Syntax-Independent Logic Infrastructure

Andrei Popescu Dmitriy Traytel

March 17, 2025

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*: $B \cap C = \{\} \Longrightarrow A \cup B - C = A - C \cup B$ **by** *auto*

lemma *Diff_Diff_Un*: $A - B - C = A - (B \cup C)$ **by** *auto*

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*)
  by (*induct xs ys rule*: *list_induct2*) *auto*

lemma *snd_set_zip_map_snd*:
  length xs = *length ys* ⟹ *snd* ‘ (*set* (*zip xs* (*map snd ys*))) = *snd* ‘ (*set ys*)
  by (*induct xs ys rule*: *list_induct2*) *auto*

lemma *snd_set_zip*:
  length xs = *length ys* ⟹ *snd* ‘ (*set* (*zip xs ys*)) = *set ys*
  by (*induct xs ys rule*: *list_induct2*) *auto*

lemma *set_zip_D*: (*x*, *y*) ∈ *set* (*zip xs ys*) ⟹ *x* ∈ *set xs* ∧ *y* ∈ *set ys*
  using *set_zip_leftD set_zip_rightD* **by** *auto*

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*)
using *a* **proof** (*induct xs arbitrary*: *ys x1 y1*)
  case (*Cons x xs yys*)
  thus *?case* **using** *i* **unfolding** *inj_on_def* **by** (*cases yys*) *auto*
qed (*insert i*, *auto*)

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 $(t, x) \in set\ txs$
using *assms*(5,1−4)
by (*induct us txs arbitrary*: *u x t rule*: *list_induct2*)
(*auto dest*: *set_zip_leftD set_zip_rightD*)

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 $(t, u) \in set\ (zip\ (map\ fst\ txs)\ us)$
using *assms*(5,1−4)
by (*induct us txs arbitrary*: *u x t rule*: *list_induct2*)
(*auto dest*: *set_zip_rightD simp*: *image_iff*)

lemma *set_zip_length_map*:
assumes $(x1, y1) \in set\ (zip\ xs\ ys)$ **and** *length xs* = *length ys*
shows $(f\ x1, y1) \in set\ (zip\ (map\ f\ xs)\ ys)$
using *assms*(2,1) **by** (*induct xs ys arbitrary*: *x1 y1 rule*: *list_induct2*) *auto*

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

lemma *asList*[*simp*,*intro*!]: $finite\ A \Longrightarrow set\ (asList\ A) = A$
unfolding *asList_def* **by** (*meson finite_list tfl_some*)

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

definition *toN* **where** $toN\ n \equiv [0..<(Suc\ n)]$

lemma *set_toN*[*simp*]: $set\ (toN\ n) = \{0..n\}$
unfolding *toN_def* **by** *auto*

declare *list.map_cong*[*cong*]

1.2 Some Proof Infrastructure

ML ‹
exception TAC of term

val simped = Thm.rule_attribute [] (fn context => fn thm =>
let
val ctxt = Context.proof_of context;
val (thm', ctxt') = yield_singleton (apfst snd oo Variable.import false) thm ctxt;
val full_goal = Thm.prop_of thm';
val goal = Goal.prove ctxt' [] [] full_goal (fn {context = ctxt, prems = _} =>
HEADGOAL (asm_full_simp_tac ctxt THEN' TRY o SUBGOAL (fn (goal, _) => raise (TAC goal))))
|> K (HOLogic.mk_Trueprop @{term True})
handle TAC goal => goal;
val thm = Goal.prove ctxt' [] [] goal (fn {context = ctxt, prems = _} =>
HEADGOAL (Method.insert_tac ctxt [thm'] THEN' asm_full_simp_tac ctxt))
|> singleton (Variable.export ctxt' ctxt);
in thm end)

val _ = Theory.setup

(*Attrib.setup* ***binding*** ‹*simped*› (*pair simped*) *simped rule*);
›

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*: $var \neq \{\}$
using *infinite_var* **by** *auto*

lemma *Var_injD*: $Var\ x = Var\ y \Longrightarrow x \in var \Longrightarrow y \in var \Longrightarrow x = y$
by *auto*

lemma *FvarsT_VarD*: $x \in FvarsT\ (Var\ y) \Longrightarrow y \in var \Longrightarrow x = y$
by *auto*

lemma *FvarsT'*: $t \in trm \Longrightarrow x \in FvarsT\ t \Longrightarrow x \in var$
using *FvarsT* **by** *auto*

lemma *Fvars'*: $\varphi \in fmla \Longrightarrow x \in Fvars\ \varphi \Longrightarrow x \in var$
using *Fvars* **by** *auto*

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

lemma *in_Fvars_substD*:
$y \in Fvars\ (subst\ \varphi\ t\ x) \Longrightarrow \varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var$
$\Longrightarrow y \in (Fvars\ \varphi - \{x\}) \cup (if\ x \in Fvars\ \varphi\ then\ FvarsT\ t\ else\ \{\})$
using *Fvars_subst* **by** *auto*

lemma *inj_on_Var*: *inj_on Var var*
using *inj_Var* **unfolding** *inj_on_def* **by** *auto*

lemma *subst_compose_same*:
$\bigwedge\ \varphi\ t1\ t2\ x.\ \varphi \in fmla \Longrightarrow t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow x \in var \Longrightarrow$
$subst\ (subst\ \varphi\ t1\ x)\ t2\ x = subst\ \varphi\ (substT\ t1\ t2\ x)\ x$
using *subst_compose_eq_or* **by** *blast*

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 $subst\ (subst\ \varphi\ (Var\ y)\ x)\ t\ y = subst\ \varphi\ t\ x$
using *subst_compose_eq_or*[*OF* φ _ t x y, *of Var y*] **using** *subst_notIn yy* **by** *simp*

lemma *subst_comp*:
$\bigwedge x\ y\ \varphi\ t.\ \varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow y \in var \Longrightarrow$
$x \neq y \Longrightarrow y \notin FvarsT\ t \Longrightarrow$
$subst\ (subst\ \varphi\ (Var\ x)\ y)\ t\ x = subst\ (subst\ \varphi\ t\ x)\ t\ y$
by (*simp add*: *subst_compose_diff*)

lemma *exists_nat_var*:
$\exists\ f{::}nat \Rightarrow 'var.\ inj\ f \wedge range\ f \subseteq var$
by (*simp add*: *infinite_countable_subset infinite_var*)

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

lemma *Variable_inj_var*:
$inj\ Variable \wedge range\ Variable \subseteq var$
unfolding *Variable_def* **using** *someI_ex*[*OF exists_nat_var*] **.**

lemma *inj_Variable*[*simp*]: $\bigwedge i\ j.\ Variable\ i = Variable\ j \longleftrightarrow i = j$
and *Variable*[*simp*,*intro*!]: $\bigwedge i.\ Variable\ i \in var$
using *Variable_inj_var image_def* **unfolding** *inj_def* **by** *auto*

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': $zz' \in var$
by *auto*

lemma *vars_distinct*[*simp*]:
$xx \neq yy\ yy \neq xx\ xx \neq zz\ zz \neq xx\ xx \neq xx'\ xx' \neq xx\ xx \neq yy'\ yy' \neq xx\ xx \neq zz'\ zz' \neq xx$
$yy \neq zz\ zz \neq yy\ yy \neq xx'\ xx' \neq yy\ yy \neq yy'\ yy' \neq yy\ yy \neq zz'\ zz' \neq yy$
$zz \neq xx'\ xx' \neq zz\ zz \neq yy'\ yy' \neq zz\ zz \neq zz'\ zz' \neq zz$
$xx' \neq yy'\ yy' \neq xx'\ xx' \neq zz'\ zz' \neq xx'$
$yy' \neq zz'\ zz' \neq yy'$
by *auto*

2.1.1 Instance Operator

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

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

unfolding *inst_def* **by** *auto*

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

lemma *getFresh*: *finite V* ⟹ *getFresh V* ∈ *var* ∧ *getFresh V* ∉ *V*
by (*metis* (*no_types*, *lifting*) *finite_subset getFresh_def infinite_var someI_ex subsetI*)

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−
have *finite* (*set xs* ∪ (⋃ (*FvarsT* ‘ *set ts*)) ∪ (⋃ (*Fvars* ‘ *set φs*)))
using *assms finite_FvarsT finite_Fvars* **by** *auto*
from *getFresh*[*OF this*] **show** *?thesis* **using** *assms* **unfolding** *getFr_def* **by** *auto*
qed

lemma *getFr*[*simp*,*intro*]:
assumes *set xs* ⊆ *var set ts* ⊆ *trm* **and** *set φs* ⊆ *fmla*
shows *getFr xs ts φs* ∈ *var*
using *assms getFr_FvarsT_Fvars* **by** *auto*

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

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

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

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*)
by (*induct n arbitrary*: *V*) (*auto simp*: *getFresh Let_def*)

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−
have *finite* (*set xs* ∪ (⋃(*FvarsT* ‘ *set ts*)) ∪ (⋃(*Fvars* ‘ *set φs*)))
using *assms finite_FvarsT finite_Fvars* **by** *auto*
from *getFreshN*[*OF this*] **show** *?thesis* **using** *assms* **unfolding** *getFrN_def* **by** *auto*
qed

lemma *getFrN*[*simp*,*intro*]:
assumes *set xs* ⊆ *var set ts* ⊆ *trm* **and** *set φs* ⊆ *fmla*
shows *set* (*getFrN xs ts φs n*) ⊆ *var*
using *assms getFrN_FvarsT_Fvars* **by** *auto*

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* = {}
using *assms getFrN_FvarsT_Fvars* **by** *auto*

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* = {}
using *assms getFrN_FvarsT_Fvars* **by** *auto*

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* = {}
using *assms getFrN_FvarsT_Fvars* **by** *auto*

lemma *getFrN_length*:
assumes *set xs* ⊆ *var set ts* ⊆ *trm* **and** *set φs* ⊆ *fmla*
shows *length* (*getFrN xs ts φs n*) = *n*
using *assms getFrN_FvarsT_Fvars* **by** *auto*

lemma *getFrN_distinct*[*simp*,*intro*]:
assumes *set xs* ⊆ *var set ts* ⊆ *trm* **and** *set φs* ⊆ *fmla*
shows *distinct* (*getFrN xs ts φs n*)
using *assms getFrN_FvarsT_Fvars* **by** *auto*

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*
using *assms* **by** (*induct txs arbitrary*: *t*) *fastforce*+

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−
  define us where us: us = getFrN (map snd txs) (t # map fst txs) [] (length txs)
  have us_facts: set us ⊆ var
    set us ∩ FvarsT t = {}
    set us ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
    set us ∩ snd ‘ (set txs) = {}
    length us = length txs
    distinct us
    using assms unfolding us
    using getFrN_FvarsT[of map snd txs t # map fst txs [] _ length txs]
      getFrN_var[of map snd txs t # map fst txs [] _ length txs]
      getFrN_length[of map snd txs t # map fst txs [] length txs]
      getFrN_distinct[of map snd txs t # map fst txs [] length txs]
   by auto (metis (no_types, opaque_lifting) IntI empty_iff image_iff old.prod.inject surjective_pairing)
  show ?thesis using assms us_facts unfolding psubstT_def
    by (force simp: Let_def us[symmetric] dest: set_zip_leftD set_zip_rightD intro!: rawpsubstT)
qed

lemma rawpsubstT_Var_not[simp]:
  assumes x ∈ var snd ‘ (set txs) ⊆ var fst ‘ (set txs) ⊆ trm
    and x ∉ snd ‘ (set txs)
  shows rawpsubstT (Var x) txs = Var x
  using assms by (induct txs) auto

lemma psubstT_Var_not[simp]:
  assumes x ∈ var snd ‘ (set txs) ⊆ var fst ‘ (set txs) ⊆ trm
    and x ∉ snd ‘ (set txs)
  shows psubstT (Var x) txs = Var x
proof−
  define us where us: us = getFrN (map snd txs) (Var x # map fst txs) [] (length txs)
  have us_facts: set us ⊆ var
    x ∉ set us
    set us ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
    set us ∩ snd ‘ (set txs) = {}
    length us = length txs
    using assms unfolding us
    using getFrN_FvarsT[of map snd txs Var x # map fst txs [] Var x length txs]
      getFrN_FvarsT[of map snd txs Var x # map fst txs [] _ length txs]
      getFrN_var[of map snd txs Var x # map fst txs [] Var x length txs]
      getFrN_length[of map snd txs Var x # map fst txs [] length txs]
    by (auto simp: set_eq_iff)
  have [simp]: rawpsubstT (Var x) (zip (map Var us) (map snd txs)) = Var x
    using assms us_facts
    by(intro rawpsubstT_Var_not) (force dest: set_zip_rightD set_zip_leftD)+
  have [simp]: rawpsubstT (Var x) (zip (map fst txs) us) = Var x
    using assms us_facts
    by(intro rawpsubstT_Var_not) (force dest: set_zip_rightD set_zip_leftD)+
  show ?thesis using assms us_facts unfolding psubstT_def
    by (auto simp: Let_def us[symmetric])
qed
```

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 $rawpsubstT\ t\ txs = t$
  using *assms* **by** (*induct txs*) *auto*

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−
  define *us* **where** *us*: $us = getFrN\ (map\ snd\ txs)\ (t\ \#\ map\ fst\ txs)\ []\ (length\ txs)$
  have *us_facts*: $set\ us \subseteq var$
    $set\ us \cap FvarsT\ t = \{\}$
    $set\ us \cap \bigcup\ (FvarsT\ `\ (fst\ `\ (set\ txs))) = \{\}$
    $set\ us \cap snd\ `\ (set\ txs) = \{\}$
    $length\ us = length\ txs$
    using *assms* **unfolding** *us*
    using *getFrN_FvarsT*[*of map snd txs t # map fst txs [] _ length txs*]
      getFrN_var[*of map snd txs t # map fst txs [] t length txs*]
      getFrN_length[*of map snd txs t # map fst txs [] length txs*]
    by (*auto simp*: *set_eq_iff*)
  have [*simp*]: $rawpsubstT\ t\ (zip\ (map\ Var\ us)\ (map\ snd\ txs)) = t$
    using *assms us_facts*
    by(*intro rawpsubstT_notIn*) (*auto 0 3 dest*: *set_zip_rightD set_zip_leftD*)
  have [*simp*]: $rawpsubstT\ t\ (zip\ (map\ fst\ txs)\ us) = t$
    using *assms us_facts*
    by(*intro rawpsubstT_notIn*) (*auto 0 3 dest*: *set_zip_rightD set_zip_leftD*)
  show *?thesis* **using** *assms us_facts* **unfolding** *psubstT_def*
    by (*auto simp*: *Let_def us*[*symmetric*])
qed

2.1.4 Parallel Formula Substitution

fun $rawpsubst :: 'fmla \Rightarrow ('trm \times 'var)\ list \Rightarrow 'fmla$ **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 $rawpsubst\ \varphi\ txs \in fmla$
  using *assms* **by** (*induct txs arbitrary*: φ) *fastforce*+

definition $psubst :: 'fmla \Rightarrow ('trm \times 'var)\ list \Rightarrow 'fmla$ **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−
  define *us* **where** *us*: $us = getFrN\ (map\ snd\ txs)\ (map\ fst\ txs)\ [\varphi]\ (length\ txs)$
  have *us_facts*: $set\ us \subseteq var$
    $set\ us \cap Fvars\ \varphi = \{\}$
    $set\ us \cap \bigcup\ (FvarsT\ `\ (fst\ `\ (set\ txs))) = \{\}$
    $set\ us \cap snd\ `\ (set\ txs) = \{\}$

