apply(nrule r: nprv_clear3_3)
by (simp add: nprv_clear2_2 prv_nprv1I unr)

have 2: prv (imp  $\varphi$  (cnj (instInp Pinp  $\langle\varphi\rangle$ )
      (instInp  $\psi$   $\langle\varphi\rangle$ )))
apply(nrule r: nprv_prvI)
apply(nrule r: nprv_impI)
apply(nrule r: nprv_cnjI)
subgoal apply(nrule r: nprv_addImpLemmaE[OF 1]) .
subgoal apply(nrule r: nprv_addImpLemmaE[OF **1(1)]) . .

define z where z  $\equiv$  Variable (Suc (Suc 0))
have z_facts[simp]: z  $\in$  var z  $\neq$  xx z  $\notin$  Fvars Pinp
  out  $\neq$  z  $\wedge$  z  $\neq$  out inp  $\neq$  z  $\wedge$  z  $\neq$  inp
  unfolding z_def by auto

have 30: subst (instInpP Pinp 0 (instInp (encF N) (Var xx)))  $\langle\varphi\rangle$  xx =
  instInpP Pinp 0 (instInp (encF N)  $\langle\varphi\rangle$ )
unfolding z_def[symmetric] instInp_def instInpP_def Let_def
by (variousSubsts4 auto
  s1: subst_compose_diff s2: subst_subst
  s3: subst_notIn[of  $\_ \_$  xx] s4: subst_compose_diff)
have 31: subst (instInp Pinp (Var xx))  $\langle\varphi\rangle$  xx =
  instInp Pinp  $\langle\varphi\rangle$  unfolding instInp_def by auto
have [simp]: instInp (instInpP Pinp (Suc 0) (encF N))  $\langle\varphi\rangle$  =
  instInpP Pinp 0 (instInp (encF N)  $\langle\varphi\rangle$ )
by (auto simp: instInp_instInpP  $\psi\_def$ )

have 3: prv (neg (cnj (instInp Pinp  $\langle\varphi\rangle$ )
      (instInp  $\psi$   $\langle\varphi\rangle$ )))
apply(nrule r: nprv_prvI)
apply(nrule r: nprv_addLemmaE[OF *, unfolded jcons_def])
apply(rule nprv_allE0[of  $\_ \_ \_$   $\langle\varphi\rangle$ ], auto)
unfolding 30 31
apply(nrule r: nprv_clear2_2)
apply(nrule r: nprv_negI)
apply(nrule r: nprv_negE0)
apply(nrule r: nprv_clear2_2)
apply(nrule r: nprv_cnjE0)
apply(nrule r: nprv_clear3_3)
apply(nrule r: nprv_cnjI)
apply(nrule r: nprv_clear2_1)
unfolding  $\psi\_def$ 
apply(nrule r: nprv_hyp) .

have ***: prv (neg  $\varphi$ )
apply(nrule r: nprv_prvI)
apply(nrule r: nprv_negI)
apply(nrule r: nprv_addImpLemmaE[OF 2])
apply(nrule r: nprv_addLemmaE[OF 3])
apply(nrule r: nprv_negE0) .

```

```

have 4: prv (instInp Pinp ⟨neg  $\varphi$ ⟩) using HBL1_inp[OF _ _ ***] by auto

have 5: prveqLPT (instInp (encF N) ⟨ $\varphi$ ⟩) ⟨neg  $\varphi$ ⟩
  using ReprF_combineWith_CapN[of  $\varphi$ ] by auto

have [simp]: instInp (encF N) ⟨ $\varphi$ ⟩ ∈ ptrm 0 using instInp by auto

have 6: prv (instInpP Pinp 0 (instInp (encF N) ⟨ $\varphi$ ⟩))
apply(nrule r: nprv_prvI)
apply(nrule r: nprv_addLemmaE[OF 4])
apply(nrule r: prveqLPT_nprv_instInpP_instInp[OF _ _ _ _ 5]) .

