Syntax-Independent Logic Infrastructure

Andrei Popescu Dmitriy Traytel

May 15, 2021

Abstract

We formalize a notion of logic whose terms and formulas are kept abstract. In particular, logical connectives, substitution, free variables, and provability are not defined, but characterized by their general properties as locale assumptions. Based on this abstract characterization, we develop further reusable reasoning infrastructure. For example, we define parallel substitution (along with proving its characterizing theorems) from single-point substitution. Similarly, we develop a natural deduction style proof system starting from the abstract Hilbert-style one. These one-time efforts benefit different concrete logics satisfying our locales' assumptions.

We instantiate the syntax-independent logic infrastructure to Robinson arithmetic (also known as Q) in the AFP entry Robinson_Arithmetic and to hereditarily finite set theory in the AFP entries Goedel_HFSet_Semantic and Goedel_HFSet_Semanticless, which are part of our formalization of Gödel's Incompleteness Theorems described in our CADE-27 paper [1].

Contents

Chapter 1

Preliminaries

1.1 Trivia

abbreviation (*input*) *any* **where** *any* ≡ *undefined*

lemma *Un_Diff2*: $B \cap C = \{\} \Longrightarrow A \cup B - C = A - C \cup B$ ⟨*proof*⟩

lemma *Diff_Diff_Un*: $A - B - C = A - (B \cup C)$ ⟨*proof*⟩

fun *first* :: *nat* ⇒ *nat list* **where**
  first 0 = []
| *first* (*Suc n*) = *n* # *first n*

Facts about zipping lists:

lemma *fst_set_zip_map_fst*:
  length xs = *length ys* ⟹ *fst* ‘ (*set* (*zip* (*map fst xs*) *ys*)) = *fst* ‘ (*set xs*)
  ⟨*proof*⟩

lemma *snd_set_zip_map_snd*:
  length xs = *length ys* ⟹ *snd* ‘ (*set* (*zip xs* (*map snd ys*))) = *snd* ‘ (*set ys*)
  ⟨*proof*⟩

lemma *snd_set_zip*:
  length xs = *length ys* ⟹ *snd* ‘ (*set* (*zip xs ys*)) = *set ys*
  ⟨*proof*⟩

lemma *set_zip_D*: (*x*, *y*) ∈ *set* (*zip xs ys*) ⟹ *x* ∈ *set xs* ∧ *y* ∈ *set ys*
  ⟨*proof*⟩

lemma *inj_on_set_zip_map*:
  assumes *i*: *inj_on f X*
    and *a*: (*f x1*, *y1*) ∈ *set* (*zip* (*map f xs*) *ys*) *set xs* ⊆ *X x1* ∈ *X length xs* = *length ys*
  shows (*x1*, *y1*) ∈ *set* (*zip xs ys*)
⟨*proof*⟩

lemma *set_zip_map_fst_snd*:
  assumes (*u*,*x*) ∈ *set* (*zip us* (*map snd txs*))
    and (*t*,*u*) ∈ *set* (*zip* (*map fst txs*) *us*)
    and *distinct* (*map snd txs*)
    and *distinct us* **and** *length us* = *length txs*
  shows (*t*, *x*) ∈ *set txs*
  ⟨*proof*⟩

lemma *set_zip_map_fst_snd2*:
assumes $(u, x) \in set\ (zip\ us\ (map\ snd\ txs))$
and $(t, x) \in set\ txs$
and *distinct* (*map snd txs*)
and *distinct us* **and** *length us* = *length txs*
shows $(t, u) \in set\ (zip\ (map\ fst\ txs)\ us)$
⟨*proof*⟩

lemma *set_zip_length_map*:
assumes $(x1, y1) \in set\ (zip\ xs\ ys)$ **and** *length xs* = *length ys*
shows $(f\ x1, y1) \in set\ (zip\ (map\ f\ xs)\ ys)$
⟨*proof*⟩

definition *asList* :: $'a\ set \Rightarrow 'a\ list$ **where**
asList A ≡ *SOME as*. *set as* = *A*

lemma *asList*[*simp*,*intro*!]: *finite A* ⟹ *set* (*asList A*) = *A*
⟨*proof*⟩

lemma *triv_Un_imp_aux*:
$(\bigwedge a.\ \varphi \Longrightarrow a \notin A \Longrightarrow a \in B \longleftrightarrow a \in C) \Longrightarrow \varphi \longrightarrow A \cup B = A \cup C$
⟨*proof*⟩

definition *toN* **where** *toN n* ≡ [*0*..<(*Suc n*)]

lemma *set_toN*[*simp*]: *set* (*toN n*) = {*0*..*n*}
⟨*proof*⟩

declare *list.map_cong*[*cong*]

1.2 Some Proof Infrastructure

⟨*ML*⟩

method *RULE* **methods** *meth* **uses** *rule* =
(*rule rule*; (*solves meth*)*?*)

TryUntilFail:

method *TUF* **methods** *meth* =
((*meth*;*fail*)+)*?*

Helping a method, usually simp or auto, with specific substitutions inserted. For auto, this is a bit like a "simp!" analogue of "intro!" and "dest!": It forces the application of an indicated simplification rule, if this is possible.

method *variousSubsts1* **methods** *meth* **uses** *s1* =
(*meth?*,(*subst s1*)*?*, *meth?*)
method *variousSubsts2* **methods** *meth* **uses** *s1 s2* =
(*meth?*,(*subst s1*)*?*, *meth?*, *subst s2*, *meth?*)
method *variousSubsts3* **methods** *meth* **uses** *s1 s2 s3* =
(*meth?*,(*subst s1*)*?*, *meth?*, (*subst s2*)*?*, *meth?*, (*subst s3*)*?*, *meth?*)
method *variousSubsts4* **methods** *meth* **uses** *s1 s2 s3 s4* =
(*meth?*,(*subst s1*)*?*, *meth?*, (*subst s2*)*?*, *meth?*, (*subst s3*)*?*, *meth?*, (*subst s4*)*?*, *meth?*)
method *variousSubsts5* **methods** *meth* **uses** *s1 s2 s3 s4 s5* =
(*meth?*,(*subst s1*)*?*, *meth?*, (*subst s2*)*?*, *meth?*, (*subst s3*)*?*, *meth?*, (*subst s4*)*?*, *meth?*, (*subst s5*)*?*, *meth?*)
method *variousSubsts6* **methods** *meth* **uses** *s1 s2 s3 s4 s5 s6* =
(*meth?*,(*subst s1*)*?*, *meth?*, (*subst s2*)*?*, *meth?*, (*subst s3*)*?*, *meth?*,

(*subst s4*)*?*, *meth?*, (*subst s5*)*?*, *meth?*, (*subst s6*)*?*, *meth?*)

Chapter 2

Syntax

2.1 Generic Syntax

We develop some generic (meta-)axioms for syntax and substitution. We only assume that the syntax of our logic has notions of variable, term and formula, which *include* subsets of "numeric" variables, terms and formulas, the latter being endowed with notions of free variables and substitution subject to some natural properties.

locale *Generic_Syntax* =
fixes
var :: *'var set* — numeric variables (i.e., variables ranging over numbers)
and *trm* :: *'trm set* — numeric trms, which include the numeric variables
and *fmla* :: *'fmla set* — numeric formulas
and *Var* :: *'var* ⇒ *'trm* — trms include at least the variables
and *FvarsT* :: *'trm* ⇒ *'var set* — free variables for trms
and *substT* :: *'trm* ⇒ *'trm* ⇒ *'var* ⇒ *'trm* — substitution for trms
and *Fvars* :: *'fmla* ⇒ *'var set* — free variables for formulas
and *subst* :: *'fmla* ⇒ *'trm* ⇒ *'var* ⇒ *'fmla* — substitution for formulas
assumes
infinite_var: *infinite var* — the variables are assumed infinite
and — Assumptions about the infrastructure (free vars, substitution and the embedding of variables into trms. NB: We need fewer assumptions for trm substitution than for formula substitution!
Var[*simp*,*intro*!]: ⋀*x*. *x* ∈ *var* ⟹ *Var x* ∈ *trm*
and
inj_Var[*simp*]: ⋀ *x y*. *x* ∈ *var* ⟹ *y* ∈ *var* ⟹ (*Var x* = *Var y* ⟷ *x* = *y*)
and
finite_FvarsT: ⋀ *t*. *t* ∈ *trm* ⟹ *finite* (*FvarsT t*)
and
FvarsT: ⋀*t*. *t* ∈ *trm* ⟹ *FvarsT t* ⊆ *var*
and
substT[*simp*,*intro*]: ⋀*t1 t x*. *t1* ∈ *trm* ⟹ *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹ *substT t1 t x* ∈ *trm*
and
FvarsT_Var[*simp*]: ⋀ *x*. *x* ∈ *var* ⟹ *FvarsT* (*Var x*) = {*x*}
and
substT_Var[*simp*]: ⋀ *x t y*. *x* ∈ *var* ⟹ *y* ∈ *var* ⟹ *t* ∈ *trm* ⟹
substT (*Var x*) *t y* = (*if x* = *y then t else Var x*)
and
substT_notIn[*simp*]:
⋀*t1 t2 x*. *x* ∈ *var* ⟹ *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹ *x* ∉ *FvarsT t1* ⟹ *substT t1 t2 x* = *t1*
and
— Assumptions about the infrastructure (free vars and substitution) on formulas
finite_Fvars: ⋀ *φ*. *φ* ∈ *fmla* ⟹ *finite* (*Fvars φ*)
and

Fvars: $\bigwedge\varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi \subseteq var$
and
subst[*simp*,*intro*]: $\bigwedge\varphi\ t\ x.\ \varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow subst\ \varphi\ t\ x \in fmla$
and
Fvars_subst_in:
$\bigwedge\ \varphi\ t\ x.\ \varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow x \in Fvars\ \varphi \Longrightarrow$
$Fvars\ (subst\ \varphi\ t\ x) = Fvars\ \varphi - \{x\} \cup FvarsT\ t$
and
subst_compose_eq_or:
$\bigwedge\ \varphi\ t1\ t2\ x1\ x2.\ \varphi \in fmla \Longrightarrow t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow x1 \in var \Longrightarrow x2 \in var \Longrightarrow$
$x1 = x2 \vee x2 \notin Fvars\ \varphi \Longrightarrow subst\ (subst\ \varphi\ t1\ x1)\ t2\ x2 = subst\ \varphi\ (substT\ t1\ t2\ x2)\ x1$
and
subst_compose_diff:
$\bigwedge\ \varphi\ t1\ t2\ x1\ x2.\ \varphi \in fmla \Longrightarrow t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow x1 \in var \Longrightarrow x2 \in var \Longrightarrow$
$x1 \neq x2 \Longrightarrow x1 \notin FvarsT\ t2 \Longrightarrow$
$subst\ (subst\ \varphi\ t1\ x1)\ t2\ x2 = subst\ (subst\ \varphi\ t2\ x2)\ (substT\ t1\ t2\ x2)\ x1$
and
subst_same_Var[*simp*]:
$\bigwedge\ \varphi\ x.\ \varphi \in fmla \Longrightarrow x \in var \Longrightarrow subst\ \varphi\ (Var\ x)\ x = \varphi$
and
subst_notIn[*simp*]:
$\bigwedge\ x\ \varphi\ t.\ \varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow x \notin Fvars\ \varphi \Longrightarrow subst\ \varphi\ t\ x = \varphi$
begin

lemma *var_NE*: $var \neq \{\}$
⟨*proof*⟩

lemma *Var_injD*: $Var\ x = Var\ y \Longrightarrow x \in var \Longrightarrow y \in var \Longrightarrow x = y$
⟨*proof*⟩

lemma *FvarsT_VarD*: $x \in FvarsT\ (Var\ y) \Longrightarrow y \in var \Longrightarrow x = y$
⟨*proof*⟩

lemma *FvarsT'*: $t \in trm \Longrightarrow x \in FvarsT\ t \Longrightarrow x \in var$
⟨*proof*⟩

lemma *Fvars'*: $\varphi \in fmla \Longrightarrow x \in Fvars\ \varphi \Longrightarrow x \in var$
⟨*proof*⟩

lemma *Fvars_subst*[*simp*]:
$\varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow$
$Fvars\ (subst\ \varphi\ t\ x) = (Fvars\ \varphi - \{x\}) \cup (if\ x \in Fvars\ \varphi\ then\ FvarsT\ t\ else\ \{\})$
⟨*proof*⟩

lemma *in_Fvars_substD*:
$y \in Fvars\ (subst\ \varphi\ t\ x) \Longrightarrow \varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var$
$\Longrightarrow y \in (Fvars\ \varphi - \{x\}) \cup (if\ x \in Fvars\ \varphi\ then\ FvarsT\ t\ else\ \{\})$
⟨*proof*⟩

lemma *inj_on_Var*: $inj\_on\ Var\ var$
⟨*proof*⟩

lemma *subst_compose_same*:
$\bigwedge\ \varphi\ t1\ t2\ x.\ \varphi \in fmla \Longrightarrow t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow x \in var \Longrightarrow$
$subst\ (subst\ \varphi\ t1\ x)\ t2\ x = subst\ \varphi\ (substT\ t1\ t2\ x)\ x$
⟨*proof*⟩

lemma *subst_subst*[*simp*]:

assumes φ[*simp*]: $\varphi \in fmla$ **and** t[*simp*]:$t \in trm$ **and** x[*simp*]:$x \in var$ **and** y[*simp*]:$y \in var$
assumes *yy*: $x \neq y$ $y \notin Fvars\ \varphi$
shows $subst\ (subst\ \varphi\ (Var\ y)\ x)\ t\ y = subst\ \varphi\ t\ x$
⟨*proof*⟩

lemma *subst_comp*:
$\bigwedge x\ y\ \varphi\ t.\ \varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow y \in var \Longrightarrow$
$x \neq y \Longrightarrow y \notin FvarsT\ t \Longrightarrow$
$subst\ (subst\ \varphi\ (Var\ x)\ y)\ t\ x = subst\ (subst\ \varphi\ t\ x)\ t\ y$
⟨*proof*⟩

lemma *exists_nat_var*:
$\exists\ f{::}nat \Rightarrow 'var.\ inj\ f \wedge range\ f \subseteq var$
⟨*proof*⟩

definition *Variable* :: $nat \Rightarrow 'var$ **where**
$Variable = (SOME\ f.\ inj\ f \wedge range\ f \subseteq var)$

lemma *Variable_inj_var*:
$inj\ Variable \wedge range\ Variable \subseteq var$
⟨*proof*⟩

lemma *inj_Variable*[*simp*]: $\bigwedge i\ j.\ Variable\ i = Variable\ j \longleftrightarrow i = j$
and *Variable*[*simp*,*intro*!]: $\bigwedge i.\ Variable\ i \in var$
⟨*proof*⟩

Convenient notations for some variables We reserve the first 10 indexes for any special variables we may wish to consider later.

abbreviation *xx* **where** $xx \equiv Variable\ 10$
abbreviation *yy* **where** $yy \equiv Variable\ 11$
abbreviation *zz* **where** $zz \equiv Variable\ 12$

abbreviation xx' **where** $xx' \equiv Variable\ 13$
abbreviation yy' **where** $yy' \equiv Variable\ 14$
abbreviation zz' **where** $zz' \equiv Variable\ 15$

lemma *xx*: $xx \in var$
and *yy*: $yy \in var$
and *zz*: $zz \in var$
and xx': $xx' \in var$
and yy': $yy' \in var$
and zz': $zz' \in var$
⟨*proof*⟩

lemma *vars_distinct*[*simp*]:
$xx \neq yy\ yy \neq xx\ xx \neq zz\ zz \neq xx\ xx \neq xx'\ xx' \neq xx\ xx \neq yy'\ yy' \neq xx\ xx \neq zz'\ zz' \neq xx$
$yy \neq zz\ zz \neq yy\ yy \neq xx'\ xx' \neq yy\ yy \neq yy'\ yy' \neq yy\ yy \neq zz'\ zz' \neq yy$
$zz \neq xx'\ xx' \neq zz\ zz \neq yy'\ yy' \neq zz\ zz \neq zz'\ zz' \neq zz$
$xx' \neq yy'\ yy' \neq xx'\ xx' \neq zz'\ zz' \neq xx'$
$yy' \neq zz'\ zz' \neq yy'$
⟨*proof*⟩

2.1.1 Instance Operator

definition *inst* :: $'fmla \Rightarrow 'trm \Rightarrow 'fmla$ **where**
$inst\ \varphi\ t = subst\ \varphi\ t\ xx$

lemma *inst*[*simp*]: $\varphi \in fmla \Longrightarrow t \in trm \Longrightarrow inst\ \varphi\ t \in fmla$

⟨*proof*⟩

definition *getFresh* :: *'var set* ⇒ *'var* **where**
getFresh V = (*SOME x*. *x* ∈ *var* ∧ *x* ∉ *V*)

lemma *getFresh*: *finite V* ⟹ *getFresh V* ∈ *var* ∧ *getFresh V* ∉ *V*
⟨*proof*⟩

definition *getFr* :: *'var list* ⇒ *'trm list* ⇒ *'fmla list* ⇒ *'var* **where**
getFr xs ts φs =
getFresh (*set xs* ∪ (⋃ (*FvarsT* ' *set ts*)) ∪ (⋃ (*Fvars* ' *set φs*)))

lemma *getFr_FvarsT_Fvars*:
assumes *set xs* ⊆ *var set ts* ⊆ *trm* **and** *set φs* ⊆ *fmla*
shows *getFr xs ts φs* ∈ *var* ∧
getFr xs ts φs ∉ *set xs* ∧
(*t* ∈ *set ts* ⟶ *getFr xs ts φs* ∉ *FvarsT t*) ∧
(*φ* ∈ *set φs* ⟶ *getFr xs ts φs* ∉ *Fvars φ*)
⟨*proof*⟩

lemma *getFr*[*simp*,*intro*]:
assumes *set xs* ⊆ *var set ts* ⊆ *trm* **and** *set φs* ⊆ *fmla*
shows *getFr xs ts φs* ∈ *var*
⟨*proof*⟩

lemma *getFr_var*:
assumes *set xs* ⊆ *var set ts* ⊆ *trm* **and** *set φs* ⊆ *fmla* **and** *t* ∈ *set ts*
shows *getFr xs ts φs* ∉ *set xs*
⟨*proof*⟩

lemma *getFr_FvarsT*:
assumes *set xs* ⊆ *var set ts* ⊆ *trm* **and** *set φs* ⊆ *fmla* **and** *t* ∈ *set ts*
shows *getFr xs ts φs* ∉ *FvarsT t*
⟨*proof*⟩

lemma *getFr_Fvars*:
assumes *set xs* ⊆ *var set ts* ⊆ *trm* **and** *set φs* ⊆ *fmla* **and** *φ* ∈ *set φs*
shows *getFr xs ts φs* ∉ *Fvars φ*
⟨*proof*⟩

2.1.2 Fresh Variables

fun *getFreshN* :: *'var set* ⇒ *nat* ⇒ *'var list* **where**
getFreshN V 0 = []
| *getFreshN V* (*Suc n*) = (*let u* = *getFresh V in u* # *getFreshN* (*insert u V*) *n*)

lemma *getFreshN*: *finite V* ⟹
set (*getFreshN V n*) ⊆ *var* ∧ *set* (*getFreshN V n*) ∩ *V* = {} ∧ *length* (*getFreshN V n*) = *n* ∧ *distinct* (*getFreshN V n*)
⟨*proof*⟩

definition *getFrN* :: *'var list* ⇒ *'trm list* ⇒ *'fmla list* ⇒ *nat* ⇒ *'var list* **where**
getFrN xs ts φs n =
getFreshN (*set xs* ∪ (⋃ (*FvarsT* ' *set ts*)) ∪ (⋃ (*Fvars* ' *set φs*))) *n*

lemma *getFrN_FvarsT_Fvars*:
assumes *set xs* ⊆ *var set ts* ⊆ *trm* **and** *set φs* ⊆ *fmla*
shows *set* (*getFrN xs ts φs n*) ⊆ *var* ∧

set (getFrN xs ts φs n) ∩ set xs = {} ∧
(t ∈ set ts ⟶ set (getFrN xs ts φs n) ∩ FvarsT t = {}) ∧
(φ ∈ set φs ⟶ set (getFrN xs ts φs n) ∩ Fvars φ = {}) ∧
length (getFrN xs ts φs n) = n ∧
distinct (getFrN xs ts φs n)
⟨*proof*⟩

lemma *getFrN*[*simp*,*intro*]:
assumes *set xs ⊆ var set ts ⊆ trm* **and** *set φs ⊆ fmla*
shows *set (getFrN xs ts φs n) ⊆ var*
⟨*proof*⟩

lemma *getFrN_var*:
assumes *set xs ⊆ var set ts ⊆ trm* **and** *set φs ⊆ fmla* **and** *t ∈ set ts*
shows *set (getFrN xs ts φs n) ∩ set xs = {}*
⟨*proof*⟩

lemma *getFrN_FvarsT*:
assumes *set xs ⊆ var set ts ⊆ trm* **and** *set φs ⊆ fmla* **and** *t ∈ set ts*
shows *set (getFrN xs ts φs n) ∩ FvarsT t = {}*
⟨*proof*⟩

lemma *getFrN_Fvars*:
assumes *set xs ⊆ var set ts ⊆ trm* **and** *set φs ⊆ fmla* **and** *φ ∈ set φs*
shows *set (getFrN xs ts φs n) ∩ Fvars φ = {}*
⟨*proof*⟩

lemma *getFrN_length*:
assumes *set xs ⊆ var set ts ⊆ trm* **and** *set φs ⊆ fmla*
shows *length (getFrN xs ts φs n) = n*
⟨*proof*⟩

lemma *getFrN_distinct*[*simp*,*intro*]:
assumes *set xs ⊆ var set ts ⊆ trm* **and** *set φs ⊆ fmla*
shows *distinct (getFrN xs ts φs n)*
⟨*proof*⟩

2.1.3 Parallel Term Substitution

fun *rawpsubstT* :: *'trm ⇒ ('trm × 'var) list ⇒ 'trm* **where**
rawpsubstT t [] = t
| *rawpsubstT t ((t1,x1) # txs) = rawpsubstT (substT t t1 x1) txs*

lemma *rawpsubstT*[*simp*]:
assumes *t ∈ trm* **and** *snd ' (set txs) ⊆ var* **and** *fst ' (set txs) ⊆ trm*
shows *rawpsubstT t txs ∈ trm*
⟨*proof*⟩

definition *psubstT* :: *'trm ⇒ ('trm × 'var) list ⇒ 'trm* **where**
psubstT t txs =
(let xs = map snd txs; ts = map fst txs; us = getFrN xs (t # ts) [] (length xs) in
rawpsubstT (rawpsubstT t (zip (map Var us) xs)) (zip ts us))

The psubstT versions of the subst axioms.

lemma *psubstT*[*simp*,*intro*]:
assumes *t ∈ trm* **and** *snd ' (set txs) ⊆ var* **and** *fst ' (set txs) ⊆ trm*
shows *psubstT t txs ∈ trm*
⟨*proof*⟩

lemma *rawpsubstT_Var_not*[*simp*]:
  assumes $x \in var\ snd\ `\ (set\ txs) \subseteq var\ fst\ `\ (set\ txs) \subseteq trm$
    and $x \notin snd\ `\ (set\ txs)$
  shows $rawpsubstT\ (Var\ x)\ txs = Var\ x$
  ⟨*proof*⟩

lemma *psubstT_Var_not*[*simp*]:
  assumes $x \in var\ snd\ `\ (set\ txs) \subseteq var\ fst\ `\ (set\ txs) \subseteq trm$
    and $x \notin snd\ `\ (set\ txs)$
  shows $psubstT\ (Var\ x)\ txs = Var\ x$
⟨*proof*⟩

lemma *rawpsubstT_notIn*[*simp*]:
  assumes $x \in var\ snd\ `\ (set\ txs) \subseteq var\ fst\ `\ (set\ txs) \subseteq trm\ t \in trm$
    and $FvarsT\ t \cap snd\ `\ (set\ txs) = \{\}$
  shows $rawpsubstT\ t\ txs = t$
  ⟨*proof*⟩

lemma *psubstT_notIn*[*simp*]:
  assumes $x \in var\ snd\ `\ (set\ txs) \subseteq var\ fst\ `\ (set\ txs) \subseteq trm\ t \in trm$
    and $FvarsT\ t \cap snd\ `\ (set\ txs) = \{\}$
  shows $psubstT\ t\ txs = t$
⟨*proof*⟩

2.1.4 Parallel Formula Substitution

fun *rawpsubst* :: *'fmla* ⇒ (*'trm* × *'var*) *list* ⇒ *'fmla* **where**
  $rawpsubst\ \varphi\ [] = \varphi$
| $rawpsubst\ \varphi\ ((t1,x1)\ \#\ txs) = rawpsubst\ (subst\ \varphi\ t1\ x1)\ txs$

lemma *rawpsubst*[*simp*]:
  assumes $\varphi \in fmla$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq trm$
  shows $rawpsubst\ \varphi\ txs \in fmla$
  ⟨*proof*⟩

definition *psubst* :: *'fmla* ⇒ (*'trm* × *'var*) *list* ⇒ *'fmla* **where**
  $psubst\ \varphi\ txs =$
    $(let\ xs = map\ snd\ txs;\ ts = map\ fst\ txs;\ us = getFrN\ xs\ ts\ [\varphi]\ (length\ xs)\ in$
      $rawpsubst\ (rawpsubst\ \varphi\ (zip\ (map\ Var\ us)\ xs))\ (zip\ ts\ us))$

The psubst versions of the subst axioms.

lemma *psubst*[*simp*,*intro*]:
  assumes $\varphi \in fmla$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq trm$
  shows $psubst\ \varphi\ txs \in fmla$
⟨*proof*⟩

lemma *Fvars_rawpsubst_su*:
  assumes $\varphi \in fmla$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq trm$
  shows $Fvars\ (rawpsubst\ \varphi\ txs) \subseteq$
      $(Fvars\ \varphi - snd\ `\ (set\ txs)) \cup (\bigcup\ \{FvarsT\ t \mid t\ x\ .\ (t,x) \in set\ txs\})$
  ⟨*proof*⟩

lemma *in_Fvars_rawpsubst_imp*:
  assumes $y \in Fvars\ (rawpsubst\ \varphi\ txs)$
    and $\varphi \in fmla$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq trm$
  shows $(y \in Fvars\ \varphi - snd\ `\ (set\ txs)) \vee$
    $(y \in \bigcup\ \{\ FvarsT\ t \mid t\ x\ .\ (t,x) \in set\ txs\})$

⟨*proof*⟩

lemma *Fvars_rawpsubst*:
assumes $\varphi \in fmla$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq trm$
and $distinct\ (map\ snd\ txs)$ **and** $\forall\ x \in snd\ `\ (set\ txs).\ \forall\ t \in fst\ `\ (set\ txs).\ x \notin FvarsT\ t$
shows $Fvars\ (rawpsubst\ \varphi\ txs) =$
$(Fvars\ \varphi - snd\ `\ (set\ txs)) \cup$
$(\bigcup \{if\ x \in Fvars\ \varphi\ then\ FvarsT\ t\ else\ \{\} \mid t\ x\ .\ (t,x) \in set\ txs\})$
⟨*proof*⟩

lemma *in_Fvars_rawpsubstD*:
assumes $y \in Fvars\ (rawpsubst\ \varphi\ txs)$
and $\varphi \in fmla$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq trm$
and $distinct\ (map\ snd\ txs)$ **and** $\forall\ x \in snd\ `\ (set\ txs).\ \forall\ t \in fst\ `\ (set\ txs).\ x \notin FvarsT\ t$
shows $(y \in Fvars\ \varphi - snd\ `\ (set\ txs)) \vee$
$(y \in \bigcup \{if\ x \in Fvars\ \varphi\ then\ FvarsT\ t\ else\ \{\} \mid t\ x\ .\ (t,x) \in set\ txs\})$
⟨*proof*⟩

lemma *Fvars_psubst*:
assumes $\varphi \in fmla$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq trm$
and $distinct\ (map\ snd\ txs)$
shows $Fvars\ (psubst\ \varphi\ txs) =$
$(Fvars\ \varphi - snd\ `\ (set\ txs)) \cup$
$(\bigcup \{if\ x \in Fvars\ \varphi\ then\ FvarsT\ t\ else\ \{\} \mid t\ x\ .\ (t,x) \in set\ txs\})$
⟨*proof*⟩

lemma *in_Fvars_psubstD*:
assumes $y \in Fvars\ (psubst\ \varphi\ txs)$
and $\varphi \in fmla$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq trm$
and $distinct\ (map\ snd\ txs)$
shows $y \in (Fvars\ \varphi - snd\ `\ (set\ txs)) \cup$
$(\bigcup \{if\ x \in Fvars\ \varphi\ then\ FvarsT\ t\ else\ \{\} \mid t\ x\ .\ (t,x) \in set\ txs\})$
⟨*proof*⟩

lemma *subst2_fresh_switch*:
assumes $\varphi \in fmla\ t \in trm\ s \in trm\ x \in var\ y \in var$
and $x \neq y\ x \notin FvarsT\ s\ y \notin FvarsT\ t$
shows $subst\ (subst\ \varphi\ s\ y)\ t\ x = subst\ (subst\ \varphi\ t\ x)\ s\ y$ (**is** *?L = ?R*)
⟨*proof*⟩

lemma *rawpsubst2_fresh_switch*:
assumes $\varphi \in fmla\ t \in trm\ s \in trm\ x \in var\ y \in var$
and $x \neq y\ x \notin FvarsT\ s\ y \notin FvarsT\ t$
shows $rawpsubst\ \varphi\ ([(s,y),(t,x)]) = rawpsubst\ \varphi\ ([(t,x),(s,y)])$
⟨*proof*⟩

lemma *rawpsubst_compose*:
assumes $\varphi \in fmla$ **and** $snd\ `\ (set\ txs1) \subseteq var$ **and** $fst\ `\ (set\ txs1) \subseteq trm$
and $snd\ `\ (set\ txs2) \subseteq var$ **and** $fst\ `\ (set\ txs2) \subseteq trm$
shows $rawpsubst\ (rawpsubst\ \varphi\ txs1)\ txs2 = rawpsubst\ \varphi\ (txs1\ @\ txs2)$
⟨*proof*⟩

lemma *rawpsubst_subst_fresh_switch*:
assumes $\varphi \in fmla\ snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq trm$
and $\forall\ x \in snd\ `\ (set\ txs).\ x \notin FvarsT\ s$
and $\forall\ t \in fst\ `\ (set\ txs).\ y \notin FvarsT\ t$
and $distinct\ (map\ snd\ txs)$
and $s \in trm$ **and** $y \in var\ y \notin snd\ `\ (set\ txs)$

shows *rawpsubst* (*subst* φ *s y*) *txs* = *rawpsubst* φ (*txs* @ [(*s*,*y*)])
⟨*proof*⟩

lemma *subst_rawpsubst_fresh_switch*:
assumes $\varphi \in$ *fmla snd* ‘ (*set txs*) ⊆ *var* **and** *fst* ‘ (*set txs*) ⊆ *trm*
and ∀ *x* ∈ *snd* ‘ (*set txs*). *x* ∉ *FvarsT s*
and ∀ *t* ∈ *fst* ‘ (*set txs*). *y* ∉ *FvarsT t*
and *distinct* (*map snd txs*)
and *s* ∈ *trm* **and** *y* ∈ *var y* ∉ *snd* ‘ (*set txs*)
shows *subst* (*rawpsubst* φ *txs*) *s y* = *rawpsubst* φ ((*s*,*y*) # *txs*)
⟨*proof*⟩

lemma *rawpsubst_compose_freshVar*:
assumes $\varphi \in$ *fmla snd* ‘ (*set txs*) ⊆ *var* **and** *fst* ‘ (*set txs*) ⊆ *trm*
and *distinct* (*map snd txs*)
and ⋀ *i j*. *i* < *j* ⟹ *j* < *length txs* ⟹ *snd* (*txs*!*j*) ∉ *FvarsT* (*fst* (*txs*!*i*))
and *us_facts*: *set us* ⊆ *var*
set us ∩ *Fvars* φ = {}
set us ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set us ∩ *snd* ‘ (*set txs*) = {}
length us = *length txs*
distinct us
shows *rawpsubst* (*rawpsubst* φ (*zip* (*map Var us*) (*map snd txs*))) (*zip* (*map fst txs*) *us*) = *rawpsubst* φ *txs*
⟨*proof*⟩

lemma *rawpsubst_compose_freshVar2_aux*:
assumes φ[*simp*]: $\varphi \in$ *fmla*
and *ts*: *set ts* ⊆ *trm*
and *xs*: *set xs* ⊆ *var* *distinct xs*
and *us_facts*: *set us* ⊆ *var* *distinct us*
set us ∩ *Fvars* φ = {}
set us ∩ ⋃ (*FvarsT* ‘ (*set ts*)) = {}
set us ∩ *set xs* = {}
and *vs_facts*: *set vs* ⊆ *var* *distinct vs*
set vs ∩ *Fvars* φ = {}
set vs ∩ ⋃ (*FvarsT* ‘ (*set ts*)) = {}
set vs ∩ *set xs* = {}
and *l*: *length us* = *length xs length vs* = *length xs length ts* = *length xs*
and *d*: *set us* ∩ *set vs* = {}
shows *rawpsubst* (*rawpsubst* φ (*zip* (*map Var us*) *xs*)) (*zip ts us*) =
rawpsubst (*rawpsubst* φ (*zip* (*map Var vs*) *xs*)) (*zip ts vs*)
⟨*proof*⟩

... now getting rid of the disjointness hypothesis:

lemma *rawpsubst_compose_freshVar2*:
assumes φ[*simp*]: $\varphi \in$ *fmla*
and *ts*: *set ts* ⊆ *trm*
and *xs*: *set xs* ⊆ *var* *distinct xs*
and *us_facts*: *set us* ⊆ *var* *distinct us*
set us ∩ *Fvars* φ = {}
set us ∩ ⋃ (*FvarsT* ‘ (*set ts*)) = {}
set us ∩ *set xs* = {}
and *vs_facts*: *set vs* ⊆ *var* *distinct vs*
set vs ∩ *Fvars* φ = {}
set vs ∩ ⋃ (*FvarsT* ‘ (*set ts*)) = {}
set vs ∩ *set xs* = {}
and *l*: *length us* = *length xs length vs* = *length xs length ts* = *length xs*

shows *rawpsubst* (*rawpsubst* φ (*zip* (*map Var us*) *xs*)) (*zip ts us*) =
rawpsubst (*rawpsubst* φ (*zip* (*map Var vs*) *xs*)) (*zip ts vs*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *psubst_subst_fresh_switch*:
assumes $\varphi \in$ *fmla snd ‘ set txs* $\subseteq$ *var fst ‘ set txs* $\subseteq$ *trm*
and $\forall x \in$ *snd ‘ set txs. x* $\notin$ *FvarsT s* $\forall t \in$ *fst ‘ set txs. y* $\notin$ *FvarsT t*
and *distinct* (*map snd txs*)
and *s* $\in$ *trm y* $\in$ *var y* $\notin$ *snd ‘ set txs*
shows *psubst* (*subst* φ *s y*) *txs* = *subst* (*psubst* φ *txs*) *s y*
⟨*proof*⟩