```
    length us = length txs
    distinct us
    using assms unfolding us
    using getFrN_FvarsT[of map snd txs map fst txs [φ] _ length txs]
      getFrN_Fvars[of map snd txs map fst txs [φ] φ length txs]
      getFrN_var[of map snd txs map fst txs [φ] _ length txs]
      getFrN_length[of map snd txs map fst txs [φ] length txs]
      getFrN_distinct[of map snd txs map fst txs [φ] length txs]
    by (auto 8 0 simp: set_eq_iff image_iff Bex_def Ball_def)
  show ?thesis using assms us_facts unfolding psubst_def
    by (auto 0 3 simp: Let_def us[symmetric] dest: set_zip_rightD set_zip_leftD intro!: rawpsubst)
qed

lemma Fvars_rawpsubst_su:
  assumes φ ∈ fmla and snd ` (set txs) ⊆ var and fst ` (set txs) ⊆ trm
  shows Fvars (rawpsubst φ txs) ⊆
      (Fvars φ − snd ` (set txs)) ∪ (⋃ {FvarsT t | t x . (t,x) ∈ set txs})
  using assms proof(induction txs arbitrary: φ)
  case (Cons tx txs φ)
  then obtain t x where tx: tx = (t,x) by force
  have t: t ∈ trm and x: x ∈ var using Cons.prems unfolding tx by auto
  define χ where χ = subst φ t x
  have 0: Fvars χ =  Fvars φ − {x} ∪ (if x ∈ Fvars φ then FvarsT t else {})
    using Cons.prems unfolding χ_def by (auto simp: tx t)
  have χ: χ ∈ fmla unfolding χ_def using Cons.prems t x by auto
  have Fvars (rawpsubst χ txs) ⊆
      (Fvars χ − snd ` (set txs)) ∪
      (⋃ {FvarsT t | t x . (t,x) ∈ set txs})
    using Cons.prems χ by (intro Cons.IH) auto
  also have ... ⊆ Fvars φ − insert x (snd ` set txs) ∪ ⋃{FvarsT ta |ta. ∃xa. ta = t ∧ xa = x ∨ (ta, xa)
∈ set txs}
    (is _ ⊆ ?R) by(auto simp: 0 tx Cons.prems)
  finally have 1: Fvars (rawpsubst χ txs) ⊆ ?R .
  have 2: Fvars χ = Fvars φ − {x} ∪ (if x ∈ Fvars φ then FvarsT t else {})
    using Cons.prems t x unfolding χ_def using Fvars_subst by auto
  show ?case using 1 by (simp add: tx χ_def[symmetric] 2)
qed auto

lemma in_Fvars_rawpsubst_imp:
  assumes y ∈ Fvars (rawpsubst φ txs)
    and φ ∈ fmla and snd ` (set txs) ⊆ var and fst ` (set txs) ⊆ trm
  shows (y ∈ Fvars φ − snd ` (set txs)) ∨
     (y ∈ ⋃ { FvarsT t | t x . (t,x) ∈ set txs})
  using Fvars_rawpsubst_su[OF assms(2−4)]
  using assms(1) by blast

lemma Fvars_rawpsubst:
  assumes φ ∈ fmla and snd ` (set txs) ⊆ var and fst ` (set txs) ⊆ trm
    and distinct (map snd txs) and ∀ x ∈ snd ` (set txs). ∀ t ∈ fst ` (set txs). x ∉ FvarsT t
  shows Fvars (rawpsubst φ txs) =
      (Fvars φ − snd ` (set txs)) ∪
      (⋃ {if x ∈ Fvars φ then FvarsT t else {} | t x . (t,x) ∈ set txs})
  using assms proof(induction txs arbitrary: φ)
  case (Cons a txs φ)
  then obtain t x where a: a = (t,x) by force
  have t: t ∈ trm and x: x ∈ var using Cons.prems unfolding a by auto
  have x_txs: ⋀ta xa. (ta, xa) ∈ set txs ⟹ x ≠ xa using ‹distinct (map snd (a # txs))›
   unfolding a by (auto simp: rev_image_eqI)
```

```
  have xt: x ∉ FvarsT t ∧ snd ‘ set txs ∩ FvarsT t = {} using Cons.prems unfolding a by auto
  hence 0: Fvars φ − {x} ∪ FvarsT t − snd ‘ set txs =
          Fvars φ − insert x (snd ‘ set txs) ∪ FvarsT t
    by auto
  define χ where χ_def: χ = subst φ t x
  have χ: χ ∈ fmla unfolding χ_def using Cons.prems t x by auto
  have 1: Fvars (rawpsubst χ txs) =
      (Fvars χ − snd ‘ (set txs)) ∪
      (⋃ {if x ∈ Fvars χ then FvarsT t else {} | t x . (t,x) ∈ set txs})
    using Cons.prems χ by (intro Cons.IH) auto
  have 2: Fvars χ = Fvars φ − {x} ∪ (if x ∈ Fvars φ then FvarsT t else {})
    using Cons.prems t x unfolding χ_def using Fvars_subst by auto

  define f where f ≡ λta xa. if xa ∈ Fvars φ then FvarsT ta else {}

  have 3: ⋃ {f ta xa  |ta xa. (ta, xa) ∈ set ((t, x) # txs)} =
       f t x ∪ (⋃ {f ta xa  |ta xa. (ta, xa) ∈ set txs}) by auto
  have 4: snd ‘ set ((t, x) # txs) = {x} ∪ snd ‘ set txs by auto
  have 5: f t x ∩ snd ‘ set txs = {} unfolding f_def using xt by auto
  have 6: ⋃ {if xa ∈ Fvars φ − {x} ∪ f t x then FvarsT ta else {} | ta xa. (ta, xa) ∈ set txs}
    = (⋃ {f ta xa | ta xa. (ta, xa) ∈ set txs})
  unfolding f_def using xt x_txs by (fastforce split: if_splits)

  have Fvars φ − {x} ∪ f t x  − snd ‘ set txs ∪
    ⋃ {if xa ∈ Fvars φ − {x} ∪ f t x then FvarsT ta else {}
        | ta xa. (ta, xa) ∈ set txs} =
       Fvars φ − snd ‘ set ((t, x) # txs) ∪
    ⋃ {f ta xa  |ta xa. (ta, xa) ∈ set ((t, x) # txs)}
  unfolding 3 4 6 unfolding Un_Diff2[OF 5] Un_assoc unfolding Diff_Diff_Un ..

  thus ?case unfolding a rawpsubst.simps 1 2 χ_def[symmetric] f_def by simp
qed auto

lemma in_Fvars_rawpsubstD:
  assumes y ∈ Fvars (rawpsubst φ txs)
    and φ ∈ fmla and snd ‘ (set txs) ⊆ var and fst ‘ (set txs) ⊆ trm
    and distinct (map snd txs) and ∀ x ∈ snd ‘ (set txs). ∀ t ∈ fst ‘ (set txs). x ∉ FvarsT t
  shows (y ∈ Fvars φ − snd ‘ (set txs)) ∨
     (y ∈ ⋃ {if x ∈ Fvars φ then FvarsT t else {} | t x . (t,x) ∈ set txs})
  using Fvars_rawpsubst assms by auto

lemma Fvars_psubst:
  assumes φ ∈ fmla and snd ‘ (set txs) ⊆ var and fst ‘ (set txs) ⊆ trm
    and distinct (map snd txs)
  shows Fvars (psubst φ txs) =
      (Fvars φ − snd ‘ (set txs)) ∪
      (⋃ {if x ∈ Fvars φ then FvarsT t else {} | t x . (t,x) ∈ set txs})
proof−
  define us where us: us = getFrN (map snd txs) (map fst txs) [φ] (length txs)
  have 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
    using assms unfolding us
    using getFrN_FvarsT[of map snd txs map fst txs [φ] _ length txs]
      getFrN_Fvars[of map snd txs map fst txs [φ] _ length txs]
```

```
    getFrN_var[of map snd txs map fst txs [φ] _ length txs]
    getFrN_length[of map snd txs map fst txs [φ] length txs]
    getFrN_length[of map snd txs map fst txs [φ] length txs]
  by (auto 9 0 simp: set_eq_iff image_iff)
define χ where χ_def: χ = rawpsubst φ (zip (map Var us) (map snd txs))
have χ: χ ∈ fmla unfolding χ_def using assms us_facts
  by (intro rawpsubst) (auto dest!: set_zip_D)
have set_us: set us = snd ‘ (set (zip (map fst txs) us))
   using us_facts by (intro snd_set_zip[symmetric]) auto
have set_txs: snd ‘ set txs = snd ‘ (set (zip (map Var us) (map snd txs)))
   using us_facts by (intro snd_set_zip_map_snd[symmetric]) auto
have ⋀ t x. (t, x) ∈ set (zip (map Var us) (map snd txs)) ⟹ ∃ u. t = Var u
  using us_facts set_zip_leftD by fastforce
hence 00: ⋀ t x. (t, x) ∈ set (zip (map Var us) (map snd txs))
  ⟷ (∃ u ∈ var. t = Var u ∧ (Var u, x) ∈ set (zip (map Var us) (map snd txs)))
  using us_facts set_zip_leftD by fastforce
have Fvars χ =
     Fvars φ − snd ‘ set txs ∪
     ⋃{if x ∈ Fvars φ then FvarsT t else {} |t x.
         (t, x) ∈ set (zip (map Var us) (map snd txs))}
  unfolding χ_def set_txs using assms us_facts
  apply(intro Fvars_rawpsubst)
  subgoal by auto
  subgoal by (auto dest!: set_zip_rightD)
  subgoal by (auto dest!: set_zip_leftD)
  subgoal by auto
  subgoal  by (auto 0 6 simp: set_txs[symmetric] set_eq_iff subset_eq image_iff in_set_zip
    dest: spec[where P=λx. x ∈ set us ⟶ (∀ y ∈ set txs. x ≠ snd y), THEN mp[OF _ nth_mem]]) .
also have ... =
  Fvars φ − snd ‘ set txs ∪
 ⋃{if x ∈ Fvars φ then {u} else {} |u x. u ∈ var ∧ (Var u, x) ∈ set (zip (map Var us) (map snd txs))}
  (is ... = ?R)
  using FvarsT_Var by (metis (no_types, opaque_lifting) 00)
finally have 0: Fvars χ = ?R .
have 1: Fvars (rawpsubst χ (zip (map fst txs) us)) =
     (Fvars χ − set us) ∪
     (⋃ {if u ∈ Fvars χ then FvarsT t else {} | t u . (t,u) ∈ set (zip (map fst txs) us)})
  unfolding set_us using us_facts assms χ
  apply (intro Fvars_rawpsubst)
  subgoal by (auto dest: set_zip_rightD)
  subgoal by (auto dest: set_zip_rightD)
  subgoal by (auto dest!: set_zip_leftD)
  subgoal by (auto dest!: set_zip_leftD)
  subgoal by (metis IntI Union_iff empty_iff fst_set_zip_map_fst image_eqI set_us) .
have 2: Fvars χ − set us = Fvars φ − snd ‘ set txs
  unfolding 0 using us_facts(1,2)
  by (fastforce dest!: set_zip_leftD split: if_splits)
have 3:
  (⋃ {if u ∈ Fvars χ then FvarsT t else {} | t u . (t,u) ∈ set (zip (map fst txs) us)}) =
 (⋃ {if x ∈ Fvars φ then FvarsT t else {} | t x . (t,x) ∈ set txs})
proof safe
  fix xx tt y
  assume xx: xx ∈ (if y ∈ Fvars χ then FvarsT tt else {})
    and ty: (tt, y) ∈ set (zip (map fst txs) us)
  have ttin: tt ∈ fst ‘ set txs using ty using set_zip_leftD by fastforce
  have yin: y ∈ set us using ty by (meson set_zip_D)
  have yvar: y ∈ var using us_facts yin by auto
  have ynotin: y ∉ snd ‘ set txs y ∉ Fvars φ using yin us_facts by auto
```