note lem = **1(2)[unfolded  $\psi\_def$ ]
have prv  $\varphi$ 
apply(nrule r: nprv_prvI)
apply(nrule r: nprv_addLemmaE[OF 6])
apply(nrule r: nprv_addImpLemmaE[OF lem]) .

from this *** ⟨consistent⟩ show False unfolding consistent_def3 by auto
qed

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 $enc \langle \langle _ \rangle \rangle$
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
 $CapSS:$
 $\bigwedge \psi f. \psi \in fmla \Longrightarrow Fvars \psi = \{xx\} \Longrightarrow f \in F \Longrightarrow$
 $ssap \psi (tenc (encF f)) = \langle inst \psi (encF f (tenc (encF f))) \rangle$
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 $tJ \psi \in trm$
using $assms tJ_def$ **by** $auto$

lemma $FvarsT_tJ[simp]:$
assumes $\psi \in fmla$ **and** $Fvars \psi = \{xx\}$
shows $FvarsT (tJ \psi) = \{\}$
using $assms tJ_def$ **by** $auto$

lemma $\varphi J[simp]:$
assumes $\psi \in fmla$ **and** $Fvars \psi = \{xx\}$
shows $\varphi J \psi \in fmla$
using $assms \varphi J_def$ **by** $auto$

lemma $Fvars_ \varphi J[simp]:$
assumes $\psi \in fmla$ **and** $Fvars \psi = \{xx\}$
shows $Fvars (\varphi J \psi) = \{\}$
using $assms \varphi J_def$ **by** $auto$

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

proof
define fJ **where** $fJ \equiv ssap \psi$
have $fJ[simp]: fJ \in F$ **unfolding** fJ_def **using** $assms$ **by** $auto$

```

have fJ (tenc (encF fJ)) = ⟨inst ψ (tJ ψ)⟩
by (simp add: CapSS assms fJ_def tJ_def)
thus **: prv (eql (tJ ψ) ⟨inst ψ (tJ ψ)⟩)
using ReprF fJ fJ_def tJ_def by fastforce
show prv (equ (φJ ψ) (inst ψ ⟨φJ ψ⟩))
using assms prv_eql_subst_trm_eqv[OF xx _ _ _ **, of ψ]
by (auto simp: φJ_def inst_def)
qed

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: ∧t. t ∈ trm ⇒ FvarsT t = {} ⇒ prv (imp (PP t) (PP ⟨PP t⟩))
begin

```

Consistency formula a la Jeroslow:

```

definition jcons :: 'fmla where
  jcons ≡ all xx (neg (cnj (PP (Var xx)) (PP (encF N (Var (xx))))))

```

```

lemma prv_eql_subst_trm3:
  x ∈ var ⇒ φ ∈ fmla ⇒ t1 ∈ trm ⇒ t2 ∈ trm ⇒
  prv (eql t1 t2) ⇒ prv (subst φ t1 x) ⇒ prv (subst φ t2 x)
using prv_eql_subst_trm2
by (meson subst prv_imp_mp)