For many cases, the simpler rawpsubst can replace psubst:

lemma *psubst_eq_rawpsubst*:
assumes $\varphi \in$ *fmla snd ‘* (*set txs*) $\subseteq$ *var* **and** *fst ‘* (*set txs*) $\subseteq$ *trm*
and *distinct* (*map snd txs*)

and $\bigwedge$ *i j. i < j* $\Longrightarrow$ *j < length txs* $\Longrightarrow$ *snd* (*txs*!*j*) $\notin$ *FvarsT* (*fst* (*txs*!*i*))
shows *psubst* φ *txs* = *rawpsubst* φ *txs*
⟨*proof*⟩

Some particular cases:

lemma *psubst_eq_subst*:
assumes $\varphi \in$ *fmla x* $\in$ *var* **and** *t* $\in$ *trm*
shows *psubst* φ [(*t*,*x*)] = *subst* φ *t x*
⟨*proof*⟩

lemma *psubst_eq_rawpsubst2*:
assumes $\varphi \in$ *fmla x1* $\in$ *var x2* $\in$ *var t1* $\in$ *trm t2* $\in$ *trm*
and *x1* $\neq$ *x2 x2* $\notin$ *FvarsT t1*
shows *psubst* φ [(*t1*,*x1*),(*t2*,*x2*)] = *rawpsubst* φ [(*t1*,*x1*),(*t2*,*x2*)]
⟨*proof*⟩

lemma *psubst_eq_rawpsubst3*:
assumes $\varphi \in$ *fmla x1* $\in$ *var x2* $\in$ *var x3* $\in$ *var t1* $\in$ *trm t2* $\in$ *trm t3* $\in$ *trm*
and *x1* $\neq$ *x2 x1* $\neq$ *x3 x2* $\neq$ *x3*
x2 $\notin$ *FvarsT t1 x3* $\notin$ *FvarsT t1 x3* $\notin$ *FvarsT t2*
shows *psubst* φ [(*t1*,*x1*),(*t2*,*x2*),(*t3*,*x3*)] = *rawpsubst* φ [(*t1*,*x1*),(*t2*,*x2*),(*t3*,*x3*)]
⟨*proof*⟩

lemma *psubst_eq_rawpsubst4*:
assumes $\varphi \in$ *fmla x1* $\in$ *var x2* $\in$ *var x3* $\in$ *var x4* $\in$ *var*
t1 $\in$ *trm t2* $\in$ *trm t3* $\in$ *trm t4* $\in$ *trm*
and *x1* $\neq$ *x2 x1* $\neq$ *x3 x2* $\neq$ *x3 x1* $\neq$ *x4 x2* $\neq$ *x4 x3* $\neq$ *x4*
x2 $\notin$ *FvarsT t1 x3* $\notin$ *FvarsT t1 x3* $\notin$ *FvarsT t2 x4* $\notin$ *FvarsT t1 x4* $\notin$ *FvarsT t2 x4* $\notin$ *FvarsT t3*
shows *psubst* φ [(*t1*,*x1*),(*t2*,*x2*),(*t3*,*x3*),(*t4*,*x4*)] = *rawpsubst* φ [(*t1*,*x1*),(*t2*,*x2*),(*t3*,*x3*),(*t4*,*x4*)]
⟨*proof*⟩

lemma *rawpsubst_same_Var*[*simp*]:
assumes $\varphi \in$ *fmla set xs* $\subseteq$ *var*
shows *rawpsubst* φ (*map* (λx. (*Var x*,*x*)) *xs*) = φ
⟨*proof*⟩

lemma *psubst_same_Var*[*simp*]:
assumes $\varphi \in$ *fmla set xs* $\subseteq$ *var* **and** *distinct xs*
shows *psubst* φ (*map* (λx. (*Var x*,*x*)) *xs*) = φ
⟨*proof*⟩

lemma *rawpsubst_notIn*[*simp*]:
assumes *snd* ‘ (*set txs*) ⊆ *var fst* ‘ (*set txs*) ⊆ *trm* φ ∈ *fmla*
and *Fvars* φ ∩ *snd* ‘ (*set txs*) = {}
shows *rawpsubst* φ *txs* = φ
⟨*proof*⟩

lemma *psubst_notIn*[*simp*]:
assumes *x* ∈ *var snd* ‘ (*set txs*) ⊆ *var fst* ‘ (*set txs*) ⊆ *trm* φ ∈ *fmla*
and *Fvars* φ ∩ *snd* ‘ (*set txs*) = {}
shows *psubst* φ *txs* = φ
⟨*proof*⟩

end — context *Generic_Syntax*

2.2 Adding Numerals to the Generic Syntax

locale *Syntax_with_Numerals* =
Generic_Syntax var trm fmla Var FvarsT substT Fvars subst
for *var* :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
+
fixes
— Abstract notion of numerals, as a subset of the ground terms:
num :: *'trm set*
assumes
numNE: *num* ≠ {}
and
num: *num* ⊆ *trm*
and
FvarsT_num[*simp*, *intro*!]: ⋀*n*. *n* ∈ *num* ⟹ *FvarsT n* = {}
begin

lemma *substT_num1*[*simp*]: *t* ∈ *trm* ⟹ *y* ∈ *var* ⟹ *n* ∈ *num* ⟹ *substT n t y* = *n*
⟨*proof*⟩

lemma *in_num*[*simp*]: *n* ∈ *num* ⟹ *n* ∈ *trm* ⟨*proof*⟩

lemma *subst_comp_num*:
assumes φ ∈ *fmla x* ∈ *var y* ∈ *var n* ∈ *num*
shows *x* ≠ *y* ⟹ *subst* (*subst* φ (*Var x*) *y*) *n x* = *subst* (*subst* φ *n x*) *n y*
⟨*proof*⟩

lemma *rawpsubstT_num*:
assumes *snd* ‘ (*set txs*) ⊆ *var fst* ‘ (*set txs*) ⊆ *trm n* ∈ *num*
shows *rawpsubstT n txs* = *n*
⟨*proof*⟩

lemma *psubstT_num*[*simp*]:
assumes *snd* ‘ (*set txs*) ⊆ *var fst* ‘ (*set txs*) ⊆ *trm n* ∈ *num*
shows *psubstT n txs* = *n*
⟨*proof*⟩

end — context *Syntax_with_Numerals*

2.3 Adding Connectives and Quantifiers

locale *Syntax_with_Connectives* =

Generic_Syntax var trm fmla Var FvarsT substT Fvars subst
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
+
fixes
— Logical connectives
eql :: *'trm* ⇒ *'trm* ⇒ *'fmla*
and
cnj :: *'fmla* ⇒ *'fmla* ⇒ *'fmla*
and
imp :: *'fmla* ⇒ *'fmla* ⇒ *'fmla*
and
all :: *'var* ⇒ *'fmla* ⇒ *'fmla*
and
exi :: *'var* ⇒ *'fmla* ⇒ *'fmla*
assumes
eql[*simp*,*intro*]: ⋀ *t1 t2*. *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹ *eql t1 t2* ∈ *fmla*
and
cnj[*simp*,*intro*]: ⋀ *φ1 φ2*. *φ1* ∈ *fmla* ⟹ *φ2* ∈ *fmla* ⟹ *cnj φ1 φ2* ∈ *fmla*
and
imp[*simp*,*intro*]: ⋀ *φ1 φ2*. *φ1* ∈ *fmla* ⟹ *φ2* ∈ *fmla* ⟹ *imp φ1 φ2* ∈ *fmla*
and
all[*simp*,*intro*]: ⋀ *x φ*. *x* ∈ *var* ⟹ *φ* ∈ *fmla* ⟹ *all x φ* ∈ *fmla*
and
exi[*simp*,*intro*]: ⋀ *x φ*. *x* ∈ *var* ⟹ *φ* ∈ *fmla* ⟹ *exi x φ* ∈ *fmla*
and
Fvars_eql[*simp*]:
⋀ *t1 t2*. *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹ *Fvars* (*eql t1 t2*) = *FvarsT t1* ∪ *FvarsT t2*
and
Fvars_cnj[*simp*]:
⋀ *φ χ*. *φ* ∈ *fmla* ⟹ *χ* ∈ *fmla* ⟹ *Fvars* (*cnj φ χ*) = *Fvars φ* ∪ *Fvars χ*
and
Fvars_imp[*simp*]:
⋀ *φ χ*. *φ* ∈ *fmla* ⟹ *χ* ∈ *fmla* ⟹ *Fvars* (*imp φ χ*) = *Fvars φ* ∪ *Fvars χ*
and
Fvars_all[*simp*]:
⋀ *x φ*. *x* ∈ *var* ⟹ *φ* ∈ *fmla* ⟹ *Fvars* (*all x φ*) = *Fvars φ* − {*x*}
and
Fvars_exi[*simp*]:
⋀ *x φ*. *x* ∈ *var* ⟹ *φ* ∈ *fmla* ⟹ *Fvars* (*exi x φ*) = *Fvars φ* − {*x*}
and
— Assumed properties of substitution
subst_cnj[*simp*]:
⋀ *x φ χ t*. *φ* ∈ *fmla* ⟹ *χ* ∈ *fmla* ⟹ *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹
subst (*cnj φ χ*) *t x* = *cnj* (*subst φ t x*) (*subst χ t x*)
and
subst_imp[*simp*]:
⋀ *x φ χ t*. *φ* ∈ *fmla* ⟹ *χ* ∈ *fmla* ⟹ *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹
subst (*imp φ χ*) *t x* = *imp* (*subst φ t x*) (*subst χ t x*)
and
subst_all[*simp*]:
⋀ *x y φ t*. *φ* ∈ *fmla* ⟹ *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹ *y* ∈ *var* ⟹
x ≠ *y* ⟹ *x* ∉ *FvarsT t* ⟹ *subst* (*all x φ*) *t y* = *all x* (*subst φ t y*)
and
subst_exi[*simp*]:
⋀ *x y φ t*. *φ* ∈ *fmla* ⟹ *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹ *y* ∈ *var* ⟹
x ≠ *y* ⟹ *x* ∉ *FvarsT t* ⟹ *subst* (*exi x φ*) *t y* = *exi x* (*subst φ t y*)

and
subst_eql[*simp*]:
$\bigwedge$ *t1 t2 t x*. $t \in trm \Longrightarrow t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow x \in var \Longrightarrow$
subst (*eql t1 t2*) *t x* = *eql* (*substT t1 t x*) (*substT t2 t x*)
begin

Formula equivalence, $\longleftrightarrow$, a derived connective

definition *eqv* :: *'fmla* ⇒ *'fmla* ⇒ *'fmla* **where**
eqv φ χ = *cnj* (*imp* φ χ) (*imp* χ φ)

lemma
eqv[*simp*]: $\bigwedge\varphi$ χ. $\varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow eqv\ \varphi\ \chi \in fmla$
and
Fvars_eqv[*simp*]: $\bigwedge\varphi$ χ. $\varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow$
Fvars (*eqv* φ χ) = *Fvars* φ ∪ *Fvars* χ
and
subst_eqv[*simp*]:
$\bigwedge\varphi$ χ *t x*. $\varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow$
subst (*eqv* φ χ) *t x* = *eqv* (*subst* φ *t x*) (*subst* χ *t x*)
⟨*proof*⟩

lemma *subst_all_idle*[*simp*]:
assumes [*simp*]: $x \in var$ $\varphi \in fmla$ $t \in trm$
shows *subst* (*all x* φ) *t x* = *all x* φ
⟨*proof*⟩

lemma *subst_exi_idle*[*simp*]:
assumes [*simp*]: $x \in var$ $\varphi \in fmla$ $t \in trm$
shows *subst* (*exi x* φ) *t x* = *exi x* φ
⟨*proof*⟩

Parallel substitution versus connectives and quantifiers.

lemma *rawpsubst_cnj*:
assumes $\varphi 1 \in fmla$ $\varphi 2 \in fmla$
and *snd* ' (*set txs*) ⊆ *var* *fst* ' (*set txs*) ⊆ *trm*
shows *rawpsubst* (*cnj* $\varphi 1$ $\varphi 2$) *txs* = *cnj* (*rawpsubst* $\varphi 1$ *txs*) (*rawpsubst* $\varphi 2$ *txs*)
⟨*proof*⟩

lemma *psubst_cnj*[*simp*]:
assumes $\varphi 1 \in fmla$ $\varphi 2 \in fmla$
and *snd* ' (*set txs*) ⊆ *var* *fst* ' (*set txs*) ⊆ *trm*
and *distinct* (*map snd txs*)
shows *psubst* (*cnj* $\varphi 1$ $\varphi 2$) *txs* = *cnj* (*psubst* $\varphi 1$ *txs*) (*psubst* $\varphi 2$ *txs*)
⟨*proof*⟩

lemma *rawpsubst_imp*:
assumes $\varphi 1 \in fmla$ $\varphi 2 \in fmla$
and *snd* ' (*set txs*) ⊆ *var* *fst* ' (*set txs*) ⊆ *trm*
shows *rawpsubst* (*imp* $\varphi 1$ $\varphi 2$) *txs* = *imp* (*rawpsubst* $\varphi 1$ *txs*) (*rawpsubst* $\varphi 2$ *txs*)
⟨*proof*⟩

lemma *psubst_imp*[*simp*]:
assumes $\varphi 1 \in fmla$ $\varphi 2 \in fmla$
and *snd* ' (*set txs*) ⊆ *var* *fst* ' (*set txs*) ⊆ *trm*
and *distinct* (*map snd txs*)
shows *psubst* (*imp* $\varphi 1$ $\varphi 2$) *txs* = *imp* (*psubst* $\varphi 1$ *txs*) (*psubst* $\varphi 2$ *txs*)
⟨*proof*⟩

lemma *rawpsubst_eqv*:
assumes $\varphi 1 \in fmla$ $\varphi 2 \in fmla$
and $snd\ `\ (set\ txs) \subseteq var$ $fst\ `\ (set\ txs) \subseteq trm$
shows $rawpsubst\ (eqv\ \varphi 1\ \varphi 2)\ txs = eqv\ (rawpsubst\ \varphi 1\ txs)\ (rawpsubst\ \varphi 2\ txs)$
⟨*proof*⟩

lemma *psubst_eqv*[*simp*]:
assumes $\varphi 1 \in fmla$ $\varphi 2 \in fmla$
and $snd\ `\ (set\ txs) \subseteq var$ $fst\ `\ (set\ txs) \subseteq trm$
and $distinct\ (map\ snd\ txs)$
shows $psubst\ (eqv\ \varphi 1\ \varphi 2)\ txs = eqv\ (psubst\ \varphi 1\ txs)\ (psubst\ \varphi 2\ txs)$
⟨*proof*⟩

lemma *rawpsubst_all*:
assumes $\varphi \in fmla$ $y \in var$
and $snd\ `\ (set\ txs) \subseteq var$ $fst\ `\ (set\ txs) \subseteq trm$
and $y \notin snd\ `\ (set\ txs)$ $y \notin \bigcup\ (FvarsT\ `\ fst\ `\ (set\ txs))$
shows $rawpsubst\ (all\ y\ \varphi)\ txs = all\ y\ (rawpsubst\ \varphi\ txs)$
⟨*proof*⟩

lemma *psubst_all*[*simp*]:
assumes $\varphi \in fmla$ $y \in var$
and $snd\ `\ (set\ txs) \subseteq var$ $fst\ `\ (set\ txs) \subseteq trm$
and $y \notin snd\ `\ (set\ txs)$ $y \notin \bigcup\ (FvarsT\ `\ fst\ `\ (set\ txs))$
and $distinct\ (map\ snd\ txs)$
shows $psubst\ (all\ y\ \varphi)\ txs = all\ y\ (psubst\ \varphi\ txs)$
⟨*proof*⟩

lemma *rawpsubst_exi*:
assumes $\varphi \in fmla$ $y \in var$
and $snd\ `\ (set\ txs) \subseteq var$ $fst\ `\ (set\ txs) \subseteq trm$
and $y \notin snd\ `\ (set\ txs)$ $y \notin \bigcup\ (FvarsT\ `\ fst\ `\ (set\ txs))$
shows $rawpsubst\ (exi\ y\ \varphi)\ txs = exi\ y\ (rawpsubst\ \varphi\ txs)$
⟨*proof*⟩

lemma *psubst_exi*[*simp*]:
assumes $\varphi \in fmla$ $y \in var$
and $snd\ `\ (set\ txs) \subseteq var$ $fst\ `\ (set\ txs) \subseteq trm$
and $y \notin snd\ `\ (set\ txs)$ $y \notin \bigcup\ (FvarsT\ `\ fst\ `\ (set\ txs))$
and $distinct\ (map\ snd\ txs)$
shows $psubst\ (exi\ y\ \varphi)\ txs = exi\ y\ (psubst\ \varphi\ txs)$
⟨*proof*⟩

end — context *Syntax_with_Connectives*

locale *Syntax_with_Numerals_and_Connectives* =
Syntax_with_Numerals
var trm fmla Var FvarsT substT Fvars subst
num
+
Syntax_with_Connectives
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
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*
begin

lemma *subst_all_num*[*simp*]:
assumes $\varphi \in fmla\ x \in var\ y \in var\ n \in num$
shows $x \neq y \Longrightarrow subst\ (all\ x\ \varphi)\ n\ y = all\ x\ (subst\ \varphi\ n\ y)$
⟨*proof*⟩

lemma *subst_exi_num*[*simp*]:
assumes $\varphi \in fmla\ x \in var\ y \in var\ n \in num$
shows $x \neq y \Longrightarrow subst\ (exi\ x\ \varphi)\ n\ y = exi\ x\ (subst\ \varphi\ n\ y)$
⟨*proof*⟩

The "soft substitution" function:

definition *softSubst* :: *'fmla* ⇒ *'trm* ⇒ *'var* ⇒ *'fmla* **where**
$softSubst\ \varphi\ t\ x = exi\ x\ (cnj\ (eql\ (Var\ x)\ t)\ \varphi)$

lemma *softSubst*[*simp*,*intro*]: $\varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow softSubst\ \varphi\ t\ x \in fmla$
⟨*proof*⟩

lemma *Fvars_softSubst*[*simp*]:
$\varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow$
$Fvars\ (softSubst\ \varphi\ t\ x) = (Fvars\ \varphi \cup FvarsT\ t - \{x\})$
⟨*proof*⟩

lemma *Fvars_softSubst_subst_in*:
$\varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow x \notin FvarsT\ t \Longrightarrow x \in Fvars\ \varphi \Longrightarrow$
$Fvars\ (softSubst\ \varphi\ t\ x) = Fvars\ (subst\ \varphi\ t\ x)$
⟨*proof*⟩

lemma *Fvars_softSubst_subst_notIn*:
$\varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow x \notin FvarsT\ t \Longrightarrow x \notin Fvars\ \varphi \Longrightarrow$
$Fvars\ (softSubst\ \varphi\ t\ x) = Fvars\ (subst\ \varphi\ t\ x) \cup FvarsT\ t$
⟨*proof*⟩

end — context *Syntax_with_Connectives*

The addition of False among logical connectives

locale *Syntax_with_Connectives_False* =
Syntax_with_Connectives
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
+
fixes *fls*::*'fmla*
assumes
fls[*simp*,*intro*!]: $fls \in fmla$
and
Fvars_fls[*simp*,*intro*!]: $Fvars\ fls = \{\}$
and
subst_fls[*simp*]:
$\bigwedge t\ x.\ t \in trm \Longrightarrow x \in var \Longrightarrow subst\ fls\ t\ x = fls$
begin

Negation as a derrived connective:

definition *neg* :: *'fmla* ⇒ *'fmla* **where**
neg φ = *imp φ fls*

lemma
neg[*simp*]: ⋀*φ*. *φ* ∈ *fmla* ⟹ *neg φ* ∈ *fmla*
and
Fvars_neg[*simp*]: ⋀*φ*. *φ* ∈ *fmla* ⟹ *Fvars* (*neg φ*) = *Fvars φ*
and
subst_neg[*simp*]:
⋀*φ t x*. *φ* ∈ *fmla* ⟹ *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹
subst (*neg φ*) *t x* = *neg* (*subst φ t x*)
⟨*proof*⟩

True as a derived connective:

definition *tru* **where** *tru* = *neg fls*

lemma
tru[*simp*,*intro*!]: *tru* ∈ *fmla*
and
Fvars_tru[*simp*]: *Fvars tru* = {}
and
subst_tru[*simp*]: ⋀ *t x*. *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹ *subst tru t x* = *tru*
⟨*proof*⟩

2.3.1 Iterated conjunction

First we define list-based conjunction:

fun *lcnj* :: *'fmla list* ⇒ *'fmla* **where**
lcnj [] = *tru*
| *lcnj* (*φ* # *φs*) = *cnj φ* (*lcnj φs*)

lemma *lcnj*[*simp*,*intro*!]: *set φs* ⊆ *fmla* ⟹ *lcnj φs* ∈ *fmla*
⟨*proof*⟩

lemma *Fvars_lcnj*[*simp*]:
set φs ⊆ *fmla* ⟹ *finite F* ⟹ *Fvars* (*lcnj φs*) = ⋃ (*set* (*map Fvars φs*))
⟨*proof*⟩

lemma *subst_lcnj*[*simp*]:
set φs ⊆ *fmla* ⟹ *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹
subst (*lcnj φs*) *t x* = *lcnj* (*map* (*λφ*. *subst φ t x*) *φs*)
⟨*proof*⟩

Then we define (finite-)set-based conjunction:

definition *scnj* :: *'fmla set* ⇒ *'fmla* **where**
scnj F = *lcnj* (*asList F*)

lemma *scnj*[*simp*,*intro*!]: *F* ⊆ *fmla* ⟹ *finite F* ⟹ *scnj F* ∈ *fmla*
⟨*proof*⟩

lemma *Fvars_scnj*[*simp*]:
F ⊆ *fmla* ⟹ *finite F* ⟹*Fvars* (*scnj F*) = ⋃ (*Fvars* ‘ *F*)
⟨*proof*⟩

2.3.2 Parallel substitution versus the new connectives

lemma *rawpsubst_fls*:

snd ' (set txs) ⊆ var ⟹ fst ' (set txs) ⊆ trm ⟹ rawpsubst fls txs = fls
⟨*proof*⟩

lemma *psubst_fls*[*simp*]:
assumes *snd ' (set txs) ⊆ var* **and** *fst ' (set txs) ⊆ trm*
shows *psubst fls txs = fls*
⟨*proof*⟩

lemma *psubst_neg*[*simp*]:
assumes *φ ∈ fmla*
and *snd ' (set txs) ⊆ var fst ' (set txs) ⊆ trm*
and *distinct (map snd txs)*
shows *psubst (neg φ) txs = neg (psubst φ txs)*
⟨*proof*⟩

lemma *psubst_tru*[*simp*]:
assumes *snd ' (set txs) ⊆ var* **and** *fst ' (set txs) ⊆ trm*
and *distinct (map snd txs)*
shows *psubst tru txs = tru*
⟨*proof*⟩

lemma *psubst_lcnj*[*simp*]:
set φs ⊆ fmla ⟹ snd ' (set txs) ⊆ var ⟹ fst ' (set txs) ⊆ trm ⟹
distinct (map snd txs) ⟹
psubst (lcnj φs) txs = lcnj (map (λφ. psubst φ txs) φs)
⟨*proof*⟩

end — context *Syntax_with_Connectives_False*

2.4 Adding Disjunction

NB: In intuitionistic logic, disjunction is not definable from the other connectives.

locale *Syntax_with_Connectives_False_Disj* =
Syntax_with_Connectives_False
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
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*
+
fixes *dsj* :: *'fmla ⇒ 'fmla ⇒ 'fmla*
assumes
dsj[*simp*]: *⋀φ χ. φ ∈ fmla ⟹ χ ∈ fmla ⟹ dsj φ χ ∈ fmla*
and
Fvars_dsj[*simp*]: *⋀φ χ. φ ∈ fmla ⟹ χ ∈ fmla ⟹*
Fvars (dsj φ χ) = Fvars φ ∪ Fvars χ
and
subst_dsj[*simp*]:
⋀ x φ χ t. φ ∈ fmla ⟹ χ ∈ fmla ⟹ t ∈ trm ⟹ x ∈ var ⟹
subst (dsj φ χ) t x = dsj (subst φ t x) (subst χ t x)
begin

2.4.1 Iterated disjunction

First we define list-based disjunction:

fun *ldsj* :: *'fmla list* ⇒ *'fmla* **where**
  ldsj [] = *fls*
| *ldsj* (*φ* # *φs*) = *dsj φ* (*ldsj φs*)

lemma *ldsj*[*simp*,*intro*!]: *set φs* ⊆ *fmla* ⟹ *ldsj φs* ∈ *fmla*
  ⟨*proof*⟩

lemma *Fvars_ldsj*[*simp*]:
  set φs ⊆ *fmla* ⟹ *Fvars* (*ldsj φs*) = ⋃ (*set* (*map Fvars φs*))
  ⟨*proof*⟩

lemma *subst_ldsj*[*simp*]:
  set φs ⊆ *fmla* ⟹ *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹
 subst (*ldsj φs*) *t x* = *ldsj* (*map* (λ*φ*. *subst φ t x*) *φs*)
  ⟨*proof*⟩

Then we define (finite-)set-based disjunction:

definition *sdsj* :: *'fmla set* ⇒ *'fmla* **where**
  sdsj F = *ldsj* (*asList F*)

lemma *sdsj*[*simp*,*intro*!]: *F* ⊆ *fmla* ⟹ *finite F* ⟹ *sdsj F* ∈ *fmla*
  ⟨*proof*⟩

lemma *Fvars_sdsj*[*simp*]:
  F ⊆ *fmla* ⟹ *finite F* ⟹ *Fvars* (*sdsj F*) = ⋃ (*Fvars* ‘ *F*)
  ⟨*proof*⟩

2.4.2 Parallel substitution versus the new connectives

lemma *rawpsubst_dsj*:
  assumes *φ1* ∈ *fmla φ2* ∈ *fmla*
    and *snd* ‘ (*set txs*) ⊆ *var fst* ‘ (*set txs*) ⊆ *trm*
  shows *rawpsubst* (*dsj φ1 φ2*) *txs* = *dsj* (*rawpsubst φ1 txs*) (*rawpsubst φ2 txs*)
  ⟨*proof*⟩

lemma *psubst_dsj*[*simp*]:
  assumes *φ1* ∈ *fmla φ2* ∈ *fmla*
    and *snd* ‘ (*set txs*) ⊆ *var fst* ‘ (*set txs*) ⊆ *trm*
    and *distinct* (*map snd txs*)
  shows *psubst* (*dsj φ1 φ2*) *txs* = *dsj* (*psubst φ1 txs*) (*psubst φ2 txs*)
⟨*proof*⟩

lemma *psubst_ldsj*[*simp*]:
  set φs ⊆ *fmla* ⟹ *snd* ‘ (*set txs*) ⊆ *var* ⟹ *fst* ‘ (*set txs*) ⊆ *trm* ⟹
 distinct (*map snd txs*) ⟹
 psubst (*ldsj φs*) *txs* = *ldsj* (*map* (λ*φ*. *psubst φ txs*) *φs*)
  ⟨*proof*⟩

end — context *Syntax_with_Connectives_False_Disj*

2.5 Adding an Ordering-Like Formula

locale *Syntax_with_Numerals_and_Connectives_False_Disj* =
  Syntax_with_Connectives_False_Disj

var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
+
Syntax_with_Numerals_and_Connectives
var trm fmla Var FvarsT substT Fvars subst
num
eql cnj imp all exi
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 in addition a formula expressing order (think: less than or equal to)

locale *Syntax_PseudoOrder* =
Syntax_with_Numerals_and_Connectives_False_Disj
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
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*
+
fixes
— Lq is a formula with free variables xx yy:
Lq :: *'fmla*
assumes
Lq[*simp*,*intro*!]: *Lq* ∈ *fmla*
and
Fvars_Lq[*simp*]: *Fvars Lq* = {*zz*,*yy*}
begin

definition *LLq* **where** *LLq t1 t2* = *psubst Lq* [(*t1*,*zz*), (*t2*,*yy*)]

lemma *LLq_def2*: *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹ *yy* ∉ *FvarsT t1* ⟹
LLq t1 t2 = *subst* (*subst Lq t1 zz*) *t2 yy*
⟨*proof*⟩

lemma *LLq*[*simp*,*intro*]:
assumes *t1* ∈ *trm t2* ∈ *trm*
shows *LLq t1 t2* ∈ *fmla*
⟨*proof*⟩

lemma *LLq2*[*simp*,*intro*!]:
n ∈ *num* ⟹ *LLq n* (*Var yy'*) ∈ *fmla*
⟨*proof*⟩

lemma *Fvars_LLq*[*simp*]: *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹ *yy* ∉ *FvarsT t1* ⟹

Fvars (LLq t1 t2) = FvarsT t1 ∪ FvarsT t2
⟨*proof*⟩

lemma *LLq_simps*[*simp*]:
m ∈ num ⟹ n ∈ num ⟹ subst (LLq m (Var yy)) n yy = LLq m n
m ∈ num ⟹ n ∈ num ⟹ subst (LLq m (Var yy′)) n yy = LLq m (Var yy′)
m ∈ num ⟹ subst (LLq m (Var yy′)) (Var yy) yy′ = LLq m (Var yy)
n ∈ num ⟹ subst (LLq (Var xx) (Var yy)) n xx = LLq n (Var yy)
n ∈ num ⟹ subst (LLq (Var zz) (Var yy)) n yy = LLq (Var zz) n
m ∈ num ⟹ subst (LLq (Var zz) (Var yy)) m zz = LLq m (Var yy)
m ∈ num ⟹ n ∈ num ⟹ subst (LLq (Var zz) n) m xx = LLq (Var zz) n
⟨*proof*⟩

end — context *Syntax_PseudoOrder*

2.6 Allowing the Renaming of Quantified Variables

So far, we did not need any renaming axiom for the quantifiers. However, our axioms for substitution implicitly assume the irrelevance of the bound names; in other words, their usual instances would have this property; and since this assumption greatly simplifies the formal development, we make it at this point.

locale *Syntax_with_Connectives_Rename* =
Syntax_with_Connectives
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
+
assumes *all_rename*:
⋀*φ x y. φ ∈ fmla ⟹ x ∈ var ⟹ y ∈ var ⟹ y ∉ Fvars φ ⟹*
all x φ = all y (subst φ (Var y) x)
and *exi_rename*:
⋀*φ x y. φ ∈ fmla ⟹ x ∈ var ⟹ y ∈ var ⟹ y ∉ Fvars φ ⟹*
exi x φ = exi y (subst φ (Var y) x)
begin

lemma *all_rename2*:
φ ∈ fmla ⟹ x ∈ var ⟹ y ∈ var ⟹ (y = x ∨ y ∉ Fvars φ) ⟹
all x φ = all y (subst φ (Var y) x)
⟨*proof*⟩

lemma *exi_rename2*:
φ ∈ fmla ⟹ x ∈ var ⟹ y ∈ var ⟹ (y = x ∨ y ∉ Fvars φ) ⟹
exi x φ = exi y (subst φ (Var y) x)
⟨*proof*⟩

2.7 The Exists-Unique Quantifier

It is phrased in such a way as to avoid substitution:

definition *exu* :: *'var ⇒ 'fmla ⇒ 'fmla* **where**
exu x φ ≡ let y = getFr [*x*] [] [*φ*] *in*
cnj (exi x φ) (exi y (all x (imp φ (eql (Var x) (Var y)))))

lemma *exu*[*simp*,*intro*]:
$x \in var \Longrightarrow \varphi \in fmla \Longrightarrow exu\ x\ \varphi \in fmla$
⟨*proof*⟩

lemma *Fvars_exu*[*simp*]:
$x \in var \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ (exu\ x\ \varphi) = Fvars\ \varphi - \{x\}$
⟨*proof*⟩

lemma *exu_def_var*:
assumes [*simp*]: $x \in var\ y \in var\ y \neq x\ y \notin Fvars\ \varphi\ \varphi \in fmla$
shows
$exu\ x\ \varphi = cnj\ (exi\ x\ \varphi)\ (exi\ y\ (all\ x\ (imp\ \varphi\ (eql\ (Var\ x)\ (Var\ y)))))$
⟨*proof*⟩

lemma *subst_exu*[*simp*]:
assumes [*simp*]: $\varphi \in fmla\ t \in trm\ x \in var\ y \in var\ x \neq y\ x \notin FvarsT\ t$
shows $subst\ (exu\ x\ \varphi)\ t\ y = exu\ x\ (subst\ \varphi\ t\ y)$
⟨*proof*⟩

lemma *subst_exu_idle*[*simp*]:
assumes [*simp*]: $x \in var\ \varphi \in fmla\ t \in trm$
shows $subst\ (exu\ x\ \varphi)\ t\ x = exu\ x\ \varphi$
⟨*proof*⟩

lemma *exu_rename*:
assumes [*simp*]: $\varphi \in fmla\ x \in var\ y \in var\ y \notin Fvars\ \varphi$
shows $exu\ x\ \varphi = exu\ y\ (subst\ \varphi\ (Var\ y)\ x)$
⟨*proof*⟩

lemma *exu_rename2*:
$\varphi \in fmla \Longrightarrow x \in var \Longrightarrow y \in var \Longrightarrow (y = x \vee y \notin Fvars\ \varphi) \Longrightarrow$
$exu\ x\ \varphi = exu\ y\ (subst\ \varphi\ (Var\ y)\ x)$
⟨*proof*⟩

end — context *Syntax_with_Connectives_Rename*

Chapter 3

Deduction

We formalize deduction in a logical system that (shallowly) embeds intuitionistic logic connectives and quantifiers over a signature containing the numerals.