```
    show xx ∈ ⋃{if x ∈ Fvars φ then FvarsT t else {} |t x. (t, x) ∈ set txs}
    proof(cases y ∈ Fvars χ)
      case True note y = True
      hence xx: xx ∈ FvarsT tt using xx by simp
      obtain x where xφ: x ∈ Fvars φ
        and yx: (Var y, x) ∈ set (zip (map Var us) (map snd txs))
        using y ynotin unfolding 0 by auto (metis empty_iff insert_iff)
      have yx: (y, x) ∈ set (zip us (map snd txs))
      using yvar us_facts by (intro inj_on_set_zip_map[OF inj_on_Var yx]) auto
      have (tt, x) ∈ set txs apply(rule set_zip_map_fst_snd[OF yx ty])
        using ‹distinct (map snd txs)› us_facts by auto
      thus ?thesis using xx xφ by auto
    qed(insert xx, auto)
  next
    fix y tt xx
    assume y: y ∈ (if xx ∈ Fvars φ then FvarsT tt else {})
      and tx: (tt, xx) ∈ set txs
    hence xxsnd: xx ∈ snd ` set txs by force
    obtain u where uin: u ∈ set us and uxx: (u, xx) ∈ set (zip us (map snd txs))
      by (metis xxsnd in_set_impl_in_set_zip2 length_map set_map set_zip_leftD us_facts(5))
    hence uvar: u ∈ var using us_facts by auto
    show y ∈ ⋃{if u ∈ Fvars χ then FvarsT t else {} |t u. (t, u) ∈ set (zip (map fst txs) us)}
    proof(cases xx ∈ Fvars φ)
      case True note xx = True
      hence y: y ∈ FvarsT tt using y by auto
      have (Var u, xx) ∈ set (zip (map Var us) (map snd txs))
      using us_facts by (intro set_zip_length_map[OF uxx]) auto
      hence uχ: u ∈ Fvars χ using uin xx uvar unfolding 0 by auto
      have ttu: (tt, u) ∈ set (zip (map fst txs) us)
      using assms us_facts by (intro set_zip_map_fst_snd2[OF uxx tx]) auto
      show ?thesis using uχ ttu y by auto
    qed(insert y, auto)
  qed
  show ?thesis
  by (simp add: psubst_def Let_def us[symmetric] χ_def[symmetric] 1 2 3)
qed

lemma in_Fvars_psubstD:
  assumes y ∈ Fvars (psubst φ txs)
    and φ ∈ fmla and snd ` (set txs) ⊆ var and fst ` (set txs) ⊆ trm
    and distinct (map snd txs)
  shows y ∈ (Fvars φ − snd ` (set txs)) ∪
          (⋃ {if x ∈ Fvars φ then FvarsT t else {} | t x . (t,x) ∈ set txs})
  using assms Fvars_psubst by auto

lemma subst2_fresh_switch:
  assumes φ ∈ fmla t ∈ trm s ∈ trm x ∈ var y ∈ var
    and x ≠ y x ∉ FvarsT s y ∉ FvarsT t
  shows subst (subst φ s y) t x = subst (subst φ t x) s y   (is ?L = ?R)
  using assms by (simp add: subst_compose_diff[of φ s t y x])

lemma rawpsubst2_fresh_switch:
  assumes φ ∈ fmla t ∈ trm s ∈ trm x ∈ var y ∈ var
    and x ≠ y x ∉ FvarsT s y ∉ FvarsT t
  shows rawpsubst φ ([(s,y),(t,x)]) = rawpsubst φ ([(t,x),(s,y)])
  using assms by (simp add: subst2_fresh_switch)

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 $rawpsubst\ (rawpsubst\ \varphi\ txs1)\ txs2 = rawpsubst\ \varphi\ (txs1\ @\ txs2)$
using *assms* **by** (*induct txs1 arbitrary*: *txs2* φ) *auto*

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 $rawpsubst\ (subst\ \varphi\ s\ y)\ txs = rawpsubst\ \varphi\ (txs\ @\ [(s,y)])$
using *assms* **proof**(*induction txs arbitrary*: φ *s y*)
case (*Cons tx txs*)
obtain *t x* **where** *tx*[*simp*]: $tx = (t,x)$ **by** *force*
have *x*: $x \in var$ **and** *t*: $t \in trm$ **using** *Cons* **unfolding** *tx* **by** *auto*
have $rawpsubst\ \varphi\ ((s,\ y)\ \#\ (t,\ x)\ \#\ txs) = rawpsubst\ \varphi\ ([(s,\ y),(t,\ x)]\ @\ txs)$ **by** *simp*
also have $\ldots = rawpsubst\ (rawpsubst\ \varphi\ [(s,\ y),(t,\ x)])\ txs$
using *Cons* **by** *auto*
also have $rawpsubst\ \varphi\ [(s,\ y),(t,\ x)] = rawpsubst\ \varphi\ [(t,\ x),(s,\ y)]$
using *Cons* **by** (*intro rawpsubst2_fresh_switch*) *auto*
also have $rawpsubst\ (rawpsubst\ \varphi\ [(t,\ x),(s,\ y)])\ txs = rawpsubst\ \varphi\ ([(t,\ x),(s,\ y)]\ @\ txs)$
using *Cons* **by** *auto*
also have $\ldots = rawpsubst\ (subst\ \varphi\ t\ x)\ (txs\ @\ [(s,y)])$ **using** *Cons* **by** *auto*
also have $\ldots = rawpsubst\ \varphi\ (((t,\ x)\ \#\ txs)\ @\ [(s,\ y)])$ **by** *simp*
finally show *?case* **unfolding** *tx* **by** *auto*
qed *auto*

lemma *subst_rawpsubst_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 $subst\ (rawpsubst\ \varphi\ txs)\ s\ y = rawpsubst\ \varphi\ ((s,y)\ \#\ txs)$
using *assms* **proof**(*induction txs arbitrary*: φ *s y*)
case (*Cons tx txs*)
obtain *t x* **where** *tx*[*simp*]: $tx = (t,x)$ **by** *force*
have *x*: $x \in var$ **and** *t*: $t \in trm$ **using** *Cons* **unfolding** *tx* **by** *auto*
have $subst\ (rawpsubst\ (subst\ \varphi\ t\ x)\ txs)\ s\ y = rawpsubst\ (subst\ \varphi\ t\ x)\ ((s,y)\ \#\ txs)$
using *Cons.prems* **by** (*intro Cons.IH*) *auto*
also have $\ldots = rawpsubst\ (rawpsubst\ \varphi\ [(t,x)])\ ((s,y)\ \#\ txs)$ **by** *simp*
also have $\ldots = rawpsubst\ \varphi\ ([(t,x)]\ @\ ((s,y)\ \#\ txs))$
using *Cons.prems* **by** *auto*
also have $\ldots = rawpsubst\ \varphi\ ([(t,x),(s,y)]\ @\ txs)$ **by** *simp*
also have $\ldots = rawpsubst\ (rawpsubst\ \varphi\ [(t,x),(s,y)])\ txs$
using *Cons.prems* **by** *auto*
also have $rawpsubst\ \varphi\ [(t,x),(s,y)] = rawpsubst\ \varphi\ [(s,y),(t,x)]$
using *Cons.prems* **by** (*intro rawpsubst2_fresh_switch*) *auto*
also have $rawpsubst\ (rawpsubst\ \varphi\ [(s,y),(t,x)])\ txs = rawpsubst\ \varphi\ ([(s,y),(t,x)]\ @\ txs)$
using *Cons.prems* **by** *auto*
finally show *?case* **by** *simp*
qed *auto*

lemma *rawpsubst_compose_freshVar*:
assumes $\varphi \in fmla\ snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq trm$
and $distinct\ (map\ snd\ txs)$
and $\bigwedge i\ j.\ i < j \Longrightarrow j < length\ txs \Longrightarrow snd\ (txs!j) \notin FvarsT\ (fst\ (txs!i))$

```
    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
  using assms proof(induction txs arbitrary: us φ)
  case (Cons tx txs uus φ)
  obtain t x where tx[simp]: tx = (t,x) by force
  obtain u us where uus[simp]: uus = u # us using Cons by (cases uus) auto
  have 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 and u_facts: u ∈ var u ∉ Fvars φ
    u ∉ ⋃ (FvarsT ‘ (fst ‘ (set txs)))
    u ∉ snd ‘ (set txs) u ∉ set us
    using Cons by auto
  let ?uxs = zip (map Var us) (map snd txs)
  have 1: rawpsubst (subst φ (Var u) x) ?uxs = rawpsubst φ (?uxs @ [(Var u,x)])
    using u_facts Cons.prems
    by (intro rawpsubst_subst_fresh_switch) (auto simp: subsetD dest!: set_zip_D)
  let ?uuxs = zip (map Var uus) (map snd (tx # txs))
  let ?tus = zip (map fst txs) us  let ?ttxs = zip (map fst (tx # txs)) uus
  have 2: u ∈ Fvars (rawpsubst φ (zip (map Var us) (map snd txs))) ⟹ False
  using Cons.prems apply− apply(drule in_Fvars_rawpsubstD)
  subgoal by auto
  subgoal by (auto dest!: set_zip_D)
  subgoal by (auto dest!: set_zip_D)
  subgoal by auto
  subgoal premises prems using us_facts(1,4,5)
    by (auto 0 3 simp: in_set_zip subset_eq set_eq_iff image_iff
      dest: spec[where P=λx. x ∈ set us ⟶ (∀ y ∈ set txs. x ≠ snd y),
        THEN mp[OF _ nth_mem], THEN bspec[OF _ nth_mem]])
  subgoal
    by (auto simp: in_set_zip subset_eq split: if_splits) .

  have 3: ⋀ xx tt. xx ∈ FvarsT t ⟹ (tt, xx) ∉ set txs
  using Cons.prems(4,5) tx unfolding set_conv_nth
  by simp (metis One_nat_def Suc_leI diff_Suc_1 fst_conv le_imp_less_Suc
          nth_Cons_0 snd_conv zero_less_Suc)

  have 00: rawpsubst (rawpsubst φ ?uuxs) ?ttxs = rawpsubst (subst (rawpsubst φ (?uxs @ [(Var u, x)]))
t u) ?tus
    by (simp add: 1)

  have rawpsubst φ (?uxs @ [(Var u, x)]) = rawpsubst (rawpsubst φ ?uxs) [(Var u, x)]
  using Cons.prems by (intro rawpsubst_compose[symmetric]) (auto intro!: rawpsubst dest!: set_zip_D)
  also have rawpsubst (rawpsubst φ ?uxs) [(Var u, x)] = subst (rawpsubst φ ?uxs) (Var u) x by simp
  finally have subst (rawpsubst φ (?uxs @ [(Var u, x)])) t u =
               subst (subst (rawpsubst φ ?uxs) (Var u) x) t u by simp
  also have ... = subst (rawpsubst φ ?uxs) t x
    using Cons 2 by (intro subst_subst) (auto intro!: rawpsubst dest!: set_zip_D)
  also have ... = rawpsubst φ ((t,x) # ?uxs)
  using Cons.prems 3 apply(intro subst_rawpsubst_fresh_switch)
```

```
    subgoal by (auto dest!: set_zip_D)
    subgoal by (auto dest!: set_zip_D)
    subgoal by (auto dest!: set_zip_D)
    subgoal by (auto dest!: set_zip_D)
    subgoal by (fastforce dest!: set_zip_D)
    by (auto dest!: set_zip_D)
  also have ... =  rawpsubst φ ([(t,x)] @ ?uxs) by simp
  also have ... = rawpsubst (rawpsubst φ [(t,x)]) ?uxs
    using Cons.prems by (intro rawpsubst_compose[symmetric]) (auto dest!: set_zip_D)
  finally have rawpsubst (subst (rawpsubst φ (?uxs @ [(Var u, x)])) t u) ?tus =
              rawpsubst (rawpsubst (rawpsubst φ [(t,x)]) ?uxs) ?tus by auto
  hence rawpsubst (rawpsubst φ ?uuxs) ?ttxs = rawpsubst (rawpsubst (rawpsubst φ [(t,x)]) ?uxs) ?tus
    using 00 by auto
  also have ... = rawpsubst (rawpsubst φ [(t,x)]) txs
    using Cons.prems apply (intro Cons.IH rawpsubst)
    subgoal by (auto dest!: set_zip_D in_Fvars_substD)
    subgoal by (auto dest!: set_zip_D in_Fvars_substD)
    subgoal by (auto dest!: set_zip_D in_Fvars_substD)
    subgoal by (auto dest!: set_zip_D in_Fvars_substD)
    subgoal by (auto dest!: set_zip_D in_Fvars_substD)
    subgoal by (auto dest!: set_zip_D in_Fvars_substD)
    subgoal by (metis Suc_mono diff_Suc_1 length_Cons nat.simps(3) nth_Cons')
    by (auto dest!: set_zip_D in_Fvars_substD)
  also have ... = rawpsubst φ ([(t,x)] @ txs)
    using Cons.prems by (intro rawpsubst_compose) (auto dest!: set_zip_D)
  finally show ?case by simp
qed auto

lemma rawpsubst_compose_freshVar2_aux:
  assumes φ[simp]: φ ∈ 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)
  using assms proof(induction xs arbitrary: φ ts us vs)
  case (Cons x xs φ tts uus vvs)
  obtain t ts u us v vs where tts[simp]: tts = t # ts and lts[simp]: length ts = length xs
    and uus[simp]: uus = u # us and lus[simp]: length us = length xs
    and vvs[simp]: vvs = v # vs and lvs[simp]: length vs = length xs
    using ‹length uus = length (x # xs)› ‹length vvs = length (x # xs)› ‹length tts = length (x # xs)›
    apply(cases tts)
    subgoal by auto
    subgoal apply(cases uus)
      subgoal by auto
      subgoal by (cases vvs) auto . .

  let ?φux = subst φ (Var u) x   let ?φvx = subst φ (Var v) x
```

```
  have 0: rawpsubst (rawpsubst ?φux (zip (map Var us) xs)) (zip ts us) =
          rawpsubst (rawpsubst ?φux (zip (map Var vs) xs)) (zip ts vs)
    apply(rule Cons.IH) using Cons.prems by (auto intro!: rawpsubst dest!: set_zip_D)

  have 1: rawpsubst ?φux (zip (map Var vs) xs) =
          subst (rawpsubst φ (zip (map Var vs) xs)) (Var u) x
    using Cons.prems
    by (intro subst_rawpsubst_fresh_switch[simplified,symmetric])
       (force intro!: rawpsubst dest!: set_zip_D simp: subset_eq)+

  have 11: rawpsubst ?φvx (zip (map Var vs) xs) =
          subst (rawpsubst φ (zip (map Var vs) xs)) (Var v) x
    using Cons.prems
    by (intro subst_rawpsubst_fresh_switch[simplified,symmetric])
       (auto intro!: rawpsubst dest!: set_zip_D simp: subset_eq)

  have subst (subst (rawpsubst φ (zip (map Var vs) xs)) (Var u) x) t u =
       subst (rawpsubst φ (zip (map Var vs) xs)) t x
  using Cons.prems
  by (intro subst_subst) (force intro!: rawpsubst dest!: set_zip_D in_Fvars_rawpsubst_imp simp: Fvars_rawpsubst)+
  also have ... = subst (subst (rawpsubst φ (zip (map Var vs) xs)) (Var v) x) t v
  using Cons.prems
  by (intro subst_subst[symmetric])
     (force intro!: rawpsubst dest!: set_zip_D in_Fvars_rawpsubst_imp simp: Fvars_rawpsubst)+

  finally have
    2: subst (subst (rawpsubst φ (zip (map Var vs) xs)) (Var u) x) t u =
       subst (subst (rawpsubst φ (zip (map Var vs) xs)) (Var v) x) t v  .

  have rawpsubst (subst (rawpsubst ?φux (zip (map Var us) xs)) t u) (zip ts us) =
       subst (rawpsubst (rawpsubst ?φux (zip (map Var us) xs)) (zip ts us)) t u
    using Cons.prems
   by (intro subst_rawpsubst_fresh_switch[simplified,symmetric]) (auto intro!: rawpsubst dest!: set_zip_D)
  also have ... = subst (rawpsubst (rawpsubst ?φux (zip (map Var vs) xs)) (zip ts vs)) t u
    unfolding 0 ..
  also have ... = rawpsubst (subst (rawpsubst ?φux (zip (map Var vs) xs)) t u) (zip ts vs)
    using Cons.prems
    by (intro subst_rawpsubst_fresh_switch[simplified]) (auto intro!: rawpsubst dest!: set_zip_D)
  also have ... = rawpsubst (subst (subst (rawpsubst φ (zip (map Var vs) xs)) (Var u) x) t u) (zip ts vs)
    unfolding 1 ..
  also have ... = rawpsubst (subst (subst (rawpsubst φ (zip (map Var vs) xs)) (Var v) x) t v) (zip ts vs)
    unfolding 2 ..
  also have ... = rawpsubst (subst (rawpsubst ?φvx (zip (map Var vs) xs)) t v) (zip ts vs)
    unfolding 11 ..
  finally have rawpsubst (subst (rawpsubst ?φux (zip (map Var us) xs)) t u) (zip ts us) =
       rawpsubst (subst (rawpsubst ?φvx (zip (map Var vs) xs)) t v) (zip ts vs) .
  thus ?case by simp
qed auto
```