```

```

lemma prv_eql_neg_encF_N:
assumes  $\varphi \in \text{fm}la$  and  $Fvars \varphi = \{\}$ 
shows  $prv (eql \langle neg \varphi \rangle (encF N \langle \varphi \rangle))$ 
  unfolding CapN[symmetric, OF assms]
  by (rule prv_prv_eql_sym) (auto simp: assms intro: ReprF)

lemma prv_imp_neg_encF_N_aux:
assumes  $\varphi \in \text{fm}la$  and  $Fvars \varphi = \{\}$ 
shows  $prv (imp (PP \langle neg \varphi \rangle) (PP (encF N \langle \varphi \rangle)))$ 
using assms prv_eql_subst_trm2[OF _ _ _ _ prv_eql_neg_encF_N[OF assms],
  of xx PP (Var xx)]
  unfolding PP_def by auto

lemma prv_cnj_neg_encF_N_aux:
assumes  $\varphi \in \text{fm}la$  and  $Fvars \varphi = \{\}$   $\chi \in \text{fm}la$   $Fvars \chi = \{\}$ 
and  $prv (neg (cnj \chi (PP \langle neg \varphi \rangle)))$ 
shows  $prv (neg (cnj \chi (PP (encF N \langle \varphi \rangle))))$ 
using assms prv_eql_subst_trm3[OF _ _ _ _ prv_eql_neg_encF_N,
  of xx neg (cnj \chi (PP (Var xx)))]
  unfolding PP_def by auto

theorem jeroslow_godel_second:
assumes consistent
shows  $\neg prv jcons$ 
proof
  assume *: prv jcons
  define  $\psi$  where  $\psi \equiv PP (encF N (Var xx))$ 
  define  $t$  where  $t \equiv tJ \psi$ 
  have  $\psi[simp,intro]: \psi \in \text{fm}la$   $Fvars \psi = \{xx\}$ 
  and  $t[simp,intro]: t \in \text{trm}$   $Fvars T t = \{\}$ 
  unfolding  $\psi\_def t\_def$  by auto

  have  $sPP[simp]: subst (PP (encF N (Var xx))) \langle PP (encF N t) \rangle xx =$ 
     $PP (encF N \langle PP (encF N t) \rangle)$ 
  unfolding PP_def by (subst subst_compose_eq_or) auto
  have  $sPP2[simp]: subst (PP (encF N (Var xx))) t xx = PP (encF N t)$ 
  unfolding PP_def by (subst subst_compose_eq_or) auto
  have  $00: PP (encF N t) = inst \psi t$  unfolding  $\psi\_def inst\_def PP\_def$ 
  by (subst subst_compose_eq_or) auto

  define  $\varphi$  where  $\varphi \equiv \varphi J \psi$ 
  have  $[\text{simp}]: \varphi \in \text{fm}la$   $Fvars \varphi = \{\}$  unfolding  $\varphi\_def$  by auto
  have *:  $prv (eql t \langle \varphi \rangle)$ 
  unfolding  $00 \varphi\_def$ 
  using  $\varphi J\_def diagonalization inst\_def t\_def$  by auto
  have  $\varphi: \varphi = PP (encF N t)$  unfolding  $\varphi\_def \varphi J\_def t\_def \psi\_def$ 
  using  $sPP2 \psi\_def t\_def$  by blast
  have  $1: prv (imp \varphi (PP \langle \varphi \rangle))$  using SHBL3[of encF N t]
  using  $00 \varphi J\_def \varphi\_def \psi\_def inst\_def t\_def$  by auto
  have  $eqv\_varphi: prv (eqv \varphi (PP (encF N \langle \varphi \rangle)))$  using diagonalization
  by (metis 00 sPP \varphi J\_def \varphi\_def \psi \psi\_def diagonalization inst\_def t\_def)
  have  $2: prv (imp \varphi (PP (encF N \langle \varphi \rangle)))$ 
  using  $prv\_cnjEL[OF _ _ eqv\_varphi[unfolded eqv\_def]]$  by auto
  have  $prv (imp (PP (encF N \langle \varphi \rangle)) \varphi)$ 
  using  $prv\_cnjER[OF _ _ eqv\_varphi[unfolded eqv\_def]]$  by auto
from  $prv\_prv\_imp\_trans[OF _ _ _ prv\_imp\_neg\_encF\_N\_aux this]$ 
  have  $22: prv (imp (PP \langle neg \varphi \rangle) \varphi)$  by auto
  have  $3: prv (imp \varphi (cnj (PP \langle \varphi \rangle) (PP (encF N \langle \varphi \rangle))))$ 