3.1 Positive Logic Deduction

locale *Deduct* =
Syntax_with_Numerals_and_Connectives
  var trm fmla Var FvarsT substT Fvars subst
  num
  eql cnj imp all exi
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*
+
fixes
— Provability of numeric formulas:
prv :: *'fmla* ⇒ *bool*
— Hilbert-style system for intuitionistic logic over ⟶,∧,∀,∃. (⊥, ¬ and ∨ will be included later.) Hilbert-style is preferred since it requires the least amount of infrastructure. (Later, natural deduction rules will also be defined.)
assumes
— Propositional rules and axioms. There is a single propositional rule, modus ponens.
— The modus ponens rule:
prv_imp_mp:
⋀ *φ χ*. *φ* ∈ *fmla* ⟹ *χ* ∈ *fmla* ⟹
  prv (*imp φ χ*) ⟹ *prv φ* ⟹ *prv χ*
and
— The propositional intuitionitic axioms:
prv_imp_imp_triv:
⋀*φ χ*. *φ* ∈ *fmla* ⟹ *χ* ∈ *fmla* ⟹
  prv (*imp φ* (*imp χ φ*))
and
prv_imp_trans:
⋀ *φ χ ψ*. *φ* ∈ *fmla* ⟹ *χ* ∈ *fmla* ⟹ *ψ* ∈ *fmla* ⟹
 prv (*imp* (*imp φ* (*imp χ ψ*))
        (*imp* (*imp φ χ*) (*imp φ ψ*)))
and
prv_imp_cnjL:
⋀ *φ χ*. *φ* ∈ *fmla* ⟹ *χ* ∈ *fmla* ⟹

prv (*imp* (*cnj* φ χ) φ)
and
prv_imp_cnjR:
$\bigwedge \varphi\ \chi.\ \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow$
prv (*imp* (*cnj* φ χ) χ)
and
prv_imp_cnjI:
$\bigwedge \varphi\ \chi.\ \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow$
prv (*imp* φ (*imp* χ (*cnj* φ χ)))
and

— Predicate calculus (quantifier) rules and axioms
— The rules of universal and existential generalization:
prv_all_imp_gen:
$\bigwedge x\ \varphi\ \chi.\ x \notin Fvars\ \varphi \Longrightarrow prv\ (imp\ \varphi\ \chi) \Longrightarrow prv\ (imp\ \varphi\ (all\ x\ \chi))$
and
prv_exi_imp_gen:
$\bigwedge x\ \varphi\ \chi.\ x \in var \Longrightarrow \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow$
$x \notin Fvars\ \chi \Longrightarrow prv\ (imp\ \varphi\ \chi) \Longrightarrow prv\ (imp\ (exi\ x\ \varphi)\ \chi)$
and
— Two quantifier instantiation axioms:
prv_all_inst:
$\bigwedge x\ \varphi\ t.$
$x \in var \Longrightarrow \varphi \in fmla \Longrightarrow t \in trm \Longrightarrow$
prv (*imp* (*all* x φ) (*subst* φ t x))
and
prv_exi_inst:
$\bigwedge x\ \varphi\ t.$
$x \in var \Longrightarrow \varphi \in fmla \Longrightarrow t \in trm \Longrightarrow$
prv (*imp* (*subst* φ t x) (*exi* x φ))
and
— The equality axioms:
prv_eql_refl:
$\bigwedge x.\ x \in var \Longrightarrow$
prv (*eql* (*Var* x) (*Var* x))
and
prv_eql_subst:
$\bigwedge \varphi\ x\ y.$
$x \in var \Longrightarrow y \in var \Longrightarrow \varphi \in fmla \Longrightarrow$
prv ((*imp* (*eql* (*Var* x) (*Var* y))
(*imp* φ (*subst* φ (*Var* y) x))))
begin

3.1.1 Properties of the propositional fragment

lemma *prv_imp_triv*:
assumes *phi*: $\varphi \in fmla$ **and** *psi*: $\psi \in fmla$
shows *prv* $\psi \Longrightarrow$ *prv* (*imp* φ ψ)
⟨*proof*⟩

lemma *prv_imp_refl*:
assumes *phi*: $\varphi \in fmla$
shows *prv* (*imp* φ φ)
⟨*proof*⟩

lemma *prv_imp_refl2*: $\varphi \in fmla \Longrightarrow \psi \in fmla \Longrightarrow \varphi = \psi \Longrightarrow$ *prv* (*imp* φ ψ)
⟨*proof*⟩

lemma *prv_cnjI*:
assumes *phi*: $\varphi \in fmla$ **and** *chi*: $\chi \in fmla$
shows $prv\ \varphi \Longrightarrow prv\ \chi \Longrightarrow prv\ (cnj\ \varphi\ \chi)$
⟨*proof*⟩

lemma *prv_cnjEL*:
assumes *phi*: $\varphi \in fmla$ **and** *chi*: $\chi \in fmla$
shows $prv\ (cnj\ \varphi\ \chi) \Longrightarrow prv\ \varphi$
⟨*proof*⟩

lemma *prv_cnjER*:
assumes *phi*: $\varphi \in fmla$ **and** *chi*: $\chi \in fmla$
shows $prv\ (cnj\ \varphi\ \chi) \Longrightarrow prv\ \chi$
⟨*proof*⟩

lemma *prv_prv_imp_trans*:
assumes *phi*: $\varphi \in fmla$ **and** *chi*: $\chi \in fmla$ **and** *psi*: $\psi \in fmla$
assumes *1*: $prv\ (imp\ \varphi\ \chi)$ **and** *2*: $prv\ (imp\ \chi\ \psi)$
shows $prv\ (imp\ \varphi\ \psi)$
⟨*proof*⟩

lemma *prv_imp_trans1*:
assumes *phi*: $\varphi \in fmla$ **and** *chi*: $\chi \in fmla$ **and** *psi*: $\psi \in fmla$
shows $prv\ (imp\ (imp\ \chi\ \psi)\ (imp\ (imp\ \varphi\ \chi)\ (imp\ \varphi\ \psi)))$
⟨*proof*⟩

lemma *prv_imp_com*:
assumes *phi*: $\varphi \in fmla$ **and** *chi*: $\chi \in fmla$ **and** *psi*: $\psi \in fmla$
assumes $prv\ (imp\ \varphi\ (imp\ \chi\ \psi))$
shows $prv\ (imp\ \chi\ (imp\ \varphi\ \psi))$
⟨*proof*⟩

lemma *prv_imp_trans2*:
assumes *phi*: $\varphi \in fmla$ **and** *chi*: $\chi \in fmla$ **and** *psi*: $\psi \in fmla$
shows $prv\ (imp\ (imp\ \varphi\ \chi)\ (imp\ (imp\ \chi\ \psi)\ (imp\ \varphi\ \psi)))$
⟨*proof*⟩

lemma *prv_imp_cnj*:
assumes $\varphi \in fmla$ **and** $\chi \in fmla$ **and** $\psi \in fmla$
shows $prv\ (imp\ \varphi\ \psi) \Longrightarrow prv\ (imp\ \varphi\ \chi) \Longrightarrow prv\ (imp\ \varphi\ (cnj\ \psi\ \chi))$
⟨*proof*⟩

lemma *prv_imp_imp_com*:
assumes $\varphi \in fmla$ **and** $\chi \in fmla$ **and** $\psi \in fmla$
shows
$prv\ (imp\ (imp\ \varphi\ (imp\ \chi\ \psi))$
$\quad (imp\ \chi\ (imp\ \varphi\ \psi)))$
⟨*proof*⟩

lemma *prv_cnj_imp_monoR2*:
assumes $\varphi \in fmla$ **and** $\chi \in fmla$ **and** $\psi \in fmla$
assumes $prv\ (imp\ \varphi\ (imp\ \chi\ \psi))$
shows $prv\ (imp\ (cnj\ \varphi\ \chi)\ \psi)$
⟨*proof*⟩

lemma *prv_imp_imp_imp_cnj*:
assumes $\varphi \in fmla$ **and** $\chi \in fmla$ **and** $\psi \in fmla$
shows

prv (*imp* (*imp* φ (*imp* χ ψ))
(*imp* (*cnj* φ χ) ψ))
⟨*proof*⟩

lemma *prv_imp_cnj_imp*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla* **and** $\psi \in$ *fmla*
shows
prv (*imp* (*imp* (*cnj* φ χ) ψ)
(*imp* φ (*imp* χ ψ)))
⟨*proof*⟩

lemma *prv_cnj_imp*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla* **and** $\psi \in$ *fmla*
assumes *prv* (*imp* (*cnj* φ χ) ψ)
shows *prv* (*imp* φ (*imp* χ ψ))
⟨*proof*⟩

Monotonicy of conjunction w.r.t. implication:

lemma *prv_cnj_imp_monoR*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla* **and** $\psi \in$ *fmla*
shows *prv* (*imp* (*imp* φ χ) (*imp* (*imp* φ ψ) (*imp* φ (*cnj* χ ψ))))
⟨*proof*⟩

lemma *prv_imp_cnj3L*:
assumes $\varphi 1 \in$ *fmla* **and** $\varphi 2 \in$ *fmla* **and** $\chi \in$ *fmla*
shows *prv* (*imp* (*imp* $\varphi 1$ χ) (*imp* (*cnj* $\varphi 1$ $\varphi 2$) χ))
⟨*proof*⟩

lemma *prv_imp_cnj3R*:
assumes $\varphi 1 \in$ *fmla* **and** $\varphi 2 \in$ *fmla* **and** $\chi \in$ *fmla*
shows *prv* (*imp* (*imp* $\varphi 2$ χ) (*imp* (*cnj* $\varphi 1$ $\varphi 2$) χ))
⟨*proof*⟩

lemma *prv_cnj_imp_mono*:
assumes $\varphi 1 \in$ *fmla* **and** $\varphi 2 \in$ *fmla* **and** $\chi 1 \in$ *fmla* **and** $\chi 2 \in$ *fmla*
shows *prv* (*imp* (*imp* $\varphi 1$ $\chi 1$) (*imp* (*imp* $\varphi 2$ $\chi 2$) (*imp* (*cnj* $\varphi 1$ $\varphi 2$) (*cnj* $\chi 1$ $\chi 2$))))
⟨*proof*⟩

lemma *prv_cnj_mono*:
assumes $\varphi 1 \in$ *fmla* **and** $\varphi 2 \in$ *fmla* **and** $\chi 1 \in$ *fmla* **and** $\chi 2 \in$ *fmla*
assumes *prv* (*imp* $\varphi 1$ $\chi 1$) **and** *prv* (*imp* $\varphi 2$ $\chi 2$)
shows *prv* (*imp* (*cnj* $\varphi 1$ $\varphi 2$) (*cnj* $\chi 1$ $\chi 2$))
⟨*proof*⟩

lemma *prv_cnj_imp_monoR4*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla* **and** $\psi 1 \in$ *fmla* **and** $\psi 2 \in$ *fmla*
shows
prv (*imp* (*imp* φ (*imp* χ $\psi 1$))
(*imp* (*imp* φ (*imp* χ $\psi 2$)) (*imp* φ (*imp* χ (*cnj* $\psi 1$ $\psi 2$)))))
⟨*proof*⟩

lemma *prv_imp_cnj4*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla* **and** $\psi 1 \in$ *fmla* **and** $\psi 2 \in$ *fmla*
shows *prv* (*imp* φ (*imp* χ $\psi 1$)) $\Longrightarrow$ *prv* (*imp* φ (*imp* χ $\psi 2$)) $\Longrightarrow$ *prv* (*imp* φ (*imp* χ (*cnj* $\psi 1$ $\psi 2$)))
⟨*proof*⟩

lemma *prv_cnj_imp_monoR5*:
assumes $\varphi 1 \in$ *fmla* **and** $\varphi 2 \in$ *fmla* **and** $\chi 1 \in$ *fmla* **and** $\chi 2 \in$ *fmla*

shows *prv* (*imp* (*imp* $\varphi 1$ $\chi 1$) (*imp* (*imp* $\varphi 2$ $\chi 2$) (*imp* $\varphi 1$ (*imp* $\varphi 2$ (*cnj* $\chi 1$ $\chi 2$)))))
⟨*proof*⟩

lemma *prv_imp_cnj5*:
assumes $\varphi 1 \in$ *fmla* **and** $\varphi 2 \in$ *fmla* **and** $\chi 1 \in$ *fmla* **and** $\chi 2 \in$ *fmla*
assumes *prv* (*imp* $\varphi 1$ $\chi 1$) **and** *prv* (*imp* $\varphi 2$ $\chi 2$)
shows *prv* (*imp* $\varphi 1$ (*imp* $\varphi 2$ (*cnj* $\chi 1$ $\chi 2$)))
⟨*proof*⟩

Properties of formula equivalence:

lemma *prv_eqv_imp*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla*
shows *prv* (*imp* (*eqv* φ χ) (*eqv* χ φ))
⟨*proof*⟩

lemma *prv_eqv_eqv*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla*
shows *prv* (*eqv* (*eqv* φ χ) (*eqv* χ φ))
⟨*proof*⟩

lemma *prv_imp_eqvEL*:
$\varphi 1 \in$ *fmla* $\Longrightarrow$ $\varphi 2 \in$ *fmla* $\Longrightarrow$ *prv* (*eqv* $\varphi 1$ $\varphi 2$) $\Longrightarrow$ *prv* (*imp* $\varphi 1$ $\varphi 2$)
⟨*proof*⟩

lemma *prv_imp_eqvER*:
$\varphi 1 \in$ *fmla* $\Longrightarrow$ $\varphi 2 \in$ *fmla* $\Longrightarrow$ *prv* (*eqv* $\varphi 1$ $\varphi 2$) $\Longrightarrow$ *prv* (*imp* $\varphi 2$ $\varphi 1$)
⟨*proof*⟩

lemma *prv_eqv_imp_trans*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla* **and** $\psi \in$ *fmla*
shows *prv* (*imp* (*eqv* φ χ) (*imp* (*eqv* χ ψ) (*eqv* φ ψ)))
⟨*proof*⟩

lemma *prv_eqv_cnj_trans*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla* **and** $\psi \in$ *fmla*
shows *prv* (*imp* (*cnj* (*eqv* φ χ) (*eqv* χ ψ)) (*eqv* φ ψ))
⟨*proof*⟩

lemma *prv_eqvI*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla*
assumes *prv* (*imp* φ χ) **and** *prv* (*imp* χ φ)
shows *prv* (*eqv* φ χ)
⟨*proof*⟩

Formula equivalence is a congruence (i.e., an equivalence that is compatible with the other connectives):

lemma *prv_eqv_refl*: $\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*eqv* φ φ)
⟨*proof*⟩

lemma *prv_eqv_sym*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla*
shows *prv* (*eqv* φ χ) $\Longrightarrow$ *prv* (*eqv* χ φ)
⟨*proof*⟩

lemma *prv_eqv_trans*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla* **and** $\psi \in$ *fmla*
shows *prv* (*eqv* φ χ) $\Longrightarrow$ *prv* (*eqv* χ ψ) $\Longrightarrow$ *prv* (*eqv* φ ψ)
⟨*proof*⟩

lemma *imp_imp_compat_eqvL*:
assumes $\varphi 1 \in fmla$ **and** $\varphi 2 \in fmla$ **and** $\chi \in fmla$
shows $prv\ (imp\ (eqv\ \varphi 1\ \varphi 2)\ (eqv\ (imp\ \varphi 1\ \chi)\ (imp\ \varphi 2\ \chi)))$
⟨*proof*⟩

lemma *imp_imp_compat_eqvR*:
assumes $\varphi \in fmla$ **and** $\chi 1 \in fmla$ **and** $\chi 2 \in fmla$
shows $prv\ (imp\ (eqv\ \chi 1\ \chi 2)\ (eqv\ (imp\ \varphi\ \chi 1)\ (imp\ \varphi\ \chi 2)))$
⟨*proof*⟩

lemma *imp_imp_compat_eqv*:
assumes $\varphi 1 \in fmla$ **and** $\varphi 2 \in fmla$ **and** $\chi 1 \in fmla$ **and** $\chi 2 \in fmla$
shows $prv\ (imp\ (eqv\ \varphi 1\ \varphi 2)\ (imp\ (eqv\ \chi 1\ \chi 2)\ (eqv\ (imp\ \varphi 1\ \chi 1)\ (imp\ \varphi 2\ \chi 2))))$
⟨*proof*⟩

lemma *imp_compat_eqvL*:
assumes $\varphi 1 \in fmla$ **and** $\varphi 2 \in fmla$ **and** $\chi \in fmla$
assumes $prv\ (eqv\ \varphi 1\ \varphi 2)$
shows $prv\ (eqv\ (imp\ \varphi 1\ \chi)\ (imp\ \varphi 2\ \chi))$
⟨*proof*⟩

lemma *imp_compat_eqvR*:
assumes $\varphi \in fmla$ **and** $\chi 1 \in fmla$ **and** $\chi 2 \in fmla$
assumes $prv\ (eqv\ \chi 1\ \chi 2)$
shows $prv\ (eqv\ (imp\ \varphi\ \chi 1)\ (imp\ \varphi\ \chi 2))$
⟨*proof*⟩

lemma *imp_compat_eqv*:
assumes $\varphi 1 \in fmla$ **and** $\varphi 2 \in fmla$ **and** $\chi 1 \in fmla$ **and** $\chi 2 \in fmla$
assumes $prv\ (eqv\ \varphi 1\ \varphi 2)$ **and** $prv\ (eqv\ \chi 1\ \chi 2)$
shows $prv\ (eqv\ (imp\ \varphi 1\ \chi 1)\ (imp\ \varphi 2\ \chi 2))$
⟨*proof*⟩

lemma *imp_cnj_compat_eqvL*:
assumes $\varphi 1 \in fmla$ **and** $\varphi 2 \in fmla$ **and** $\chi \in fmla$
shows $prv\ (imp\ (eqv\ \varphi 1\ \varphi 2)\ (eqv\ (cnj\ \varphi 1\ \chi)\ (cnj\ \varphi 2\ \chi)))$
⟨*proof*⟩

lemma *imp_cnj_compat_eqvR*:
assumes $\varphi \in fmla$ **and** $\chi 1 \in fmla$ **and** $\chi 2 \in fmla$
shows $prv\ (imp\ (eqv\ \chi 1\ \chi 2)\ (eqv\ (cnj\ \varphi\ \chi 1)\ (cnj\ \varphi\ \chi 2)))$
⟨*proof*⟩

lemma *imp_cnj_compat_eqv*:
assumes $\varphi 1 \in fmla$ **and** $\varphi 2 \in fmla$ **and** $\chi 1 \in fmla$ **and** $\chi 2 \in fmla$
shows $prv\ (imp\ (eqv\ \varphi 1\ \varphi 2)\ (imp\ (eqv\ \chi 1\ \chi 2)\ (eqv\ (cnj\ \varphi 1\ \chi 1)\ (cnj\ \varphi 2\ \chi 2))))$
⟨*proof*⟩

lemma *cnj_compat_eqvL*:
assumes $\varphi 1 \in fmla$ **and** $\varphi 2 \in fmla$ **and** $\chi \in fmla$
assumes $prv\ (eqv\ \varphi 1\ \varphi 2)$
shows $prv\ (eqv\ (cnj\ \varphi 1\ \chi)\ (cnj\ \varphi 2\ \chi))$
⟨*proof*⟩

lemma *cnj_compat_eqvR*:

assumes $\varphi \in fmla$ **and** $\chi 1 \in fmla$ **and** $\chi 2 \in fmla$
assumes $prv\ (eqv\ \chi 1\ \chi 2)$
shows $prv\ (eqv\ (cnj\ \varphi\ \chi 1)\ (cnj\ \varphi\ \chi 2))$
⟨*proof*⟩

lemma *cnj_compat_eqv*:
assumes $\varphi 1 \in fmla$ **and** $\varphi 2 \in fmla$ **and** $\chi 1 \in fmla$ **and** $\chi 2 \in fmla$
assumes $prv\ (eqv\ \varphi 1\ \varphi 2)$ **and** $prv\ (eqv\ \chi 1\ \chi 2)$
shows $prv\ (eqv\ (cnj\ \varphi 1\ \chi 1)\ (cnj\ \varphi 2\ \chi 2))$
⟨*proof*⟩

lemma *prv_eqv_prv*:
assumes $\varphi \in fmla$ **and** $\chi \in fmla$
assumes $prv\ \varphi$ **and** $prv\ (eqv\ \varphi\ \chi)$
shows $prv\ \chi$
⟨*proof*⟩

lemma *prv_eqv_prv_rev*:
assumes $\varphi \in fmla$ **and** $\chi \in fmla$
assumes $prv\ \varphi$ **and** $prv\ (eqv\ \chi\ \varphi)$
shows $prv\ \chi$
⟨*proof*⟩

lemma *prv_imp_eqv_transi*:
assumes $\varphi \in fmla$ **and** $\chi 1 \in fmla$ **and** $\chi 2 \in fmla$
assumes $prv\ (imp\ \varphi\ \chi 1)$ **and** $prv\ (eqv\ \chi 1\ \chi 2)$
shows $prv\ (imp\ \varphi\ \chi 2)$
⟨*proof*⟩

lemma *prv_imp_eqv_transi_rev*:
assumes $\varphi \in fmla$ **and** $\chi 1 \in fmla$ **and** $\chi 2 \in fmla$
assumes $prv\ (imp\ \varphi\ \chi 2)$ **and** $prv\ (eqv\ \chi 1\ \chi 2)$
shows $prv\ (imp\ \varphi\ \chi 1)$
⟨*proof*⟩

lemma *prv_eqv_imp_transi*:
assumes $\varphi 1 \in fmla$ **and** $\varphi 2 \in fmla$ **and** $\chi \in fmla$
assumes $prv\ (eqv\ \varphi 1\ \varphi 2)$ **and** $prv\ (imp\ \varphi 2\ \chi)$
shows $prv\ (imp\ \varphi 1\ \chi)$
⟨*proof*⟩

lemma *prv_eqv_imp_transi_rev*:
assumes $\varphi 1 \in fmla$ **and** $\varphi 2 \in fmla$ **and** $\chi \in fmla$
assumes $prv\ (eqv\ \varphi 1\ \varphi 2)$ **and** $prv\ (imp\ \varphi 1\ \chi)$
shows $prv\ (imp\ \varphi 2\ \chi)$
⟨*proof*⟩

lemma *prv_imp_monoL*: $\varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
$prv\ (imp\ \chi\ \psi) \Longrightarrow prv\ (imp\ (imp\ \varphi\ \chi)\ (imp\ \varphi\ \psi))$
⟨*proof*⟩

lemma *prv_imp_monoR*: $\varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
$prv\ (imp\ \psi\ \chi) \Longrightarrow prv\ (imp\ (imp\ \chi\ \varphi)\ (imp\ \psi\ \varphi))$
⟨*proof*⟩

More properties involving conjunction:

lemma *prv_cnj_com_imp*:
assumes $\varphi[simp]$: $\varphi \in fmla$ **and** $\chi[simp]$: $\chi \in fmla$

shows *prv* (*imp* (*cnj* φ χ) (*cnj* χ φ))
⟨*proof*⟩

lemma *prv_cnj_com*:
assumes φ[*simp*]: $\varphi \in$ *fmla* **and** χ[*simp*]: $\chi \in$ *fmla*
shows *prv* (*eqv* (*cnj* φ χ) (*cnj* χ φ))
⟨*proof*⟩

lemma *prv_cnj_assoc_imp1*:
assumes φ[*simp*]: $\varphi \in$ *fmla* **and** χ[*simp*]: $\chi \in$ *fmla* **and** ψ[*simp*]: $\psi \in$ *fmla*
shows *prv* (*imp* (*cnj* φ (*cnj* χ ψ)) (*cnj* (*cnj* φ χ) ψ))
⟨*proof*⟩

lemma *prv_cnj_assoc_imp2*:
assumes φ[*simp*]: $\varphi \in$ *fmla* **and** χ[*simp*]: $\chi \in$ *fmla* **and** ψ[*simp*]: $\psi \in$ *fmla*
shows *prv* (*imp* (*cnj* (*cnj* φ χ) ψ) (*cnj* φ (*cnj* χ ψ)))
⟨*proof*⟩

lemma *prv_cnj_assoc*:
assumes φ[*simp*]: $\varphi \in$ *fmla* **and** χ[*simp*]: $\chi \in$ *fmla* **and** ψ[*simp*]: $\psi \in$ *fmla*
shows *prv* (*eqv* (*cnj* φ (*cnj* χ ψ)) (*cnj* (*cnj* φ χ) ψ))
⟨*proof*⟩

lemma *prv_cnj_com_imp3*:
assumes *φ1* ∈ *fmla* *φ2* ∈ *fmla* *φ3* ∈ *fmla*
shows *prv* (*imp* (*cnj* *φ1* (*cnj* *φ2* *φ3*))
(*cnj* *φ2* (*cnj* *φ1* *φ3*)))
⟨*proof*⟩

3.1.2 Properties involving quantifiers

Fundamental properties:

lemma *prv_allE*:
assumes $x \in$ *var* **and** $\varphi \in$ *fmla* **and** $t \in$ *trm*
shows *prv* (*all* x φ) $\Longrightarrow$ *prv* (*subst* φ t x)
⟨*proof*⟩

lemma *prv_exiI*:
assumes $x \in$ *var* **and** $\varphi \in$ *fmla* **and** $t \in$ *trm*
shows *prv* (*subst* φ t x) $\Longrightarrow$ *prv* (*exi* x φ)
⟨*proof*⟩

lemma *prv_imp_imp_exi*:
assumes $x \in$ *var* **and** $\varphi \in$ *fmla* **and** $\chi \in$ *fmla*
assumes $x \notin$ *Fvars* φ
shows *prv* (*imp* (*exi* x (*imp* φ χ)) (*imp* φ (*exi* x χ)))
⟨*proof*⟩

lemma *prv_imp_exi*:
assumes $x \in$ *var* **and** $\varphi \in$ *fmla* **and** $\chi \in$ *fmla*
shows $x \notin$ *Fvars* $\varphi \Longrightarrow$ *prv* (*exi* x (*imp* φ χ)) $\Longrightarrow$ *prv* (*imp* φ (*exi* x χ))
⟨*proof*⟩

lemma *prv_exi_imp*:
assumes x: $x \in$ *var* **and** $\varphi \in$ *fmla* **and** $\chi \in$ *fmla*
assumes $x \notin$ *Fvars* χ **and** d: *prv* (*all* x (*imp* φ χ))
shows *prv* (*imp* (*exi* x φ) χ)
⟨*proof*⟩

lemma *prv_all_imp*:
assumes x: $x \in var$ **and** $\varphi \in fmla$ **and** $\chi \in fmla$
assumes $x \notin Fvars\ \varphi$ **and** $prv\ (all\ x\ (imp\ \varphi\ \chi))$
shows $prv\ (imp\ \varphi\ (all\ x\ \chi))$
⟨*proof*⟩

lemma *prv_exi_inst_same*:
assumes $\varphi \in fmla\ \chi \in fmla\ x \in var$
shows $prv\ (imp\ \varphi\ (exi\ x\ \varphi))$
⟨*proof*⟩

lemma *prv_exi_cong*:
assumes $\varphi \in fmla\ \chi \in fmla\ x \in var$
and $prv\ (imp\ \varphi\ \chi)$
shows $prv\ (imp\ (exi\ x\ \varphi)\ (exi\ x\ \chi))$
⟨*proof*⟩

lemma *prv_exi_congW*:
assumes $\varphi \in fmla\ \chi \in fmla\ x \in var$
and $prv\ (imp\ \varphi\ \chi)\ prv\ (exi\ x\ \varphi)$
shows $prv\ (exi\ x\ \chi)$
⟨*proof*⟩

lemma *prv_all_cong*:
assumes $\varphi \in fmla\ \chi \in fmla\ x \in var$
and $prv\ (imp\ \varphi\ \chi)$
shows $prv\ (imp\ (all\ x\ \varphi)\ (all\ x\ \chi))$
⟨*proof*⟩

lemma *prv_all_congW*:
assumes $\varphi \in fmla\ \chi \in fmla\ x \in var$
and $prv\ (imp\ \varphi\ \chi)\ prv\ (all\ x\ \varphi)$
shows $prv\ (all\ x\ \chi)$
⟨*proof*⟩

Quantifiers versus free variables and substitution:

lemma *exists_no_Fvars*: $\exists\ \varphi.\ \varphi \in fmla \wedge prv\ \varphi \wedge Fvars\ \varphi = \{\}$
⟨*proof*⟩

lemma *prv_all_gen*:
assumes $x \in var$ **and** $\varphi \in fmla$
assumes $prv\ \varphi$ **shows** $prv\ (all\ x\ \varphi)$
⟨*proof*⟩

lemma *all_subst_rename_prv*:
$\varphi \in fmla \Longrightarrow x \in var \Longrightarrow y \in var \Longrightarrow$
$y \notin Fvars\ \varphi \Longrightarrow prv\ (all\ x\ \varphi) \Longrightarrow prv\ (all\ y\ (subst\ \varphi\ (Var\ y)\ x))$
⟨*proof*⟩

lemma *allE_id*:
assumes $y \in var$ **and** $\varphi \in fmla$
assumes $prv\ (all\ y\ \varphi)$
shows $prv\ \varphi$
⟨*proof*⟩

lemma *prv_subst*:
assumes $x \in var$ **and** $\varphi \in fmla$ **and** $t \in trm$

shows *prv* $\varphi \Longrightarrow$ *prv* (*subst* φ *t x*)
⟨*proof*⟩

lemma *prv_rawpsubst*:
assumes $\varphi \in$ *fmla* **and** *snd* ‘ (*set txs*) $\subseteq$ *var* **and** *fst* ‘ (*set txs*) $\subseteq$ *trm*
and *prv* φ
shows *prv* (*rawpsubst* φ *txs*)
⟨*proof*⟩

lemma *prv_psubst*:
assumes $\varphi \in$ *fmla* **and** *snd* ‘ (*set txs*) $\subseteq$ *var* **and** *fst* ‘ (*set txs*) $\subseteq$ *trm*
and *prv* φ
shows *prv* (*psubst* φ *txs*)
⟨*proof*⟩

lemma *prv_eqv_rawpsubst*:
$\varphi \in$ *fmla* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$ *snd* ‘ *set txs* $\subseteq$ *var* $\Longrightarrow$ *fst* ‘ *set txs* $\subseteq$ *trm* $\Longrightarrow$ *prv* (*eqv* φ ψ) $\Longrightarrow$
prv (*eqv* (*rawpsubst* φ *txs*) (*rawpsubst* ψ *txs*))
⟨*proof*⟩

lemma *prv_eqv_psubst*:
$\varphi \in$ *fmla* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$ *snd* ‘ *set txs* $\subseteq$ *var* $\Longrightarrow$ *fst* ‘ *set txs* $\subseteq$ *trm* $\Longrightarrow$ *prv* (*eqv* φ ψ) $\Longrightarrow$
distinct (*map snd txs*) $\Longrightarrow$
prv (*eqv* (*psubst* φ *txs*) (*psubst* ψ *txs*))
⟨*proof*⟩

lemma *prv_all_imp_trans*:
assumes $x \in$ *var* **and** $\varphi \in$ *fmla* **and** $\chi \in$ *fmla* **and** $\psi \in$ *fmla*
shows *prv* (*all x* (*imp* φ χ)) $\Longrightarrow$ *prv* (*all x* (*imp* χ ψ)) $\Longrightarrow$ *prv* (*all x* (*imp* φ ψ))
⟨*proof*⟩

lemma *prv_all_imp_cnj*:
assumes $x \in$ *var* **and** $\varphi \in$ *fmla* **and** $\chi \in$ *fmla* **and** $\psi \in$ *fmla*
shows *prv* (*all x* (*imp* φ (*imp* ψ χ))) $\Longrightarrow$ *prv* (*all x* (*imp* (*cnj* ψ φ) χ))
⟨*proof*⟩

lemma *prv_all_imp_cnj_rev*:
assumes $x \in$ *var* **and** $\varphi \in$ *fmla* **and** $\chi \in$ *fmla* **and** $\psi \in$ *fmla*
shows *prv* (*all x* (*imp* (*cnj* φ ψ) χ)) $\Longrightarrow$ *prv* (*all x* (*imp* φ (*imp* ψ χ)))
⟨*proof*⟩

3.1.3 Properties concerning equality

lemma *prv_eql_reflT*:
assumes *t*: $t \in$ *trm*
shows *prv* (*eql t t*)
⟨*proof*⟩

lemma *prv_eql_subst_trm*:
assumes *xx*: $x \in$ *var* **and** φ: $\varphi \in$ *fmla* **and** *t1* $\in$ *trm* **and** *t2* $\in$ *trm*
shows *prv* ((*imp* (*eql t1 t2*)
(*imp* (*subst* φ *t1 x*) (*subst* φ *t2 x*))))
⟨*proof*⟩

lemma *prv_eql_subst_trm2*:
assumes $x \in$ *var* **and** $\varphi \in$ *fmla* **and** *t1* $\in$ *trm* **and** *t2* $\in$ *trm*
assumes *prv* (*eql t1 t2*)
shows *prv* (*imp* (*subst* φ *t1 x*) (*subst* φ *t2 x*))

⟨*proof*⟩

lemma *prv_eql_sym*:
assumes [*simp*]: *t1* ∈ *trm t2* ∈ *trm*
shows *prv* (*imp* (*eql t1 t2*) (*eql t2 t1*))
⟨*proof*⟩

lemma *prv_prv_eql_sym*: *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹ *prv* (*eql t1 t2*) ⟹ *prv* (*eql t2 t1*)
⟨*proof*⟩

lemma *prv_all_eql*:
assumes *x* ∈ *var* **and** *y* ∈ *var* **and** *φ* ∈ *fmla* **and** *t1* ∈ *trm* **and** *t2* ∈ *trm*
shows *prv* (*all x* ((*imp* (*eql t1 t2*)
(*imp* (*subst φ t2 y*) (*subst φ t1 y*)))))
⟨*proof*⟩

lemma *prv_eql_subst_trm_rev*:
assumes *t1* ∈ *trm* **and** *t2* ∈ *trm* **and** *φ* ∈ *fmla* **and** *y* ∈ *var*
shows
prv ((*imp* (*eql t1 t2*)
(*imp* (*subst φ t2 y*) (*subst φ t1 y*))))
⟨*proof*⟩

lemma *prv_eql_subst_trm_rev2*:
assumes *x* ∈ *var* **and** *φ* ∈ *fmla* **and** *t1* ∈ *trm* **and** *t2* ∈ *trm*
assumes *prv* (*eql t1 t2*)
shows *prv* (*imp* (*subst φ t2 x*) (*subst φ t1 x*))
⟨*proof*⟩

lemma *prv_eql_subst_trm_eqv*:
assumes *x* ∈ *var* **and** *φ* ∈ *fmla* **and** *t1* ∈ *trm* **and** *t2* ∈ *trm*
assumes *prv* (*eql t1 t2*)
shows *prv* (*eqv* (*subst φ t1 x*) (*subst φ t2 x*))
⟨*proof*⟩

lemma *prv_eql_subst_trm_id*:
assumes *y* ∈ *var φ* ∈ *fmla* **and** *n* ∈ *num*
shows *prv* (*imp* (*eql* (*Var y*) *n*) (*imp φ* (*subst φ n y*)))
⟨*proof*⟩

lemma *prv_eql_subst_trm_id_back*:
assumes *y* ∈ *var φ* ∈ *fmla* **and** *n* ∈ *num*
shows *prv* (*imp* (*eql* (*Var y*) *n*) (*imp* (*subst φ n y*) *φ*))
⟨*proof*⟩

lemma *prv_eql_subst_trm_id_rev*:
assumes *y* ∈ *var φ* ∈ *fmla* **and** *n* ∈ *num*
shows *prv* (*imp* (*eql n* (*Var y*)) (*imp φ* (*subst φ n y*)))
⟨*proof*⟩

lemma *prv_eql_subst_trm_id_back_rev*:
assumes *y* ∈ *var φ* ∈ *fmla* **and** *n* ∈ *num*
shows *prv* (*imp* (*eql n* (*Var y*)) (*imp* (*subst φ n y*) *φ*))
⟨*proof*⟩

lemma *prv_eql_imp_trans_rev*:
assumes *t1*[*simp*]: *t1* ∈ *trm* **and** *t2*[*simp*]: *t2* ∈ *trm* **and** *t3*[*simp*]: *t3* ∈ *trm*
shows *prv* (*imp* (*eql t1 t2*) (*imp* (*eql t1 t3*) (*eql t2 t3*)))