... now getting rid of the disjointness hypothesis:

```
lemma rawpsubst_compose_freshVar2:
  assumes φ[simp]: φ ∈ 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−
  define ws where ws = getFrN (xs @ us @ vs) ts [φ] (length xs)
  note fv = getFrN_Fvars[of xs @ us @ vs ts [φ] _ length xs]
  and fvt = getFrN_FvarsT[of xs @ us @ vs ts [φ] _ length xs]
  and var = getFrN_var[of xs @ us @ vs ts [φ] _ length xs]
  and l = getFrN_length[of xs @ us @ vs ts [φ] length xs]
  have ws_facts: set ws ⊆ var  distinct ws
    set ws ∩ Fvars φ = {}
    set ws ∩ ⋃ (FvarsT ` (set ts)) = {}
    set ws ∩ set xs = {} set ws ∩ set us = {} set ws ∩ set vs = {}
    length ws = length xs using assms unfolding ws_def
    apply −
    subgoal by auto
    subgoal by auto
    subgoal using fv by auto
    subgoal using fvt IntI empty_iff by fastforce
    subgoal using var IntI empty_iff by fastforce
    subgoal using var IntI empty_iff by fastforce
    subgoal using var IntI empty_iff by fastforce
    subgoal using l by auto  .
  have ?L = rawpsubst (rawpsubst φ (zip (map Var ws) xs)) (zip ts ws)
    apply(rule rawpsubst_compose_freshVar2_aux) using assms ws_facts by auto
  also have ... = ?R
    apply(rule rawpsubst_compose_freshVar2_aux) using assms ws_facts by auto
  finally show ?thesis .
qed

lemma psubst_subst_fresh_switch:
  assumes φ ∈ fmla snd ` set txs ⊆ var fst ` set txs ⊆ trm
    and ∀ x∈snd ` set txs. x ∉ FvarsT s ∀ t∈fst ` set txs. y ∉ FvarsT t
    and distinct (map snd txs)
    and s ∈ trm y ∈ var y ∉ snd ` set txs
  shows psubst (subst φ s y) txs = subst (psubst φ txs) s y
proof−
  define us where us: us = getFrN (map snd txs) (map fst txs) [φ] (length txs)
  note fvt = getFrN_FvarsT[of map snd txs map fst txs [φ] _ length txs]
  and fv = getFrN_Fvars[of map snd txs map fst txs [φ] _ length txs]
  and var = getFrN_var[of map snd txs map fst txs [φ] _ length txs]
  and l = getFrN_length[of map snd txs map fst txs [φ] length txs]

  have 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
  using assms unfolding us apply −
  subgoal by auto
  subgoal using fv by (cases txs, auto)
  subgoal using fvt by (cases txs, auto)
  subgoal using var by (cases txs, auto)
```

subgoal using *l* **by** *auto*
subgoal by *auto* **.**

define *vs* **where** *vs*: *vs* = *getFrN* (*map snd txs*) (*map fst txs*) [*subst* φ *s y*] (*length txs*)
note *fvt* = *getFrN_FvarsT*[*of map snd txs map fst txs* [*subst* φ *s y*] _ *length txs*]
and *fv* = *getFrN_Fvars*[*of map snd txs map fst txs* [*subst* φ *s y*] _ *length txs*]
and *var* = *getFrN_var*[*of map snd txs map fst txs* [*subst* φ *s y*] _ *length txs*]
and *l* = *getFrN_length*[*of map snd txs map fst txs* [*subst* φ *s y*] *length txs*]

have *vs_facts*: *set vs* $\subseteq$ *var*
set vs $\cap$ *Fvars* (*subst* φ *s y*) = {}
set vs $\cap \bigcup$ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set vs $\cap$ *snd* ‘ (*set txs*) = {}
length vs = *length txs*
distinct vs
using *assms* **unfolding** *vs* **apply** −
subgoal by *auto*
subgoal using *fv* **by** (*cases txs*, *auto*)
subgoal using *fvt* **by** (*cases txs*, *auto*)
subgoal using *var* **by** (*cases txs*, *auto*)
subgoal using *l* **by** *auto*
subgoal by *auto* **.**

define *ws* **where** *ws*: *ws* = *getFrN* (*y* # *map snd txs*) (*s* # *map fst txs*) [φ] (*length txs*)
note *fvt* = *getFrN_FvarsT*[*of y* # *map snd txs s* # *map fst txs* [φ] _ *length txs*]
and *fv* = *getFrN_Fvars*[*of y* # *map snd txs s* # *map fst txs* [φ] _ *length txs*]
and *var* = *getFrN_var*[*of y* # *map snd txs s* # *map fst txs* [φ] _ *length txs*]
and *l* = *getFrN_length*[*of y* # *map snd txs s* # *map fst txs* [φ] *length txs*]

have *ws_facts*: *set ws* $\subseteq$ *var*
set ws $\cap$ *Fvars* φ = {} *y* $\notin$ *set ws set ws* $\cap$ *FvarsT s* = {}
set ws $\cap \bigcup$ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set ws $\cap$ *snd* ‘ (*set txs*) = {}
length ws = *length txs*
distinct ws
using *assms* **unfolding** *ws* **apply** −
subgoal by *auto*
subgoal using *fv* **by** (*cases txs*, *auto*)
subgoal using *var* **by** (*cases txs*, *auto*)
subgoal using *fvt* **by** (*cases txs*, *auto*)
subgoal using *fvt* **by** (*cases txs*, *auto*)
subgoal using *var* **by** (*cases txs*, *auto*)
subgoal using *l* **by** (*cases txs*, *auto*)
by *auto*

let *?vxs* = *zip* (*map Var vs*) (*map snd txs*)
let *?tvs* = (*zip* (*map fst txs*) *vs*)
let *?uxs* = *zip* (*map Var us*) (*map snd txs*)
let *?tus* = (*zip* (*map fst txs*) *us*)
let *?wxs* = *zip* (*map Var ws*) (*map snd txs*)
let *?tws* = (*zip* (*map fst txs*) *ws*)

have *0*: *rawpsubst* (*subst* φ *s y*) *?wxs* = *subst* (*rawpsubst* φ *?wxs*) *s y*
apply(*subst rawpsubst_compose*[*of* φ *?wxs* [(*s*,*y*)],*simplified*])
using *assms ws_facts* **apply** −
subgoal by *auto*
subgoal by (*auto dest*!: *set_zip_D*)
subgoal by (*auto dest*!: *set_zip_D*)

```
  subgoal by auto
  subgoal by auto
  subgoal apply(subst rawpsubst_subst_fresh_switch)
    by (auto dest!: set_zip_D simp: subset_eq rawpsubst_subst_fresh_switch) .

  have 1: rawpsubst (rawpsubst φ ?wxs) ?tws = rawpsubst (rawpsubst φ ?uxs) ?tus
    using assms ws_facts us_facts by (intro rawpsubst_compose_freshVar2) (auto simp: subset_eq)

  have rawpsubst (rawpsubst (subst φ s y) ?vxs) ?tvs =
        rawpsubst (rawpsubst (subst φ s y) ?wxs) ?tws
    using assms ws_facts vs_facts
    by (intro rawpsubst_compose_freshVar2)  (auto simp: subset_eq)
  also have ... = rawpsubst (subst (rawpsubst φ ?wxs) s y) ?tws unfolding 0 ..
  also have ... = subst (rawpsubst (rawpsubst φ ?wxs) ?tws) s y
  apply(subst rawpsubst_compose[of rawpsubst φ ?wxs ?tws  [(s,y)],simplified])
  using assms ws_facts apply −
  subgoal by (auto dest!: set_zip_D simp: subset_eq intro!: rawpsubst)
  subgoal by (auto dest!: set_zip_D)
  subgoal by (auto dest!: set_zip_D)
  subgoal by auto
  subgoal by auto
  subgoal by (subst rawpsubst_subst_fresh_switch)
    (auto dest!: set_zip_D simp: subset_eq rawpsubst_subst_fresh_switch
          intro!: rawpsubst) .
  also have ... = subst (rawpsubst (rawpsubst φ ?uxs) ?tus) s y unfolding 1 ..
  finally show  ?thesis unfolding psubst_def by (simp add: Let_def vs[symmetric] us[symmetric])
qed
```

For many cases, the simpler rawpsubst can replace psubst:

```
lemma psubst_eq_rawpsubst:
  assumes φ ∈ 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))
  shows psubst φ txs = rawpsubst φ txs
proof−
  define us where us: us = getFrN (map snd txs) (map fst txs) [φ] (length txs)
  note fvt = getFrN_FvarsT[of map snd txs map fst txs [φ] _ length txs]
  and fv = getFrN_Fvars[of map snd txs map fst txs [φ] _ length txs]
  and var = getFrN_var[of map snd txs map fst txs [φ] _ length txs]
  and l = getFrN_length[of map snd txs map fst txs [φ] length txs]
  have 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
    using assms unfolding us
    apply −
    subgoal by auto
    subgoal using fv by auto
    subgoal using fvt by force
    subgoal using var by (force simp: image_iff)
    using l by auto
  show ?thesis
    using rawpsubst_compose_freshVar assms us_facts
    by (simp add: psubst_def Let_def us[symmetric])
qed
```

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−
have *psubst* φ $[(t,x)]$ = *rawpsubst* φ $[(t,x)]$ **apply**(*rule psubst_eq_rawpsubst*)
using *assms* **by** *auto*
thus *?thesis* **by** *auto*
qed

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 $[(t1,x1),(t2,x2)]$
apply(*rule psubst_eq_rawpsubst*)
using *assms* **using** *less_SucE* **by** *force*+

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 $[(t1,x1),(t2,x2),(t3,x3)]$
using *assms* **using** *less_SucE* **apply**(*intro psubst_eq_rawpsubst*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *i j*
apply(*cases j*)
subgoal by *auto*
subgoal by (*simp add*: *nth_Cons'*) . .

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 $[(t1,x1),(t2,x2),(t3,x3),(t4,x4)]$
using *assms* **using** *less_SucE* **apply**(*intro psubst_eq_rawpsubst*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *i j*
apply(*cases j*)
subgoal by *auto*
subgoal by (*simp add*: *nth_Cons'*) . .

lemma *rawpsubst_same_Var*[*simp*]:
assumes $\varphi \in fmla\ set\ xs \subseteq var$
shows *rawpsubst* φ (*map* ($\lambda x.$ (*Var x*,*x*)) *xs*) = φ
using *assms* **by** (*induct xs*) *auto*