```

```

  by (rule prv_imp_cnj[OF _ _ _ 1 2]) (auto simp:  $\varphi\_def$ )
have 4: prv (neg (cnj (PP  $\langle\varphi\rangle$ ) (PP (encF N  $\langle\varphi\rangle$ ))))
  using prv_allE[OF _ _ _ *[unfolded jcons_def], of  $\langle\varphi\rangle$ ]
by (simp add:  $\varphi\_psi\_def$ )
have 5: prv (neg  $\varphi$ )
  unfolding neg_def
  by (rule prv_prv_imp_trans[OF _ _ _ 3 4[unfolded neg_def]]) auto
hence prv (PP  $\langle neg \varphi \rangle$ ) using
  HBL1_PP[OF _ _ 5] by auto
hence prv  $\varphi$  using prv_imp_mp[OF _ _ 22] by auto
with 5 assms show False unfolding consistent_def3 by auto
qed

```

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*: $\bigwedge \varphi1 \varphi2. \{\varphi1, \varphi2\} \subseteq fmla \implies Fvars \varphi1 = \{\} \implies Fvars \varphi2 = \{\} \implies$
 $prv (imp (cnj (PP \langle\varphi1\rangle) (PP \langle\varphi2\rangle))) (PP \langle cnj \varphi1 \varphi2 \rangle))$

and *WHBL2*: $\bigwedge \varphi1 \varphi2. \{\varphi1, \varphi2\} \subseteq fmla \implies Fvars \varphi1 = \{\} \implies Fvars \varphi2 = \{\} \implies$
 $prv (imp \varphi1 \varphi2) \implies prv (imp (PP \langle\varphi1\rangle) (PP \langle\varphi2\rangle))$

shows $\neg prv (neg (PP \langle fls \rangle))$

proof

assume *: $prv (neg (PP \langle fls \rangle))$

define ψ **where** $\psi \equiv PP (encF N (Var xx))$

define t **where** $t \equiv tJ \psi$

have $\psi[simp, intro]: \psi \in fmla \ Fvars \psi = \{xx\}$

and $t[simp, intro]: t \in trm \ FvarsT t = \{\}$

unfolding $\psi_def \ t_def$ **by** *auto*

have $[simp]: subst (PP (encF N (Var xx))) \langle PP (encF N t) \rangle xx =$
 $PP (encF N \langle PP (encF N t) \rangle)$

unfolding PP_def **by** (*subst subst_compose_eq_or*) *auto*

have $[simp]: subst (PP (encF N (Var xx))) t xx = PP (encF N t)$

unfolding PP_def **by** (*subst subst_compose_eq_or*) *auto*

have $00: PP (encF N t) = inst \psi t$ **unfolding** $\psi_def \ inst_def \ PP_def$
by (*subst subst_compose_eq_or*) *auto*

define φ **where** $\varphi = PP (encF N t)$

have $[simp]: \varphi \in fmla \ Fvars \varphi = \{\}$ **unfolding** φ_def **by** *auto*

have **: $prv (eql t \langle PP (encF N t) \rangle)$

unfolding 00 **by** (*simp add: diagonalization t_def*)

have 1: $prv (imp \varphi (PP \langle\varphi\rangle))$ **using** *SHBL3[of encF N t]*

by (*auto simp: φ_def*)

have 2: $prv (imp \varphi (PP (encF N \langle\varphi\rangle)))$

using $prv_eql_subst_trm2[OF xx _ _ _ **, of PP (encF N (Var xx))]$

by (*auto simp: φ_def*)

have $prv (imp (PP (encF N \langle\varphi\rangle)) \varphi)$

using $prv_eql_subst_trm_rev2[OF xx _ _ _ **, of PP (encF N (Var xx))]$

by (*auto simp: φ_def*)

from $prv_prv_imp_trans[OF _ _ _ prv_imp_neg_encF_N_aux this]$

have 22: $prv (imp (PP \langle neg \varphi \rangle) \varphi)$ **by** *auto*

have 3: $prv (imp \varphi (cnj (PP \langle\varphi\rangle) (PP (encF N \langle\varphi\rangle))))$

by (*rule prv_imp_cnj[OF _ _ _ 1 2]*) (*auto simp: φ_def*)