⟨*proof*⟩

lemma *prv_eql_imp_trans*:
assumes *t1*[*simp*]: *t1* ∈ *trm* **and** *t2*[*simp*]: *t2* ∈ *trm* **and** *t3*[*simp*]: *t3* ∈ *trm*
shows *prv* (*imp* (*eql t1 t2*) (*imp* (*eql t2 t3*) (*eql t1 t3*)))
⟨*proof*⟩

lemma *prv_eql_trans*:
assumes *t1*[*simp*]: *t1* ∈ *trm* **and** *t2*[*simp*]: *t2* ∈ *trm* **and** *t3*[*simp*]: *t3* ∈ *trm*
and *prv* (*eql t1 t2*) **and** *prv* (*eql t2 t3*)
shows *prv* (*eql t1 t3*)
⟨*proof*⟩

3.1.4 The equivalence between soft substitution and substitution

lemma *prv_subst_imp_softSubst*:
assumes [*simp*,*intro*!]: *x* ∈ *var t* ∈ *trm* φ ∈ *fmla x* ∉ *FvarsT t*
shows *prv* (*imp* (*subst* φ *t x*) (*softSubst* φ *t x*))
⟨*proof*⟩

lemma *prv_subst_implies_softSubst*:
assumes *x* ∈ *var t* ∈ *trm* φ ∈ *fmla*
and *x* ∉ *FvarsT t*
and *prv* (*subst* φ *t x*)
shows *prv* (*softSubst* φ *t x*)
⟨*proof*⟩

lemma *prv_softSubst_imp_subst*:
assumes [*simp*,*intro*!]: *x* ∈ *var t* ∈ *trm* φ ∈ *fmla x* ∉ *FvarsT t*
shows *prv* (*imp* (*softSubst* φ *t x*) (*subst* φ *t x*))
⟨*proof*⟩

lemma *prv_softSubst_implies_subst*:
assumes *x* ∈ *var t* ∈ *trm* φ ∈ *fmla*
and *x* ∉ *FvarsT t*
and *prv* (*softSubst* φ *t x*)
shows *prv* (*subst* φ *t x*)
⟨*proof*⟩

lemma *prv_softSubst_eqv_subst*:
assumes [*simp*,*intro*!]: *x* ∈ *var t* ∈ *trm* φ ∈ *fmla x* ∉ *FvarsT t*
shows *prv* (*eqv* (*softSubst* φ *t x*) (*subst* φ *t x*))
⟨*proof*⟩

end — context *Deduct*

3.2 Deduction Considering False

locale *Deduct_with_False* =
Syntax_with_Connectives_False
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
+
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 *eql cnj imp all exi*
and *fls*
and *num*
and *prv*
+
assumes
prv_fls[*simp*,*intro*]: $\bigwedge\varphi$. *prv* (*imp fls* φ)
begin

3.2.1 Basic properties of False (fls)

lemma *prv_expl*:
assumes $\varphi \in$ *fmla*
assumes *prv fls*
shows *prv* φ
⟨*proof*⟩

lemma *prv_cnjR_fls*: $\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*eqv* (*cnj fls* φ) *fls*)
⟨*proof*⟩

lemma *prv_cnjL_fls*: $\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*eqv* (*cnj* φ *fls*) *fls*)
⟨*proof*⟩

3.2.2 Properties involving negation

Recall that negation has been defined from implication and False.

lemma *prv_imp_neg_fls*:
assumes $\varphi \in$ *fmla*
shows *prv* (*imp* φ (*imp* (*neg* φ) *fls*))
⟨*proof*⟩

lemma *prv_neg_fls*:
assumes $\varphi \in$ *fmla*
assumes *prv* φ **and** *prv* (*neg* φ)
shows *prv fls*
⟨*proof*⟩

lemma *prv_imp_neg_neg*:
assumes $\varphi \in$ *fmla*
shows *prv* (*imp* φ (*neg* (*neg* φ)))
⟨*proof*⟩

lemma *prv_neg_neg*:
assumes $\varphi \in$ *fmla*
assumes *prv* φ
shows *prv* (*neg* (*neg* φ))
⟨*proof*⟩

lemma *prv_imp_imp_neg_rev*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla*
shows *prv* (*imp* (*imp* φ χ)
(*imp* (*neg* χ) (*neg* φ)))
⟨*proof*⟩

lemma *prv_imp_neg_rev*:
assumes $\varphi \in fmla$ **and** $\chi \in fmla$
assumes $prv\ (imp\ \varphi\ \chi)$
shows $prv\ (imp\ (neg\ \chi)\ (neg\ \varphi))$
⟨*proof*⟩

lemma *prv_eqv_neg_prv_fls*:
$\varphi \in fmla \Longrightarrow$
$prv\ (eqv\ \varphi\ (neg\ \varphi)) \Longrightarrow prv\ fls$
⟨*proof*⟩

lemma *prv_eqv_eqv_neg_prv_fls*:
$\varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow$
$prv\ (eqv\ \varphi\ \chi) \Longrightarrow prv\ (eqv\ \varphi\ (neg\ \chi)) \Longrightarrow prv\ fls$
⟨*proof*⟩

lemma *prv_eqv_eqv_neg_prv_fls2*:
$\varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow$
$prv\ (eqv\ \varphi\ \chi) \Longrightarrow prv\ (eqv\ \chi\ (neg\ \varphi)) \Longrightarrow prv\ fls$
⟨*proof*⟩

lemma *prv_neg_imp_imp_trans*:
assumes $\varphi \in fmla\ \chi \in fmla\ \ \psi \in fmla$
and $prv\ (neg\ \varphi)$
and $prv\ (imp\ \chi\ (imp\ \psi\ \varphi))$
shows $prv\ (imp\ \chi\ (neg\ \psi))$
⟨*proof*⟩

lemma *prv_imp_neg_imp_neg_imp*:
assumes $\varphi \in fmla\ \chi \in fmla$
and $prv\ (neg\ \varphi)$
shows $prv\ ((imp\ \chi\ (neg\ (imp\ \chi\ \varphi))))$
⟨*proof*⟩

lemma *prv_prv_neg_imp_neg*:
assumes $\varphi \in fmla\ \chi \in fmla$
and $prv\ \varphi$ **and** $prv\ \chi$
shows $prv\ (neg\ (imp\ \varphi\ (neg\ \chi)))$
⟨*proof*⟩

lemma *prv_imp_neg_imp_cnjL*:
assumes $\varphi \in fmla\ \varphi 1 \in fmla\ \varphi 2 \in fmla$
and $prv\ (imp\ \varphi\ (neg\ \varphi 1))$
shows $prv\ (imp\ \varphi\ (neg\ (cnj\ \varphi 1\ \varphi 2)))$
⟨*proof*⟩

lemma *prv_imp_neg_imp_cnjR*:
assumes $\varphi \in fmla\ \varphi 1 \in fmla\ \varphi 2 \in fmla$
and $prv\ (imp\ \varphi\ (neg\ \varphi 2))$
shows $prv\ (imp\ \varphi\ (neg\ (cnj\ \varphi 1\ \varphi 2)))$
⟨*proof*⟩

Negation versus quantifiers:

lemma *prv_all_neg_imp_neg_exi*:
assumes x: $x \in var$ **and** φ: $\varphi \in fmla$
shows $prv\ (imp\ (all\ x\ (neg\ \varphi))\ (neg\ (exi\ x\ \varphi)))$
⟨*proof*⟩

lemma *prv_neg_exi_imp_all_neg*:
assumes x: $x \in var$ **and** φ: $\varphi \in fmla$
shows $prv\ (imp\ (neg\ (exi\ x\ \varphi))\ (all\ x\ (neg\ \varphi)))$
⟨*proof*⟩

lemma *prv_neg_exi_eqv_all_neg*:
assumes x: $x \in var$ **and** φ: $\varphi \in fmla$
shows $prv\ (eqv\ (neg\ (exi\ x\ \varphi))\ (all\ x\ (neg\ \varphi)))$
⟨*proof*⟩

lemma *prv_neg_exi_implies_all_neg*:
assumes x: $x \in var$ **and** φ: $\varphi \in fmla$ **and** $prv\ (neg\ (exi\ x\ \varphi))$
shows $prv\ (all\ x\ (neg\ \varphi))$
⟨*proof*⟩

lemma *prv_neg_neg_exi*:
assumes $x \in var\ \varphi \in fmla$
and $prv\ (neg\ \varphi)$
shows $prv\ (neg\ (exi\ x\ \varphi))$
⟨*proof*⟩

lemma *prv_exi_imp_exiI*:
assumes [*simp*]: $x \in var\ y \in var\ \varphi \in fmla$ **and** *yx*: $y = x \vee y \notin Fvars\ \varphi$
shows $prv\ (imp\ (exi\ x\ \varphi)\ (exi\ y\ (subst\ \varphi\ (Var\ y)\ x)))$
⟨*proof*⟩

lemma *prv_imp_neg_allI*:
assumes $\varphi \in fmla\ \chi \in fmla\ t \in trm\ x \in var$
and $prv\ (imp\ \varphi\ (neg\ (subst\ \chi\ t\ x)))$
shows $prv\ (imp\ \varphi\ (neg\ (all\ x\ \chi)))$
⟨*proof*⟩

lemma *prv_imp_neg_allWI*:
assumes $\chi \in fmla\ t \in trm\ x \in var$
and $prv\ (neg\ (subst\ \chi\ t\ x))$
shows $prv\ (neg\ (all\ x\ \chi))$
⟨*proof*⟩

3.2.3 Properties involving True (tru)

lemma *prv_imp_tru*: $\varphi \in fmla \Longrightarrow prv\ (imp\ \varphi\ tru)$
⟨*proof*⟩

lemma *prv_tru_imp*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ (imp\ tru\ \varphi)\ \varphi)$
⟨*proof*⟩

lemma *prv_fls_neg_tru*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ fls\ (neg\ tru))$
⟨*proof*⟩

lemma *prv_tru_neg_fls*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ tru\ (neg\ fls))$
⟨*proof*⟩

lemma *prv_cnjR_tru*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ (cnj\ tru\ \varphi)\ \varphi)$
⟨*proof*⟩

lemma *prv_cnjL_tru*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ (cnj\ \varphi\ tru)\ \varphi)$
⟨*proof*⟩

3.2.4 Property of set-based conjunctions

These are based on properties of the auxiliary list conjunctions.

lemma *prv_lcnj_imp_in*:
assumes *set φs ⊆ fmla*
and *φ ∈ set φs*
shows *prv (imp (lcnj φs) φ)*
⟨*proof*⟩

lemma *prv_lcnj_imp*:
assumes *χ ∈ fmla* **and** *set φs ⊆ fmla*
and *φ ∈ set φs* **and** *prv (imp φ χ)*
shows *prv (imp (lcnj φs) χ)*
⟨*proof*⟩

lemma *prv_imp_lcnj*:
assumes *χ ∈ fmla* **and** *set φs ⊆ fmla*
and *⋀φ. φ ∈ set φs ⟹ prv (imp χ φ)*
shows *prv (imp χ (lcnj φs))*
⟨*proof*⟩

lemma *prv_lcnj_imp_inner*:
assumes *φ ∈ fmla set φ1s ⊆ fmla set φ2s ⊆ fmla*
shows *prv (imp (cnj φ (lcnj (φ1s @ φ2s))) (lcnj (φ1s @ φ # φ2s)))*
⟨*proof*⟩

lemma *prv_lcnj_imp_remdups*:
assumes *set φs ⊆ fmla*
shows *prv (imp (lcnj (remdups φs)) (lcnj φs))*
⟨*proof*⟩

lemma *prv_lcnj_mono*:
assumes *φ1s*: *set φ1s ⊆ fmla* **and** *set φ2s ⊆ set φ1s*
shows *prv (imp (lcnj φ1s) (lcnj φ2s))*
⟨*proof*⟩

lemma *prv_lcnj_eqv*:
assumes *set φ1s ⊆ fmla* **and** *set φ2s = set φ1s*
shows *prv (eqv (lcnj φ1s) (lcnj φ2s))*
⟨*proof*⟩

lemma *prv_lcnj_mono_imp*:
assumes *set φ1s ⊆ fmla set φ2s ⊆ fmla* **and** *∀ φ2 ∈ set φ2s. ∃ φ1 ∈ set φ1s. prv (imp φ1 φ2)*
shows *prv (imp (lcnj φ1s) (lcnj φ2s))*
⟨*proof*⟩

Set-based conjunction commutes with substitution only up to provably equivalence:

lemma *prv_subst_scnj*:
assumes *F ⊆ fmla finite F t ∈ trm x ∈ var*
shows *prv (eqv (subst (scnj F) t x) (scnj ((λφ. subst φ t x) ‘ F)))*
⟨*proof*⟩

lemma *prv_imp_subst_scnj*:
assumes *F ⊆ fmla finite F t ∈ trm x ∈ var*
shows *prv (imp (subst (scnj F) t x) (scnj ((λφ. subst φ t x) ‘ F)))*
⟨*proof*⟩

lemma *prv_subst_scnj_imp*:

assumes $F \subseteq fmla\ finite\ F\ t \in trm\ x \in var$
shows $prv\ (imp\ (scnj\ ((\lambda\varphi.\ subst\ \varphi\ t\ x)\ `\ F))\ (subst\ (scnj\ F)\ t\ x))$
⟨*proof*⟩

lemma *prv_scnj_imp_in*:
assumes $F \subseteq fmla\ finite\ F$
and $\varphi \in F$
shows $prv\ (imp\ (scnj\ F)\ \varphi)$
⟨*proof*⟩

lemma *prv_scnj_imp*:
assumes $\chi \in fmla$ **and** $F \subseteq fmla\ finite\ F$
and $\varphi \in F$ **and** $prv\ (imp\ \varphi\ \chi)$
shows $prv\ (imp\ (scnj\ F)\ \chi)$
⟨*proof*⟩

lemma *prv_imp_scnj*:
assumes $\chi \in fmla$ **and** $F \subseteq fmla\ finite\ F$
and $\bigwedge\varphi.\ \varphi \in F \Longrightarrow prv\ (imp\ \chi\ \varphi)$
shows $prv\ (imp\ \chi\ (scnj\ F))$
⟨*proof*⟩

lemma *prv_scnj_mono*:
assumes $F1 \subseteq fmla$ **and** $F2 \subseteq F1$ **and** $finite\ F1$
shows $prv\ (imp\ (scnj\ F1)\ (scnj\ F2))$
⟨*proof*⟩

lemma *prv_scnj_mono_imp*:
assumes $F1 \subseteq fmla\ F2 \subseteq fmla\ finite\ F1\ finite\ F2$
and $\forall\ \varphi2 \in F2.\ \exists\ \varphi1 \in F1.\ prv\ (imp\ \varphi1\ \varphi2)$
shows $prv\ (imp\ (scnj\ F1)\ (scnj\ F2))$
⟨*proof*⟩

Commutation with parallel substitution:

lemma *prv_imp_scnj_insert*:
assumes $F \subseteq fmla$ **and** $finite\ F$ **and** $\varphi \in fmla$
shows $prv\ (imp\ (scnj\ (insert\ \varphi\ F))\ (cnj\ \varphi\ (scnj\ F)))$
⟨*proof*⟩

lemma *prv_implies_scnj_insert*:
assumes $F \subseteq fmla$ **and** $finite\ F$ **and** $\varphi \in fmla$
and $prv\ (scnj\ (insert\ \varphi\ F))$
shows $prv\ (cnj\ \varphi\ (scnj\ F))$
⟨*proof*⟩

lemma *prv_imp_cnj_scnj*:
assumes $F \subseteq fmla$ **and** $finite\ F$ **and** $\varphi \in fmla$
shows $prv\ (imp\ (cnj\ \varphi\ (scnj\ F))\ (scnj\ (insert\ \varphi\ F)))$
⟨*proof*⟩

lemma *prv_implies_cnj_scnj*:
assumes $F \subseteq fmla$ **and** $finite\ F$ **and** $\varphi \in fmla$
and $prv\ (cnj\ \varphi\ (scnj\ F))$
shows $prv\ (scnj\ (insert\ \varphi\ F))$
⟨*proof*⟩

lemma *prv_eqv_scnj_insert*:
assumes $F \subseteq fmla$ **and** $finite\ F$ **and** $\varphi \in fmla$

shows *prv* (*eqv* (*scnj* (*insert* φ *F*)) (*cnj* φ (*scnj F*)))
⟨*proof*⟩

lemma *prv_scnj1_imp*:
$\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* (*scnj* {φ}) φ)
⟨*proof*⟩

lemma *prv_imp_scnj1*:
$\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* φ (*scnj* {φ}))
⟨*proof*⟩

lemma *prv_scnj2_imp_cnj*:
$\varphi \in$ *fmla* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* (*scnj* {φ,ψ}) (*cnj* φ ψ))
⟨*proof*⟩

lemma *prv_cnj_imp_scnj2*:
$\varphi \in$ *fmla* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* (*cnj* φ ψ) (*scnj* {φ,ψ}))
⟨*proof*⟩

lemma *prv_imp_imp_scnj2*:
$\varphi \in$ *fmla* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* φ (*imp* ψ (*scnj* {φ,ψ})))
⟨*proof*⟩

lemma *prv_rawpsubst_scnj*:
assumes *F* $\subseteq$ *fmla finite F*
and *snd* ‘ (*set txs*) $\subseteq$ *var fst* ‘ (*set txs*) $\subseteq$ *trm*
shows *prv* (*eqv* (*rawpsubst* (*scnj F*) *txs*) (*scnj* ((λφ. *rawpsubst* φ *txs*) ‘ *F*)))
⟨*proof*⟩

lemma *prv_psubst_scnj*:
assumes *F* $\subseteq$ *fmla finite F*
and *snd* ‘ (*set txs*) $\subseteq$ *var fst* ‘ (*set txs*) $\subseteq$ *trm*
and *distinct* (*map snd txs*)
shows *prv* (*eqv* (*psubst* (*scnj F*) *txs*) (*scnj* ((λφ. *psubst* φ *txs*) ‘ *F*)))
⟨*proof*⟩

lemma *prv_imp_psubst_scnj*:
assumes *F* $\subseteq$ *fmla finite F snd* ‘ *set txs* $\subseteq$ *var fst* ‘ *set txs* $\subseteq$ *trm*
and *distinct* (*map snd txs*)
shows *prv* (*imp* (*psubst* (*scnj F*) *txs*) (*scnj* ((λφ. *psubst* φ *txs*) ‘ *F*)))
⟨*proof*⟩

lemma *prv_psubst_scnj_imp*:
assumes *F* $\subseteq$ *fmla finite F snd* ‘ *set txs* $\subseteq$ *var fst* ‘ *set txs* $\subseteq$ *trm*
and *distinct* (*map snd txs*)
shows *prv* (*imp* (*scnj* ((λφ. *psubst* φ *txs*) ‘ *F*)) (*psubst* (*scnj F*) *txs*))
⟨*proof*⟩

3.2.5 Consistency and ω-consistency

definition *consistent* :: *bool* **where**
consistent $\equiv \neg$ *prv fls*

lemma *consistent_def2*: *consistent* $\longleftrightarrow$ ($\exists\, \varphi \in$ *fmla*. $\neg$ *prv* φ)
⟨*proof*⟩

lemma *consistent_def3*: *consistent* $\longleftrightarrow$ $(\forall\, \varphi \in fmla.\ \neg\ (prv\ \varphi \wedge prv\ (neg\ \varphi)))$
⟨*proof*⟩

definition ω*consistent* :: *bool* **where**
ω*consistent* $\equiv$
$\forall\ \varphi \in fmla.\ \forall\ x \in var.\ Fvars\ \varphi = \{x\} \longrightarrow$
$(\forall\ n \in num.\ prv\ (neg\ (subst\ \varphi\ n\ x)))$
$\longrightarrow$
$\neg\ prv\ (neg\ (neg\ (exi\ x\ \varphi)))$

The above particularly strong version of ω*consistent* is used for the sake of working without assuming classical logic. It of course implies the more standard formulations for classical logic:

definition ω*consistentStd1* :: *bool* **where**
ω*consistentStd1* $\equiv$
$\forall\ \varphi \in fmla.\ \forall\ x \in var.\ Fvars\ \varphi = \{x\} \longrightarrow$
$(\forall\ n \in num.\ prv\ (neg\ (subst\ \varphi\ n\ x))) \longrightarrow \neg\ prv\ (exi\ x\ \varphi)$

definition ω*consistentStd2* :: *bool* **where**
ω*consistentStd2* $\equiv$
$\forall\ \varphi \in fmla.\ \forall\ x \in var.\ Fvars\ \varphi = \{x\} \longrightarrow$
$(\forall\ n \in num.\ prv\ (subst\ \varphi\ n\ x)) \longrightarrow \neg\ prv\ (exi\ x\ (neg\ \varphi))$

lemma ω*consistent_impliesStd1*:
ω*consistent* $\Longrightarrow$
ω*consistentStd1*
⟨*proof*⟩

lemma ω*consistent_impliesStd2*:
ω*consistent* $\Longrightarrow$
ω*consistentStd2*
⟨*proof*⟩

In the presence of classical logic deduction, the stronger condition is equivalent to the standard ones:

lemma ω*consistent_iffStd1*:
assumes $\bigwedge \varphi.\ \varphi \in fmla \Longrightarrow prv\ (imp\ (neg\ (neg\ \varphi))\ \varphi)$
shows ω*consistent* $\longleftrightarrow$ ω*consistentStd1*
⟨*proof*⟩

lemma ω*consistent_iffStd2*:
assumes $\bigwedge \varphi.\ \varphi \in fmla \Longrightarrow prv\ (imp\ (neg\ (neg\ \varphi))\ \varphi)$
shows ω*consistent* $\longleftrightarrow$ ω*consistentStd2*
⟨*proof*⟩

ω-consistency implies consistency:

lemma ω*consistentStd1_implies_consistent*:
assumes ω*consistentStd1*
shows *consistent*
⟨*proof*⟩

lemma ω*consistentStd2_implies_consistent*:
assumes ω*consistentStd2*
shows *consistent*
⟨*proof*⟩

corollary ω*consistent_implies_consistent*:
assumes ω*consistent*

shows *consistent*
⟨*proof*⟩

end — context *Deduct_with_False*

3.3 Deduction Considering False and Disjunction

locale *Deduct_with_False_Disj* =
Syntax_with_Connectives_False_Disj
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
+
Deduct_with_False
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
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*
+
assumes
prv_dsj_impL:
⋀ *φ χ*. *φ* ∈ *fmla* ⟹ *χ* ∈ *fmla* ⟹
prv (*imp φ* (*dsj φ χ*))
and
prv_dsj_impR:
⋀ *φ χ*. *φ* ∈ *fmla* ⟹ *χ* ∈ *fmla* ⟹
prv (*imp χ* (*dsj φ χ*))
and
prv_imp_dsjE:
⋀ *φ χ ψ*. *φ* ∈ *fmla* ⟹ *χ* ∈ *fmla* ⟹ *ψ* ∈ *fmla* ⟹
prv (*imp* (*imp φ ψ*) (*imp* (*imp χ ψ*) (*imp* (*dsj φ χ*) *ψ*)))
begin

lemma *prv_imp_dsjEE*:
assumes *φ*[*simp*]: *φ* ∈ *fmla* **and** *χ*[*simp*]: *χ* ∈ *fmla* **and** *ψ*[*simp*]: *ψ* ∈ *fmla*
assumes *prv* (*imp φ ψ*) **and** *prv* (*imp χ ψ*)
shows *prv* (*imp* (*dsj φ χ*) *ψ*)
⟨*proof*⟩

lemma *prv_dsj_cases*:
assumes *φ1* ∈ *fmla φ2* ∈ *fmla χ* ∈ *fmla*
and *prv* (*dsj φ1 φ2*) **and** *prv* (*imp φ1 χ*) **and** *prv* (*imp φ2 χ*)
shows *prv χ*
⟨*proof*⟩

3.3.1 Disjunction vs. disjunction

lemma *prv_dsj_com_imp*:

assumes φ[*simp*]: $\varphi \in$ *fmla* **and** χ[*simp*]: $\chi \in$ *fmla*
shows *prv* (*imp* (*dsj* φ χ) (*dsj* χ φ))
⟨*proof*⟩

lemma *prv_dsj_com*:
assumes φ[*simp*]: $\varphi \in$ *fmla* **and** χ[*simp*]: $\chi \in$ *fmla*
shows *prv* (*eqv* (*dsj* φ χ) (*dsj* χ φ))
⟨*proof*⟩

lemma *prv_dsj_assoc_imp1*:
assumes φ[*simp*]: $\varphi \in$ *fmla* **and** χ[*simp*]: $\chi \in$ *fmla* **and** ψ[*simp*]: $\psi \in$ *fmla*
shows *prv* (*imp* (*dsj* φ (*dsj* χ ψ)) (*dsj* (*dsj* φ χ) ψ))
⟨*proof*⟩

lemma *prv_dsj_assoc_imp2*:
assumes φ[*simp*]: $\varphi \in$ *fmla* **and** χ[*simp*]: $\chi \in$ *fmla* **and** ψ[*simp*]: $\psi \in$ *fmla*
shows *prv* (*imp* (*dsj* (*dsj* φ χ) ψ) (*dsj* φ (*dsj* χ ψ)))
⟨*proof*⟩

lemma *prv_dsj_assoc*:
assumes φ[*simp*]: $\varphi \in$ *fmla* **and** χ[*simp*]: $\chi \in$ *fmla* **and** ψ[*simp*]: $\psi \in$ *fmla*
shows *prv* (*eqv* (*dsj* φ (*dsj* χ ψ)) (*dsj* (*dsj* φ χ) ψ))
⟨*proof*⟩

lemma *prv_dsj_com_imp3*:
assumes $\varphi 1 \in$ *fmla* $\varphi 2 \in$ *fmla* $\varphi 3 \in$ *fmla*
shows *prv* (*imp* (*dsj* $\varphi 1$ (*dsj* $\varphi 2$ $\varphi 3$))
(*dsj* $\varphi 2$ (*dsj* $\varphi 1$ $\varphi 3$)))
⟨*proof*⟩

lemma *prv_dsj_mono*:
$\varphi 1 \in$ *fmla* $\Longrightarrow$ $\varphi 2 \in$ *fmla* $\Longrightarrow$ $\chi 1 \in$ *fmla* $\Longrightarrow$ $\chi 2 \in$ *fmla* $\Longrightarrow$
prv (*imp* $\varphi 1$ $\chi 1$) $\Longrightarrow$ *prv* (*imp* $\varphi 2$ $\chi 2$) $\Longrightarrow$ *prv* (*imp* (*dsj* $\varphi 1$ $\varphi 2$) (*dsj* $\chi 1$ $\chi 2$))
⟨*proof*⟩

3.3.2 Disjunction vs. conjunction

lemma *prv_cnj_dsj_distrib1*:
assumes φ[*simp*]: $\varphi \in$ *fmla* **and** $\chi 1$[*simp*]: $\chi 1 \in$ *fmla* **and** $\chi 2$[*simp*]: $\chi 2 \in$ *fmla*
shows *prv* (*imp* (*cnj* φ (*dsj* $\chi 1$ $\chi 2$)) (*dsj* (*cnj* φ $\chi 1$) (*cnj* φ $\chi 2$)))
⟨*proof*⟩

lemma *prv_cnj_dsj_distrib2*:
assumes φ[*simp*]: $\varphi \in$ *fmla* **and** $\chi 1$[*simp*]: $\chi 1 \in$ *fmla* **and** $\chi 2$[*simp*]: $\chi 2 \in$ *fmla*
shows *prv* (*imp* (*dsj* (*cnj* φ $\chi 1$) (*cnj* φ $\chi 2$)) (*cnj* φ (*dsj* $\chi 1$ $\chi 2$)))
⟨*proof*⟩

lemma *prv_cnj_dsj_distrib*:
assumes φ[*simp*]: $\varphi \in$ *fmla* **and** $\chi 1$[*simp*]: $\chi 1 \in$ *fmla* **and** $\chi 2$[*simp*]: $\chi 2 \in$ *fmla*
shows *prv* (*eqv* (*cnj* φ (*dsj* $\chi 1$ $\chi 2$)) (*dsj* (*cnj* φ $\chi 1$) (*cnj* φ $\chi 2$)))
⟨*proof*⟩

lemma *prv_dsj_cnj_distrib1*:
assumes φ[*simp*]: $\varphi \in$ *fmla* **and** $\chi 1$[*simp*]: $\chi 1 \in$ *fmla* **and** $\chi 2$[*simp*]: $\chi 2 \in$ *fmla*
shows *prv* (*imp* (*dsj* φ (*cnj* $\chi 1$ $\chi 2$)) (*cnj* (*dsj* φ $\chi 1$) (*dsj* φ $\chi 2$)))
⟨*proof*⟩

lemma *prv_dsj_cnj_distrib2*:

assumes φ[*simp*]: $\varphi \in$ *fmla* **and** $\chi 1$[*simp*]: $\chi 1 \in$ *fmla* **and** $\chi 2$[*simp*]: $\chi 2 \in$ *fmla*
shows *prv* (*imp* (*cnj* (*dsj* φ $\chi 1$) (*dsj* φ $\chi 2$)) (*dsj* φ (*cnj* $\chi 1$ $\chi 2$)))
⟨*proof*⟩

lemma *prv_dsj_cnj_distrib*:
assumes φ[*simp*]: $\varphi \in$ *fmla* **and** $\chi 1$[*simp*]: $\chi 1 \in$ *fmla* **and** $\chi 2$[*simp*]: $\chi 2 \in$ *fmla*
shows *prv* (*eqv* (*dsj* φ (*cnj* $\chi 1$ $\chi 2$)) (*cnj* (*dsj* φ $\chi 1$) (*dsj* φ $\chi 2$)))
⟨*proof*⟩

3.3.3 Disjunction vs. True and False

lemma *prv_dsjR_fls*: $\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*eqv* (*dsj fls* φ) φ)
⟨*proof*⟩

lemma *prv_dsjL_fls*: $\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*eqv* (*dsj* φ *fls*) φ)
⟨*proof*⟩

lemma *prv_dsjR_tru*: $\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*eqv* (*dsj tru* φ) *tru*)
⟨*proof*⟩

lemma *prv_dsjL_tru*: $\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*eqv* (*dsj* φ *tru*) *tru*)
⟨*proof*⟩

3.3.4 Set-based disjunctions

Just like for conjunctions, these are based on properties of the auxiliary list disjunctions.

lemma *prv_imp_ldsj_in*:
assumes *set* $\varphi s \subseteq$ *fmla*
and $\varphi \in$ *set* φs
shows *prv* (*imp* φ (*ldsj* φs))
⟨*proof*⟩

lemma *prv_imp_ldsj*:
assumes $\chi \in$ *fmla* **and** *set* $\varphi s \subseteq$ *fmla*
and $\varphi \in$ *set* φs **and** *prv* (*imp* χ φ)
shows *prv* (*imp* χ (*ldsj* φs))
⟨*proof*⟩

lemma *prv_ldsj_imp*:
assumes $\chi \in$ *fmla* **and** *set* $\varphi s \subseteq$ *fmla*
and $\bigwedge \varphi$. $\varphi \in$ *set* $\varphi s \Longrightarrow$ *prv* (*imp* φ χ)
shows *prv* (*imp* (*ldsj* φs) χ)
⟨*proof*⟩

lemma *prv_ldsj_imp_inner*:
assumes $\varphi \in$ *fmla* *set* $\varphi 1s \subseteq$ *fmla* *set* $\varphi 2s \subseteq$ *fmla*
shows *prv* (*imp* (*ldsj* ($\varphi 1s$ @ φ # $\varphi 2s$)) (*dsj* φ (*ldsj* ($\varphi 1s$ @ $\varphi 2s$))))
⟨*proof*⟩

lemma *prv_ldsj_imp_remdups*:
assumes *set* $\varphi s \subseteq$ *fmla*
shows *prv* (*imp* (*ldsj* φs) (*ldsj* (*remdups* φs)))
⟨*proof*⟩

lemma *prv_ldsj_mono*:
assumes $\varphi 2s$: *set* $\varphi 2s \subseteq$ *fmla* **and** *set* $\varphi 1s \subseteq$ *set* $\varphi 2s$
shows *prv* (*imp* (*ldsj* $\varphi 1s$) (*ldsj* $\varphi 2s$))
⟨*proof*⟩

lemma *prv_ldsj_eqv*:
assumes *set φ1s ⊆ fmla* **and** *set φ2s = set φ1s*
shows *prv* (*eqv* (*ldsj φ1s*) (*ldsj φ2s*))
⟨*proof*⟩

lemma *prv_ldsj_mono_imp*:
assumes *set φ1s ⊆ fmla set φ2s ⊆ fmla* **and** ∀ *φ1* ∈ *set φ1s*. ∃ *φ2* ∈ *set φ2s*. *prv* (*imp φ1 φ2*)
shows *prv* (*imp* (*ldsj φ1s*) (*ldsj φ2s*))
⟨*proof*⟩

Just like set-based conjunction, set-based disjunction commutes with substitution only up to provably equivalence:

lemma *prv_subst_sdsj*:
F ⊆ fmla ⟹ *finite F* ⟹ *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹
prv (*eqv* (*subst* (*sdsj F*) *t x*) (*sdsj* ((λ*φ*. *subst φ t x*) ‘ *F*)))
⟨*proof*⟩

lemma *prv_imp_sdsj_in*:
assumes *φ* ∈ *fmla* **and** *F ⊆ fmla finite F*
and *φ* ∈ *F*
shows *prv* (*imp φ* (*sdsj F*))
⟨*proof*⟩

lemma *prv_imp_sdsj*:
assumes *χ* ∈ *fmla* **and** *F ⊆ fmla finite F*
and *φ* ∈ *F* **and** *prv* (*imp χ φ*)
shows *prv* (*imp χ* (*sdsj F*))
⟨*proof*⟩

lemma *prv_sdsj_imp*:
assumes *χ* ∈ *fmla* **and** *F ⊆ fmla finite F*
and ⋀*φ*. *φ* ∈ *F* ⟹ *prv* (*imp φ χ*)
shows *prv* (*imp* (*sdsj F*) *χ*)
⟨*proof*⟩

lemma *prv_sdsj_mono*:
assumes *F2 ⊆ fmla* **and** *F1 ⊆ F2* **and** *finite F2*
shows *prv* (*imp* (*sdsj F1*) (*sdsj F2*))
⟨*proof*⟩

lemma *prv_sdsj_mono_imp*:
assumes *F1 ⊆ fmla F2 ⊆ fmla finite F1 finite F2*
and ∀ *φ1* ∈ *F1*. ∃ *φ2* ∈ *F2*. *prv* (*imp φ1 φ2*)
shows *prv* (*imp* (*sdsj F1*) (*sdsj F2*))
⟨*proof*⟩

lemma *prv_sdsj_cases*:
assumes *F ⊆ fmla finite F ψ* ∈ *fmla*
and *prv* (*sdsj F*) **and** ⋀ *φ*. *φ* ∈ *F* ⟹ *prv* (*imp φ ψ*)
shows *prv ψ*
⟨*proof*⟩

lemma *prv_sdsj1_imp*:
φ ∈ *fmla* ⟹ *prv* (*imp* (*sdsj* {*φ*}) *φ*)
⟨*proof*⟩

lemma *prv_imp_sdsj1*:

$\varphi \in fmla \Longrightarrow prv\ (imp\ \varphi\ (sdsj\ \{\varphi\}))$
⟨*proof*⟩

lemma *prv_sdsj2_imp_dsj*:
$\varphi \in fmla \Longrightarrow \psi \in fmla \Longrightarrow prv\ (imp\ (sdsj\ \{\varphi,\psi\})\ (dsj\ \varphi\ \psi))$
⟨*proof*⟩

lemma *prv_dsj_imp_sdsj2*:
$\varphi \in fmla \Longrightarrow \psi \in fmla \Longrightarrow prv\ (imp\ (dsj\ \varphi\ \psi)\ (sdsj\ \{\varphi,\psi\}))$
⟨*proof*⟩

Commutation with parallel substitution:

lemma *prv_rawpsubst_sdsj*:
assumes $F \subseteq fmla$ *finite* F
and $snd\ `\ (set\ txs) \subseteq var \times trm$
shows $prv\ (eqv\ (rawpsubst\ (sdsj\ F)\ txs)\ (sdsj\ ((\lambda\varphi.\ rawpsubst\ \varphi\ txs)\ `\ F)))$
⟨*proof*⟩

lemma *prv_psubst_sdsj*:
assumes $F \subseteq fmla$ *finite* F
and $snd\ `\ (set\ txs) \subseteq var \times trm$
and *distinct* $(map\ snd\ txs)$
shows $prv\ (eqv\ (psubst\ (sdsj\ F)\ txs)\ (sdsj\ ((\lambda\varphi.\ psubst\ \varphi\ txs)\ `\ F)))$
⟨*proof*⟩

end — context *Deduct_with_False_Disj*

3.4 Deduction with Quantified Variable Renaming Included

locale *Deduct_with_False_Disj_Rename* =
Deduct_with_False_Disj
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
prv
+
Syntax_with_Connectives_Rename
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
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*

3.5 Deduction with PseudoOrder Axioms Included

We assume a two-variable formula Lq that satisfies two axioms resembling the properties of the strict or nonstrict ordering on naturals. The choice of these axioms is motivated by an abstract account of Rosser's trick to improve on Gödel's First Incompleteness Theorem, reported in our CADE 2019 paper [1].

locale *Deduct_with_PseudoOrder* =
Deduct_with_False_Disj
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
prv
+
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*
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*. *prv* (*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*. *prv* (*subst φ p zz*)
shows *prv* (*all zz* (*imp* (*LLq* (*Var zz*) *q*) *φ*))
⟨*proof*⟩

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

end — context *Deduct_with_PseudoOrder*

Chapter 4

Natural Deduction

We develop a natural deduction system based on the Hilbert system.

context *Deduct_with_False_Disj*
begin

4.1 Natural Deduction from the Hilbert System

definition *nprv* :: *'fmla set* ⇒ *'fmla* ⇒ *bool* **where**
nprv F φ ≡ *prv* (*imp* (*scnj F*) *φ*)

lemma *nprv_hyp*[*simp*,*intro*]:
φ ∈ *F* ⟹ *F* ⊆ *fmla* ⟹ *finite F* ⟹ *nprv F φ*
⟨*proof*⟩

4.2 Structural Rules for the Natural Deduction Relation

lemma *prv_nprv0I*: *prv φ* ⟹ *φ* ∈ *fmla* ⟹ *nprv* {} *φ*
⟨*proof*⟩

lemma *prv_nprv_emp*: *φ* ∈ *fmla* ⟹ *prv φ* ⟷ *nprv* {} *φ*
⟨*proof*⟩

lemma *nprv_mono*:
assumes *nprv G φ*
and *F* ⊆ *fmla finite F G* ⊆ *F φ* ∈ *fmla*
shows *nprv F φ*
⟨*proof*⟩

lemma *nprv_cut*:
assumes *nprv F φ* **and** *nprv* (*insert φ F*) *ψ*
and *F* ⊆ *fmla finite F φ* ∈ *fmla ψ* ∈ *fmla*
shows *nprv F ψ*
⟨*proof*⟩

lemma *nprv_strong_cut2*:
nprv F φ1 ⟹ *nprv* (*insert φ1 F*) *φ2* ⟹ *nprv* (*insert φ2* (*insert φ1 F*)) *ψ* ⟹
F ⊆ *fmla* ⟹ *finite F* ⟹ *φ1* ∈ *fmla* ⟹ *φ2* ∈ *fmla* ⟹ *ψ* ∈ *fmla* ⟹
nprv F ψ
⟨*proof*⟩

lemma *nprv_cut2*:

nprv F φ1 ⟹ nprv F φ2 ⟹
F ⊆ fmla ⟹ finite F ⟹ φ1 ∈ fmla ⟹ φ2 ∈ fmla ⟹ ψ ∈ fmla ⟹
nprv (insert φ2 (insert φ1 F)) ψ ⟹ nprv F ψ
⟨*proof*⟩

Useful for fine control of the eigenformula:

lemma *nprv_insertShiftI*:
nprv (insert φ1 (insert φ2 F)) ψ ⟹ nprv (insert φ2 (insert φ1 F)) ψ
⟨*proof*⟩

lemma *nprv_insertShift2I*:
nprv (insert φ3 (insert φ1 (insert φ2 F))) ψ ⟹ nprv (insert φ1 (insert φ2 (insert φ3 F))) ψ
⟨*proof*⟩

4.3 Back and Forth between Hilbert and Natural Deduction

This is now easy, thanks to the large number of facts we have proved for Hilbert-style deduction

lemma *prv_nprvI*: *prv φ ⟹ φ ∈ fmla ⟹ F ⊆ fmla ⟹ finite F ⟹ nprv F φ*
⟨*proof*⟩

thm *prv_nprv0I*

lemma *prv_nprv1I*:
assumes *φ ∈ fmla ψ ∈ fmla* **and** *prv (imp φ ψ)*
shows *nprv {φ} ψ*
⟨*proof*⟩

lemma *prv_nprv2I*:
assumes *prv (imp φ1 (imp φ2 ψ)) φ1 ∈ fmla φ2 ∈ fmla ψ ∈ fmla*
shows *nprv {φ1,φ2} ψ*
⟨*proof*⟩

lemma *nprv_prvI*: *nprv {} φ ⟹ φ ∈ fmla ⟹ prv φ*
⟨*proof*⟩

4.4 More Structural Properties

lemma *nprv_clear*: *nprv {} φ ⟹ F ⊆ fmla ⟹ finite F ⟹ φ ∈ fmla ⟹ nprv F φ*
⟨*proof*⟩

lemma *nprv_cut_set*:
assumes *F*: *finite F F ⊆ fmla* **and** *G*: *finite G G ⊆ fmla χ ∈ fmla*
and *n1*: *⋀ ψ. ψ ∈ G ⟹ nprv F ψ* **and** *n2*: *nprv (G ∪ F) χ*
shows *nprv F χ*
⟨*proof*⟩

lemma *nprv_clear2_1*:
nprv {φ2} ψ ⟹ φ1 ∈ fmla ⟹ φ2 ∈ fmla ⟹ ψ ∈ fmla ⟹
nprv {φ1,φ2} ψ
⟨*proof*⟩

lemma *nprv_clear2_2*:
nprv {φ1} ψ ⟹ φ1 ∈ fmla ⟹ φ2 ∈ fmla ⟹ ψ ∈ fmla ⟹
nprv {φ1,φ2} ψ
⟨*proof*⟩

lemma *nprv_clear3_1*:
$nprv\ \{\varphi 2,\varphi 3\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3\}\ \psi$
⟨*proof*⟩

lemma *nprv_clear3_2*:
$nprv\ \{\varphi 1,\varphi 3\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3\}\ \psi$
⟨*proof*⟩

lemma *nprv_clear3_3*:
$nprv\ \{\varphi 1,\varphi 2\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3\}\ \psi$
⟨*proof*⟩

lemma *nprv_clear4_1*:
$nprv\ \{\varphi 2,\varphi 3,\varphi 4\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \varphi 4 \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4\}\ \psi$
⟨*proof*⟩

lemma *nprv_clear4_2*:
$nprv\ \{\varphi 1,\varphi 3,\varphi 4\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \varphi 4 \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4\}\ \psi$
⟨*proof*⟩

lemma *nprv_clear4_3*:
$nprv\ \{\varphi 1,\varphi 2,\varphi 4\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \varphi 4 \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4\}\ \psi$
⟨*proof*⟩

lemma *nprv_clear4_4*:
$nprv\ \{\varphi 1,\varphi 2,\varphi 3\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \varphi 4 \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4\}\ \psi$
⟨*proof*⟩

lemma *nprv_clear5_1*:
$nprv\ \{\varphi 2,\varphi 3,\varphi 4,\varphi 5\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \varphi 4 \in fmla \Longrightarrow \varphi 5 \in fmla$
$\Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4,\varphi 5\}\ \psi$
⟨*proof*⟩

lemma *nprv_clear5_2*:
$nprv\ \{\varphi 1,\varphi 3,\varphi 4,\varphi 5\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \varphi 4 \in fmla \Longrightarrow \varphi 5 \in fmla$
$\Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4,\varphi 5\}\ \psi$
⟨*proof*⟩

lemma *nprv_clear5_3*:
$nprv\ \{\varphi 1,\varphi 2,\varphi 4,\varphi 5\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \varphi 4 \in fmla \Longrightarrow \varphi 5 \in fmla$
$\Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4,\varphi 5\}\ \psi$
⟨*proof*⟩

lemma *nprv_clear5_4*:
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 5\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \varphi 4 \in fmla \Longrightarrow \varphi 5 \in fmla$
$\Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4,\varphi 5\}\ \psi$
⟨*proof*⟩

lemma *nprv_clear5_5*:
nprv $\{\varphi 1,\varphi 2,\varphi 3,\varphi 4\}$ $\psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \varphi 4 \in fmla \Longrightarrow \varphi 5 \in fmla$
$\Longrightarrow \psi \in fmla \Longrightarrow$
nprv $\{\varphi 1,\varphi 2,\varphi 3,\varphi 4,\varphi 5\}$ ψ
⟨*proof*⟩

4.5 Properties Involving Substitution

lemma *nprv_subst*:
assumes $x \in var$ $t \in trm$ $\psi \in fmla$ *finite* F $F \subseteq fmla$
and *1*: *nprv* F ψ
shows *nprv* $((\lambda\varphi.\ subst\ \varphi\ t\ x)\ `\ F)\ (subst\ \psi\ t\ x)$
⟨*proof*⟩

lemma *nprv_subst_fresh*:
assumes *0*: $x \in var$ $t \in trm$ $\psi \in fmla$ *finite* F $F \subseteq fmla$
nprv F ψ **and** *1*: $x \notin \bigcup\ (Fvars\ `\ F)$
shows *nprv* F $(subst\ \psi\ t\ x)$
⟨*proof*⟩

lemma *nprv_subst_rev*:
assumes *0*: $x \in var$ $y \in var$ $\psi \in fmla$ *finite* F $F \subseteq fmla$
and *f*: $y = x \vee (y \notin Fvars\ \psi \wedge y \notin \bigcup\ (Fvars\ `\ F))$
and *1*: *nprv* $((\lambda\varphi.\ subst\ \varphi\ (Var\ y)\ x)\ `\ F)\ (subst\ \psi\ (Var\ y)\ x)$
shows *nprv* F ψ
⟨*proof*⟩

lemma *nprv_psubst*:
assumes *0*: *snd* ` *set txs* $\subseteq$ *var fst* ` *set txs* $\subseteq$ *trm* $\psi \in fmla$ *finite* F $F \subseteq fmla$
distinct (*map snd txs*)
and *1*: *nprv* F ψ
shows *nprv* $((\lambda\varphi.\ psubst\ \varphi\ txs)\ `\ F)\ (psubst\ \psi\ txs)$
⟨*proof*⟩

4.6 Introduction and Elimination Rules

We systematically leave the side-conditions at the end, to simplify reasoning.

lemma *nprv_impI*:
nprv $(insert\ \varphi\ F)\ \psi \Longrightarrow$
$F \subseteq fmla \Longrightarrow$ *finite* $F \Longrightarrow \varphi \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
nprv F $(imp\ \varphi\ \psi)$
⟨*proof*⟩

lemma *nprv_impI_rev*:
assumes *nprv* F $(imp\ \varphi\ \psi)$
and $F \subseteq fmla$ **and** *finite* F **and** $\varphi \in fmla$ **and** $\psi \in fmla$
shows *nprv* $(insert\ \varphi\ F)\ \psi$
⟨*proof*⟩

lemma *nprv_impI_rev2*:
assumes *nprv* F $(imp\ \varphi\ \psi)$ **and** *G*: *insert* $\varphi\ F \subseteq G$
and $G \subseteq fmla$ **and** *finite* G **and** $\varphi \in fmla$ **and** $\psi \in fmla$
shows *nprv* G ψ
⟨*proof*⟩

lemma *nprv_mp*:
nprv F (imp φ ψ) ⟹ nprv F φ ⟹
F ⊆ fmla ⟹ finite F ⟹ φ ∈ fmla ⟹ ψ ∈ fmla ⟹
nprv F ψ
⟨*proof*⟩

lemma *nprv_impE*:
nprv F (imp φ ψ) ⟹ nprv F φ ⟹ nprv (insert ψ F) χ ⟹
F ⊆ fmla ⟹ finite F ⟹ φ ∈ fmla ⟹ ψ ∈ fmla ⟹ χ ∈ fmla ⟹
nprv F χ
⟨*proof*⟩

lemmas *nprv_impE0 = nprv_impE*[*OF nprv_hyp _ _*, *simped*]
lemmas *nprv_impE1 = nprv_impE*[*OF _ nprv_hyp _*, *simped*]
lemmas *nprv_impE2 = nprv_impE*[*OF _ _ nprv_hyp*, *simped*]
lemmas *nprv_impE01 = nprv_impE*[*OF nprv_hyp nprv_hyp _*, *simped*]
lemmas *nprv_impE02 = nprv_impE*[*OF nprv_hyp _ nprv_hyp*, *simped*]
lemmas *nprv_impE12 = nprv_impE*[*OF _ nprv_hyp nprv_hyp*, *simped*]
lemmas *nprv_impE012 = nprv_impE*[*OF nprv_hyp nprv_hyp nprv_hyp*, *simped*]

lemma *nprv_cnjI*:
nprv F φ ⟹ nprv F ψ ⟹
F ⊆ fmla ⟹ finite F ⟹ φ ∈ fmla ⟹ ψ ∈ fmla ⟹
nprv F (cnj φ ψ)
⟨*proof*⟩

lemma *nprv_cnjE*:
nprv F (cnj φ1 φ2) ⟹ nprv (insert φ1 (insert φ2 F)) ψ ⟹
F ⊆ fmla ⟹ finite F ⟹ φ1 ∈ fmla ⟹ φ2 ∈ fmla ⟹ ψ ∈ fmla ⟹
nprv F ψ
⟨*proof*⟩

lemmas *nprv_cnjE0 = nprv_cnjE*[*OF nprv_hyp _*, *simped*]
lemmas *nprv_cnjE1 = nprv_cnjE*[*OF _ nprv_hyp*, *simped*]
lemmas *nprv_cnjE01 = nprv_cnjE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemma *nprv_dsjIL*:
nprv F φ ⟹
F ⊆ fmla ⟹ finite F ⟹ φ ∈ fmla ⟹ ψ ∈ fmla ⟹
nprv F (dsj φ ψ)
⟨*proof*⟩

lemma *nprv_dsjIR*:
nprv F ψ ⟹
F ⊆ fmla ⟹ finite F ⟹ φ ∈ fmla ⟹ ψ ∈ fmla ⟹
nprv F (dsj φ ψ)
⟨*proof*⟩

lemma *nprv_dsjE*:
assumes *nprv F (dsj φ ψ)*
and *nprv (insert φ F) χ nprv (insert ψ F) χ*
and *F ⊆ fmla finite F φ ∈ fmla ψ ∈ fmla χ ∈ fmla*
shows *nprv F χ*
⟨*proof*⟩

lemmas *nprv_dsjE0 = nprv_dsjE*[*OF nprv_hyp _ _*, *simped*]
lemmas *nprv_dsjE1 = nprv_dsjE*[*OF _ nprv_hyp _*, *simped*]
lemmas *nprv_dsjE2 = nprv_dsjE*[*OF _ _ nprv_hyp*, *simped*]

lemmas *nprv_dsjE01* = *nprv_dsjE*[*OF nprv_hyp nprv_hyp _*, *simped*]
lemmas *nprv_dsjE02* = *nprv_dsjE*[*OF nprv_hyp _ nprv_hyp*, *simped*]
lemmas *nprv_dsjE12* = *nprv_dsjE*[*OF _ nprv_hyp nprv_hyp*, *simped*]
lemmas *nprv_dsjE012* = *nprv_dsjE*[*OF nprv_hyp nprv_hyp nprv_hyp*, *simped*]

lemma *nprv_flsE*: *nprv F fls* $\Longrightarrow$ $F \subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ $\varphi \in$ *fmla* $\Longrightarrow$ *nprv F* φ
⟨*proof*⟩

lemmas *nprv_flsE0* = *nprv_flsE*[*OF nprv_hyp*, *simped*]

lemma *nprv_truI*: $F \subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ *nprv F tru*
⟨*proof*⟩

lemma *nprv_negI*:
nprv (*insert* φ *F*) *fls* $\Longrightarrow$
$F \subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ $\varphi \in$ *fmla* $\Longrightarrow$
nprv F (*neg* φ)
⟨*proof*⟩

lemma *nprv_neg_fls*:
nprv F (*neg* φ) $\Longrightarrow$ *nprv F* φ $\Longrightarrow$
$F \subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ $\varphi \in$ *fmla* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$
nprv F fls
⟨*proof*⟩

lemma *nprv_negE*:
nprv F (*neg* φ) $\Longrightarrow$ *nprv F* φ $\Longrightarrow$
$F \subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ $\varphi \in$ *fmla* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$
nprv F ψ
⟨*proof*⟩

lemmas *nprv_negE0* = *nprv_negE*[*OF nprv_hyp _*, *simped*]
lemmas *nprv_negE1* = *nprv_negE*[*OF _ nprv_hyp*, *simped*]
lemmas *nprv_negE01* = *nprv_negE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemma *nprv_scnjI*:
($\bigwedge \psi$. $\psi \in G \Longrightarrow$ *nprv F* ψ) $\Longrightarrow$
$F \subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ $G \subseteq$ *fmla* $\Longrightarrow$ *finite G* $\Longrightarrow$
nprv F (*scnj G*)
⟨*proof*⟩

lemma *nprv_scnjE*:
nprv F (*scnj G*) $\Longrightarrow$ *nprv* ($G \cup F$) ψ $\Longrightarrow$
$F \subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ $G \subseteq$ *fmla* $\Longrightarrow$ *finite G* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$
nprv F ψ
⟨*proof*⟩

lemmas *nprv_scnjE0* = *nprv_scnjE*[*OF nprv_hyp _*, *simped*]
lemmas *nprv_scnjE1* = *nprv_scnjE*[*OF _ nprv_hyp*, *simped*]
lemmas *nprv_scnjE01* = *nprv_scnjE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemma *nprv_lcnjI*:
($\bigwedge \psi$. $\psi \in$ *set* $\psi s \Longrightarrow$ *nprv F* ψ) $\Longrightarrow$
$F \subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ *set* $\psi s \subseteq$ *fmla* $\Longrightarrow$
nprv F (*lcnj* ψs)
⟨*proof*⟩

lemma *nprv_lcnjE*:

$nprv\ F\ (lcnj\ \varphi s) \Longrightarrow nprv\ (set\ \varphi s \cup F)\ \psi \Longrightarrow$
$F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow set\ \varphi s \subseteq fmla \Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ F\ \psi$
⟨*proof*⟩

lemmas *nprv_lcnjE0* = *nprv_lcnjE*[*OF nprv_hyp* _, *simped*]
lemmas *nprv_lcnjE1* = *nprv_lcnjE*[*OF* _ *nprv_hyp*, *simped*]
lemmas *nprv_lcnjE01* = *nprv_lcnjE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemma *nprv_sdsjI*:
$nprv\ F\ \varphi \Longrightarrow$
$F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow G \subseteq fmla \Longrightarrow finite\ G \Longrightarrow \varphi \in G \Longrightarrow$
$nprv\ F\ (sdsj\ G)$
⟨*proof*⟩

lemma *nprv_sdsjE*:
assumes $nprv\ F\ (sdsj\ G)$
and $\bigwedge \psi.\ \psi \in G \Longrightarrow nprv\ (insert\ \psi\ F)\ \chi$
and $F \subseteq fmla\ finite\ F\ G \subseteq fmla\ finite\ G\ \chi \in fmla$
shows $nprv\ F\ \chi$
⟨*proof*⟩

lemmas *nprv_sdsjE0* = *nprv_sdsjE*[*OF nprv_hyp* _, *simped*]
lemmas *nprv_sdsjE1* = *nprv_sdsjE*[*OF* _ *nprv_hyp*, *simped*]
lemmas *nprv_sdsjE01* = *nprv_sdsjE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemma *nprv_ldsjI*:
$nprv\ F\ \varphi \Longrightarrow$
$F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow set\ \varphi s \subseteq fmla \Longrightarrow\ \varphi \in set\ \varphi s \Longrightarrow$
$nprv\ F\ (ldsj\ \varphi s)$
⟨*proof*⟩

lemma *nprv_ldsjE*:
assumes $nprv\ F\ (ldsj\ \psi s)$
and $\bigwedge \psi.\ \psi \in set\ \psi s \Longrightarrow nprv\ (insert\ \psi\ F)\ \chi$
and $F \subseteq fmla\ finite\ F\ set\ \psi s \subseteq fmla\ \ \chi \in fmla$
shows $nprv\ F\ \chi$
⟨*proof*⟩

lemmas *nprv_ldsjE0* = *nprv_ldsjE*[*OF nprv_hyp* _, *simped*]
lemmas *nprv_ldsjE1* = *nprv_ldsjE*[*OF* _ *nprv_hyp*, *simped*]
lemmas *nprv_ldsjE01* = *nprv_ldsjE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemma *nprv_allI*:
$nprv\ F\ \varphi \Longrightarrow$
$F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow \varphi \in fmla \Longrightarrow x \in var \Longrightarrow x \notin \bigcup\ (Fvars\ `\ F) \Longrightarrow$
$nprv\ F\ (all\ x\ \varphi)$
⟨*proof*⟩

lemma *nprv_allE*:
assumes $nprv\ F\ (all\ x\ \varphi)\ nprv\ (insert\ (subst\ \varphi\ t\ x)\ F)\ \psi$
$F \subseteq fmla\ finite\ F\ \varphi \in fmla\ t \in trm\ x \in var\ \psi \in fmla$
shows $nprv\ F\ \psi$
⟨*proof*⟩

lemmas *nprv_allE0* = *nprv_allE*[*OF nprv_hyp* _, *simped*]
lemmas *nprv_allE1* = *nprv_allE*[*OF* _ *nprv_hyp*, *simped*]
lemmas *nprv_allE01* = *nprv_allE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemma *nprv_exiI*:
nprv F (subst φ t x) $\Longrightarrow$
F $\subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ *φ* $\in$ *fmla* $\Longrightarrow$ *t* $\in$ *trm* $\Longrightarrow$ *x* $\in$ *var* $\Longrightarrow$
nprv F (exi x φ)
⟨*proof*⟩

lemma *nprv_exiE*:
assumes *n*: *nprv F (exi x φ)*
and *nn*: *nprv (insert φ F) ψ*
and *0*[*simp*]: *F* $\subseteq$ *fmla finite F φ* $\in$ *fmla x* $\in$ *var ψ* $\in$ *fmla*
and *x*: *x* $\notin \bigcup$ *(Fvars ' F) x* $\notin$ *Fvars ψ*
shows *nprv F ψ*
⟨*proof*⟩

lemmas *nprv_exiE0* = *nprv_exiE*[*OF nprv_hyp _*, *simped*]
lemmas *nprv_exiE1* = *nprv_exiE*[*OF _ nprv_hyp*, *simped*]
lemmas *nprv_exiE01* = *nprv_exiE*[*OF nprv_hyp nprv_hyp*, *simped*]

4.7 Adding Lemmas of Various Shapes into the Proof Context

lemma *nprv_addLemmaE*:
assumes *prv φ nprv (insert φ F) ψ*
and *φ* $\in$ *fmla ψ* $\in$ *fmla* **and** *F* $\subseteq$ *fmla* **and** *finite F*
shows *nprv F ψ*
⟨*proof*⟩

lemmas *nprv_addLemmaE1* = *nprv_addLemmaE*[*OF _ nprv_hyp*, *simped*]

lemma *nprv_addImpLemmaI*:
assumes *prv (imp φ1 φ2)*
and *F* $\subseteq$ *fmla finite F φ1* $\in$ *fmla φ2* $\in$ *fmla*
and *nprv F φ1*
shows *nprv F φ2*
⟨*proof*⟩

lemma *nprv_addImpLemmaE*:
assumes *prv (imp φ1 φ2)* **and** *nprv F φ1* **and** *nprv ((insert φ2) F) ψ*
and *F* $\subseteq$ *fmla finite F φ1* $\in$ *fmla φ2* $\in$ *fmla ψ* $\in$ *fmla*
shows *nprv F ψ*
⟨*proof*⟩

lemmas *nprv_addImpLemmaE1* = *nprv_addImpLemmaE*[*OF _ nprv_hyp _*, *simped*]
lemmas *nprv_addImpLemmaE2* = *nprv_addImpLemmaE*[*OF _ _ nprv_hyp*, *simped*]
lemmas *nprv_addImpLemmaE12* = *nprv_addImpLemmaE*[*OF _ nprv_hyp nprv_hyp*, *simped*]

lemma *nprv_addImp2LemmaI*:
assumes *prv (imp φ1 (imp φ2 φ3))*
and *F* $\subseteq$ *fmla finite F φ1* $\in$ *fmla φ2* $\in$ *fmla φ3* $\in$ *fmla*
and *nprv F φ1 nprv F φ2*
shows *nprv F φ3*
⟨*proof*⟩

lemma *nprv_addImp2LemmaE*:
assumes *prv (imp φ1 (imp φ2 φ3))* **and** *nprv F φ1* **and** *nprv F φ2* **and** *nprv ((insert φ3) F) ψ*
and *F* $\subseteq$ *fmla finite F φ1* $\in$ *fmla φ2* $\in$ *fmla φ3* $\in$ *fmla ψ* $\in$ *fmla*

shows *nprv F ψ*
⟨*proof*⟩

lemmas *nprv_addImp2LemmaE1 = nprv_addImp2LemmaE*[*OF _ nprv_hyp _ _*, *simped*]
lemmas *nprv_addImp2LemmaE2 = nprv_addImp2LemmaE*[*OF _ _ nprv_hyp _*, *simped*]
lemmas *nprv_addImp2LemmaE3 = nprv_addImp2LemmaE*[*OF _ _ _ nprv_hyp*, *simped*]
lemmas *nprv_addImp2LemmaE12 = nprv_addImp2LemmaE*[*OF _ nprv_hyp nprv_hyp _*, *simped*]
lemmas *nprv_addImp2LemmaE13 = nprv_addImp2LemmaE*[*OF _ nprv_hyp _ nprv_hyp*, *simped*]
lemmas *nprv_addImp2LemmaE23 = nprv_addImp2LemmaE*[*OF _ _ nprv_hyp nprv_hyp*, *simped*]
lemmas *nprv_addImp2LemmaE123 = nprv_addImp2LemmaE*[*OF _ nprv_hyp nprv_hyp nprv_hyp*, *simped*]

lemma *nprv_addImp3LemmaI*:
assumes *prv* (*imp φ1* (*imp φ2* (*imp φ3 φ4*)))
and *F* ⊆ *fmla finite F φ1* ∈ *fmla φ2* ∈ *fmla φ3* ∈ *fmla φ4* ∈ *fmla*
and *nprv F φ1 nprv F φ2 nprv F φ3*
shows *nprv F φ4*
⟨*proof*⟩

lemma *nprv_addImp3LemmaE*:
assumes *prv* (*imp φ1* (*imp φ2* (*imp φ3 φ4*))) **and** *nprv F φ1* **and** *nprv F φ2* **and** *nprv F φ3*
and *nprv* ((*insert φ4*) *F*) *ψ*
and *F* ⊆ *fmla finite F φ1* ∈ *fmla φ2* ∈ *fmla φ3* ∈ *fmla φ4* ∈ *fmla ψ* ∈ *fmla*
shows *nprv F ψ*
⟨*proof*⟩

lemmas *nprv_addImp3LemmaE1 = nprv_addImp3LemmaE*[*OF _ nprv_hyp _ _ _*, *simped*]
lemmas *nprv_addImp3LemmaE2 = nprv_addImp3LemmaE*[*OF _ _ nprv_hyp _ _*, *simped*]
lemmas *nprv_addImp3LemmaE3 = nprv_addImp3LemmaE*[*OF _ _ _ nprv_hyp _*, *simped*]
lemmas *nprv_addImp3LemmaE4 = nprv_addImp3LemmaE*[*OF _ _ _ _ nprv_hyp*, *simped*]
lemmas *nprv_addImp3LemmaE12 = nprv_addImp3LemmaE*[*OF _ nprv_hyp nprv_hyp _ _*, *simped*]
lemmas *nprv_addImp3LemmaE13 = nprv_addImp3LemmaE*[*OF _ nprv_hyp _ nprv_hyp _*, *simped*]
lemmas *nprv_addImp3LemmaE14 = nprv_addImp3LemmaE*[*OF _ nprv_hyp _ _ nprv_hyp*, *simped*]
lemmas *nprv_addImp3LemmaE23 = nprv_addImp3LemmaE*[*OF _ _ nprv_hyp nprv_hyp _*, *simped*]
lemmas *nprv_addImp3LemmaE24 = nprv_addImp3LemmaE*[*OF _ _ nprv_hyp _ nprv_hyp*, *simped*]
lemmas *nprv_addImp3LemmaE34 = nprv_addImp3LemmaE*[*OF _ _ _ nprv_hyp nprv_hyp*, *simped*]
lemmas *nprv_addImp3LemmaE123 = nprv_addImp3LemmaE*[*OF _ nprv_hyp nprv_hyp nprv_hyp _*, *simped*]
lemmas *nprv_addImp3LemmaE124 = nprv_addImp3LemmaE*[*OF _ nprv_hyp nprv_hyp _ nprv_hyp*, *simped*]
lemmas *nprv_addImp3LemmaE134 = nprv_addImp3LemmaE*[*OF _ nprv_hyp _ nprv_hyp nprv_hyp*, *simped*]
lemmas *nprv_addImp3LemmaE234 = nprv_addImp3LemmaE*[*OF _ _ nprv_hyp nprv_hyp nprv_hyp*, *simped*]
lemmas *nprv_addImp3LemmaE1234 = nprv_addImp3LemmaE*[*OF _ nprv_hyp nprv_hyp nprv_hyp nprv_hyp*, *simped*]

lemma *nprv_addDsjLemmaE*:
assumes *prv* (*dsj φ1 φ2*) **and** *nprv* (*insert φ1 F*) *ψ* **and** *nprv* ((*insert φ2*) *F*) *ψ*
and *F* ⊆ *fmla finite F φ1* ∈ *fmla φ2* ∈ *fmla ψ* ∈ *fmla*
shows *nprv F ψ*
⟨*proof*⟩

lemmas *nprv_addDsjLemmaE1 = nprv_addDsjLemmaE*[*OF _ nprv_hyp _*, *simped*]
lemmas *nprv_addDsjLemmaE2 = nprv_addDsjLemmaE*[*OF _ _ nprv_hyp*, *simped*]
lemmas *nprv_addDsjLemmaE12 = nprv_addDsjLemmaE*[*OF _ nprv_hyp nprv_hyp*, *simped*]

4.8 Rules for Equality

Reflexivity:

lemma *nprv_eql_reflI*: $F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow t \in trm \Longrightarrow nprv\ F\ (eql\ t\ t)$
⟨*proof*⟩

lemma *nprv_eq_eqlI*: $t1 = t2 \Longrightarrow F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow t1 \in trm \Longrightarrow nprv\ F\ (eql\ t1\ t2)$
⟨*proof*⟩

Symmetry:

lemmas *nprv_eql_symI* = *nprv_addImpLemmaI*[*OF prv_eql_sym*, *simped*, *rotated 4*]
lemmas *nprv_eql_symE* = *nprv_addImpLemmaE*[*OF prv_eql_sym*, *simped*, *rotated 2*]

lemmas *nprv_eql_symE0* = *nprv_eql_symE*[*OF nprv_hyp _*, *simped*]
lemmas *nprv_eql_symE1* = *nprv_eql_symE*[*OF _ nprv_hyp*, *simped*]
lemmas *nprv_eql_symE01* = *nprv_eql_symE*[*OF nprv_hyp nprv_hyp*, *simped*]

Transitivity:

lemmas *nprv_eql_transI* = *nprv_addImp2LemmaI*[*OF prv_eql_imp_trans*, *simped*, *rotated 5*]
lemmas *nprv_eql_transE* = *nprv_addImp2LemmaE*[*OF prv_eql_imp_trans*, *simped*, *rotated 3*]

lemmas *nprv_eql_transE0* = *nprv_eql_transE*[*OF nprv_hyp _ _*, *simped*]
lemmas *nprv_eql_transE1* = *nprv_eql_transE*[*OF _ nprv_hyp _*, *simped*]
lemmas *nprv_eql_transE2* = *nprv_eql_transE*[*OF _ _ nprv_hyp*, *simped*]
lemmas *nprv_eql_transE01* = *nprv_eql_transE*[*OF nprv_hyp nprv_hyp _*, *simped*]
lemmas *nprv_eql_transE02* = *nprv_eql_transE*[*OF nprv_hyp _ nprv_hyp*, *simped*]
lemmas *nprv_eql_transE12* = *nprv_eql_transE*[*OF _ nprv_hyp nprv_hyp*, *simped*]
lemmas *nprv_eql_transE012* = *nprv_eql_transE*[*OF nprv_hyp nprv_hyp nprv_hyp*, *simped*]