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

```
  using assms by (intro psubst_eq_rawpsubst) (auto simp: nth_eq_iff_index_eq subsetD)
  thus ?thesis using assms by auto
qed

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
  using assms by (induct txs) auto

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−
  define us where us: us = getFrN (map snd txs) (map fst txs) [φ] (length txs)
  have us_facts: set us ⊆ var
    set us ∩ Fvars φ = {}
    set us ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
    set us ∩ snd ‘ (set txs) = {}
    length us = length txs
    using getFrN_Fvars[of map snd txs map fst txs [φ] _ length txs]
      getFrN_FvarsT[of map snd txs map fst txs [φ] _ length txs]
      getFrN_var[of map snd txs map fst txs [φ] _ length txs]
      getFrN_length[of map snd txs map fst txs [φ] length txs]
    using assms unfolding us apply −
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (fastforce simp: image_iff)
    subgoal by auto .

  have [simp]: rawpsubst φ (zip (map Var us) (map snd txs)) = φ
  using assms us_facts apply(intro rawpsubst_notIn)
  subgoal by (auto dest!: set_zip_rightD)
  subgoal by (auto dest!: set_zip_leftD)
  subgoal by auto
  subgoal by (auto dest!: set_zip_rightD) .
  have [simp]: rawpsubst φ (zip (map fst txs) us) = φ
    using assms us_facts apply(intro rawpsubst_notIn)
  subgoal by (auto dest!: set_zip_rightD)
  subgoal by (auto dest!: set_zip_leftD)
  subgoal by auto
  subgoal by (auto dest!: set_zip_rightD) .
  show ?thesis using assms us_facts unfolding psubst_def
    by(auto simp: Let_def us[symmetric])
qed

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*
using *num* **by** *auto*

lemma *in_num*[*simp*]: *n* ∈ *num* ⟹ *n* ∈ *trm* **using** *num* **by** *auto*

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*
using *assms* **by** (*simp add*: *subst_comp*)

lemma *rawpsubstT_num*:
assumes *snd* ‘ (*set txs*) ⊆ *var fst* ‘ (*set txs*) ⊆ *trm n* ∈ *num*
shows *rawpsubstT n txs* = *n*
using *assms* **by** (*induct txs*) *auto*

lemma *psubstT_num*[*simp*]:
assumes *snd* ‘ (*set txs*) ⊆ *var fst* ‘ (*set txs*) ⊆ *trm n* ∈ *num*
shows *psubstT n txs* = *n*
proof−
define *us* **where** *us*: *us* = *getFrN* (*map snd txs*) (*n* # *map fst txs*) [] (*length txs*)
have *us_facts*: *set us* ⊆ *var*
set us ∩ *FvarsT n* = {}
set us ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set us ∩ *snd* ‘ (*set txs*) = {}
length us = *length txs*
using *assms* **unfolding** *us*
using *getFrN_Fvars*[*of map snd txs n* # *map fst txs* [] _ *length txs*]
getFrN_FvarsT[*of map snd txs n* # *map fst txs* [] _ *length txs*]
getFrN_var[*of map snd txs n* # *map fst txs* [] _ *length txs*]
getFrN_length[*of map snd txs n* # *map fst txs* [] *length txs*]
by (*auto 7 0 simp*: *set_eq_iff image_iff*)
let *?t* = *rawpsubstT n* (*zip* (*map Var us*) (*map snd txs*))
have *t*: *?t* = *n*
using *assms us_facts* **apply**(*intro rawpsubstT_num*)
subgoal by (*auto dest*!: *set_zip_rightD*)
subgoal by (*auto dest*!: *set_zip_leftD*)
subgoal by *auto* **.**
have *rawpsubstT ?t* (*zip* (*map fst txs*) *us*) = *n*
unfolding *t* **using** *assms us_facts* **apply**(*intro rawpsubstT_num*)
subgoal by (*auto dest*!: *set_zip_rightD*)
subgoal by (*auto dest*!: *set_zip_leftD*)
subgoal by *auto* **.**
thus *?thesis* **unfolding** *psubstT_def* **by**(*simp add*: *Let_def us*[*symmetric*])
qed

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*]:

$\bigwedge x\ y\ \varphi\ t.\ \varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow y \in var \Longrightarrow$
$x \neq y \Longrightarrow x \notin FvarsT\ t \Longrightarrow subst\ (exi\ x\ \varphi)\ t\ y = exi\ x\ (subst\ \varphi\ t\ y)$
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**
$eqv\ \varphi\ \chi = cnj\ (imp\ \varphi\ \chi)\ (imp\ \chi\ \varphi)$

lemma
eqv[*simp*]: $\bigwedge\varphi\ \chi.\ \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow eqv\ \varphi\ \chi \in fmla$
and
Fvars_eqv[*simp*]: $\bigwedge\varphi\ \chi.\ \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow$
$Fvars\ (eqv\ \varphi\ \chi) = Fvars\ \varphi \cup Fvars\ \chi$
and
subst_eqv[*simp*]:
$\bigwedge\varphi\ \chi\ t\ x.\ \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow$
$subst\ (eqv\ \varphi\ \chi)\ t\ x = eqv\ (subst\ \varphi\ t\ x)\ (subst\ \chi\ t\ x)$
unfolding *eqv_def* **by** *auto*

lemma *subst_all_idle*[*simp*]:
assumes [*simp*]: $x \in var\ \varphi \in fmla\ t \in trm$
shows $subst\ (all\ x\ \varphi)\ t\ x = all\ x\ \varphi$
by (*intro subst_notIn*) *auto*

lemma *subst_exi_idle*[*simp*]:
assumes [*simp*]: $x \in var\ \varphi \in fmla\ t \in trm$
shows $subst\ (exi\ x\ \varphi)\ t\ x = exi\ x\ \varphi$
by (*rule subst_notIn*) *auto*

Parallel substitution versus connectives and quantifiers.

lemma *rawpsubst_cnj*:
assumes $\varphi1 \in fmla\ \varphi2 \in fmla$
and $snd\ `\ (set\ txs) \subseteq var\ fst\ `\ (set\ txs) \subseteq trm$
shows $rawpsubst\ (cnj\ \varphi1\ \varphi2)\ txs = cnj\ (rawpsubst\ \varphi1\ txs)\ (rawpsubst\ \varphi2\ txs)$
using *assms* **by** (*induct txs arbitrary*: $\varphi1\ \varphi2$) *auto*

lemma *psubst_cnj*[*simp*]:
assumes $\varphi1 \in fmla\ \varphi2 \in fmla$
and $snd\ `\ (set\ txs) \subseteq var\ fst\ `\ (set\ txs) \subseteq trm$
and $distinct\ (map\ snd\ txs)$
shows $psubst\ (cnj\ \varphi1\ \varphi2)\ txs = cnj\ (psubst\ \varphi1\ txs)\ (psubst\ \varphi2\ txs)$
proof−
define *us* **where** *us*: $us = getFrN\ (map\ snd\ txs)\ (map\ fst\ txs)\ [cnj\ \varphi1\ \varphi2]\ (length\ txs)$
have *us_facts*: $set\ us \subseteq var$
$set\ us \cap Fvars\ \varphi1 = \{\}$
$set\ us \cap Fvars\ \varphi2 = \{\}$
$set\ us \cap \bigcup\ (FvarsT\ `\ (fst\ `\ (set\ txs))) = \{\}$
$set\ us \cap snd\ `\ (set\ txs) = \{\}$
$length\ us = length\ txs$
$distinct\ us$
using *assms* **unfolding** *us*
using *getFrN_Fvars*[*of map snd txs map fst txs* [*cnj* $\varphi1\ \varphi2$] _ *length txs*]
getFrN_FvarsT[*of map snd txs map fst txs* [*cnj* $\varphi1\ \varphi2$] _ *length txs*]
getFrN_var[*of map snd txs map fst txs* [*cnj* $\varphi1\ \varphi2$] _ *length txs*]

```
    getFrN_length[of map snd txs map fst txs [cnj φ1 φ2] length txs]
  apply −
  subgoal by auto
  subgoal by fastforce
  subgoal by fastforce
  subgoal by fastforce
  subgoal by (fastforce simp: image_iff)
  subgoal by auto
  subgoal by auto .
define vs1 where vs1: vs1 = getFrN (map snd txs) (map fst txs) [φ1] (length txs)
have vs1_facts: set vs1 ⊆ var
  set vs1 ∩ Fvars φ1 = {}
  set vs1 ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
  set vs1 ∩ snd ‘ (set txs) = {}
  length vs1 = length txs
  distinct vs1
  using assms unfolding vs1
  using getFrN_Fvars[of map snd txs map fst txs [φ1] _ length txs]
    getFrN_FvarsT[of map snd txs map fst txs [φ1] _ length txs]
    getFrN_var[of map snd txs map fst txs [φ1] _ length txs]
    getFrN_length[of map snd txs map fst txs [φ1] length txs]
  apply −
  subgoal by auto
  subgoal by auto
  subgoal by fastforce
  subgoal by force
  subgoal by auto
  subgoal by auto .

define vs2 where vs2: vs2 = getFrN (map snd txs) (map fst txs) [φ2] (length txs)
have vs2_facts: set vs2 ⊆ var
  set vs2 ∩ Fvars φ2 = {}
  set vs2 ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
  set vs2 ∩ snd ‘ (set txs) = {}
  length vs2 = length txs
  distinct vs2
  using assms unfolding vs2
  using getFrN_Fvars[of map snd txs map fst txs [φ2] _ length txs]
    getFrN_FvarsT[of map snd txs map fst txs [φ2] _ length txs]
    getFrN_var[of map snd txs map fst txs [φ2] _ length txs]
    getFrN_length[of map snd txs map fst txs [φ2] length txs]
  apply −
  subgoal by auto
  subgoal by auto
  subgoal by fastforce
  subgoal by force
  subgoal by auto
  subgoal by auto .

let ?tus = zip (map fst txs) us
let ?uxs = zip (map Var us) (map snd txs)
let ?tvs1 = zip (map fst txs) vs1
let ?vxs1 = zip (map Var vs1) (map snd txs)
let ?tvs2 = zip (map fst txs) vs2
let ?vxs2 = zip (map Var vs2) (map snd txs)

let ?c = rawpsubst (cnj φ1 φ2) ?uxs
have c: ?c = cnj (rawpsubst φ1 ?uxs) (rawpsubst φ2 ?uxs)
```

```
    using assms us_facts
    by (intro rawpsubst_cnj) (auto intro!: rawpsubstT dest!: set_zip_D)
  have 0: rawpsubst ?c ?tus =
    cnj (rawpsubst (rawpsubst φ1 ?uxs) ?tus) (rawpsubst (rawpsubst φ2 ?uxs) ?tus)
    unfolding c using assms us_facts
    by (intro rawpsubst_cnj) (auto dest!: set_zip_D intro!: rawpsubst)

  have 1: rawpsubst (rawpsubst φ1 ?uxs) ?tus = rawpsubst (rawpsubst φ1 ?vxs1) ?tvs1
    using assms vs1_facts us_facts
    by (intro rawpsubst_compose_freshVar2) (auto intro!: rawpsubst)
  have 2: rawpsubst (rawpsubst φ2 ?uxs) ?tus = rawpsubst (rawpsubst φ2 ?vxs2) ?tvs2
    using assms vs2_facts us_facts
    by (intro rawpsubst_compose_freshVar2)(auto intro!: rawpsubst)
 show ?thesis unfolding psubst_def by (simp add: Let_def us[symmetric] vs1[symmetric] vs2[symmetric]
0 1 2)
qed

lemma rawpsubst_imp:
  assumes φ1 ∈ fmla φ2 ∈ fmla
    and snd ‘ (set txs) ⊆ var fst ‘ (set txs) ⊆ trm
  shows rawpsubst (imp φ1 φ2) txs = imp (rawpsubst φ1 txs) (rawpsubst φ2 txs)
  using assms apply (induct txs arbitrary: φ1 φ2)
  subgoal by auto
  subgoal for tx txs φ1 φ2 by (cases tx) auto .

lemma psubst_imp[simp]:
  assumes φ1 ∈ fmla φ2 ∈ fmla
    and snd ‘ (set txs) ⊆ var fst ‘ (set txs) ⊆ trm
    and distinct (map snd txs)
  shows psubst (imp φ1 φ2) txs = imp (psubst φ1 txs) (psubst φ2 txs)
proof−
  define us where us: us = getFrN (map snd txs) (map fst txs) [imp φ1 φ2] (length txs)
  have us_facts: set us ⊆ var
    set us ∩ Fvars φ1 = {}
    set us ∩ Fvars φ2 = {}
    set us ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
    set us ∩ snd ‘ (set txs) = {}
    length us = length txs
    distinct us
    using assms unfolding us
    using getFrN_Fvars[of map snd txs map fst txs [imp φ1 φ2] _ length txs]
      getFrN_FvarsT[of map snd txs map fst txs [imp φ1 φ2] _ length txs]
      getFrN_var[of map snd txs map fst txs [imp φ1 φ2] _ length txs]
      getFrN_length[of map snd txs map fst txs [imp φ1 φ2] length txs]
    apply −
    subgoal by auto
    subgoal by fastforce
    subgoal by fastforce
    subgoal by fastforce
    subgoal by (fastforce simp: image_iff)
    by auto

  define vs1 where vs1: vs1 = getFrN (map snd txs) (map fst txs) [φ1] (length txs)
  have vs1_facts: set vs1 ⊆ var
    set vs1 ∩ Fvars φ1 = {}
    set vs1 ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
    set vs1 ∩ snd ‘ (set txs) = {}
    length vs1 = length txs
```

```
    distinct vs1
    using assms unfolding vs1
    using getFrN_Fvars[of map snd txs map fst txs [φ1] _ length txs]
      getFrN_FvarsT[of map snd txs map fst txs [φ1] _ length txs]
      getFrN_var[of map snd txs map fst txs [φ1] _ length txs]
      getFrN_length[of map snd txs map fst txs [φ1] length txs]
    apply −
    subgoal by auto
    subgoal by auto
    subgoal by fastforce
    subgoal by force
    by auto

  define vs2 where vs2: vs2 = getFrN (map snd txs) (map fst txs) [φ2] (length txs)
  have vs2_facts: set vs2 ⊆ var
    set vs2 ∩ Fvars φ2 = {}
    set vs2 ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
    set vs2 ∩ snd ‘ (set txs) = {}
    length vs2 = length txs
    distinct vs2
    using assms unfolding vs2
    using getFrN_Fvars[of map snd txs map fst txs [φ2] _ length txs]
      getFrN_FvarsT[of map snd txs map fst txs [φ2] _ length txs]
      getFrN_var[of map snd txs map fst txs [φ2] _ length txs]
      getFrN_length[of map snd txs map fst txs [φ2] length txs]
    apply −
    subgoal by auto
    subgoal by auto
    subgoal by fastforce
    subgoal by force
    by auto

  let ?tus = zip (map fst txs) us
  let ?uxs = zip (map Var us) (map snd txs)
  let ?tvs1 = zip (map fst txs) vs1
  let ?vxs1 = zip (map Var vs1) (map snd txs)
  let ?tvs2 = zip (map fst txs) vs2
  let ?vxs2 = zip (map Var vs2) (map snd txs)