— This is the modification from the proof of *consistent* $\implies \neg \text{prv } j\text{cons}$:

```

have 41: prv (imp (cnj (PP  $\langle \varphi \rangle$ ) (PP  $\langle \text{neg } \varphi \rangle$ )) (PP  $\langle \text{cnj } \varphi (\text{neg } \varphi) \rangle$ ))
using HBL4[of  $\varphi$  neg  $\varphi$ ] by auto
have prv (imp (cnj  $\varphi$  (neg  $\varphi$ )) (fls))
  by (simp add: prv_cnj_imp_monoR2 prv_imp_neg_fls)
from WHBL2[OF _ _ _ this]
have 42: prv (imp (PP  $\langle \text{cnj } \varphi (\text{neg } \varphi) \rangle$ ) (PP  $\langle \text{fls} \rangle$ )) by auto
from prv_prv_imp_trans[OF _ _ _ 41 42]
have prv (imp (cnj (PP  $\langle \varphi \rangle$ ) (PP  $\langle \text{neg } \varphi \rangle$ )) (PP  $\langle \text{fls} \rangle$ )) by auto
from prv_prv_imp_trans[OF _ _ _ this *[unfolded neg_def]]
have prv (neg (cnj (PP  $\langle \varphi \rangle$ ) (PP  $\langle \text{neg } \varphi \rangle$ )))
unfolding neg_def by auto
from prv_cnj_neg_encF_N_aux[OF _ _ _ _ this]
have 4: prv (neg (cnj (PP  $\langle \varphi \rangle$ ) (PP (encF N  $\langle \varphi \rangle$ )))) by auto
— End modification

```

```

have 5: prv (neg  $\varphi$ )
  unfolding neg_def
  by (rule prv_prv_imp_trans[OF _ _ _ 3 4 [unfolded neg_def]]) auto
hence prv (PP  $\langle \text{neg } \varphi \rangle$ ) using HBL1_PP[OF _ _ 5] by auto
hence prv  $\varphi$  using prv_imp_mp[OF _ _ 22] by auto
with 5 assms show False unfolding consistent_def3 by auto
qed

```

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

definition *noContr* :: *bool* **where**

noContr $\equiv \forall \varphi \in \text{fmla}. \text{Fvars } \varphi = \{\} \longrightarrow \text{prv } (\text{neg } (\text{cnj } (\text{PP } \langle \varphi \rangle) (\text{PP } \langle \text{neg } \varphi \rangle)))$

lemma *jcons_noContr*:

assumes *j*: *prv jcons*

shows *noContr*

unfolding *noContr_def* **proof** *safe*

fix φ **assume** φ [*simp*]: $\varphi \in \text{fmla}$ *Fvars* $\varphi = \{\}$

have [*simp*]: *subst* (*PP* (*encF* *N* (*Var* *xx*))) $\langle \varphi \rangle$ *xx* =
PP (*encF* *N* $\langle \varphi \rangle$)

unfolding *PP_def* **by** (*simp* *add*: *subst_compose_same*)

note *j* = *alle_id*[*OF* _ _ *j*[*unfolded jcons_def*], *simplified*]

have 0: *prv* (*neg* (*cnj* (*PP* $\langle \varphi \rangle$)
(*PP* (*encF* *N* $\langle \varphi \rangle$))))

(*is* *prv* (*neg* (*cnj* (*PP* $\langle \varphi \rangle$) ?*j*)))

using *prv_subst*[*OF* _ _ _ *j*, *of* *xx* $\langle \varphi \rangle$] **by** *simp*

have 1: *prv* (*imp* (*PP* $\langle \text{neg } \varphi \rangle$) ?*j*)

using *prv_eqL_neg_encF_N*[*of* φ , *simplified*]

using *prv_imp_neg_encF_N_aux* **by** *auto*

have 2: *prv* (*imp* (*cnj* (*PP* $\langle \varphi \rangle$) (*PP* $\langle \text{neg } \varphi \rangle$))
(*cnj* (*PP* $\langle \varphi \rangle$) ?*j*))

using 0 1 **by** (*simp* *add*: *prv_cnj_mono* *prv_imp_refl*)