Substitutivity:

lemmas *nprv_eql_substI* =
nprv_addImp2LemmaI[*OF prv_eql_subst_trm_rev*, *simped*, *rotated 6*]
lemmas *nprv_eql_substE* = *nprv_addImp2LemmaE*[*OF prv_eql_subst_trm_rev*, *simped*, *rotated 4*]

lemmas *nprv_eql_substE0* = *nprv_eql_substE*[*OF nprv_hyp _ _*, *simped*]
lemmas *nprv_eql_substE1* = *nprv_eql_substE*[*OF _ nprv_hyp _*, *simped*]
lemmas *nprv_eql_substE2* = *nprv_eql_substE*[*OF _ _ nprv_hyp*, *simped*]
lemmas *nprv_eql_substE01* = *nprv_eql_substE*[*OF nprv_hyp nprv_hyp _*, *simped*]
lemmas *nprv_eql_substE02* = *nprv_eql_substE*[*OF nprv_hyp _ nprv_hyp*, *simped*]
lemmas *nprv_eql_substE12* = *nprv_eql_substE*[*OF _ nprv_hyp nprv_hyp*, *simped*]
lemmas *nprv_eql_substE012* = *nprv_eql_substE*[*OF nprv_hyp nprv_hyp nprv_hyp*, *simped*]

4.9 Other Rules

lemma *nprv_cnjH*:
$nprv\ (insert\ \varphi1\ (insert\ \varphi2\ F))\ \psi \Longrightarrow$
$F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow \varphi1 \in fmla \Longrightarrow \varphi2 \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ (insert\ (cnj\ \varphi1\ \varphi2)\ F)\ \psi$
⟨*proof*⟩

lemma *nprv_exi_commute*:
assumes [*simp*]: $x \in var\ y \in var\ \varphi \in fmla$
shows $nprv\ \{exi\ x\ (exi\ y\ \varphi)\}\ (exi\ y\ (exi\ x\ \varphi))$
⟨*proof*⟩

lemma *prv_exi_commute*:

assumes [*simp*]: $x \in var\ y \in var\ \varphi \in fmla$
shows $prv\ (imp\ (exi\ x\ (exi\ y\ \varphi))\ (exi\ y\ (exi\ x\ \varphi)))$
⟨*proof*⟩

end

4.10 Natural Deduction for the Exists-Unique Quantifier

context *Deduct_with_False_Disj_Rename*
begin

lemma *nprv_exuI*:
assumes *n1*: $nprv\ F\ (subst\ \varphi\ t\ x)$ **and** *n2*: $nprv\ (insert\ \varphi\ F)\ (eql\ (Var\ x)\ t)$
and *i*[*simp*]: $F \subseteq fmla\ finite\ F\ \varphi \in fmla\ t \in trm\ x \in var \quad x \notin FvarsT\ t$
and *u*: $x \notin (\bigcup \varphi \in F.\ Fvars\ \varphi)$
shows $nprv\ F\ (exu\ x\ \varphi)$
⟨*proof*⟩

lemma *nprv_exuI_var*:
assumes *n1*: $nprv\ F\ (subst\ \varphi\ t\ x)$ **and** *n2*: $nprv\ (insert\ (subst\ \varphi\ (Var\ y)\ x)\ F)\ (eql\ (Var\ y)\ t)$
and *i*[*simp*]: $F \subseteq fmla\ finite\ F\ \varphi \in fmla\ t \in trm\ x \in var$
$y \in var\ y \notin FvarsT\ t$ **and** *u*: $y \notin (\bigcup \varphi \in F.\ Fvars\ \varphi)$ **and** *yx*: $y = x \vee y \notin Fvars\ \varphi$
shows $nprv\ F\ (exu\ x\ \varphi)$
⟨*proof*⟩

This turned out to be the most useful introduction rule for arithmetic:

lemma *nprv_exuI_exi*:
assumes *n1*: $nprv\ F\ (exi\ x\ \varphi)$ **and** *n2*: $nprv\ (insert\ (subst\ \varphi\ (Var\ y)\ x)\ (insert\ \varphi\ F))\ (eql\ (Var\ y)\ (Var\ x))$
and *i*[*simp*]: $F \subseteq fmla\ finite\ F\ \varphi \in fmla\ x \in var\ y \in var\ y \neq x\ y \notin Fvars\ \varphi$
and *u*: $x \notin (\bigcup \varphi \in F.\ Fvars\ \varphi)\ y \notin (\bigcup \varphi \in F.\ Fvars\ \varphi)$
shows $nprv\ F\ (exu\ x\ \varphi)$
⟨*proof*⟩

lemma *prv_exu_imp_exi*:
assumes [*simp*]: $\varphi \in fmla\ x \in var$
shows $prv\ (imp\ (exu\ x\ \varphi)\ (exi\ x\ \varphi))$
⟨*proof*⟩

lemma *prv_exu_exi*:
assumes $x \in var\ \varphi \in fmla\ prv\ (exu\ x\ \varphi)$
shows $prv\ (exi\ x\ \varphi)$
⟨*proof*⟩

This is just exu behaving for elimination and forward like exi:

lemma *nprv_exuE_exi*:
assumes *n1*: $nprv\ F\ (exu\ x\ \varphi)$ **and** *n2*: $nprv\ (insert\ \varphi\ F)\ \psi$
and *i*[*simp*]: $F \subseteq fmla\ finite\ F\ \varphi \in fmla\ x \in var\ \psi \in fmla\ x \notin Fvars\ \psi$
and *u*: $x \notin (\bigcup \varphi \in F.\ Fvars\ \varphi)$
shows $nprv\ F\ \psi$
⟨*proof*⟩

lemma *nprv_exuF_exi*:
assumes *n1*: $exu\ x\ \varphi \in F$ **and** *n2*: $nprv\ (insert\ \varphi\ F)\ \psi$
and *i*[*simp*]: $F \subseteq fmla\ finite\ F\ \varphi \in fmla\ x \in var\ \psi \in fmla\ x \notin Fvars\ \psi$
and *u*: $x \notin (\bigcup \varphi \in F.\ Fvars\ \varphi)$
shows $nprv\ F\ \psi$

⟨*proof*⟩

lemma *prv_exu_uni*:
assumes [*simp*]: $\varphi \in fmla\ x \in var\ t1 \in trm\ t2 \in trm$
shows *prv* (*imp* (*exu x* φ) (*imp* (*subst* φ *t1 x*) (*imp* (*subst* φ *t2 x*) (*eql t1 t2*))))
⟨*proof*⟩

lemmas *nprv_exuE_uni* = *nprv_addImp3LemmaE*[*OF prv_exu_uni*,*simped*,*rotated 4*]
lemmas *nprv_exuF_uni* = *nprv_exuE_uni*[*OF nprv_hyp*,*simped*]

end — context *Deduct_with_False_Disj*

4.11 Eisbach Notation for Natural Deduction Proofs

The proof pattern will be: On a goal of the form *nprv F* φ, we apply a rule (usually an introduction, elimination, or cut/lemma-addition rule), then discharge the side-conditions with *auto*, ending up with zero, one or two goals of the same nprv-shape. This process is abstracted away in the Eisbach nrule method:

method *nrule* **uses** *r* = (*rule r*, *auto?*)

method *nrule2* **uses** *r* = (*rule r*, *auto?*)

Methods for chaining several nrule applications:

method *nprover2* **uses** *r1 r2* =
(−,(((*nrule r*: *r1*)*?*, (*nrule r*: *r2*)*?*) ; *fail*))
method *nprover3* **uses** *r1 r2 r3* =
(−,(((*nrule r*: *r1*)*?*, (*nrule r*: *r2*)*?*, (*nrule r*: *r3*)*?*) ; *fail*))
method *nprover4* **uses** *r1 r2 r3 r4* =
(−,(((*nrule r*: *r1*)*?*, (*nrule r*: *r2*)*?*, (*nrule r*: *r3*)*?*, (*nrule r*: *r4*)*?*) ; *fail*))
method *nprover5* **uses** *r1 r2 r3 r4 r5* =
(−,((*nrule r*: *r1*)*?*, (*nrule r*: *r2*)*?*, (*nrule r*: *r3*)*?*,
(*nrule r*: *r4*)*?*, (*nrule r*: *r5*)*?*) ; *fail*)
method *nprover6* **uses** *r1 r2 r3 r4 r5 r6* =
(−,((*nrule r*: *r1*)*?*, (*nrule r*: *r2*)*?*, (*nrule r*: *r3*)*?*,
(*nrule r*: *r4*)*?*, (*nrule r*: *r5*)*?*, (*nrule r*: *r6*)*?*) ; *fail*)

Chapter 5

Pseudo-Terms

Pseudo-terms are formulas that satisfy the exists-unique property on one of their variables.

5.1 Basic Setting

context *Generic_Syntax*
begin

We choose a specific variable, out, that will represent the "output" of pseudo-terms, i.e., the variable on which the exists-unique property holds:

abbreviation *out* ≡ *Variable 0*

Many facts will involve pseudo-terms with only one additional "input" variable, inp:

abbreviation *inp* ≡ *Variable* (*Suc 0*)

lemma *out_inp_distinct*[*simp*]:
out ≠ *inp inp* ≠ *out*
out ≠ *xx out* ≠ *yy yy* ≠ *out out* ≠ *zz zz* ≠ *out out* ≠ *xx′ xx′* ≠ *out*
out ≠ *yy′ yy′* ≠ *out out* ≠ *zz′ zz′* ≠ *out*
inp ≠ *xx inp* ≠ *yy yy* ≠ *inp inp* ≠ *zz zz* ≠ *inp inp* ≠ *xx′ xx′* ≠ *inp*
inp ≠ *yy′ yy′* ≠ *inp inp* ≠ *zz′ zz′* ≠ *inp*
⟨*proof*⟩

end

context *Deduct_with_False_Disj_Rename*
begin

Pseudo-terms over the first $n + 1$ variables, i.e., having n input variables (Variable 1 to Variable n), and an output variable, out (which is an abbreviation for Variable 0).

definition *ptrm* :: *nat* ⇒ *'fmla set* **where**
ptrm n ≡ {*σ* ∈ *fmla* . *Fvars σ* = *Variable* ‘ {*0..n*} ∧ *prv* (*exu out σ*)}

lemma *ptrm*[*intro*,*simp*]: *σ* ∈ *ptrm n* ⟹ *σ* ∈ *fmla*
⟨*proof*⟩

lemma *ptrm_1_Fvars*[*simp*]: *σ* ∈ *ptrm* (*Suc 0*) ⟹ *Fvars σ* = {*out*,*inp*}
⟨*proof*⟩

lemma *ptrm_prv_exu*: $\sigma \in ptrm\ n \Longrightarrow prv\ (exu\ out\ \sigma)$
⟨*proof*⟩

lemma *ptrm_prv_exi*: $\sigma \in ptrm\ n \Longrightarrow prv\ (exi\ out\ \sigma)$
⟨*proof*⟩

lemma *nprv_ptrmE_exi*:
$\sigma \in ptrm\ n \Longrightarrow nprv\ (insert\ \sigma\ F)\ \psi \Longrightarrow$
$F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow$
$\psi \in fmla \Longrightarrow out \notin Fvars\ \psi \Longrightarrow out \notin \bigcup\ (Fvars\ `\ F) \Longrightarrow nprv\ F\ \psi$
⟨*proof*⟩

lemma *nprv_ptrmE_uni*:
$\sigma \in ptrm\ n \Longrightarrow nprv\ F\ (subst\ \sigma\ t1\ out) \Longrightarrow nprv\ F\ (subst\ \sigma\ t2\ out) \Longrightarrow$
$nprv\ (insert\ (eql\ t1\ t2)\ F)\ \psi \Longrightarrow$
$F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow \psi \in fmla \Longrightarrow t1 \in trm \Longrightarrow t2 \in trm$
$\Longrightarrow nprv\ F\ \psi$
⟨*proof*⟩

lemma *nprv_ptrmE_uni0*:
$\sigma \in ptrm\ n \Longrightarrow nprv\ F\ \sigma \Longrightarrow nprv\ F\ (subst\ \sigma\ t\ out) \Longrightarrow$
$nprv\ (insert\ (eql\ (Var\ out)\ t)\ F)\ \psi \Longrightarrow$
$F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow \psi \in fmla \Longrightarrow t \in trm$
$\Longrightarrow nprv\ F\ \psi$
⟨*proof*⟩

lemma *nprv_ptrmE0_uni0*:
$\sigma \in ptrm\ n \Longrightarrow \sigma \in F \Longrightarrow nprv\ F\ (subst\ \sigma\ t\ out) \Longrightarrow$
$nprv\ (insert\ (eql\ (Var\ out)\ t)\ F)\ \psi \Longrightarrow$
$F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow \psi \in fmla \Longrightarrow t \in trm$
$\Longrightarrow nprv\ F\ \psi$
⟨*proof*⟩

5.2 The ∀-∃ Equivalence

There are two natural ways to state that (unique) "output" of a pseudo-term σ satisfies a property φ: (1) using ∃: there exists an "out" such that σ and φ hold for it; (2) using ∀: for all "out" such that σ holds for it, φ holds for it as well.

We prove the well-known fact that these two ways are equivalent. (Intuitionistic logic suffice to prove that.)

lemma *ptrm_nprv_exi*:
assumes σ: $\sigma \in ptrm\ n$ **and** [*simp*]: $\varphi \in fmla$
shows $nprv\ \{\sigma,\ exi\ out\ (cnj\ \sigma\ \varphi)\}\ \varphi$
⟨*proof*⟩

lemma *ptrm_nprv_exi_all*:
assumes σ: $\sigma \in ptrm\ n$ **and** [*simp*]: $\varphi \in fmla$
shows $nprv\ \{exi\ out\ (cnj\ \sigma\ \varphi)\}\ (all\ out\ (imp\ \sigma\ \varphi))$
⟨*proof*⟩

lemma *ptrm_prv_exi_imp_all*:
assumes σ: $\sigma \in ptrm\ n$ **and** [*simp*]: $\varphi \in fmla$
shows $prv\ (imp\ (exi\ out\ (cnj\ \sigma\ \varphi))\ (all\ out\ (imp\ \sigma\ \varphi)))$
⟨*proof*⟩

lemma *ptrm_nprv_all_imp_exi*:

assumes σ: $\sigma \in ptrm\ n$ **and** [*simp*]: $\varphi \in fmla$
shows *nprv* {*all out* (*imp* σ φ)} (*exi out* (*cnj* σ φ))
⟨*proof*⟩

lemma *ptrm_prv_all_imp_exi*:
assumes σ: $\sigma \in ptrm\ n$ **and** [*simp*]: $\varphi \in fmla$
shows *prv* (*imp* (*all out* (*imp* σ φ)) (*exi out* (*cnj* σ φ)))
⟨*proof*⟩

end — context *Deduct_with_False_Disj_Rename*

5.3 Instantiation

We define the notion of instantiating the "inp" variable of a formula (in particular, of a pseudo-term): – first with a term); – then with a pseudo-term.

5.3.1 Instantiation with terms

Instantiation with terms is straightforward using substitution. In the name of the operator, the suffix "Inp" is a reminder that we instantiate φ on its variable "inp".

context *Generic_Syntax*
begin

definition *instInp* :: *'fmla* ⇒ *'trm* ⇒ *'fmla* **where**
instInp φ $t \equiv$ *subst* φ t *inp*

lemma *instInp_fmla*[*simp*,*intro*]:
assumes $\varphi \in fmla$ **and** $t \in trm$
shows *instInp* φ $t \in fmla$
⟨*proof*⟩

lemma *Fvars_instInp*[*simp*,*intro*]:
assumes $\varphi \in fmla$ **and** $t \in trm$ *Fvars* $\varphi = \{inp\}$
shows *Fvars* (*instInp* φ t) = *FvarsT* t
⟨*proof*⟩

end — context *Generic_Syntax*

context *Deduct_with_False_Disj_Rename*
begin

lemma *Fvars_instInp_ptrm_1*[*simp*,*intro*]:
assumes τ: $\tau \in ptrm\ (Suc\ 0)$ **and** $t \in trm$
shows *Fvars* (*instInp* τ t) = *insert out* (*FvarsT* t)
⟨*proof*⟩

lemma *instInp*:
assumes τ: $\tau \in ptrm\ (Suc\ 0)$ **and** [*simp*]: $t \in trm$
and [*simp*]: *FvarsT* t = *Variable* ‘ $\{(Suc\ 0)..n\}$
shows *instInp* τ $t \in ptrm\ n$
⟨*proof*⟩

lemma *instInp_0*:
assumes τ: $\tau \in ptrm\ (Suc\ 0)$ **and** $t \in trm$ **and** *FvarsT* $t = \{\}$
shows *instInp* τ $t \in ptrm\ 0$

⟨*proof*⟩

lemma *instInp_1*:
assumes τ: $\tau \in ptrm\ (Suc\ 0)$ **and** $t \in trm$ **and** $FvarsT\ t = \{inp\}$
shows $instInp\ \tau\ t \in ptrm\ (Suc\ 0)$
⟨*proof*⟩

5.3.2 Instantiation with pseudo-terms

Instantiation of a formula φ with a pseudo-term τ yields a formula that could be casually written $\varphi(\tau)$. It states the existence of an output *zz* of τ on which φ holds. Instead of $\varphi(\tau)$, we write $instInpP\ \varphi\ n\ \tau$ where n is the number of input variables of τ. In the name *instInpP*, *Inp* is as before a reminder that we instantiate φ on its variable "inp" and the suffix "P" stands for "Pseudo".

definition $instInpP :: 'fmla \Rightarrow nat \Rightarrow 'fmla \Rightarrow 'fmla$ **where**
$instInpP\ \varphi\ n\ \tau \equiv let\ zz = Variable\ (Suc\ (Suc\ n))\ in$
  $exi\ zz\ (cnj\ (subst\ \tau\ (Var\ zz)\ out)\ (subst\ \varphi\ (Var\ zz)\ inp))$

lemma *instInpP_fmla*[*simp*, *intro*]:
assumes $\varphi \in fmla$ **and** $\tau \in fmla$
shows $instInpP\ \varphi\ n\ \tau \in fmla$
⟨*proof*⟩

lemma *Fvars_instInpP*[*simp*]:
assumes $\varphi \in fmla$ **and** τ: $\tau \in ptrm\ n$ $Variable\ (Suc\ (Suc\ n)) \notin Fvars\ \varphi$
shows $Fvars\ (instInpP\ \varphi\ n\ \tau) = Fvars\ \varphi - \{inp\} \cup Variable\ `\ \{(Suc\ 0)..n\}$
⟨*proof*⟩

lemma *Fvars_instInpP2*[*simp*]:
assumes $\varphi \in fmla$ **and** τ: $\tau \in ptrm\ n$ **and** $Fvars\ \varphi \subseteq \{inp\}$
shows $Fvars\ (instInpP\ \varphi\ n\ \tau) = Fvars\ \varphi - \{inp\} \cup Variable\ `\ \{(Suc\ 0)..n\}$
⟨*proof*⟩

5.3.3 Closure and compositionality properties of instantiation

Instantiating a 1-pseudo-term with an n-pseudo-term yields an n pseudo-term:

lemma *instInpP1*[*simp*,*intro*]:
assumes σ: $\sigma \in ptrm\ (Suc\ 0)$ **and** τ: $\tau \in ptrm\ n$
shows $instInpP\ \sigma\ n\ \tau \in ptrm\ n$
⟨*proof*⟩

Term and pseudo-term instantiation compose smoothly:

lemma *instInp_instInpP*:
assumes φ: $\varphi \in fmla\ Fvars\ \varphi \subseteq \{inp\}$ **and** τ: $\tau \in ptrm\ (Suc\ 0)$
and $t \in trm$ **and** $FvarsT\ t = \{\}$
shows $instInp\ (instInpP\ \varphi\ (Suc\ 0)\ \tau)\ t = instInpP\ \varphi\ 0\ (instInp\ \tau\ t)$
⟨*proof*⟩

Pseudo-term instantiation also composes smoothly with itself:

lemma *nprv_instInpP_compose*:
assumes [*simp*]: $\chi \in fmla\ Fvars\ \chi = \{inp\}$
and σ[*simp*]: $\sigma \in ptrm\ (Suc\ 0)$ **and** τ[*simp*]: $\tau \in ptrm\ 0$
shows $nprv\ \{instInpP\ (instInpP\ \chi\ (Suc\ 0)\ \sigma)\ 0\ \tau\}$
  $(instInpP\ \chi\ 0\ (instInpP\ \sigma\ 0\ \tau))$ (**is** *?A*)
and
  $nprv\ \{instInpP\ \chi\ 0\ (instInpP\ \sigma\ 0\ \tau)\}$
  $(instInpP\ (instInpP\ \chi\ (Suc\ 0)\ \sigma)\ 0\ \tau)$ (**is** *?B*)

⟨*proof*⟩

lemma *prv_instInpP_compose*:
assumes [*simp*]: $\chi \in$ *fmla Fvars* $\chi = \{inp\}$
and σ[*simp*]: $\sigma \in$ *ptrm* (*Suc 0*) **and** τ[*simp*]: $\tau \in$ *ptrm 0*
shows *prv* (*imp* (*instInpP* (*instInpP* χ (*Suc 0*) σ) *0* τ)
(*instInpP* χ *0* (*instInpP* σ *0* τ))) (**is** *?A*)
and
prv (*imp* (*instInpP* χ *0* (*instInpP* σ *0* τ))
(*instInpP* (*instInpP* χ (*Suc 0*) σ) *0* τ)) (**is** *?B*)
and
prv (*eqv* (*instInpP* (*instInpP* χ (*Suc 0*) σ) *0* τ)
(*instInpP* χ *0* (*instInpP* σ *0* τ))) (**is** *?C*)
⟨*proof*⟩

5.4 Equality between Pseudo-Terms and Terms

Casually, the equality between a pseudo-term τ and a term t can be written as $\vdash \tau = t$. This is in fact the (provability of) the instantiation of τ with t on τ's output variable out. Indeed, this formula says that the unique entity denoted by τ is exactly t.

definition *prveqlPT* :: *'fmla* ⇒ *'trm* ⇒ *bool* **where**
prveqlPT τ *t* ≡ *prv* (*subst* τ *t out*)

We prove that term–pseudo-term equality indeed acts like an equality, in that it satisfies the substitutivity principle (shown only in the particular case of formula-input instantiation).

lemma *prveqlPT_nprv_instInp_instInpP*:
assumes[*simp*]: $\varphi \in$ *fmla* **and** *f*: *Fvars* $\varphi \subseteq \{inp\}$ **and** τ: $\tau \in$ *ptrm 0*
and [*simp*]: *t* ∈ *trm FvarsT t* = {}
and τt: *prveqlPT* τ *t*
shows *nprv* {*instInpP* φ *0* τ} (*instInp* φ *t*)
⟨*proof*⟩

lemma *prveqlPT_prv_instInp_instInpP*:
assumes$\varphi \in$ *fmla* **and** *f*: *Fvars* $\varphi \subseteq \{inp\}$ **and** τ: $\tau \in$ *ptrm 0*
and *t* ∈ *trm FvarsT t* = {}
and τt: *prveqlPT* τ *t*
shows *prv* (*imp* (*instInpP* φ *0* τ) (*instInp* φ *t*))
⟨*proof*⟩

lemma *prveqlPT_nprv_instInpP_instInp*:
assumes[*simp*]: $\varphi \in$ *fmla* **and** *f*: *Fvars* $\varphi \subseteq \{inp\}$ **and** τ: $\tau \in$ *ptrm 0*
and [*simp*]: *t* ∈ *trm FvarsT t* = {}
and τt: *prveqlPT* τ *t*
shows *nprv* {*instInp* φ *t*} (*instInpP* φ *0* τ)
⟨*proof*⟩

lemma *prveqlPT_prv_instInpP_instInp*:
assumes$\varphi \in$ *fmla* **and** *f*: *Fvars* $\varphi \subseteq \{inp\}$ **and** τ: $\tau \in$ *ptrm 0*
and *t* ∈ *trm FvarsT t* = {}
and τt: *prveqlPT* τ *t*
shows *prv* (*imp* (*instInp* φ *t*) (*instInpP* φ *0* τ))
⟨*proof*⟩

lemma *prveqlPT_prv_instInp_eqv_instInpP*:
assumes$\varphi \in$ *fmla* **and** *f*: *Fvars* $\varphi \subseteq \{inp\}$ **and** τ: $\tau \in$ *ptrm 0*
and *t* ∈ *trm FvarsT t* = {}
and τt: *prveqlPT* τ *t*

shows *prv* (*eqv* (*instInpP* φ *0* τ) (*instInp* φ *t*))
⟨*proof*⟩

end — context *Deduct_with_False_Disj_Rename*

Chapter 6

Truth in a Standard Model

Abstract notion of standard model and truth.

First some minimal assumptions, involving implication, negation and (universal and existential) quantification:

locale *Minimal_Truth* =
Syntax_with_Numerals_and_Connectives_False_Disj
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
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*
+
— The notion of truth for sentences:
fixes *isTrue* :: *'fmla* ⇒ *bool*
assumes
not_isTrue_fls: ¬ *isTrue fls*
and
isTrue_imp:
⋀*φ ψ*. *φ* ∈ *fmla* ⟹ *ψ* ∈ *fmla* ⟹ *Fvars φ* = {} ⟹ *Fvars ψ* = {} ⟹
  isTrue φ ⟹ *isTrue* (*imp φ ψ*) ⟹ *isTrue ψ*
and
isTrue_all:
⋀*x φ*. *x* ∈ *var* ⟹ *φ* ∈ *fmla* ⟹ *Fvars φ* = {*x*} ⟹
  (∀ *n* ∈ *num*. *isTrue* (*subst φ n x*)) ⟹ *isTrue* (*all x φ*)
and
isTrue_exi:
⋀*x φ*. *x* ∈ *var* ⟹ *φ* ∈ *fmla* ⟹ *Fvars φ* = {*x*} ⟹
  isTrue (*exi x φ*) ⟹ (∃ *n* ∈ *num*. *isTrue* (*subst φ n x*))
and
isTrue_neg:
⋀*φ*. *φ* ∈ *fmla* ⟹ *Fvars φ* = {} ⟹
  isTrue φ ∨ *isTrue* (*neg φ*)
begin

lemma *isTrue_neg_excl*:

$\varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow$
$isTrue\ \varphi \Longrightarrow isTrue\ (neg\ \varphi) \Longrightarrow False$
⟨*proof*⟩

lemma *isTrue_neg_neg*:
assumes $\varphi \in fmla\ Fvars\ \varphi = \{\}$
and *isTrue* (*neg* (*neg* φ))
shows *isTrue* φ
⟨*proof*⟩

end — context *Minimal_Truth*

locale *Minimal_Truth_Soundness* =
Minimal_Truth
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
isTrue
+
Deduct_with_False_Disj
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
prv
for
var :: *'var set* **and** *trm* :: *'trm set* **and** *fmla* :: *'fmla set*
and *Var FvarsT substT Fvars subst*
and *eql cnj imp all exi*
and *fls*
and *dsj*
and *num*
and *prv*
and *isTrue*
+
assumes
— We assume soundness of the provability for sentences (w.r.t. truth):
sound_isTrue: $\bigwedge\varphi.\ \varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow prv\ \varphi \Longrightarrow isTrue\ \varphi$
begin

For sound theories, consistency is a fact rather than a hypothesis:

lemma *consistent*: *consistent*
⟨*proof*⟩

lemma *prv_neg_excl*:
$\varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow prv\ \varphi \Longrightarrow prv\ (neg\ \varphi) \Longrightarrow False$
⟨*proof*⟩

lemma *prv_imp_implies_isTrue*:
assumes [*simp*]: $\varphi \in fmla\ \chi \in fmla\ Fvars\ \varphi = \{\}\ Fvars\ \chi = \{\}$
and *p*: *prv* (*imp* $\varphi\ \chi$) **and** *i*: *isTrue* φ
shows *isTrue* χ
⟨*proof*⟩

Sound theories are not only consistent, but also ω-consistent (in the strong, intuitionistic sense):

lemma *ωconsistent*: *ωconsistent*
⟨*proof*⟩

lemma *ωconsistentStd1*: *ωconsistentStd1*
⟨*proof*⟩

lemma *ωconsistentStd2*: *ωconsistentStd2*
⟨*proof*⟩

end — context *Minimal_Truth_Soundness*

Chapter 7

Arithmetic Constructs

Less genereric syntax, more committed towards embedding arithmetics

(An embedding of) the syntax of arithmetic, obtained by adding plus and times

locale *Syntax_Arith_aux* =
Syntax_with_Connectives_Rename
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
+
Syntax_with_Numerals_and_Connectives_False_Disj
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
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*
+
fixes
zer :: *'trm*
and
suc :: *'trm* ⇒ *'trm*
and
pls :: *'trm* ⇒ *'trm* ⇒ *'trm*
and
tms :: *'trm* ⇒ *'trm* ⇒ *'trm*
assumes
Fvars_zero[*simp*,*intro*!]: *FvarsT zer* = {}
and
substT_zer[*simp*]: ⋀ *t x*. *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹
substT zer t x = *zer*
and
suc[*simp*]: ⋀*t*. *t* ∈ *trm* ⟹ *suc t* ∈ *trm*
and
FvarsT_suc[*simp*]: ⋀ *t*. *t* ∈ *trm* ⟹
FvarsT (*suc t*) = *FvarsT t*
and
substT_suc[*simp*]: ⋀ *t1 t x*. *t1* ∈ *trm* ⟹ *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹

substT (*suc t1*) *t x* = *suc* (*substT t1 t x*)
and
pls[*simp*]: ⋀ *t1 t2*. *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹ *pls t1 t2* ∈ *trm*
and
Fvars_pls[*simp*]: ⋀ *t1 t2*. *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹
FvarsT (*pls t1 t2*) = *FvarsT t1* ∪ *FvarsT t2*
and
substT_pls[*simp*]: ⋀ *t1 t2 t x*. *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹ *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹
substT (*pls t1 t2*) *t x* = *pls* (*substT t1 t x*) (*substT t2 t x*)
and
tms[*simp*]: ⋀ *t1 t2*. *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹ *tms t1 t2* ∈ *trm*
and
Fvars_tms[*simp*]: ⋀ *t1 t2*. *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹
FvarsT (*tms t1 t2*) = *FvarsT t1* ∪ *FvarsT t2*
and
substT_tms[*simp*]: ⋀ *t1 t2 t x*. *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹ *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹
substT (*tms t1 t2*) *t x* = *tms* (*substT t1 t x*) (*substT t2 t x*)
begin

The embedding of numbers into our abstract notion of numerals (not required to be surjective)

fun *Num* :: *nat* ⇒ *'trm* **where**
Num 0 = *zer*
|*Num* (*Suc n*) = *suc* (*Num n*)

end — context *Syntax_Arith_aux*

locale *Syntax_Arith* =
Syntax_Arith_aux
var trm fmla Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
zer suc pls tms
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*
zer suc pls tms
+
assumes
— We assume that numbers are the only numerals:
num_Num: *num* = *range Num*
begin

lemma *Num*[*simp*,*intro*!]: *Num n* ∈ *num*
⟨*proof*⟩

lemma *FvarsT_Num*[*simp*]: *FvarsT* (*Num n*) = {}
⟨*proof*⟩

lemma *substT_Num*[*simp*]: *x* ∈ *var* ⟹ *t* ∈ *trm* ⟹ *substT* (*Num n*) *t x* = *Num n*
⟨*proof*⟩

lemma *zer*[*simp*,*intro*!]: $zer \in num$
and *suc_num*[*simp*]: $\bigwedge n.\ n \in num \Longrightarrow suc\ n \in num$
⟨*proof*⟩

7.1 Arithmetic Terms

Arithmetic terms are inductively defined to contain the numerals and the variables and be closed under the arithmetic operators:

inductive_set *atrm* :: *'trm set* **where**
atrm_num[*simp*]: $n \in num \Longrightarrow n \in atrm$
|*atrm_Var*[*simp*,*intro*]: $x \in var \Longrightarrow Var\ x \in atrm$
|*atrm_suc*[*simp*,*intro*]: $t \in atrm \Longrightarrow suc\ t \in atrm$
|*atrm_pls*[*simp*,*intro*]: $t \in atrm \Longrightarrow t' \in atrm \Longrightarrow pls\ t\ t' \in atrm$
|*atrm_tms*[*simp*,*intro*]: $t \in atrm \Longrightarrow t' \in atrm \Longrightarrow tms\ t\ t' \in atrm$

lemma *atrm_imp_trm*[*simp*]: **assumes** $t \in atrm$ **shows** $t \in trm$
⟨*proof*⟩

lemma *atrm_trm*: $atrm \subseteq trm$
⟨*proof*⟩

lemma *zer_atrm*[*simp*]: $zer \in atrm$ ⟨*proof*⟩

lemma *Num_atrm*[*simp*]: $Num\ n \in atrm$
⟨*proof*⟩

lemma *substT_atrm*[*simp*]:
assumes $r \in atrm$ **and** $x \in var$ **and** $t \in atrm$
shows $substT\ r\ t\ x \in atrm$
⟨*proof*⟩

Whereas we did not assume the rich set of formula-substitution properties to hold for all terms, we can prove that these properties hold for arithmetic terms.