  let ?c = rawpsubst (imp φ1 φ2) ?uxs
  have c: ?c = imp (rawpsubst φ1 ?uxs) (rawpsubst φ2 ?uxs)
    apply(rule rawpsubst_imp) using assms us_facts apply (auto intro!: rawpsubstT)
    apply(drule set_zip_rightD) apply simp apply blast
    apply(drule set_zip_leftD) apply simp apply blast .
  have 0: rawpsubst ?c ?tus =
    imp (rawpsubst (rawpsubst φ1 ?uxs) ?tus) (rawpsubst (rawpsubst φ2 ?uxs) ?tus)
    unfolding c
    using assms us_facts
    by (intro rawpsubst_imp) (auto intro!: rawpsubst dest!: set_zip_D)
  have 1: rawpsubst (rawpsubst φ1 ?uxs) ?tus = rawpsubst (rawpsubst φ1 ?vxs1) ?tvs1
    using assms vs1_facts us_facts
    by (intro rawpsubst_compose_freshVar2) (auto intro!: rawpsubst)
  have 2: rawpsubst (rawpsubst φ2 ?uxs) ?tus = rawpsubst (rawpsubst φ2 ?vxs2) ?tvs2
    using assms vs2_facts us_facts
    by (intro rawpsubst_compose_freshVar2) (auto intro!: rawpsubst)
 show ?thesis unfolding psubst_def by (simp add: Let_def us[symmetric] vs1[symmetric] vs2[symmetric]
0 1 2)
qed
```

lemma *rawpsubst_eqv*:
assumes $\varphi 1 \in fmla$ $\varphi 2 \in fmla$
and $snd\ `\ (set\ txs) \subseteq var$ $fst\ `\ (set\ txs) \subseteq trm$
shows $rawpsubst\ (eqv\ \varphi 1\ \varphi 2)\ txs = eqv\ (rawpsubst\ \varphi 1\ txs)\ (rawpsubst\ \varphi 2\ txs)$
using *assms* **apply** (*induct txs arbitrary*: $\varphi 1\ \varphi 2$)
subgoal by *auto*
subgoal for *tx txs* $\varphi 1\ \varphi 2$ **by** (*cases tx*) *auto* **.**

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−
define *us* **where** *us*: $us = getFrN\ (map\ snd\ txs)\ (map\ fst\ txs)\ [eqv\ \varphi 1\ \varphi 2]\ (length\ txs)$
have *us_facts*: $set\ us \subseteq var$
$set\ us \cap Fvars\ \varphi 1 = \{\}$
$set\ us \cap Fvars\ \varphi 2 = \{\}$
$set\ us \cap \bigcup\ (FvarsT\ `\ (fst\ `\ (set\ txs))) = \{\}$
$set\ us \cap snd\ `\ (set\ txs) = \{\}$
$length\ us = length\ txs$
distinct us
using *assms* **unfolding** *us*
using *getFrN_Fvars*[*of map snd txs map fst txs* [*eqv* $\varphi 1\ \varphi 2$] _ *length txs*]
getFrN_FvarsT[*of map snd txs map fst txs* [*eqv* $\varphi 1\ \varphi 2$] _ *length txs*]
getFrN_var[*of map snd txs map fst txs* [*eqv* $\varphi 1\ \varphi 2$] _ *length txs*]
getFrN_length[*of map snd txs map fst txs* [*eqv* $\varphi 1\ \varphi 2$] *length txs*]
apply −
subgoal by *auto*
subgoal by *fastforce*
subgoal by *fastforce*
subgoal by *fastforce*
subgoal by (*fastforce simp*: *image_iff*)
by *auto*

define *vs1* **where** *vs1*: $vs1 = getFrN\ (map\ snd\ txs)\ (map\ fst\ txs)\ [\varphi 1]\ (length\ txs)$
have *vs1_facts*: $set\ vs1 \subseteq var$
$set\ vs1 \cap Fvars\ \varphi 1 = \{\}$
$set\ vs1 \cap \bigcup\ (FvarsT\ `\ (fst\ `\ (set\ txs))) = \{\}$
$set\ vs1 \cap snd\ `\ (set\ txs) = \{\}$
$length\ vs1 = length\ txs$
distinct vs1
using *assms* **unfolding** *vs1*
using *getFrN_Fvars*[*of map snd txs map fst txs* [$\varphi 1$] _ *length txs*]
getFrN_FvarsT[*of map snd txs map fst txs* [$\varphi 1$] _ *length txs*]
getFrN_var[*of map snd txs map fst txs* [$\varphi 1$] _ *length txs*]
getFrN_length[*of map snd txs map fst txs* [$\varphi 1$] *length txs*]
apply −
subgoal by *auto*
subgoal by *auto*
subgoal by *fastforce*
subgoal by *force*
by *auto*

define *vs2* **where** *vs2*: $vs2 = getFrN\ (map\ snd\ txs)\ (map\ fst\ txs)\ [\varphi 2]\ (length\ txs)$
have *vs2_facts*: $set\ vs2 \subseteq var$
$set\ vs2 \cap Fvars\ \varphi 2 = \{\}$

```
    set vs2 ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
    set vs2 ∩ snd ‘ (set txs) = {}
    length vs2 = length txs
    distinct vs2
    using assms unfolding vs2
    using getFrN_Fvars[of map snd txs map fst txs [φ2] _ length txs]
      getFrN_FvarsT[of map snd txs map fst txs [φ2] _ length txs]
      getFrN_var[of map snd txs map fst txs [φ2] _ length txs]
      getFrN_length[of map snd txs map fst txs [φ2] length txs]
    apply −
    subgoal by auto
    subgoal by auto
    subgoal by fastforce
    subgoal by force
    by auto

  let ?tus = zip (map fst txs) us
  let ?uxs = zip (map Var us) (map snd txs)
  let ?tvs1 = zip (map fst txs) vs1
  let ?vxs1 = zip (map Var vs1) (map snd txs)
  let ?tvs2 = zip (map fst txs) vs2
  let ?vxs2 = zip (map Var vs2) (map snd txs)

  let ?c = rawpsubst (eqv φ1 φ2) ?uxs
  have c: ?c = eqv (rawpsubst φ1 ?uxs) (rawpsubst φ2 ?uxs)
    using assms us_facts
    by (intro rawpsubst_eqv) (auto intro!: rawpsubstT dest!: set_zip_D)
  have 0: rawpsubst ?c ?tus =
    eqv (rawpsubst (rawpsubst φ1 ?uxs) ?tus) (rawpsubst (rawpsubst φ2 ?uxs) ?tus)
    unfolding c using assms us_facts
    by (intro rawpsubst_eqv) (auto intro!: rawpsubst dest!: set_zip_D)
  have 1: rawpsubst (rawpsubst φ1 ?uxs) ?tus = rawpsubst (rawpsubst φ1 ?vxs1) ?tvs1
    using assms vs1_facts us_facts
    by (intro rawpsubst_compose_freshVar2) (auto intro!: rawpsubst)
  have 2: rawpsubst (rawpsubst φ2 ?uxs) ?tus = rawpsubst (rawpsubst φ2 ?vxs2) ?tvs2
    using assms vs2_facts us_facts
    by (intro rawpsubst_compose_freshVar2) (auto intro!: rawpsubst)
 show ?thesis unfolding psubst_def by (simp add: Let_def us[symmetric] vs1[symmetric] vs2[symmetric]
0 1 2)
qed

lemma rawpsubst_all:
  assumes φ ∈ fmla y ∈ var
    and snd ‘ (set txs) ⊆ var fst ‘ (set txs) ⊆ trm
    and y ∉ snd ‘ (set txs) y ∉ ⋃ (FvarsT ‘ fst ‘ (set txs))
  shows rawpsubst (all y φ) txs = all y (rawpsubst φ txs)
  using assms apply (induct txs arbitrary: φ)
  subgoal by auto
  subgoal for tx txs φ by (cases tx) auto .

lemma psubst_all[simp]:
  assumes φ ∈ fmla y ∈ var
    and snd ‘ (set txs) ⊆ var fst ‘ (set txs) ⊆ trm
    and y ∉ snd ‘ (set txs) y ∉ ⋃ (FvarsT ‘ fst ‘ (set txs))
    and distinct (map snd txs)
  shows psubst (all y φ) txs = all y (psubst φ txs)
proof−
  define us where us: us = getFrN (map snd txs) (map fst txs) [all y φ] (length txs)
```

have *us_facts*: *set us* ⊆ *var*
set us ∩ (*Fvars* φ − {*y*}) = {}
set us ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set us ∩ *snd* ‘ (*set txs*) = {}
length us = *length txs*
distinct us
using *assms* **unfolding** *us*
using *getFrN_Fvars*[*of map snd txs map fst txs* [*all y* φ] _ *length txs*]
getFrN_FvarsT[*of map snd txs map fst txs* [*all y* φ] _ *length txs*]
getFrN_var[*of map snd txs map fst txs* [*all y* φ] _ *length txs*]
getFrN_length[*of map snd txs map fst txs* [*all y* φ] *length txs*]
apply −
subgoal by *auto*
subgoal by *fastforce*
subgoal by *fastforce*
subgoal by (*fastforce simp*: *image_iff*)
by *auto*

define *vs* **where** *vs*: *vs* = *getFrN* (*map snd txs*) (*map fst txs*) [φ] (*length txs*)
have *vs_facts*: *set vs* ⊆ *var*
set vs ∩ *Fvars* φ = {}
set vs ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set vs ∩ *snd* ‘ (*set txs*) = {}
length vs = *length txs*
distinct vs
using *assms* **unfolding** *vs*
using *getFrN_Fvars*[*of map snd txs map fst txs* [φ] _ *length txs*]
getFrN_FvarsT[*of map snd txs map fst txs* [φ] _ *length txs*]
getFrN_var[*of map snd txs map fst txs* [φ] _ *length txs*]
getFrN_length[*of map snd txs map fst txs* [φ] *length txs*]
apply −
subgoal by *auto*
subgoal by *fastforce*
subgoal by *fastforce*
subgoal by (*fastforce simp*: *image_iff*)
by *auto*

define *ws* **where** *ws*: *ws* = *getFrN* (*y* # *map snd txs*) (*map fst txs*) [φ] (*length txs*)
have *ws_facts*: *set ws* ⊆ *var*
set ws ∩ *Fvars* φ = {} *y* ∉ *set ws*
set ws ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set ws ∩ *snd* ‘ (*set txs*) = {}
length ws = *length txs*
distinct ws
using *assms* **unfolding** *ws*
using *getFrN_Fvars*[*of y* # *map snd txs map fst txs* [φ] _ *length txs*]
getFrN_FvarsT[*of y* # *map snd txs map fst txs* [φ] _ *length txs*]
getFrN_var[*of y* # *map snd txs map fst txs* [φ] _ *length txs*]
getFrN_length[*of y* # *map snd txs map fst txs* [φ] *length txs*]
apply −
subgoal by *auto*
subgoal by *fastforce*
subgoal by *fastforce*
subgoal by (*fastforce simp*: *image_iff*)
subgoal by (*fastforce simp*: *image_iff*)
by *auto*

have *0*: *rawpsubst* (*all y* φ) (*zip* (*map Var ws*) (*map snd txs*)) =

```
        all y (rawpsubst φ (zip (map Var ws) (map snd txs)))
    using assms ws_facts  apply(intro rawpsubst_all)
    subgoal by auto
    subgoal by auto
    subgoal by (auto dest!: set_zip_D)
    subgoal by (auto dest!: set_zip_D)
    subgoal by (auto dest!: set_zip_D)
    subgoal by (fastforce dest: set_zip_D) .

  have 1: rawpsubst ((rawpsubst φ (zip (map Var ws) (map snd txs)))) (zip (map fst txs) ws) =
        rawpsubst ((rawpsubst φ (zip (map Var vs) (map snd txs)))) (zip (map fst txs) vs)
    apply(rule rawpsubst_compose_freshVar2)
    using assms ws_facts vs_facts by (auto intro!: rawpsubst)
  have rawpsubst (rawpsubst (all y φ) (zip (map Var us) (map snd txs))) (zip (map fst txs) us) =
      rawpsubst (rawpsubst (all y φ) (zip (map Var ws) (map snd txs))) (zip (map fst txs) ws)
    using assms ws_facts us_facts
    by (intro rawpsubst_compose_freshVar2) (auto intro!: rawpsubst)
  also have
    ... = all y (rawpsubst ((rawpsubst φ (zip (map Var ws) (map snd txs)))) (zip (map fst txs) ws))
    unfolding 0 using assms ws_facts
    by (intro rawpsubst_all) (auto dest!: set_zip_D intro!: rawpsubst)
  also have
    ... = all y (rawpsubst (rawpsubst φ (zip (map Var vs) (map snd txs))) (zip (map fst txs) vs))
    unfolding 1 ..
  finally show ?thesis unfolding psubst_def by (simp add: Let_def us[symmetric] vs[symmetric])
qed

lemma rawpsubst_exi:
  assumes φ ∈ fmla y ∈ var
    and snd ` (set txs) ⊆ var fst ` (set txs) ⊆ trm
    and y ∉ snd ` (set txs) y ∉ ⋃ (FvarsT ` fst ` (set txs))
  shows rawpsubst (exi y φ) txs = exi y (rawpsubst φ txs)
  using assms apply (induct txs arbitrary: φ)
  subgoal by auto
  subgoal for tx txs φ by (cases tx) auto .