have *prv* (*imp* (*cnj* (*PP* $\langle \varphi \rangle$) (*PP* $\langle \text{neg } \varphi \rangle$))
(*cnj* (*PP* $\langle \varphi \rangle$) ?*j*))

by (*simp* *add*: 2 *prv_cnj_mono* *prv_imp_refl*)

thus *prv* (*neg* (*cnj* (*PP* $\langle \varphi \rangle$) (*PP* $\langle \text{neg } \varphi \rangle$))) **using** 0

unfolding *neg_def*

by (*elim* *prv_prv_imp_trans*[*rotated* 3]) *auto*

qed

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

lemma *noContr_implies_neg_PP_fls*:

```

assumes noContr
shows prv (neg (PP ⟨fls⟩))
proof –
  have prv (neg (cnj (PP ⟨fls⟩) (PP ⟨neg fls⟩)))
    using assms unfolding noContr_def by auto
  thus ?thesis
  using Fvars_tru enc in_num tru_def PP PP_def fls imp HBL1 neg_def
    prv_cnj_imp prv_fls prv_imp_com prv_imp_mp
  by (metis Encode.enc HBL1_axioms HBL1_def)
qed

```

```

corollary jcons_implies_neg_PP_fls:
assumes prv jcons
shows prv (neg (PP ⟨fls⟩))
by (simp add: assms noContr_implies_neg_PP_fls jcons_noContr)

```

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:
 $\bigwedge \varphi \chi \psi. \varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies \psi \in \text{fmla} \implies$ 
 $\text{Fvars } \varphi = \{\} \implies \text{Fvars } \chi = \{\} \implies \text{Fvars } \psi = \{\} \implies$ 
 $\text{prv } (\text{imp } \varphi (\text{imp } \chi \psi)) \implies \text{prv } (\text{imp } (\text{PP } \langle \varphi \rangle) (\text{imp } (\text{PP } \langle \chi \rangle) (\text{PP } \langle \psi \rangle)))$ 
assumes p: prv (neg (PP ⟨fls⟩))
shows noContr
unfolding noContr_def proof safe
  fix  $\varphi$  assume  $\varphi[\text{simp}]$ :  $\varphi \in \text{fmla}$   $\text{Fvars } \varphi = \{\}$ 
  have 0: prv (imp  $\varphi$  (imp (neg  $\varphi$ ) fls))
    by (simp add: prv_imp_neg_fls)
  have 1: prv (imp (PP ⟨ $\varphi$ ⟩) (imp (PP ⟨neg  $\varphi$ ⟩) (PP ⟨fls⟩)))
    using WWHBL22[OF _ _ _ _ 0] by auto
  show prv (neg (cnj (PP ⟨ $\varphi$ ⟩) (PP ⟨neg  $\varphi$ ⟩))) using 1 p
    unfolding neg_def
    by (elim prv_cnj_imp_monoR2[rotated 3, OF prv_prv_imp_trans[rotated 3]])
      (auto intro!: prv_imp_monoL)
qed

```

end — context *Jeroslow_Godel_Second*

Chapter 13

Löb Formulas

The Löb formula, parameterized by a sentence φ , is defined by diagonalizing $\text{imp } P \varphi$.

```
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 $\varphi L :: 'fmla \Rightarrow 'fmla$ **where** $\varphi L \varphi \equiv \text{diag } (\text{imp } P \varphi)$

lemma $\varphi L[\text{simp}, \text{intro}]$: $\bigwedge \varphi. \varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \varphi L \varphi \in \text{fmla}$

and

$\text{Fvars}_{\varphi L}[\text{simp}]$: $\varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{Fvars } (\varphi L \varphi) = \{\}$

unfolding φL_def PP_def **by** *auto*

lemma $\text{bprv}_{\varphi L_equiv}$:

$\varphi \in \text{fm}la \implies \text{Fvars } \varphi = \{\} \implies \text{bprv } (\text{eqv } (\varphi L \varphi) (\text{imp } (PP \langle \varphi L \varphi \rangle) \varphi))$
unfolding φL_def PP_def **using** bprv_diag_eqv [of $\text{imp } P \varphi$] **by auto**

lemma $\text{prv_}\varphi L_eqv$:

$\varphi \in \text{fm}la \implies \text{Fvars } \varphi = \{\} \implies \text{prv } (\text{eqv } (\varphi L \varphi) (\text{imp } (PP \langle \varphi L \varphi \rangle) \varphi))$
using bprv_prv [$OF _ _ \text{bprv_}\varphi L_eqv$, simplified] **by auto**

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  $\varphi$ [simp]:  $\varphi \in \text{fmla } Fvars \varphi = \{\}$  and  $p$ :  $\text{prv } (\varphi L \varphi)$ 
shows  $\text{prv } \varphi$ 
proof-
  have  $\text{prv } (\text{imp } (PP \langle \varphi L \varphi \rangle) \varphi)$  using assms prv_eqv_prv[OF  $\_ \_ p \text{ prv\_}\varphi L \_ \text{eqv}$ ] by auto
  moreover have  $\text{bprv } (PP \langle \varphi L \varphi \rangle)$  using HBL1[OF  $\varphi L$ [OF  $\varphi$ ]  $\_ p$ ] unfolding PP_def by simp
  from bprv_prv[OF  $\_ \_ \text{this, simplified}$ ] have  $\text{prv } (PP \langle \varphi L \varphi \rangle)$  .
  ultimately show ?thesis using PP  $\varphi L$  by (meson assms enc in_num prv_imp_mp)
qed
```

```
lemma loeb_aux_bprv:
```

```

assumes  $\varphi[simp]$ :  $\varphi \in fmla$   $Fvars \varphi = \{\}$  and  $p$ :  $bprv (\varphi L \varphi)$ 
shows  $bprv \varphi$ 
proof –
  note  $pp = bprv\_prv[OF \_ \_ p, simplified]$ 
  have  $bprv (imp (PP (\varphi L \varphi)) \varphi)$  using  $assms B.prv\_eqv\_prv[OF \_ \_ p bprv\_varphi\_L\_eqv]$  by auto
  moreover have  $bprv (PP (\varphi L \varphi))$  using  $HBL1[OF \varphi L[OF \varphi] \_ pp]$  unfolding  $PP\_def$  by simp
  ultimately show ?thesis using  $PP \varphi L$  by (meson  $assms enc in\_num B.prv\_imp\_mp$ )
qed

```

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

```

lemma  $P\_L$ :
assumes  $\varphi[simp]$ :  $\varphi \in fmla$   $Fvars \varphi = \{\}$ 
shows  $bprv (imp (PP (\varphi L \varphi)) (PP (\varphi)))$ 
proof –
  have  $0$ :  $prv (imp (\varphi L \varphi) (imp (PP (\varphi L \varphi)) \varphi))$ 
    using  $prv\_varphi\_L\_eqv$  by (intro  $prv\_imp\_eqvEL$ ) auto
  have  $1$ :  $bprv (PP (imp (\varphi L \varphi) (imp (PP (\varphi L \varphi)) \varphi)))$ 
    using  $HBL1\_PP[OF \_ \_ 0]$  by simp
  have  $2$ :  $bprv (imp (PP (\varphi L \varphi)) (PP (imp (PP (\varphi L \varphi)) \varphi)))$ 
    using  $HBL2\_imp2[OF \_ \_ \_ 1]$  by simp
  have  $3$ :  $bprv (imp (PP (\varphi L \varphi)) (PP (PP (\varphi L \varphi))))$ 
    using  $HBL3[OF \varphi L[OF \varphi] \_]$  by simp
  have  $23$ :  $bprv (imp (PP (\varphi L \varphi))$ 
    (cnj ( $PP (PP (\varphi L \varphi))$ 
    ( $PP (imp (PP (\varphi L \varphi)) \varphi))$ )))
    using  $B.prv\_imp\_cnj[OF \_ \_ \_ 3 2]$  by simp
  have  $4$ :  $bprv (imp (cnj (PP (PP (\varphi L \varphi)))$ 
    ( $PP (imp (PP (\varphi L \varphi)) \varphi))$ 
    ( $PP (\varphi)$ )))
    using  $HBL2[of PP (\varphi L \varphi) \varphi]$  by simp
  show ?thesis using  $B.prv\_prv\_imp\_trans[OF \_ \_ \_ 23 4]$  by simp
qed

```

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  $\varphi[simp]$ :  $\varphi \in fmla$   $Fvars \varphi = \{\}$  and  $p$ :  $bprv (imp (PP (\varphi)) \varphi)$ 
shows  $bprv \varphi$ 
proof –
  have  $bprv (imp (PP (\varphi L \varphi)) \varphi)$ 
    by (rule  $B.prv\_prv\_imp\_trans[OF \_ \_ \_ P\_L p]$ ) auto
  hence  $bprv (\varphi L \varphi)$ 
    by (rule  $B.prv\_eqv\_prv\_rev[OF \_ \_ \_ bprv\_varphi\_L\_eqv, rotated 2]$ ) auto
  thus ?thesis using  $loeb\_aux\_bprv[OF \varphi]$  by simp
qed