Properties for arithmetic terms corresponding to the axioms for formulas:

lemma *FvarsT_substT*:
assumes $s \in atrm\ t \in trm\ x \in var$
shows $FvarsT\ (substT\ s\ t\ x) = (FvarsT\ s - \{x\}) \cup (if\ x \in FvarsT\ s\ then\ FvarsT\ t\ else\ \{\})$
⟨*proof*⟩

lemma *substT_compose_eq_or*:
assumes $s \in atrm\ t1 \in trm\ t2 \in trm\ x1 \in var\ x2 \in var$
and $x1 = x2 \vee x2 \notin FvarsT\ s$
shows $substT\ (substT\ s\ t1\ x1)\ t2\ x2 = substT\ s\ (substT\ t1\ t2\ x2)\ x1$
⟨*proof*⟩

lemma *substT_compose_diff*:
assumes $s \in atrm\ t1 \in trm\ t2 \in trm\ x1 \in var\ x2 \in var$
and $x1 \neq x2\ x1 \notin FvarsT\ t2$
shows $substT\ (substT\ s\ t1\ x1)\ t2\ x2 = substT\ (substT\ s\ t2\ x2)\ (substT\ t1\ t2\ x2)\ x1$
⟨*proof*⟩

lemma *substT_same_Var*[*simp*]:
assumes $s \in atrm\ x \in var$
shows $substT\ s\ (Var\ x)\ x = s$
⟨*proof*⟩

... and corresponding to some corollaries we proved for formulas (with essentially the same proofs):

lemma *in_FvarsT_substTD*:
$y \in FvarsT\ (substT\ r\ t\ x) \Longrightarrow r \in atrm \Longrightarrow t \in trm \Longrightarrow x \in var$
$\Longrightarrow y \in (FvarsT\ r - \{x\}) \cup (if\ x \in FvarsT\ r\ then\ FvarsT\ t\ else\ \{\})$
⟨*proof*⟩

lemma *substT_compose_same*:
$\bigwedge s\ t1\ t2\ x.\ s \in atrm \Longrightarrow t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow x \in var \Longrightarrow$
$substT\ (substT\ s\ t1\ x)\ t2\ x = substT\ s\ (substT\ t1\ t2\ x)\ x$
⟨*proof*⟩

lemma *substT_substT*[*simp*]:
assumes *s*[*simp*]: $s \in atrm$ **and** *t*[*simp*]:$t \in trm$ **and** *x*[*simp*]:$x \in var$ **and** *y*[*simp*]:$y \in var$
assumes *yy*: $x \neq y\ y \notin FvarsT\ s$
shows $substT\ (substT\ s\ (Var\ y)\ x)\ t\ y = substT\ s\ t\ x$
⟨*proof*⟩

lemma *substT_comp*:
$\bigwedge x\ y\ s\ t.\ s \in atrm \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow y \in var \Longrightarrow$
$x \neq y \Longrightarrow y \notin FvarsT\ t \Longrightarrow$
$substT\ (substT\ s\ (Var\ x)\ y)\ t\ x = substT\ (substT\ s\ t\ x)\ t\ y$
⟨*proof*⟩

Now the corresponding development of parallel substitution for arithmetic terms:

lemma *rawpsubstT_atrm*[*simp*,*intro*]:
assumes $r \in atrm$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq atrm$
shows $rawpsubstT\ r\ txs \in atrm$
⟨*proof*⟩

lemma *psubstT_atrm*[*simp*,*intro*]:
assumes $r \in atrm$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq atrm$
shows $psubstT\ r\ txs \in atrm$
⟨*proof*⟩

lemma *Fvars_rawpsubst_su*:
assumes $r \in atrm$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq atrm$
shows $FvarsT\ (rawpsubstT\ r\ txs) \subseteq$
$(FvarsT\ r - snd\ `\ (set\ txs)) \cup (\bigcup\ \{FvarsT\ t \mid t\ x\ .\ (t,x) \in set\ txs\})$
⟨*proof*⟩

lemma *in_FvarsT_rawpsubstT_imp*:
assumes $y \in FvarsT\ (rawpsubstT\ r\ txs)$
and $r \in atrm$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq atrm$
shows $(y \in FvarsT\ r - snd\ `\ (set\ txs)) \vee$
$(y \in \bigcup\ \{\ FvarsT\ t \mid t\ x\ .\ (t,x) \in set\ txs\})$
⟨*proof*⟩

lemma *FvarsT_rawpsubstT*:
assumes $r \in atrm$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq atrm$
and $distinct\ (map\ snd\ txs)$ **and** $\forall\ x \in snd\ `\ (set\ txs).\ \forall\ t \in fst\ `\ (set\ txs).\ x \notin FvarsT\ t$
shows $FvarsT\ (rawpsubstT\ r\ txs) =$
$(FvarsT\ r - snd\ `\ (set\ txs)) \cup$
$(\bigcup\ \{if\ x \in FvarsT\ r\ then\ FvarsT\ t\ else\ \{\} \mid t\ x\ .\ (t,x) \in set\ txs\})$
⟨*proof*⟩

lemma *in_FvarsT_rawpsubstTD*:
assumes $y \in FvarsT\ (rawpsubstT\ r\ txs)$
and $r \in atrm$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq atrm$

and *distinct* (*map snd txs*) **and** $\forall\ x \in snd\ `\ (set\ txs).\ \forall\ t \in fst\ `\ (set\ txs).\ x \notin FvarsT\ t$
shows $(y \in FvarsT\ r - snd\ `\ (set\ txs)) \vee$
$(y \in \bigcup\ \{if\ x \in FvarsT\ r\ then\ FvarsT\ t\ else\ \{\} \mid t\ x\ .\ (t,x) \in set\ txs\})$
⟨*proof*⟩

lemma *FvarsT_psubstT*:
assumes $r \in atrm$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq atrm$
and *distinct* (*map snd txs*)
shows $FvarsT\ (psubstT\ r\ txs) =$
$(FvarsT\ r - snd\ `\ (set\ txs)) \cup$
$(\bigcup\ \{if\ x \in FvarsT\ r\ then\ FvarsT\ t\ else\ \{\} \mid t\ x\ .\ (t,x) \in set\ txs\})$
⟨*proof*⟩

lemma *in_FvarsT_psubstTD*:
assumes $y \in FvarsT\ (psubstT\ r\ txs)$
and $r \in atrm$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq atrm$
and *distinct* (*map snd txs*)
shows $y \in (FvarsT\ r - snd\ `\ (set\ txs)) \cup$
$(\bigcup\ \{if\ x \in FvarsT\ r\ then\ FvarsT\ t\ else\ \{\} \mid t\ x\ .\ (t,x) \in set\ txs\})$
⟨*proof*⟩

lemma *substT2_fresh_switch*:
assumes $r \in atrm\ t \in trm\ s \in trm\ x \in var\ y \in var$
and $x \neq y\ x \notin FvarsT\ s\ y \notin FvarsT\ t$
shows $substT\ (substT\ r\ s\ y)\ t\ x = substT\ (substT\ r\ t\ x)\ s\ y$ (**is** *?L = ?R*)
⟨*proof*⟩

lemma *rawpsubst2_fresh_switch*:
assumes $r \in atrm\ t \in trm\ s \in trm\ x \in var\ y \in var$
and $x \neq y\ x \notin FvarsT\ s\ y \notin FvarsT\ t$
shows $rawpsubstT\ r\ ([(s,y),(t,x)]) = rawpsubstT\ r\ ([(t,x),(s,y)])$
⟨*proof*⟩

lemma *rawpsubstT_compose*:
assumes $t \in trm$ **and** $snd\ `\ (set\ txs1) \subseteq var$ **and** $fst\ `\ (set\ txs1) \subseteq atrm$
and $snd\ `\ (set\ txs2) \subseteq var$ **and** $fst\ `\ (set\ txs2) \subseteq atrm$
shows $rawpsubstT\ (rawpsubstT\ t\ txs1)\ txs2 = rawpsubstT\ t\ (txs1\ @\ txs2)$
⟨*proof*⟩

lemma *rawpsubstT_subst_fresh_switch*:
assumes $r \in atrm\ snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq atrm$
and $\forall\ x \in snd\ `\ (set\ txs).\ x \notin FvarsT\ s$
and $\forall\ t \in fst\ `\ (set\ txs).\ y \notin FvarsT\ t$
and *distinct* (*map snd txs*)
and $s \in atrm$ **and** $y \in var\ y \notin snd\ `\ (set\ txs)$
shows $rawpsubstT\ (substT\ r\ s\ y)\ txs = rawpsubstT\ r\ (txs\ @\ [(s,y)])$
⟨*proof*⟩

lemma *substT_rawpsubstT_fresh_switch*:
assumes $r \in atrm\ snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq atrm$
and $\forall\ x \in snd\ `\ (set\ txs).\ x \notin FvarsT\ s$
and $\forall\ t \in fst\ `\ (set\ txs).\ y \notin FvarsT\ t$
and *distinct* (*map snd txs*)
and $s \in atrm$ **and** $y \in var\ y \notin snd\ `\ (set\ txs)$
shows $substT\ (rawpsubstT\ r\ txs)\ s\ y = rawpsubstT\ r\ ((s,y)\ \#\ txs)$
⟨*proof*⟩

lemma *rawpsubstT_compose_freshVar*:
assumes $r \in atrm\ snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq atrm$
and $distinct\ (map\ snd\ txs)$
and $\bigwedge i\ j.\ i < j \Longrightarrow j < length\ txs \Longrightarrow snd\ (txs!j) \notin FvarsT\ (fst\ (txs!i))$
and *us_facts*: $set\ us \subseteq var$
$set\ us \cap FvarsT\ r = \{\}$
$set\ us \cap \bigcup\ (FvarsT\ `\ (fst\ `\ (set\ txs))) = \{\}$
$set\ us \cap snd\ `\ (set\ txs) = \{\}$
$length\ us = length\ txs$
$distinct\ us$
shows $rawpsubstT\ (rawpsubstT\ r\ (zip\ (map\ Var\ us)\ (map\ snd\ txs)))\ (zip\ (map\ fst\ txs)\ us) = rawpsubstT\ r\ txs$
⟨*proof*⟩

lemma *rawpsubstT_compose_freshVar2_aux*:
assumes *r*[*simp*]: $r \in atrm$
and *ts*: $set\ ts \subseteq atrm$
and *xs*: $set\ xs \subseteq var\ \ distinct\ xs$
and *us_facts*: $set\ us \subseteq var\ \ distinct\ us$
$set\ us \cap FvarsT\ r = \{\}$
$set\ us \cap \bigcup\ (FvarsT\ `\ (set\ ts)) = \{\}$
$set\ us \cap set\ xs = \{\}$
and *vs_facts*: $set\ vs \subseteq var\ \ distinct\ vs$
$set\ vs \cap FvarsT\ r = \{\}$
$set\ vs \cap \bigcup\ (FvarsT\ `\ (set\ ts)) = \{\}$
$set\ vs \cap set\ xs = \{\}$
and *l*: $length\ us = length\ xs\ length\ vs = length\ xs\ length\ ts = length\ xs$
and *d*: $set\ us \cap set\ vs = \{\}$
shows $rawpsubstT\ (rawpsubstT\ r\ (zip\ (map\ Var\ us)\ xs))\ (zip\ ts\ us) =$
$rawpsubstT\ (rawpsubstT\ r\ (zip\ (map\ Var\ vs)\ xs))\ (zip\ ts\ vs)$
⟨*proof*⟩

lemma *rawpsubstT_compose_freshVar2*:
assumes *r*[*simp*]: $r \in atrm$
and *ts*: $set\ ts \subseteq atrm$
and *xs*: $set\ xs \subseteq var\ \ distinct\ xs$
and *us_facts*: $set\ us \subseteq var\ \ distinct\ us$
$set\ us \cap FvarsT\ r = \{\}$
$set\ us \cap \bigcup\ (FvarsT\ `\ (set\ ts)) = \{\}$
$set\ us \cap set\ xs = \{\}$
and *vs_facts*: $set\ vs \subseteq var\ \ distinct\ vs$
$set\ vs \cap FvarsT\ r = \{\}$
$set\ vs \cap \bigcup\ (FvarsT\ `\ (set\ ts)) = \{\}$
$set\ vs \cap set\ xs = \{\}$
and *l*: $length\ us = length\ xs\ length\ vs = length\ xs\ length\ ts = length\ xs$
shows $rawpsubstT\ (rawpsubstT\ r\ (zip\ (map\ Var\ us)\ xs))\ (zip\ ts\ us) =$
$rawpsubstT\ (rawpsubstT\ r\ (zip\ (map\ Var\ vs)\ xs))\ (zip\ ts\ vs)$ (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *in_fst_image*: $a \in fst\ `\ AB \longleftrightarrow (\exists\, b.\ (a,b) \in AB)$ ⟨*proof*⟩

lemma *psubstT_eq_rawpsubstT*:
assumes $r \in atrm\ snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq atrm$
and $distinct\ (map\ snd\ txs)$

and $\bigwedge i\ j.\ i < j \Longrightarrow j < length\ txs \Longrightarrow snd\ (txs!j) \notin FvarsT\ (fst\ (txs!i))$
shows $psubstT\ r\ txs = rawpsubstT\ r\ txs$
⟨*proof*⟩

lemma *psubstT_eq_substT*:
assumes $r \in atrm\ x \in var$ **and** $t \in atrm$
shows $psubstT\ r\ [(t,x)] = substT\ r\ t\ x$
⟨*proof*⟩

lemma *psubstT_eq_rawpsubst2*:
assumes $r \in atrm\ x1 \in var\ x2 \in var\ t1 \in atrm\ t2 \in atrm$
and $x1 \neq x2\ x2 \notin FvarsT\ t1$
shows $psubstT\ r\ [(t1,x1),(t2,x2)] = rawpsubstT\ r\ [(t1,x1),(t2,x2)]$
⟨*proof*⟩

lemma *psubstT_eq_rawpsubst3*:
assumes $r \in atrm\ x1 \in var\ x2 \in var\ x3 \in var\ t1 \in atrm\ t2 \in atrm\ t3 \in atrm$
and $x1 \neq x2\ x1 \neq x3\ x2 \neq x3$
$x2 \notin FvarsT\ t1\ x3 \notin FvarsT\ t1\ x3 \notin FvarsT\ t2$
shows $psubstT\ r\ [(t1,x1),(t2,x2),(t3,x3)] = rawpsubstT\ r\ [(t1,x1),(t2,x2),(t3,x3)]$
⟨*proof*⟩

lemma *psubstT_eq_rawpsubst4*:
assumes $r \in atrm\ x1 \in var\ x2 \in var\ x3 \in var\ x4 \in var$
$t1 \in atrm\ t2 \in atrm\ t3 \in atrm\ t4 \in atrm$
and $x1 \neq x2\ x1 \neq x3\ x2 \neq x3\ x1 \neq x4\ x2 \neq x4\ x3 \neq x4$
$x2 \notin FvarsT\ t1\ x3 \notin FvarsT\ t1\ x3 \notin FvarsT\ t2\ x4 \notin FvarsT\ t1\ x4 \notin FvarsT\ t2\ x4 \notin FvarsT\ t3$
shows $psubstT\ r\ [(t1,x1),(t2,x2),(t3,x3),(t4,x4)] = rawpsubstT\ r\ [(t1,x1),(t2,x2),(t3,x3),(t4,x4)]$
⟨*proof*⟩

lemma *rawpsubstT_same_Var*[*simp*]:
assumes $r \in atrm\ set\ xs \subseteq var$
shows $rawpsubstT\ r\ (map\ (\lambda x.\ (Var\ x,x))\ xs) = r$
⟨*proof*⟩

lemma *psubstT_same_Var*[*simp*]:
assumes $r \in atrm\ set\ xs \subseteq var$ **and** $distinct\ xs$
shows $psubstT\ r\ (map\ (\lambda x.\ (Var\ x,x))\ xs) = r$
⟨*proof*⟩

thm *psubstT_notIn*

lemma *rawpsubst_eql*:
assumes $t1 \in trm\ t2 \in trm$
and $snd\ `\ (set\ txs) \subseteq var\ fst\ `\ (set\ txs) \subseteq trm$
shows $rawpsubst\ (eql\ t1\ t2)\ txs = eql\ (rawpsubstT\ t1\ txs)\ (rawpsubstT\ t2\ txs)$
⟨*proof*⟩

lemma *psubst_eql*[*simp*]:
assumes $t1 \in atrm\ t2 \in atrm$
and $snd\ `\ (set\ txs) \subseteq var\ fst\ `\ (set\ txs) \subseteq atrm$
and $distinct\ (map\ snd\ txs)$

shows *psubst (eql t1 t2) txs = eql (psubstT t1 txs) (psubstT t2 txs)*
⟨*proof*⟩

lemma *psubst_exu*[*simp*]:
assumes $\varphi \in fmla$ $x \in var$ *snd ‘ set txs* $\subseteq$ *var fst ‘ set txs* $\subseteq$ *atrm*
$x \notin$ *snd ‘ set txs* $x \notin (\bigcup t \in$ *fst ‘ set txs. FvarsT t*$)$ *distinct (map snd txs)*
shows *psubst (exu x* φ*) txs = exu x (psubst* φ *txs)*
⟨*proof*⟩

thm *psubstT_Var_not*[*no_vars*]

lemma *rawpsubstT_Var_in*:
assumes *snd ‘ (set txs)* $\subseteq$ *var fst ‘ (set txs)* $\subseteq$ *trm*
and *distinct (map snd txs)* **and** $(s,y) \in$ *set txs*
and $\bigwedge i\ j.\ i < j \Longrightarrow j <$ *length txs* $\Longrightarrow$ *snd (txs!j)* $\notin$ *FvarsT (fst (txs!i))*
shows *rawpsubstT (Var y) txs = s*
⟨*proof*⟩

lemma *psubstT_Var_in*:
assumes $y \in$ *var snd ‘ (set txs)* $\subseteq$ *var fst ‘ (set txs)* $\subseteq$ *trm*
and *distinct (map snd txs)* **and** $(s,y) \in$ *set txs*
shows *psubstT (Var y) txs = s*
⟨*proof*⟩

lemma *psubstT_Var_Cons_aux*:
assumes $y \in$ *var* $x \in$ *var* $t \in$ *atrm*
snd ‘ set txs $\subseteq$ *var fst ‘ set txs* $\subseteq$ *atrm* $x \notin$ *snd ‘ set txs*
distinct (map snd txs) $y \neq x$
shows *psubstT (Var y) ((t, x) # txs) = psubstT (Var y) txs*
⟨*proof*⟩

Simplification rules for parallel substitution:

lemma *psubstT_Var_Cons*[*simp*]:
$y \in$ *var* $\Longrightarrow$ $x \in$ *var* $\Longrightarrow$ $t \in$ *atrm* $\Longrightarrow$
snd ‘ set txs $\subseteq$ *var* $\Longrightarrow$ *fst ‘ set txs* $\subseteq$ *atrm* $\Longrightarrow$ *distinct (map snd txs)* $\Longrightarrow$ $x \notin$ *snd ‘ set txs* $\Longrightarrow$
psubstT (Var y) ((t,x) # txs) = (if y = x then t else psubstT (Var y) txs)
⟨*proof*⟩

lemma *psubstT_zer*[*simp*]:
assumes *snd ‘ (set txs)* $\subseteq$ *var* **and** *fst ‘ (set txs)* $\subseteq$ *trm*
shows *psubstT zer txs = zer*
⟨*proof*⟩

lemma *rawpsubstT_suc*:
assumes $r \in$ *trm* **and** *snd ‘ (set txs)* $\subseteq$ *var* **and** *fst ‘ (set txs)* $\subseteq$ *trm*
shows *rawpsubstT (suc r) txs = suc (rawpsubstT r txs)*
⟨*proof*⟩

lemma *psubstT_suc*[*simp*]:
assumes $r \in$ *atrm* **and** *snd ‘ (set txs)* $\subseteq$ *var* **and** *fst ‘ (set txs)* $\subseteq$ *atrm*
and *distinct (map snd txs)*
shows *psubstT (suc r) txs = suc (psubstT r txs)*
⟨*proof*⟩

lemma *rawpsubstT_pls*:
assumes *r1* ∈ *trm r2* ∈ *trm* **and** *snd* ‘ (*set txs*) ⊆ *var* **and** *fst* ‘ (*set txs*) ⊆ *trm*
shows *rawpsubstT* (*pls r1 r2*) *txs* = *pls* (*rawpsubstT r1 txs*) (*rawpsubstT r2 txs*)
⟨*proof*⟩

lemma *psubstT_pls*[*simp*]:
assumes *r1* ∈ *atrm r2* ∈ *atrm* **and** *snd* ‘ (*set txs*) ⊆ *var* **and** *fst* ‘ (*set txs*) ⊆ *atrm*
and *distinct* (*map snd txs*)
shows *psubstT* (*pls r1 r2*) *txs* = *pls* (*psubstT r1 txs*) (*psubstT r2 txs*)
⟨*proof*⟩

lemma *rawpsubstT_tms*:
assumes *r1* ∈ *trm r2* ∈ *trm* **and** *snd* ‘ (*set txs*) ⊆ *var* **and** *fst* ‘ (*set txs*) ⊆ *trm*
shows *rawpsubstT* (*tms r1 r2*) *txs* = *tms* (*rawpsubstT r1 txs*) (*rawpsubstT r2 txs*)
⟨*proof*⟩

lemma *psubstT_tms*[*simp*]:
assumes *r1* ∈ *atrm r2* ∈ *atrm* **and** *snd* ‘ (*set txs*) ⊆ *var* **and** *fst* ‘ (*set txs*) ⊆ *atrm*
and *distinct* (*map snd txs*)
shows *psubstT* (*tms r1 r2*) *txs* = *tms* (*psubstT r1 txs*) (*psubstT r2 txs*)
⟨*proof*⟩

7.2 The (Nonstrict and Strict) Order Relations

Lq (less than or equal to) is a formula with free vars xx and yy. NB: Out of the two possible ways, adding zz to the left or to the right, we choose the former, since this seems to enable Q (Robinson arithmetic) to prove as many useful properties as possible.

definition *Lq* :: *'fmla* **where**
Lq ≡ *exi zz* (*eql* (*Var yy*) (*pls* (*Var zz*) (*Var xx*)))

Alternative, more flexible definition , for any non-capturing bound variable:

lemma *Lq_def2*: *z* ∈ *var* ⟹ *z* ≠ *yy* ⟹ *z* ≠ *xx* ⟹ *Lq* = *exi z* (*eql* (*Var yy*) (*pls* (*Var z*) (*Var xx*)))
⟨*proof*⟩

lemma *Lq*[*simp*,*intro*!]: *Lq* ∈ *fmla*
⟨*proof*⟩

lemma *Fvars_Lq*[*simp*]: *Fvars Lq* = {*xx*,*yy*}
⟨*proof*⟩

As usual, we also define a predicate version:

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

lemma *LLq*[*simp*,*intro*]:
assumes *t1* ∈ *trm t2* ∈ *trm*
shows *LLq t1 t2* ∈ *fmla*
⟨*proof*⟩

lemma *LLq2*[*simp*,*intro*!]:
n ∈ *num* ⟹ *LLq n* (*Var yy'*) ∈ *fmla*
⟨*proof*⟩

lemma *Fvars_LLq*[*simp*]: *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹
Fvars (*LLq t1 t2*) = *FvarsT t1* ∪ *FvarsT t2*
⟨*proof*⟩

This lemma will be the working definition of LLq:

lemma *LLq_pls*:
assumes [*simp*]: *t1* ∈ *atrm t2* ∈ *atrm z* ∈ *var z* ∉ *FvarsT t1 z* ∉ *FvarsT t2*
shows *LLq t1 t2* = *exi z* (*eql t2* (*pls* (*Var z*) *t1*))
⟨*proof*⟩

lemma *LLq_pls_zz*:
assumes *t1* ∈ *atrm t2* ∈ *atrm zz* ∉ *FvarsT t1 zz* ∉ *FvarsT t2*
shows *LLq t1 t2* = *exi zz* (*eql t2* (*pls* (*Var zz*) *t1*))
⟨*proof*⟩

If we restrict attention to arithmetic terms, we can prove a uniform substitution property for LLq:

lemma *subst_LLq*[*simp*]:
assumes [*simp*]: *t1* ∈ *atrm t2* ∈ *atrm s* ∈ *atrm x* ∈ *var*
shows *subst* (*LLq t1 t2*) *s x* = *LLq* (*substT t1 s x*) (*substT t2 s x*)
⟨*proof*⟩

lemma *psubst_LLq*[*simp*]:
assumes *1*: *t1* ∈ *atrm t2* ∈ *atrm fst* ‘ *set txs* ⊆ *atrm*
and *2*: *snd* ‘ *set txs* ⊆ *var*
and *3*: *distinct* (*map snd txs*)
shows *psubst* (*LLq t1 t2*) *txs* = *LLq* (*psubstT t1 txs*) (*psubstT t2 txs*)
⟨*proof*⟩

Lq less than) is the strict version of the order relation. We prove similar facts as for Lq

definition *Ls* :: *'fmla* **where**
Ls ≡ *cnj Lq* (*neg* (*eql* (*Var xx*) (*Var yy*)))

lemma *Ls*[*simp*,*intro*!]: *Ls* ∈ *fmla*
⟨*proof*⟩

lemma *Fvars_Ls*[*simp*]: *Fvars Ls* = {*xx*,*yy*}
⟨*proof*⟩

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

lemma *LLs*[*simp*,*intro*]:
assumes *t1* ∈ *trm t2* ∈ *trm*
shows *LLs t1 t2* ∈ *fmla*
⟨*proof*⟩

lemma *LLs2*[*simp*,*intro*!]:
n ∈ *num* ⟹ *LLs n* (*Var yy'*) ∈ *fmla*
⟨*proof*⟩

lemma *Fvars_LLs*[*simp*]: *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹
Fvars (*LLs t1 t2*) = *FvarsT t1* ∪ *FvarsT t2*
⟨*proof*⟩

The working definition of LLs:

lemma *LLs_LLq*:
t1 ∈ *atrm* ⟹ *t2* ∈ *atrm* ⟹
LLs t1 t2 = *cnj* (*LLq t1 t2*) (*neg* (*eql t1 t2*))
⟨*proof*⟩

lemma *subst_LLs*[*simp*]:
assumes [*simp*]: *t1* ∈ *atrm t2* ∈ *atrm s* ∈ *atrm x* ∈ *var*

shows *subst (LLs t1 t2) s x = LLs (substT t1 s x) (substT t2 s x)*
⟨*proof*⟩

lemma *psubst_LLs*[*simp*]:
assumes *1*: *t1 ∈ atrm t2 ∈ atrm fst ‘ set txs ⊆ atrm*
and *2*: *snd ‘ set txs ⊆ var*
and *3*: *distinct (map snd txs)*
shows *psubst (LLs t1 t2) txs = LLs (psubstT t1 txs) (psubstT t2 txs)*
⟨*proof*⟩

7.3 Bounded Quantification

Bounded forall

definition *ball* :: *'var ⇒ 'trm ⇒ 'fmla ⇒ 'fmla* **where**
ball x t φ ≡ all x (imp (LLq (Var x) t) φ)

lemma *ball*[*simp*, *intro*]: *x ∈ var ⟹ t ∈ trm ⟹ φ ∈ fmla ⟹ ball x t φ ∈ fmla*
⟨*proof*⟩

lemma *Fvars_ball*[*simp*]:
x ∈ var ⟹ φ ∈ fmla ⟹ t ∈ trm ⟹ Fvars (ball x t φ) = (Fvars φ ∪ FvarsT t) − {x}
⟨*proof*⟩

lemma *subst_ball*:
φ ∈ fmla ⟹ t ∈ atrm ⟹ t1 ∈ atrm ⟹ x ∈ var ⟹ y ∈ var ⟹ x ≠ y ⟹ x ∉ FvarsT t1 ⟹
subst (ball x t φ) t1 y = ball x (substT t t1 y) (subst φ t1 y)
⟨*proof*⟩

lemma *psubst_ball*:
φ ∈ fmla ⟹ y ∈ var ⟹ snd ‘ set txs ⊆ var ⟹ t ∈ atrm ⟹
fst ‘ set txs ⊆ trm ⟹ fst ‘ set txs ⊆ atrm ⟹ y ∉ snd ‘ set txs ⟹ y ∉ (⋃ t ∈ fst ‘ set txs. FvarsT t)
⟹
distinct (map snd txs) ⟹
psubst (ball y t φ) txs = ball y (psubstT t txs) (psubst φ txs)
⟨*proof*⟩

Bounded exists

definition *bexi* :: *'var ⇒ 'trm ⇒ 'fmla ⇒ 'fmla* **where**
bexi x t φ ≡ exi x (cnj (LLq (Var x) t) φ)

lemma *bexi*[*simp*, *intro*]: *x ∈ var ⟹ t ∈ trm ⟹ φ ∈ fmla ⟹ bexi x t φ ∈ fmla*
⟨*proof*⟩

lemma *Fvars_bexi*[*simp*]:
x ∈ var ⟹ φ ∈ fmla ⟹ t ∈ trm ⟹ Fvars (bexi x t φ) = (Fvars φ ∪ FvarsT t) − {x}
⟨*proof*⟩

lemma *subst_bexi*:
φ ∈ fmla ⟹ t ∈ atrm ⟹ t1 ∈ atrm ⟹ x ∈ var ⟹ y ∈ var ⟹ x ≠ y ⟹ x ∉ FvarsT t1 ⟹
subst (bexi x t φ) t1 y = bexi x (substT t t1 y) (subst φ t1 y)
⟨*proof*⟩

lemma *psubst_bexi*:
φ ∈ fmla ⟹ y ∈ var ⟹ snd ‘ set txs ⊆ var ⟹ t ∈ atrm ⟹
fst ‘ set txs ⊆ trm ⟹ fst ‘ set txs ⊆ atrm ⟹ y ∉ snd ‘ set txs ⟹ y ∉ (⋃ t ∈ fst ‘ set txs. FvarsT t)
⟹
distinct (map snd txs) ⟹

$psubst\ (bexi\ y\ t\ \varphi)\ txs = bexi\ y\ (psubstT\ t\ txs)\ (psubst\ \varphi\ txs)$
⟨*proof*⟩

end — context *Syntax_Arith*

Chapter 8

Deduction in a System Embedding the Intuitionistic Robinson Arithmetic

NB: Robinson arithmetic, also know as system Q, is Peano arithmetic without the induction axiom schema.

8.1 Natural Deduction with the Bounded Quantifiers

We start by simply putting together deduction with the arithmetic syntax, which allows us to reason about bounded quantifiers:

locale *Deduct_with_False_Disj_Arith* =
Syntax_Arith
  var trm fmla Var FvarsT substT Fvars subst
  eql cnj imp all exi
  fls
  dsj
  num
  zer suc pls tms
+
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 *zer suc pls tms*
and *prv*
begin

lemma *nprv_ballI*:

nprv (*insert* (*LLq* (*Var x*) *t*) *F*) $\varphi \Longrightarrow$
F $\subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ $\varphi \in$ *fmla* $\Longrightarrow$ *t* $\in$ *trm* $\Longrightarrow$ *x* $\in$ *var* $\Longrightarrow$
x $\notin$ ($\bigcup \varphi \in F$. *Fvars* φ) $\Longrightarrow$ *x* $\notin$ *FvarsT t* $\Longrightarrow$
nprv F (*ball x t* φ)
⟨*proof*⟩

lemma *nprv_ballE_aux*:
nprv F (*ball x t* φ) $\Longrightarrow$ *nprv F* (*LLq t1 t*) $\Longrightarrow$
F $\subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ $\varphi \in$ *fmla* $\Longrightarrow$ *t* $\in$ *atrm* $\Longrightarrow$ *t1* $\in$ *atrm* $\Longrightarrow$ *x* $\in$ *var* $\Longrightarrow$ *x* $\notin$ *FvarsT t* $\Longrightarrow$
nprv F (*subst* φ *t1 x*)
⟨*proof*⟩

lemma *nprv_ballE*:
nprv F (*ball x t* φ) $\Longrightarrow$ *nprv F* (*LLq t1 t*) $\Longrightarrow$ *nprv* (*insert* (*subst* φ *t1 x*) *F*) ψ $\Longrightarrow$
F $\subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ $\varphi \in$ *fmla* $\Longrightarrow$ *t* $\in$ *atrm* $\Longrightarrow$ *t1* $\in$ *atrm* $\Longrightarrow$ *x* $\in$ *var* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$
x $\notin$ *FvarsT t* $\Longrightarrow$
nprv F ψ
⟨*proof*⟩

lemmas *nprv_ballE0* = *nprv_ballE*[*OF nprv_hyp* _ _, *simped*]
lemmas *nprv_ballE1* = *nprv_ballE*[*OF* _ *nprv_hyp* _, *simped*]
lemmas *nprv_ballE2* = *nprv_ballE*[*OF* _ _ *nprv_hyp*, *simped*]
lemmas *nprv_ballE01* = *nprv_ballE*[*OF nprv_hyp nprv_hyp* _, *simped*]
lemmas *nprv_ballE02* = *nprv_ballE*[*OF nprv_hyp* _ *nprv_hyp*, *simped*]
lemmas *nprv_ballE12* = *nprv_ballE*[*OF* _ *nprv_hyp nprv_hyp*, *simped*]
lemmas *nprv_ballE012* = *nprv_ballE*[*OF nprv_hyp nprv_hyp nprv_hyp*, *simped*]

lemma *nprv_bexiI*:
nprv F (*subst* φ *t1 x*) $\Longrightarrow$ *nprv F* (*LLq t1 t*) $\Longrightarrow$
F $\subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ $\varphi \in$ *fmla* $\Longrightarrow$ *t* $\in$ *atrm* $\Longrightarrow$ *t1* $\in$ *atrm* $\Longrightarrow$ *x* $\in$ *var* $\Longrightarrow$
x $\notin$ *FvarsT t* $\Longrightarrow$
nprv F (*bexi x t* φ)
⟨*proof*⟩

lemma *nprv_bexiE*:
nprv F (*bexi x t* φ) $\Longrightarrow$ *nprv* (*insert* (*LLq* (*Var x*) *t*) (*insert* φ *F*)) ψ $\Longrightarrow$
F $\subseteq$ *fmla* $\Longrightarrow$ *finite F* $\Longrightarrow$ $\varphi \in$ *fmla* $\Longrightarrow$ *x* $\in$ *var* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$ *t* $\in$ *atrm* $\Longrightarrow$
x $\notin$ ($\bigcup \varphi \in F$. *Fvars* φ) $\Longrightarrow$ *x* $\notin$ *Fvars* ψ $\Longrightarrow$ *x* $\notin$ *FvarsT t* $\Longrightarrow$
nprv F ψ
⟨*proof*⟩

lemmas *nprv_bexiE0* = *nprv_bexiE*[*OF nprv_hyp* _, *simped*]
lemmas *nprv_bexiE1* = *nprv_bexiE*[*OF* _ *nprv_hyp*, *simped*]
lemmas *nprv_bexiE01* = *nprv_bexiE*[*OF nprv_hyp nprv_hyp*, *simped*]

end — context *Deduct_with_False_Disj*

8.2 Deduction with the Robinson Arithmetic Axioms

locale *Deduct_Q* =
Deduct_with_False_Disj_Arith
var trm fmla
Var FvarsT substT Fvars subst
eql cnj imp all exi
fls
dsj
num
zer suc pls tms

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 *zer suc pls tms*
and *prv*
+
assumes
— The Q axioms are stated for some fixed variables; we will prove more useful versions, for arbitrary terms substituting the variables.
prv_neg_zer_suc_var:
prv (*neg* (*eql zer* (*suc* (*Var xx*))))
and
prv_inj_suc_var:
prv (*imp* (*eql* (*suc* (*Var xx*)) (*suc* (*Var yy*)))
(*eql* (*Var xx*) (*Var yy*)))
and
prv_zer_dsj_suc_var:
prv (*dsj* (*eql* (*Var yy*) *zer*)
(*exi xx* (*eql* (*Var yy*) (*suc* (*Var xx*)))))
and
prv_pls_zer_var:
prv (*eql* (*pls* (*Var xx*) *zer*) (*Var xx*))
and
prv_pls_suc_var:
prv (*eql* (*pls* (*Var xx*) (*suc* (*Var yy*)))
(*suc* (*pls* (*Var xx*) (*Var yy*))))
and
prv_tms_zer_var:
prv (*eql* (*tms* (*Var xx*) *zer*) *zer*)
and
prv_tms_suc_var:
prv (*eql* (*tms* (*Var xx*) (*suc* (*Var yy*)))
(*pls* (*tms* (*Var xx*) (*Var yy*)) (*Var xx*)))
begin