lemma psubst_exi[simp]:
  assumes φ ∈ fmla y ∈ var
    and snd ` (set txs) ⊆ var fst ` (set txs) ⊆ trm
    and y ∉ snd ` (set txs) y ∉ ⋃ (FvarsT ` fst ` (set txs))
    and distinct (map snd txs)
  shows psubst (exi y φ) txs = exi y (psubst φ txs)
proof−
  define us where us: us = getFrN (map snd txs) (map fst txs) [exi y φ] (length txs)
  have us_facts: set us ⊆ var
    set us ∩ (Fvars φ − {y}) = {}
    set us ∩ ⋃ (FvarsT ` (fst ` (set txs))) = {}
    set us ∩ snd ` (set txs) = {}
    length us = length txs
    distinct us
    using assms unfolding us
    using getFrN_Fvars[of map snd txs map fst txs [exi y φ] _ length txs]
      getFrN_FvarsT[of map snd txs map fst txs [exi y φ] _ length txs]
      getFrN_var[of map snd txs map fst txs [exi y φ] _ length txs]
      getFrN_length[of map snd txs map fst txs [exi y φ] length txs]
    apply −
    subgoal by auto
    subgoal by fastforce
```

subgoal by *fastforce*
subgoal by (*fastforce simp*: *image_iff*)
subgoal by (*fastforce simp*: *image_iff*)
by *auto*

define *vs* **where** *vs*: *vs* = *getFrN* (*map snd txs*) (*map fst txs*) [φ] (*length txs*)
have *vs_facts*: *set vs* $\subseteq$ *var*
set vs $\cap$ *Fvars* φ = {}
set vs $\cap$ $\bigcup$ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set vs $\cap$ *snd* ‘ (*set txs*) = {}
length vs = *length txs*
distinct vs
using *assms* **unfolding** *vs*
using *getFrN_Fvars*[*of map snd txs map fst txs* [φ] _ *length txs*]
getFrN_FvarsT[*of map snd txs map fst txs* [φ] _ *length txs*]
getFrN_var[*of map snd txs map fst txs* [φ] _ *length txs*]
getFrN_length[*of map snd txs map fst txs* [φ] *length txs*]
apply −
subgoal by *auto*
subgoal by *fastforce*
subgoal by *fastforce*
subgoal by (*fastforce simp*: *image_iff*)
subgoal by (*fastforce simp*: *image_iff*)
by *auto*

define *ws* **where** *ws*: *ws* = *getFrN* (*y # map snd txs*) (*map fst txs*) [φ] (*length txs*)
have *ws_facts*: *set ws* $\subseteq$ *var*
set ws $\cap$ *Fvars* φ = {} *y* $\notin$ *set ws*
set ws $\cap$ $\bigcup$ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set ws $\cap$ *snd* ‘ (*set txs*) = {}
length ws = *length txs*
distinct ws
using *assms* **unfolding** *ws*
using *getFrN_Fvars*[*of y # map snd txs map fst txs* [φ] _ *length txs*]
getFrN_FvarsT[*of y # map snd txs map fst txs* [φ] _ *length txs*]
getFrN_var[*of y # map snd txs map fst txs* [φ] _ *length txs*]
getFrN_length[*of y # map snd txs map fst txs* [φ] *length txs*]
apply −
subgoal by *auto*
subgoal by *fastforce*
subgoal by *fastforce*
subgoal by (*fastforce simp*: *image_iff*)
subgoal by (*fastforce simp*: *image_iff*)
by *auto*

have *0*: *rawpsubst* (*exi y* φ) (*zip* (*map Var ws*) (*map snd txs*)) =
exi y (*rawpsubst* φ (*zip* (*map Var ws*) (*map snd txs*)))
using *assms ws_facts* **apply**(*intro rawpsubst_exi*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto dest*!: *set_zip_D*)
subgoal by (*auto dest*!: *set_zip_D*)
subgoal by (*auto dest*!: *set_zip_D*)
subgoal by (*fastforce dest*: *set_zip_D*) **.**

have *1*: *rawpsubst* ((*rawpsubst* φ (*zip* (*map Var ws*) (*map snd txs*)))) (*zip* (*map fst txs*) *ws*) =
rawpsubst ((*rawpsubst* φ (*zip* (*map Var vs*) (*map snd txs*)))) (*zip* (*map fst txs*) *vs*)
using *assms ws_facts vs_facts*

by (*intro rawpsubst_compose_freshVar2*) (*auto intro*!: *rawpsubst*)
have *rawpsubst* (*rawpsubst* (*exi y* φ) (*zip* (*map Var us*) (*map snd txs*))) (*zip* (*map fst txs*) *us*) =
rawpsubst (*rawpsubst* (*exi y* φ) (*zip* (*map Var ws*) (*map snd txs*))) (*zip* (*map fst txs*) *ws*)
using *assms ws_facts us_facts*
by (*intro rawpsubst_compose_freshVar2*) (*auto intro*!: *rawpsubst*)
also have
. . . = *exi y* (*rawpsubst* ((*rawpsubst* φ (*zip* (*map Var ws*) (*map snd txs*)))) (*zip* (*map fst txs*) *ws*))
using *assms ws_facts* **unfolding** *0*
by (*intro rawpsubst_exi*) (*auto dest*!: *set_zip_D intro*!: *rawpsubst*)
also have
. . . = *exi y* (*rawpsubst* (*rawpsubst* φ (*zip* (*map Var vs*) (*map snd txs*))) (*zip* (*map fst txs*) *vs*))
unfolding *1* **..**
finally show *?thesis* **unfolding** *psubst_def* **by** (*simp add*: *Let_def us*[*symmetric*] *vs*[*symmetric*])
qed

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 $x \neq y \Longrightarrow subst\ (all\ x\ \varphi)\ n\ y = all\ x\ (subst\ \varphi\ n\ y)$
using *assms* **by** *simp*

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

The "soft substitution" function:

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

lemma *softSubst*[*simp*,*intro*]: $\varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow softSubst\ \varphi\ t\ x \in fmla$
unfolding *softSubst_def* **by** *auto*

lemma *Fvars_softSubst*[*simp*]:
$\varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow$
$Fvars\ (softSubst\ \varphi\ t\ x) = (Fvars\ \varphi \cup FvarsT\ t - \{x\})$
unfolding *softSubst_def* **by** *auto*

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

by *auto*

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

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* $\Rightarrow$ *'fmla* **where**
neg φ = *imp* φ *fls*

lemma
neg[*simp*]: $\bigwedge\varphi.\ \varphi \in fmla \Longrightarrow neg\ \varphi \in fmla$
and
Fvars_neg[*simp*]: $\bigwedge\varphi.\ \varphi \in fmla \Longrightarrow Fvars\ (neg\ \varphi) = Fvars\ \varphi$
and
subst_neg[*simp*]:
$\bigwedge\varphi\ t\ x.\ \varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow$
$subst\ (neg\ \varphi)\ t\ x = neg\ (subst\ \varphi\ t\ x)$
unfolding *neg_def* **by** *auto*

True as a derived connective:

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

lemma
tru[*simp*,*intro*!]: *tru* $\in$ *fmla*
and
Fvars_tru[*simp*]: *Fvars tru* = {}
and
subst_tru[*simp*]: $\bigwedge\ t\ x.\ t \in trm \Longrightarrow x \in var \Longrightarrow subst\ tru\ t\ x = tru$
unfolding *tru_def* **by** *auto*

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*
by (*induct φs*) *auto*

lemma *Fvars_lcnj*[*simp*]:
set φs ⊆ *fmla* ⟹ *finite F* ⟹ *Fvars* (*lcnj φs*) = ⋃ (*set* (*map Fvars φs*))
by(*induct φs*) *auto*

lemma *subst_lcnj*[*simp*]:
set φs ⊆ *fmla* ⟹ *t* ∈ *trm* ⟹ *x* ∈ *var* ⟹
subst (*lcnj φs*) *t x* = *lcnj* (*map* (λ*φ*. *subst φ t x*) *φs*)
by(*induct φs*) *auto*

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*
unfolding *scnj_def* **by** *auto*

lemma *Fvars_scnj*[*simp*]:
F ⊆ *fmla* ⟹ *finite F* ⟹*Fvars* (*scnj F*) = ⋃ (*Fvars* ‘ *F*)
unfolding *scnj_def* **by** *auto*

2.3.2 Parallel substitution versus the new connectives

lemma *rawpsubst_fls*:
snd ‘ (*set txs*) ⊆ *var* ⟹ *fst* ‘ (*set txs*) ⊆ *trm* ⟹ *rawpsubst fls txs* = *fls*
by (*induct txs*) *auto*

lemma *psubst_fls*[*simp*]:
assumes *snd* ‘ (*set txs*) ⊆ *var* **and** *fst* ‘ (*set txs*) ⊆ *trm*
shows *psubst fls txs* = *fls*
proof−
define *us* **where** *us*: *us* = *getFrN* (*map snd txs*) (*map fst txs*) [*fls*] (*length txs*)
have *us_facts*: *set us* ⊆ *var*
set us ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set us ∩ *snd* ‘ (*set txs*) = {}
length us = *length txs*
distinct us
using *assms* **unfolding** *us*
using *getFrN_Fvars*[*of map snd txs map fst txs* [*fls*] _ *length txs*]
getFrN_FvarsT[*of map snd txs map fst txs* [*fls*] _ *length txs*]
getFrN_var[*of map snd txs map fst txs* [*fls*] _ *length txs*]
getFrN_length[*of map snd txs map fst txs* [*fls*] *length txs*]
apply −
subgoal by *auto*
subgoal by *fastforce*
subgoal by (*fastforce simp*: *image_iff*)
subgoal by (*fastforce simp*: *image_iff*)
by *auto*
have [*simp*]: *rawpsubst fls* (*zip* (*map Var us*) (*map snd txs*)) = *fls*
using *us_facts assms* **by** (*intro rawpsubst_fls*) (*auto dest*!: *set_zip_D*)
show *?thesis* **using** *assms us_facts*
unfolding *psubst_def* **by** (*auto simp add*: *Let_def us*[*symmetric*] *intro*!: *rawpsubst_fls dest*!: *set_zip_D*)

qed

lemma *psubst_neg*[*simp*]:
assumes $\varphi \in fmla$
and $snd\ `\ (set\ txs) \subseteq var\ fst\ `\ (set\ txs) \subseteq trm$
and $distinct\ (map\ snd\ txs)$
shows $psubst\ (neg\ \varphi)\ txs = neg\ (psubst\ \varphi\ txs)$
unfolding *neg_def* **using** *assms psubst_imp psubst_fls* **by** *auto*

lemma *psubst_tru*[*simp*]:
assumes $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq trm$
and $distinct\ (map\ snd\ txs)$
shows $psubst\ tru\ txs = tru$
unfolding *tru_def* **using** *assms psubst_neg*[*of fls txs*] *psubst_fls* **by** *auto*

lemma *psubst_lcnj*[*simp*]:
$set\ \varphi s \subseteq fmla \Longrightarrow snd\ `\ (set\ txs) \subseteq var \Longrightarrow fst\ `\ (set\ txs) \subseteq trm \Longrightarrow$
$distinct\ (map\ snd\ txs) \Longrightarrow$
$psubst\ (lcnj\ \varphi s)\ txs = lcnj\ (map\ (\lambda\varphi.\ psubst\ \varphi\ txs)\ \varphi s)$
by $(induct\ \varphi s)\ auto$

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* :: $'fmla \Rightarrow 'fmla \Rightarrow 'fmla$
assumes
dsj[*simp*]: $\bigwedge\varphi\ \chi.\ \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow dsj\ \varphi\ \chi \in fmla$
and
Fvars_dsj[*simp*]: $\bigwedge\varphi\ \chi.\ \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow$
$Fvars\ (dsj\ \varphi\ \chi) = Fvars\ \varphi \cup Fvars\ \chi$
and
subst_dsj[*simp*]:
$\bigwedge\ x\ \varphi\ \chi\ t.\ \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow$
$subst\ (dsj\ \varphi\ \chi)\ t\ x = dsj\ (subst\ \varphi\ t\ x)\ (subst\ \chi\ t\ x)$
begin

2.4.1 Iterated disjunction

First we define list-based disjunction:

fun *ldsj* :: $'fmla\ list \Rightarrow 'fmla$ **where**
$ldsj\ [] = fls$
| $ldsj\ (\varphi\ \#\ \varphi s) = dsj\ \varphi\ (ldsj\ \varphi s)$

lemma *ldsj*[*simp*,*intro*!]: *set* $\varphi s \subseteq$ *fmla* $\Longrightarrow$ *ldsj* $\varphi s \in$ *fmla*
by (*induct* φs) *auto*

lemma *Fvars_ldsj*[*simp*]:
set $\varphi s \subseteq$ *fmla* $\Longrightarrow$ *Fvars* (*ldsj* φs) $= \bigcup$ (*set* (*map Fvars* φs))
by(*induct* φs) *auto*

lemma *subst_ldsj*[*simp*]:
set $\varphi s \subseteq$ *fmla* $\Longrightarrow$ $t \in$ *trm* $\Longrightarrow$ $x \in$ *var* $\Longrightarrow$
subst (*ldsj* φs) t x = *ldsj* (*map* ($\lambda\varphi$. *subst* φ t x) φs)
by(*induct* φs) *auto*

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

definition *sdsj* :: *'fmla set* $\Rightarrow$ *'fmla* **where**
sdsj F = *ldsj* (*asList* F)

lemma *sdsj*[*simp*,*intro*!]: $F \subseteq$ *fmla* $\Longrightarrow$ *finite* $F \Longrightarrow$ *sdsj* $F \in$ *fmla*
unfolding *sdsj_def* **by** *auto*

lemma *Fvars_sdsj*[*simp*]:
$F \subseteq$ *fmla* $\Longrightarrow$ *finite* $F \Longrightarrow$ *Fvars* (*sdsj* F) $= \bigcup$ (*Fvars* ‘ F)
unfolding *sdsj_def* **by** *auto*

2.4.2 Parallel substitution versus the new connectives

lemma *rawpsubst_dsj*:
assumes $\varphi 1 \in$ *fmla* $\varphi 2 \in$ *fmla*
and *snd* ‘ (*set txs*) $\subseteq$ *var* *fst* ‘ (*set txs*) $\subseteq$ *trm*
shows *rawpsubst* (*dsj* $\varphi 1$ $\varphi 2$) *txs* = *dsj* (*rawpsubst* $\varphi 1$ *txs*) (*rawpsubst* $\varphi 2$ *txs*)
using *assms* **apply** (*induct txs arbitrary*: $\varphi 1$ $\varphi 2$)
subgoal by *auto*
subgoal for *tx txs* $\varphi 1$ $\varphi 2$ **apply** (*cases tx*) **by** *auto* **.**

lemma *psubst_dsj*[*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* (*dsj* $\varphi 1$ $\varphi 2$) *txs* = *dsj* (*psubst* $\varphi 1$ *txs*) (*psubst* $\varphi 2$ *txs*)
proof−