```

```

theorem  $loeb\_prv$ :
assumes  $\varphi[simp]$ :  $\varphi \in fmla$   $Fvars \varphi = \{\}$  and  $p$ :  $prv (imp (PP (\varphi)) \varphi)$ 
shows  $prv \varphi$ 
proof –
  note  $PL = bprv\_prv[OF \_ \_ P\_L, simplified]$ 
  have  $prv (imp (PP (\varphi L \varphi)) \varphi)$ 
    by (rule  $prv\_prv\_imp\_trans[OF \_ \_ \_ PL p]$ ) auto
  hence  $prv (\varphi L \varphi)$ 
    by (rule  $prv\_eqv\_prv\_rev[OF \_ \_ \_ prv\_varphi\_L\_eqv, rotated 2]$ ) auto
  thus ?thesis using  $loeb\_aux\_prv[OF \varphi]$  by simp
qed

```

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 $prv_T: \bigwedge \varphi. \varphi \in fmla \implies Fvars\ \varphi = \{\} \implies prv\ (eqv\ (subst\ T\ \langle \varphi \rangle\ xx)\ \varphi)$
begin

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

lemma $\varphi T[simp,intro!]: \varphi T \in fmla$ **and**
 $Fvars_ \varphi T[simp]: Fvars\ \varphi T = \{\}$
 unfolding $\varphi T_def\ PP_def$ **by** *auto*

lemma $bprv_ \varphi T_eqv:$
 $bprv\ (eqv\ \varphi T\ (neg\ (subst\ T\ \langle \varphi T \rangle\ xx)))$
 unfolding φT_def **using** $bprv_diag_eqv[of\ neg\ T]$ **by** *simp*

lemma $prv_ \varphi T_eqv:$
 $prv\ (eqv\ \varphi T\ (neg\ (subst\ T\ \langle \varphi T \rangle\ xx)))$
 using $d_dwf.bprv_prv'[OF_ _ bprv_ \varphi T_eqv, simplified]$.

lemma $\varphi T_prv_fls: prv\ fls$
using $prv_eqv_eqv_neg_prv_fls2[OF_ _ _ prv_T[OF\ \varphi T\ Fvars_ \varphi T]\ prv_ \varphi T_eqv]$ **by** *auto*

end — context

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

```
using  $\varphi T\_prv\_fls$ [OF assms] consistent_def by auto
```

```
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 ( $\langle \_ \rangle$ )
and S
and P
+
fixes isTrue :: 'fmla  $\Rightarrow$  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  $\neg consistent$ 
using assms prv_eq_isTrue[of eqv (subst T \langle \_ \rangle xx) \_]
  by (intro Tarski_proof_theoretic[OF 0]) auto
```

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.