Rules for quantifiers that allow changing, on the fly, the bound variable with one that is fresh for the proof context:

lemma *nprv_allI_var*:
assumes *n1*[*simp*]: *nprv F* (*subst* φ (*Var y*) *x*)
and *i*[*simp*]: $F \subseteq$ *fmla finite F* $\varphi \in$ *fmla* $x \in$ *var* $y \in$ *var*
and *u*: $y \notin (\bigcup \varphi \in F.$ *Fvars* $\varphi)$ **and** *yx*[*simp*]: $y = x \vee y \notin$ *Fvars* φ
shows *nprv F* (*all x* φ)
⟨*proof*⟩

lemma *nprv_exiE_var*:
assumes *n*: *nprv F* (*exi x* φ)
and *nn*: *nprv* (*insert* (*subst* φ (*Var y*) *x*) *F*) ψ
and *0*: $F \subseteq$ *fmla finite F* $\varphi \in$ *fmla* $x \in$ *var* $y \in$ *var* $\psi \in$ *fmla*
and *yx*: $y = x \vee y \notin$ *Fvars* φ $y \notin \bigcup$ (*Fvars* ‘ *F*) $y \notin$ *Fvars* ψ
shows *nprv F* ψ
⟨*proof*⟩

lemma *prv_neg_zer_suc*:
assumes [*simp*]: $t \in atrm$ **shows** $prv\ (neg\ (eql\ zer\ (suc\ t)))$
⟨*proof*⟩

lemma *prv_neg_suc_zer*:
assumes $t \in atrm$ **shows** $prv\ (neg\ (eql\ (suc\ t)\ zer))$
⟨*proof*⟩

lemmas *nprv_zer_suc_contrE* =
nprv_flsE[*OF nprv_addImpLemmaE*[*OF prv_neg_zer_suc*[*unfolded neg_def*]], *OF _ _ nprv_hyp*, *simped*, *rotated*]

lemmas *nprv_zer_suc_contrE0* = *nprv_zer_suc_contrE*[*OF nprv_hyp*, *simped*]

lemma *nprv_zer_suc_2contrE*:
$nprv\ F\ (eql\ t\ zer) \Longrightarrow nprv\ F\ (eql\ t\ (suc\ t1)) \Longrightarrow$
$finite\ F \Longrightarrow F \subseteq fmla \Longrightarrow t \in atrm \Longrightarrow t1 \in atrm \Longrightarrow \varphi \in fmla \Longrightarrow$
$nprv\ F\ \varphi$
⟨*proof*⟩

lemmas *nprv_zer_suc_2contrE0* = *nprv_zer_suc_2contrE*[*OF nprv_hyp _*, *simped*]
lemmas *nprv_zer_suc_2contrE1* = *nprv_zer_suc_2contrE*[*OF _ nprv_hyp*, *simped*]
lemmas *nprv_zer_suc_2contrE01* = *nprv_zer_suc_2contrE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemma *prv_inj_suc*:
$t \in atrm \Longrightarrow t' \in atrm \Longrightarrow$
$prv\ (imp\ (eql\ (suc\ t)\ (suc\ t'))$
$\quad (eql\ t\ t'))$
⟨*proof*⟩

lemmas *nprv_eql_sucI* = *nprv_addImpLemmaI*[*OF prv_inj_suc*, *simped*, *rotated 4*]
lemmas *nprv_eql_sucE* = *nprv_addImpLemmaE*[*OF prv_inj_suc*, *simped*, *rotated 2*]

lemmas *nprv_eql_sucE0* = *nprv_eql_sucE*[*OF nprv_hyp _*, *simped*]
lemmas *nprv_eql_sucE1* = *nprv_eql_sucE*[*OF _ nprv_hyp*, *simped*]
lemmas *nprv_eql_sucE01* = *nprv_eql_sucE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemma *prv_zer_dsj_suc*:
assumes *t*[*simp*]: $t \in atrm$ **and** *x*[*simp*]: $x \in var\ x \notin FvarsT\ t$
shows $prv\ (dsj\ (eql\ t\ zer)$
$\quad (exi\ x\ (eql\ t\ (suc\ (Var\ x)))))$
⟨*proof*⟩

lemma *nprv_zer_suc_casesE*:
$nprv\ (insert\ (eql\ t\ zer)\ F)\ \varphi \Longrightarrow nprv\ (insert\ (eql\ t\ (suc\ (Var\ x)))\ F)\ \varphi \Longrightarrow$
$finite\ F \Longrightarrow F \subseteq fmla \Longrightarrow \varphi \in fmla \Longrightarrow x \in var \Longrightarrow t \in atrm \Longrightarrow$
$x \notin Fvars\ \varphi \Longrightarrow x \notin FvarsT\ t \Longrightarrow x \notin \bigcup\ (Fvars\ `\ F) \Longrightarrow$
$nprv\ F\ \varphi$
⟨*proof*⟩

lemmas *nprv_zer_suc_casesE0 = nprv_zer_suc_casesE[OF nprv_hyp _, simped]*
lemmas *nprv_zer_suc_casesE1 = nprv_zer_suc_casesE[OF _ nprv_hyp, simped]*
lemmas *nprv_zer_suc_casesE01 = nprv_zer_suc_casesE[OF nprv_hyp nprv_hyp, simped]*

lemma *prv_pls_zer*:
assumes [*simp*]: *t ∈ atrm* **shows** *prv (eql (pls t zer) t)*
⟨*proof*⟩

lemma *prv_pls_suc*:
t ∈ atrm ⟹ t′ ∈ atrm ⟹
prv (eql (pls t (suc t′))
(suc (pls t t′)))
⟨*proof*⟩

lemma *prv_tms_zer*:
assumes [*simp*]: *t ∈ atrm* **shows** *prv (eql (tms t zer) zer)*
⟨*proof*⟩

lemma *prv_tms_suc*:
t ∈ atrm ⟹ t′ ∈ atrm ⟹
prv (eql (tms t (suc t′))
(pls (tms t t′) t))
⟨*proof*⟩

lemma *prv_suc_imp_cong*:
assumes *t1*[*simp*]: *t1 ∈ atrm* **and** *t2*[*simp*]: *t2 ∈ atrm*
shows *prv (imp (eql t1 t2)*
(eql (suc t1) (suc t2)))
⟨*proof*⟩

lemmas *nprv_suc_congI = nprv_addImpLemmaI[OF prv_suc_imp_cong, simped, rotated 4]*
lemmas *nprv_suc_congE = nprv_addImpLemmaE[OF prv_suc_imp_cong, simped, rotated 2]*

lemmas *nprv_suc_congE0 = nprv_suc_congE[OF nprv_hyp _, simped]*
lemmas *nprv_suc_congE1 = nprv_suc_congE[OF _ nprv_hyp, simped]*
lemmas *nprv_suc_congE01 = nprv_suc_congE[OF nprv_hyp nprv_hyp, simped]*

lemma *prv_suc_cong*:
assumes *t1*[*simp*]: *t1 ∈ atrm* **and** *t2*[*simp*]: *t2 ∈ atrm*
assumes *prv (eql t1 t2)*
shows *prv (eql (suc t1) (suc t2))*
⟨*proof*⟩

lemma *prv_pls_imp_cong*:
assumes *t1*[*simp*]: *t1 ∈ atrm* **and** *t1′*[*simp*]: *t1′ ∈ atrm*
and *t2*[*simp*]: *t2 ∈ atrm* **and** *t2′*[*simp*]: *t2′ ∈ atrm*
shows *prv (imp (eql t1 t1′)*
(imp (eql t2 t2′) (eql (pls t1 t2) (pls t1′ t2′))))
⟨*proof*⟩

lemmas *nprv_pls_congI = nprv_addImp2LemmaI[OF prv_pls_imp_cong, simped, rotated 6]*

lemmas *nprv_pls_congE* = *nprv_addImp2LemmaE*[*OF prv_pls_imp_cong*, *simped*, *rotated 4*]

lemmas *nprv_pls_congE0* = *nprv_pls_congE*[*OF nprv_hyp _ _*, *simped*]
lemmas *nprv_pls_congE1* = *nprv_pls_congE*[*OF _ nprv_hyp _*, *simped*]
lemmas *nprv_pls_congE2* = *nprv_pls_congE*[*OF _ _ nprv_hyp*, *simped*]
lemmas *nprv_pls_congE01* = *nprv_pls_congE*[*OF nprv_hyp nprv_hyp _*, *simped*]
lemmas *nprv_pls_congE02* = *nprv_pls_congE*[*OF nprv_hyp _ nprv_hyp*, *simped*]
lemmas *nprv_pls_congE12* = *nprv_pls_congE*[*OF _ nprv_hyp nprv_hyp*, *simped*]
lemmas *nprv_pls_congE012* = *nprv_pls_congE*[*OF nprv_hyp nprv_hyp nprv_hyp*, *simped*]

lemma *prv_pls_cong*:
assumes *t1* ∈ *atrm t1'* ∈ *atrm t2* ∈ *atrm t2'* ∈ *atrm*
and *prv* (*eql t1 t1'*) **and** *prv* (*eql t2 t2'*)
shows *prv* (*eql* (*pls t1 t2*) (*pls t1' t2'*))
⟨*proof*⟩

lemma *prv_pls_congL*:
t1 ∈ *atrm* ⟹ *t1'* ∈ *atrm* ⟹ *t2* ∈ *atrm* ⟹
prv (*eql t1 t1'*) ⟹ *prv* (*eql* (*pls t1 t2*) (*pls t1' t2*))
⟨*proof*⟩

lemma *prv_pls_congR*:
t1 ∈ *atrm* ⟹ *t2* ∈ *atrm* ⟹ *t2'* ∈ *atrm* ⟹
prv (*eql t2 t2'*) ⟹ *prv* (*eql* (*pls t1 t2*) (*pls t1 t2'*))
⟨*proof*⟩

lemma *nprv_pls_cong*:
assumes [*simp*]: *t1* ∈ *atrm t1'* ∈ *atrm t2* ∈ *atrm t2'* ∈ *atrm*
shows *nprv* {*eql t1 t1'*, *eql t2 t2'*} (*eql* (*pls t1 t2*) (*pls t1' t2'*))
⟨*proof*⟩

lemma *prv_tms_imp_cong*:
assumes *t1*[*simp*]: *t1* ∈ *atrm* **and** *t1'*[*simp*]: *t1'* ∈ *atrm*
and *t2*[*simp*]: *t2* ∈ *atrm* **and** *t2'*[*simp*]: *t2'* ∈ *atrm*
shows *prv* (*imp* (*eql t1 t1'*)
(*imp* (*eql t2 t2'*) (*eql* (*tms t1 t2*) (*tms t1' t2'*))))
⟨*proof*⟩

lemmas *nprv_tms_congI* = *nprv_addImp2LemmaI*[*OF prv_tms_imp_cong*, *simped*, *rotated 6*]
lemmas *nprv_tms_congE* = *nprv_addImp2LemmaE*[*OF prv_tms_imp_cong*, *simped*, *rotated 4*]

lemmas *nprv_tms_congE0* = *nprv_tms_congE*[*OF nprv_hyp _ _*, *simped*]
lemmas *nprv_tms_congE1* = *nprv_tms_congE*[*OF _ nprv_hyp _*, *simped*]
lemmas *nprv_tms_congE2* = *nprv_tms_congE*[*OF _ _ nprv_hyp*, *simped*]
lemmas *nprv_tms_congE01* = *nprv_tms_congE*[*OF nprv_hyp nprv_hyp _*, *simped*]
lemmas *nprv_tms_congE02* = *nprv_tms_congE*[*OF nprv_hyp _ nprv_hyp*, *simped*]
lemmas *nprv_tms_congE12* = *nprv_tms_congE*[*OF _ nprv_hyp nprv_hyp*, *simped*]
lemmas *nprv_tms_congE012* = *nprv_tms_congE*[*OF nprv_hyp nprv_hyp nprv_hyp*, *simped*]

lemma *prv_tms_cong*:
assumes *t1* ∈ *atrm t1'* ∈ *atrm t2* ∈ *atrm t2'* ∈ *atrm*
and *prv* (*eql t1 t1'*) **and** *prv* (*eql t2 t2'*)
shows *prv* (*eql* (*tms t1 t2*) (*tms t1' t2'*))
⟨*proof*⟩

lemma *nprv_tms_cong*:
assumes [*simp*]: *t1* ∈ *atrm t1'* ∈ *atrm t2* ∈ *atrm t2'* ∈ *atrm*

shows *nprv* {*eql t1 t1′*, *eql t2 t2′*} (*eql* (*tms t1 t2*) (*tms t1′ t2′*))
⟨*proof*⟩

lemma *prv_tms_congL*:
t1 ∈ *atrm* ⟹ *t1′* ∈ *atrm* ⟹ *t2* ∈ *atrm* ⟹
prv (*eql t1 t1′*) ⟹ *prv* (*eql* (*tms t1 t2*) (*tms t1′ t2*))
⟨*proof*⟩

lemma *prv_tms_congR*:
t1 ∈ *atrm* ⟹ *t2* ∈ *atrm* ⟹ *t2′* ∈ *atrm* ⟹
prv (*eql t2 t2′*) ⟹ *prv* (*eql* (*tms t1 t2*) (*tms t1 t2′*))
⟨*proof*⟩

8.3 Properties Provable in Q

8.3.1 General properties, unconstrained by numerals

lemma *prv_pls_suc_zer*:
t ∈ *atrm* ⟹ *prv* (*eql* (*pls t* (*suc zer*)) (*suc t*))
⟨*proof*⟩

lemma *prv_LLq_suc_imp*:
assumes [*simp*]: *t1* ∈ *atrm t2* ∈ *atrm*
shows *prv* (*imp* (*LLq* (*suc t1*) (*suc t2*)) (*LLq t1 t2*))
⟨*proof*⟩

lemmas *nprv_LLq_sucI* = *nprv_addImpLemmaI*[*OF prv_LLq_suc_imp*, *simped*, *rotated 4*]
lemmas *nprv_LLq_sucE* = *nprv_addImpLemmaE*[*OF prv_LLq_suc_imp*, *simped*, *rotated 2*]

lemmas *nprv_LLq_sucE0* = *nprv_LLq_sucE*[*OF nprv_hyp _*, *simped*]
lemmas *nprv_LLq_sucE1* = *nprv_LLq_sucE*[*OF _ nprv_hyp*, *simped*]
lemmas *nprv_LLq_sucE01* = *nprv_LLq_sucE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemma *prv_LLs_imp_LLq*:
assumes [*simp*]: *t1* ∈ *atrm t2* ∈ *atrm*
shows *prv* (*imp* (*LLs t1 t2*) (*LLq t1 t2*))
⟨*proof*⟩

lemma *prv_LLq_refl*:
prv (*LLq zer zer*)
⟨*proof*⟩

NB: Monotonicity of pls and tms w.r.t. LLq cannot be proved in Q.

lemma *prv_suc_mono_LLq*:
assumes *t1* ∈ *atrm t2* ∈ *atrm*
shows *prv* (*imp* (*LLq t1 t2*) (*LLq* (*suc t1*) (*suc t2*)))
⟨*proof*⟩

lemmas *nprv_suc_mono_LLqI* = *nprv_addImpLemmaI*[*OF prv_suc_mono_LLq*, *simped*, *rotated 4*]
lemmas *nprv_suc_mono_LLqE* = *nprv_addImpLemmaE*[*OF prv_suc_mono_LLq*, *simped*, *rotated 2*]

lemmas *nprv_suc_mono_LLqE0* = *nprv_suc_mono_LLqE*[*OF nprv_hyp _*, *simped*]
lemmas *nprv_suc_mono_LLqE1* = *nprv_suc_mono_LLqE*[*OF _ nprv_hyp*, *simped*]
lemmas *nprv_suc_mono_LLqE01* = *nprv_suc_mono_LLqE*[*OF nprv_hyp nprv_hyp*, *simped*]

8.3.2 Representability properties

Representability of number inequality

lemma *prv_neg_eql_suc_Num_zer*:
prv (*neg* (*eql* (*suc* (*Num n*)) *zer*))
⟨*proof*⟩

lemma *diff_prv_eql_Num*:
assumes $m \neq n$
shows *prv* (*neg* (*eql* (*Num m*) (*Num n*)))
⟨*proof*⟩

lemma *consistent_prv_eql_Num_equal*:
assumes *consistent* **and** *prv* (*eql* (*Num m*) (*Num n*))
shows $m = n$
⟨*proof*⟩

Representability of addition

lemma *prv_pls_zer_zer*:
prv (*eql* (*pls zer zer*) *zer*)
⟨*proof*⟩

lemma *prv_eql_pls_plus*:
prv (*eql* (*pls* (*Num m*) (*Num n*))
(*Num* (*m*+*n*)))
⟨*proof*⟩

lemma *not_plus_prv_neg_eql_pls*:
assumes $m + n \neq k$
shows *prv* (*neg* (*eql* (*pls* (*Num m*) (*Num n*)) (*Num k*)))
⟨*proof*⟩

lemma *consistent_prv_eql_pls_plus_rev*:
assumes *consistent prv* (*eql* (*pls* (*Num m*) (*Num n*)) (*Num k*))
shows $k = m + n$
⟨*proof*⟩

Representability of multiplication

lemma *prv_tms_Num_zer*:
prv (*eql* (*tms* (*Num n*) *zer*) *zer*)
⟨*proof*⟩

lemma *prv_eql_tms_times*:
prv (*eql* (*tms* (*Num m*) (*Num n*)) (*Num* ($m * n$)))
⟨*proof*⟩

lemma *ge_prv_neg_eql_pls_Num_zer*:
assumes [*simp*]: $t \in atrm$ **and** *m*: $m > k$
shows *prv* (*neg* (*eql* (*pls t* (*Num m*)) (*Num k*)))
⟨*proof*⟩

lemma *nprv_pls_Num_injectR*:
assumes [*simp*]: *t1* ∈ *atrm t2* ∈ *atrm*
shows *prv* (*imp* (*eql* (*pls t1* (*Num m*)) (*pls t2* (*Num m*)))
(*eql t1 t2*))
⟨*proof*⟩

lemmas *nprv_pls_Num_injectI* = *nprv_addImpLemmaI*[*OF nprv_pls_Num_injectR*, *simped*, *rotated 4*]
lemmas *nprv_pls_Num_injectE* = *nprv_addImpLemmaE*[*OF nprv_pls_Num_injectR*, *simped*, *rotated 2*]

lemmas *nprv_pls_Num_injectE0* = *nprv_pls_Num_injectE*[*OF nprv_hyp _*, *simped*]
lemmas *nprv_pls_Num_injectE1* = *nprv_pls_Num_injectE*[*OF _ nprv_hyp*, *simped*]
lemmas *nprv_pls_Num_injectE01* = *nprv_pls_Num_injectE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemma *not_times_prv_neg_eql_tms*:
assumes $m * n \neq k$
shows *prv* (*neg* (*eql* (*tms* (*Num m*) (*Num n*)) (*Num k*)))
⟨*proof*⟩

lemma *consistent_prv_eql_tms_times_rev*:
assumes *consistent prv* (*eql* (*tms* (*Num m*) (*Num n*)) (*Num k*))
shows $k = m * n$
⟨*proof*⟩

Representability of the order

lemma *leq_prv_LLq_Num*:
assumes $m \leq n$
shows *prv* (*LLq* (*Num m*) (*Num n*))
⟨*proof*⟩

8.3.3 The "order-adequacy" properties

These are properties Q1–O9 from Peter Smith, An Introduction to Gödel's theorems, Second Edition, Page 73.

lemma *prv_LLq_zer*: — O1
assumes [*simp*]: $t \in atrm$
shows *prv* (*LLq zer t*)
⟨*proof*⟩

lemmas *Q1* = *prv_LLq_zer*

lemma *prv_LLq_zer_imp_eql*:
assumes [*simp*]: $t \in atrm$
shows *prv* (*imp* (*LLq t zer*) (*eql t zer*))
⟨*proof*⟩

lemmas *nprv_LLq_zer_eqlI* = *nprv_addImpLemmaI*[*OF prv_LLq_zer_imp_eql*, *simped*, *rotated 3*]
lemmas *nprv_LLq_zer_eqlE* = *nprv_addImpLemmaE*[*OF prv_LLq_zer_imp_eql*, *simped*, *rotated 1*]

lemmas *nprv_LLq_zer_eqlE0* = *nprv_LLq_zer_eqlE*[*OF nprv_hyp _*, *simped*]
lemmas *nprv_LLq_zer_eqlE1* = *nprv_LLq_zer_eqlE*[*OF _ nprv_hyp*, *simped*]
lemmas *nprv_LLq_zer_eqlE01* = *nprv_LLq_zer_eqlE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemma *prv_sdsj_eql_imp_LLq*: — O2
assumes [*simp*]: $t \in atrm$
shows *prv* (*imp* (*ldsj* (*map* (λ*i*. *eql t* (*Num i*)) (*toN n*))) (*LLq t* (*Num n*)))
⟨*proof*⟩

declare *subset_eq*[*simp*]
lemmas *nprv_sdsj_eql_LLqI* = *nprv_addImpLemmaI*[*OF prv_sdsj_eql_imp_LLq*, *simped*, *rotated 3*]
lemmas *nprv_sdsj_eql_LLqE* = *nprv_addImpLemmaE*[*OF prv_sdsj_eql_imp_LLq*, *simped*, *rotated 1*]
declare *subset_eq*[*simp del*]

lemmas *nprv_sdsj_eql_LLqE0* = *nprv_sdsj_eql_LLqE*[*OF nprv_hyp _*, *simped*]
lemmas *nprv_sdsj_eql_LLqE1* = *nprv_sdsj_eql_LLqE*[*OF _ nprv_hyp*, *simped*]
lemmas *nprv_sdsj_eql_LLqE01* = *nprv_sdsj_eql_LLqE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemmas *O2I* = *nprv_sdsj_eql_LLqI*
lemmas *O2E* = *nprv_sdsj_eql_LLqE*
lemmas *O2E0* = *nprv_sdsj_eql_LLqE0*
lemmas *O2E1* = *nprv_sdsj_eql_LLqE1*
lemmas *O2E01* = *nprv_sdsj_eql_LLqE01*

lemma *prv_LLq_imp_sdsj_eql*: — O3
assumes [*simp*]: $t \in atrm$
shows *prv* (*imp* (*LLq t* (*Num n*)) (*ldsj* (*map* (λi. *eql t* (*Num i*)) (*toN n*))))
⟨*proof*⟩

declare *subset_eq*[*simp*]
lemmas *prv_LLq_sdsj_eqlI* = *nprv_addImpLemmaI*[*OF prv_LLq_imp_sdsj_eql*, *simped*, *rotated 3*]
lemmas *prv_LLq_sdsj_eqlE* = *nprv_addImpLemmaE*[*OF prv_LLq_imp_sdsj_eql*, *simped*, *rotated 1*]
declare *subset_eq*[*simp del*]

lemmas *prv_LLq_sdsj_eqlE0* = *prv_LLq_sdsj_eqlE*[*OF nprv_hyp _*, *simped*]
lemmas *prv_LLq_sdsj_eqlE1* = *prv_LLq_sdsj_eqlE*[*OF _ nprv_hyp*, *simped*]
lemmas *prv_LLq_sdsj_eqlE01* = *prv_LLq_sdsj_eqlE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemmas *O3I* = *prv_LLq_sdsj_eqlI*
lemmas *O3E* = *prv_LLq_sdsj_eqlE*
lemmas *O3E0* = *prv_LLq_sdsj_eqlE0*
lemmas *O3E1* = *prv_LLq_sdsj_eqlE1*
lemmas *O3E01* = *prv_LLq_sdsj_eqlE01*

lemma *not_leq_prv_neg_LLq_Num*:
assumes $\neg\ m \leq n$
shows *prv* (*neg* (*LLq* (*Num m*) (*Num n*)))
⟨*proof*⟩

lemma *consistent_prv_LLq_Num_leq*:
assumes *consistent* *prv* (*LLq* (*Num m*) (*Num n*))
shows $m \leq n$
⟨*proof*⟩

lemma *prv_ball_NumI*: — O4
assumes [*simp*]: $x \in var$ $\varphi \in fmla$
and [*simp*]: $\bigwedge i.\ i \leq n \Longrightarrow$ *prv* (*subst* φ (*Num i*) x)
shows *prv* (*ball x* (*Num n*) φ)
⟨*proof*⟩

lemmas *O4* = *prv_ball_NumI*

lemma *prv_bexi_NumI*: — O5

assumes [*simp*]: $x \in var\ \varphi \in fmla$
and [*simp*]: $i \leq n\ prv\ (subst\ \varphi\ (Num\ i)\ x)$
shows $prv\ (bexi\ x\ (Num\ n)\ \varphi)$
⟨*proof*⟩

lemmas *O5* = *prv_bexi_NumI*

lemma *prv_LLq_Num_imp_Suc*: — O6
assumes [*simp*]: $t \in atrm$
shows $prv\ (imp\ (LLq\ t\ (Num\ n))\ (LLq\ t\ (suc\ (Num\ n))))$
⟨*proof*⟩

lemmas *nprv_LLq_Num_SucI* = *nprv_addImpLemmaI*[*OF prv_LLq_Num_imp_Suc*, *simped*, *rotated 3*]
lemmas *nprv_LLq_Num_SucE* = *nprv_addImpLemmaE*[*OF prv_LLq_Num_imp_Suc*, *simped*, *rotated 1*]

lemmas *nprv_LLq_Num_SucE0* = *nprv_LLq_Num_SucE*[*OF nprv_hyp _*, *simped*]
lemmas *nprv_LLq_Num_SucE1* = *nprv_LLq_Num_SucE*[*OF _ nprv_hyp*, *simped*]
lemmas *nprv_LLq_Num_SucE01* = *nprv_LLq_Num_SucE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemmas *O6I* = *nprv_LLq_Num_SucI*
lemmas *O6E* = *nprv_LLq_Num_SucE*
lemmas *O6E0* = *nprv_LLq_Num_SucE0*
lemmas *O6E1* = *nprv_LLq_Num_SucE1*
lemmas *O6E01* = *nprv_LLq_Num_SucE01*

Crucial for proving O7:

lemma *prv_LLq_suc_Num_pls_Num*:
assumes [*simp*]: $t \in atrm$
shows $prv\ (LLq\ (suc\ (Num\ n))\ (pls\ (suc\ t)\ (Num\ n)))$
⟨*proof*⟩

lemma *prv_Num_LLq_imp_eql_suc*: — O7
assumes [*simp*]: $t \in atrm$
shows $prv\ (imp\ (LLq\ (Num\ n)\ t)$
$(dsj\ (eql\ (Num\ n)\ t)$
$(LLq\ (suc\ (Num\ n))\ t)))$
⟨*proof*⟩

lemma *prv_Num_LLq_eql_sucE*:
$nprv\ F\ (LLq\ (Num\ n)\ t) \Longrightarrow$
$nprv\ (insert\ (eql\ (Num\ n)\ t)\ F)\ \psi \Longrightarrow$
$nprv\ (insert\ (LLq\ (suc\ (Num\ n))\ t)\ F)\ \psi \Longrightarrow$
$t \in atrm \Longrightarrow F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow\ \psi \in fmla \Longrightarrow$
$nprv\ F\ \psi$
⟨*proof*⟩

lemmas *prv_Num_LLq_eql_sucE0* = *prv_Num_LLq_eql_sucE*[*OF nprv_hyp _ _*, *simped*]
lemmas *prv_Num_LLq_eql_sucE1* = *prv_Num_LLq_eql_sucE*[*OF _ nprv_hyp _*, *simped*]
lemmas *prv_Num_LLq_eql_sucE2* = *prv_Num_LLq_eql_sucE*[*OF _ _ nprv_hyp*, *simped*]
lemmas *prv_Num_LLq_eql_sucE01* = *prv_Num_LLq_eql_sucE*[*OF nprv_hyp nprv_hyp _*, *simped*]
lemmas *prv_Num_LLq_eql_sucE02* = *prv_Num_LLq_eql_sucE*[*OF nprv_hyp _ nprv_hyp*, *simped*]
lemmas *prv_Num_LLq_eql_sucE12* = *prv_Num_LLq_eql_sucE*[*OF _ nprv_hyp nprv_hyp*, *simped*]
lemmas *prv_Num_LLq_eql_sucE012* = *prv_Num_LLq_eql_sucE*[*OF nprv_hyp nprv_hyp nprv_hyp*, *simped*]

lemmas *O7E* = *prv_Num_LLq_eql_sucE*
lemmas *O7E0* = *prv_Num_LLq_eql_sucE0*

lemma *prv_dsj_eql_Num_neg*:
assumes $t \in atrm$
shows *prv* (*dsj* (*eql t* (*Num n*)) (*neg* (*eql t* (*Num n*))))
⟨*proof*⟩

lemmas *nprv_eql_Num_casesE* = *nprv_addDsjLemmaE*[*OF prv_dsj_eql_Num_neg*, *simped*, *rotated*]

lemmas *nprv_eql_Num_casesE0* = *nprv_eql_Num_casesE*[*OF nprv_hyp* _, *simped*]
lemmas *nprv_eql_Num_casesE1* = *nprv_eql_Num_casesE*[*OF* _ *nprv_hyp*, *simped*]
lemmas *nprv_eql_Num_casesE01* = *nprv_eql_Num_casesE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemma *prv_LLq_Num_dsj*: — O8
assumes [*simp*]: $t \in atrm$
shows *prv* (*dsj* (*LLq t* (*Num n*)) (*LLq* (*Num n*) *t*))
⟨*proof*⟩

lemma *prv_LLq_Num_casesE*:
nprv (*insert* (*LLq t* (*Num n*)) *F*) $\psi \Longrightarrow$
nprv (*insert* (*LLq* (*Num n*) *t*) *F*) $\psi \Longrightarrow$
$t \in atrm \Longrightarrow F \subseteq fmla \Longrightarrow$ *finite F* $\Longrightarrow \psi \in fmla \Longrightarrow$
nprv F ψ
⟨*proof*⟩

lemmas *prv_LLq_Num_casesE0* = *prv_LLq_Num_casesE*[*OF nprv_hyp* _, *simped*]
lemmas *prv_LLq_Num_casesE1* = *prv_LLq_Num_casesE*[*OF* _ *nprv_hyp*, *simped*]
lemmas *prv_LLq_Num_casesE01* = *prv_LLq_Num_casesE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemmas *O8E* = *prv_LLq_Num_casesE*
lemmas *O8E0* = *prv_LLq_Num_casesE0*
lemmas *O8E1* = *prv_LLq_Num_casesE1*
lemmas *O8E01* = *prv_LLq_Num_casesE01*

lemma *prv_imp_LLq_neg_Num_suc*:
assumes [*simp*]: $t \in atrm$
shows *prv* (*imp* (*LLq t* (*suc* (*Num n*)))
(*imp* ((*neg* (*eql t* (*suc* (*Num n*)))))
(*LLq t* (*Num n*))))
⟨*proof*⟩

lemmas *nprv_LLq_neg_Num_sucI* = *nprv_addImp2LemmaI*[*OF prv_imp_LLq_neg_Num_suc*, *simped*, *rotated 3*]
lemmas *nprv_LLq_neg_Num_sucE* = *nprv_addImp2LemmaE*[*OF prv_imp_LLq_neg_Num_suc*, *simped*, *rotated 1*]

lemmas *nprv_LLq_neg_Num_sucE0* = *nprv_LLq_neg_Num_sucE*[*OF nprv_hyp* _ _, *simped*]
lemmas *nprv_LLq_neg_Num_sucE1* = *nprv_LLq_neg_Num_sucE*[*OF* _ *nprv_hyp* _, *simped*]

lemmas *nprv_LLq_neg_Num_sucE2* = *nprv_LLq_neg_Num_sucE*[*OF* _ _ *nprv_hyp*, *simped*]
lemmas *nprv_LLq_neg_Num_sucE01* = *nprv_LLq_neg_Num_sucE*[*OF nprv_hyp nprv_hyp* _, *simped*]
lemmas *nprv_LLq_neg_Num_sucE02* = *nprv_LLq_neg_Num_sucE*[*OF nprv_hyp* _ *nprv_hyp*, *simped*]
lemmas *nprv_LLq_neg_Num_sucE12* = *nprv_LLq_neg_Num_sucE*[*OF* _ *nprv_hyp nprv_hyp*, *simped*]
lemmas *nprv_LLq_neg_Num_sucE012* = *nprv_LLq_neg_Num_sucE*[*OF nprv_hyp nprv_hyp nprv_hyp*, *simped*]

lemma *prv_ball_Num_imp_ball_suc*: — O9
assumes [*simp*]: $x \in var$ $\varphi \in fmla$
shows *prv* (*imp* (*ball x* (*Num n*) φ)
(*ball x* (*suc* (*Num n*)) (*imp* (*neg* (*eql* (*Var x*) (*suc* (*Num n*)))) φ)))
⟨*proof*⟩

lemmas *prv_ball_Num_ball_sucI* = *nprv_addImpLemmaI*[*OF prv_ball_Num_imp_ball_suc*, *simped*, *rotated 4*]
lemmas *prv_ball_Num_ball_sucE* = *nprv_addImpLemmaE*[*OF prv_ball_Num_imp_ball_suc*, *simped*, *rotated 2*]

lemmas *prv_ball_Num_ball_sucE0* = *prv_ball_Num_ball_sucE*[*OF nprv_hyp* _, *simped*]
lemmas *prv_ball_Num_ball_sucE1* = *prv_ball_Num_ball_sucE*[*OF* _ *nprv_hyp*, *simped*]
lemmas *prv_ball_Num_ball_sucE01* = *prv_ball_Num_ball_sucE*[*OF nprv_hyp nprv_hyp*, *simped*]

lemmas *O9I* = *prv_ball_Num_ball_sucI*
lemmas *O9E* = *prv_ball_Num_ball_sucE*
lemmas *O9E0* = *prv_ball_Num_ball_sucE0*
lemmas *O9E1* = *prv_ball_Num_ball_sucE1*
lemmas *O9E01* = *prv_ball_Num_ball_sucE01*

8.3.4 Verifying the abstract ordering assumptions

lemma *LLq_num*:
assumes φ[*simp*]: $\varphi \in fmla$ *Fvars* $\varphi = \{zz\}$ **and** *q*: $q \in num$ **and** *p*: $\forall\ p \in num.\ prv\ (subst\ \varphi\ p\ zz)$
shows *prv* (*all zz* (*imp* (*LLq* (*Var zz*) *q*) φ))
⟨*proof*⟩

lemma *LLq_num2*:
assumes $p \in num$
shows $\exists P \subseteq num.$ *finite* $P \wedge$ *prv* (*dsj* (*sdsj* {*eql* (*Var yy*) *r* |*r*. $r \in P$}) (*LLq p* (*Var yy*)))
⟨*proof*⟩

end — context *Deduct_Q*

sublocale *Deduct_Q* < *lab*: *Deduct_with_PseudoOrder* **where** *Lq* = *LLq* (*Var zz*) (*Var yy*)
⟨*proof*⟩

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.