define *us* **where** *us*: *us* = *getFrN* (*map snd txs*) (*map fst txs*) [*dsj* $\varphi 1$ $\varphi 2$] (*length txs*)
have *us_facts*: *set us* $\subseteq$ *var*
set us $\cap$ *Fvars* $\varphi 1$ = {}
set us $\cap$ *Fvars* $\varphi 2$ = {}
set us $\cap \bigcup$ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set us $\cap$ *snd* ‘ (*set txs*) = {}
length us = *length txs*
distinct us
using *assms* **unfolding** *us*
using *getFrN_Fvars*[*of map snd txs map fst txs* [*dsj* $\varphi 1$ $\varphi 2$] _ *length txs*]
getFrN_FvarsT[*of map snd txs map fst txs* [*dsj* $\varphi 1$ $\varphi 2$] _ *length txs*]
getFrN_var[*of map snd txs map fst txs* [*dsj* $\varphi 1$ $\varphi 2$] _ *length txs*]
getFrN_length[*of map snd txs map fst txs* [*dsj* $\varphi 1$ $\varphi 2$] *length txs*]
apply −
subgoal by *auto*
subgoal by *fastforce*
subgoal by (*fastforce simp*: *image_iff*)
subgoal by (*fastforce simp*: *image_iff*)
subgoal by (*fastforce simp*: *image_iff*)

by *auto*

define *vs1* **where** *vs1*: *vs1* = *getFrN* (*map snd txs*) (*map fst txs*) [*φ1*] (*length txs*)
have *vs1_facts*: *set vs1* ⊆ *var*
set vs1 ∩ *Fvars φ1* = {}
set vs1 ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set vs1 ∩ *snd* ‘ (*set txs*) = {}
length vs1 = *length txs*
distinct vs1
using *assms* **unfolding** *vs1*
using *getFrN_Fvars*[*of map snd txs map fst txs* [*φ1*] _ *length txs*]
getFrN_FvarsT[*of map snd txs map fst txs* [*φ1*] _ *length txs*]
getFrN_var[*of map snd txs map fst txs* [*φ1*] _ *length txs*]
getFrN_length[*of map snd txs map fst txs* [*φ1*] *length txs*]
apply −
subgoal by *auto*
subgoal by *fastforce*
subgoal by *fastforce*
subgoal by (*fastforce simp*: *image_iff*)
by *auto*

define *vs2* **where** *vs2*: *vs2* = *getFrN* (*map snd txs*) (*map fst txs*) [*φ2*] (*length txs*)
have *vs2_facts*: *set vs2* ⊆ *var*
set vs2 ∩ *Fvars φ2* = {}
set vs2 ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set vs2 ∩ *snd* ‘ (*set txs*) = {}
length vs2 = *length txs*
distinct vs2
using *assms* **unfolding** *vs2*
using *getFrN_Fvars*[*of map snd txs map fst txs* [*φ2*] _ *length txs*]
getFrN_FvarsT[*of map snd txs map fst txs* [*φ2*] _ *length txs*]
getFrN_var[*of map snd txs map fst txs* [*φ2*] _ *length txs*]
getFrN_length[*of map snd txs map fst txs* [*φ2*] *length txs*]
apply −
apply −
subgoal by *auto*
subgoal by *fastforce*
subgoal by *fastforce*
subgoal by (*fastforce simp*: *image_iff*)
by *auto*

let *?tus* = *zip* (*map fst txs*) *us*
let *?uxs* = *zip* (*map Var us*) (*map snd txs*)
let *?tvs1* = *zip* (*map fst txs*) *vs1*
let *?vxs1* = *zip* (*map Var vs1*) (*map snd txs*)
let *?tvs2* = *zip* (*map fst txs*) *vs2*
let *?vxs2* = *zip* (*map Var vs2*) (*map snd txs*)

let *?c* = *rawpsubst* (*dsj φ1 φ2*) *?uxs*
have *c*: *?c* = *dsj* (*rawpsubst φ1 ?uxs*) (*rawpsubst φ2 ?uxs*)
apply(*rule rawpsubst_dsj*) **using** *assms us_facts* **apply** (*auto intro*!: *rawpsubstT*)
apply(*drule set_zip_rightD*) **apply** *simp* **apply** *blast*
apply(*drule set_zip_leftD*) **apply** *simp* **apply** *blast* **.**
have *0*: *rawpsubst ?c ?tus* =
dsj (*rawpsubst* (*rawpsubst φ1 ?uxs*) *?tus*) (*rawpsubst* (*rawpsubst φ2 ?uxs*) *?tus*)
unfolding *c* **using** *assms us_facts*
by (*intro rawpsubst_dsj*) (*auto intro*!: *rawpsubst dest*!: *set_zip_D*)
have *1*: *rawpsubst* (*rawpsubst φ1 ?uxs*) *?tus* = *rawpsubst* (*rawpsubst φ1 ?vxs1*) *?tvs1*

using *assms vs1_facts us_facts*
by (*intro rawpsubst_compose_freshVar2*) (*auto intro*!: *rawpsubst*)
have *2*: *rawpsubst* (*rawpsubst* $\varphi 2$ *?uxs*) *?tus* = *rawpsubst* (*rawpsubst* $\varphi 2$ *?vxs2*) *?tvs2*
using *assms vs2_facts us_facts*
by (*intro rawpsubst_compose_freshVar2*) (*auto intro*!: *rawpsubst*)
show *?thesis* **unfolding** *psubst_def* **by** (*simp add*: *Let_def us*[*symmetric*] *vs1*[*symmetric*] *vs2*[*symmetric*] *0 1 2*)
qed

lemma *psubst_ldsj*[*simp*]:
set $\varphi s \subseteq$ *fmla* $\Longrightarrow$ *snd* ' (*set txs*) $\subseteq$ *var* $\Longrightarrow$ *fst* ' (*set txs*) $\subseteq$ *trm* $\Longrightarrow$
distinct (*map snd txs*) $\Longrightarrow$
psubst (*ldsj* φs) *txs* = *ldsj* (*map* ($\lambda\varphi$. *psubst* φ *txs*) φs)
by (*induct* φs) *auto*

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

lemma *LLq*[*simp*,*intro*]:
assumes *t1* ∈ *trm t2* ∈ *trm*
shows *LLq t1 t2* ∈ *fmla*
using *assms* **unfolding** *LLq_def* **by** *auto*

lemma *LLq2*[*simp*,*intro*!]:
n ∈ *num* ⟹ *LLq n* (*Var yy*′) ∈ *fmla*
by *auto*

lemma *Fvars_LLq*[*simp*]: *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹ *yy* ∉ *FvarsT t1* ⟹
Fvars (*LLq t1 t2*) = *FvarsT t1* ∪ *FvarsT t2*
by (*auto simp add*: *LLq_def2 subst2_fresh_switch*)

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*
by (*auto simp*: *LLq_def2 subst2_fresh_switch*)

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*:
$\varphi \in fmla \Longrightarrow x \in var \Longrightarrow y \in var \Longrightarrow (y = x \vee y \notin Fvars\ \varphi) \Longrightarrow$
$all\ x\ \varphi = all\ y\ (subst\ \varphi\ (Var\ y)\ x)$
using *all_rename* **by** (*cases* $y = x$) (*auto simp del*: *Fvars_subst*)

lemma *exi_rename2*:
$\varphi \in fmla \Longrightarrow x \in var \Longrightarrow y \in var \Longrightarrow (y = x \vee y \notin Fvars\ \varphi) \Longrightarrow$
$exi\ x\ \varphi = exi\ y\ (subst\ \varphi\ (Var\ y)\ x)$
using *exi_rename* **by** (*cases* $y = x$) (*auto simp del*: *Fvars_subst*)

2.7 The Exists-Unique Quantifier

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

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

lemma *exu*[*simp*,*intro*]:
$x \in var \Longrightarrow \varphi \in fmla \Longrightarrow exu\ x\ \varphi \in fmla$
unfolding *exu_def* **by** (*simp add*: *Let_def*)

lemma *Fvars_exu*[*simp*]:
$x \in var \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ (exu\ x\ \varphi) = Fvars\ \varphi - \{x\}$
unfolding *exu_def* **by** (*auto simp*: *Let_def getFr_Fvars*)

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−
have [*simp*]: $x \neq y$ **using** *assms* **by** *blast*
define z **where** z: $z \equiv getFr\ [x]\ []\ [\varphi]$
have *z_facts*[*simp*]: $z \in var\ z \neq x\ x \neq z\ z \notin Fvars\ \varphi$
unfolding z **using** *getFr_FvarsT_Fvars*[*of* $[x]$ $[]$ $[\varphi]$] **by** *auto*
define u **where** u: $u \equiv getFr\ [x,y,z]\ []\ [\varphi]$
have *u_facts*[*simp*]: $u \in var\ u \neq x\ u \neq z\ y \neq u\ u \neq y\ x \neq u\ z \neq u\ u \notin Fvars\ \varphi$
unfolding u **using** *getFr_FvarsT_Fvars*[*of* $[x,y,z]$ $[]$ $[\varphi]$] **by** *auto*

have $exu\ x\ \varphi = cnj\ (exi\ x\ \varphi)\ (exi\ u\ (all\ x\ (imp\ \varphi\ (eql\ (Var\ x)\ (Var\ u)))))$
by (*auto simp*: *exu_def Let_def z*[*symmetric*] *exi_rename*[*of all x* (*imp* φ (*eql* (*Var x*) (*Var z*))) *z u*])
also have $\ldots = cnj\ (exi\ x\ \varphi)\ (exi\ y\ (all\ x\ (imp\ \varphi\ (eql\ (Var\ x)\ (Var\ y)))))$
by (*auto simp*: *exi_rename*[*of all x* (*imp* φ (*eql* (*Var x*) (*Var u*))) *u y*]
split: *if_splits*)
finally show *?thesis* **.**
qed

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−
define u **where** u: $u \equiv getFr\ [x,y]\ [t]\ [\varphi]$
have *u_facts*[*simp*]: $u \in var\ u \neq x\ u \neq y\ y \neq u\ x \neq u$
$u \notin FvarsT\ t\ u \notin Fvars\ \varphi$
unfolding u **using** *getFr_FvarsT_Fvars*[*of* $[x,y]$ $[t]$ $[\varphi]$] **by** *auto*
show *?thesis*

by (*auto simp*: *Let_def exu_def_var*[*of* _ *u*] *subst_compose_diff*)
qed

lemma *subst_exu_idle*[*simp*]:
assumes [*simp*]: $x \in var$ $\varphi \in fmla$ $t \in trm$
shows *subst* (*exu x* φ) *t x* = *exu x* φ
by (*intro subst_notIn*) *auto*

lemma *exu_rename*:
assumes [*simp*]: $\varphi \in fmla$ $x \in var$ $y \in var$ $y \notin Fvars\ \varphi$
shows *exu x* φ = *exu y* (*subst* φ (*Var y*) *x*)
proof(*cases y* = *x*)
case [*simp*]: *False*
define *z* **where** *z*: *z* = *getFr* [*x*] [] [φ]
have *z_facts*[*simp*]: $z \in var$ $z \neq x$ $x \neq z$ $z \notin Fvars\ \varphi$
unfolding *z* **using** *getFr_FvarsT_Fvars*[*of* [*x*] [] [φ]] **by** *auto*
define *u* **where** *u*: *u* ≡ *getFr* [*x*,*y*,*z*] [] [φ]
have *u_facts*[*simp*]: $u \in var$ $u \neq x$ $x \neq u$ $u \neq y$ $y \neq u$ $u \neq z$ $z \neq u$
$u \notin Fvars\ \varphi$
unfolding *u* **using** *getFr_FvarsT_Fvars*[*of* [*x*,*y*,*z*] [] [φ]] **by** *auto*
show *?thesis*
by (*auto simp*: *exu_def_var*[*of* _ *u*] *exi_rename*[*of* _ _ *y*] *all_rename*[*of* _ _ *y*])
qed *auto*

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 φ = *exu y* (*subst* φ (*Var y*) *x*)
using *exu_rename* **by** (*cases y* = *x*) (*auto simp del*: *Fvars_subst*)

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)
by (*meson prv_imp_imp_triv prv_imp_mp imp phi psi*)

lemma *prv_imp_refl*:
assumes *phi*: $\varphi \in fmla$
shows *prv* (*imp* φ φ)
by (*metis prv_imp_imp_triv prv_imp_mp prv_imp_trans imp phi*)

lemma *prv_imp_refl2*: $\varphi \in fmla \Longrightarrow \psi \in fmla \Longrightarrow \varphi = \psi \Longrightarrow prv\ (imp\ \varphi\ \psi)$
using *prv_imp_refl* **by** *auto*

lemma *prv_cnjI*:
assumes *phi*: $\varphi \in fmla$ **and** *chi*: $\chi \in fmla$
shows $prv\ \varphi \Longrightarrow prv\ \chi \Longrightarrow prv\ (cnj\ \varphi\ \chi)$
by (*meson cnj prv_imp_cnjI prv_imp_mp imp phi chi*)

lemma *prv_cnjEL*:
assumes *phi*: $\varphi \in fmla$ **and** *chi*: $\chi \in fmla$
shows $prv\ (cnj\ \varphi\ \chi) \Longrightarrow prv\ \varphi$
using *chi phi prv_imp_cnjL prv_imp_mp* **by** *blast*

lemma *prv_cnjER*:
assumes *phi*: $\varphi \in fmla$ **and** *chi*: $\chi \in fmla$
shows $prv\ (cnj\ \varphi\ \chi) \Longrightarrow prv\ \chi$
using *chi phi prv_imp_cnjR prv_imp_mp* **by** *blast*

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−
have $prv\ (imp\ \varphi\ (imp\ \chi\ \psi))$ **by** (*simp add*: *2 chi prv_imp_triv phi psi*)
thus *?thesis* **by** (*metis 1 chi prv_imp_mp prv_imp_trans imp phi psi*)
qed

lemma *prv_imp_trans1*:
assumes *phi*: $\varphi \in fmla$ **and** *chi*: $\chi \in fmla$ **and** *psi*: $\psi \in fmla$
shows $prv\ (imp\ (imp\ \chi\ \psi)\ (imp\ (imp\ \varphi\ \chi)\ (imp\ \varphi\ \psi)))$
by (*meson chi prv_prv_imp_trans prv_imp_imp_triv prv_imp_trans imp phi psi*)

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 $prv\ (imp\ \chi\ (imp\ \varphi\ \psi))$
by (*metis* (*no_types*) *assms prv_prv_imp_trans prv_imp_imp_triv prv_imp_mp prv_imp_trans imp*)

lemma *prv_imp_trans2*:
assumes *phi*: $\varphi \in fmla$ **and** *chi*: $\chi \in fmla$ **and** *psi*: $\psi \in fmla$
shows $prv\ (imp\ (imp\ \varphi\ \chi)\ (imp\ (imp\ \chi\ \psi)\ (imp\ \varphi\ \psi)))$
using *prv_imp_mp prv_imp_trans prv_imp_trans1 prv_imp_imp_triv*
by (*meson chi prv_imp_com imp phi psi*)

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 −
assume $prv\ (imp\ \varphi\ \psi)$
moreover
assume $prv\ (imp\ \varphi\ \chi)$
then have $prv\ (imp\ \varphi\ (imp\ \psi\ f))$ **if** $prv\ (imp\ \chi\ f)\ f \in fmla$ **for** *f*
using *that* **by** (*metis* (*no_types*) *assms imp prv_imp_imp_triv prv_prv_imp_trans*)
moreover have $prv\ (imp\ \varphi\ (imp\ \psi\ \psi)) \Longrightarrow prv\ (imp\ \varphi\ (imp\ \varphi\ \psi))$
using ‹$prv\ (imp\ \varphi\ \psi)$› **by** (*metis* (*no_types*) *assms*(*1*,*3*) *imp prv_imp_com prv_prv_imp_trans*)
ultimately show *?thesis*
by (*metis* (*no_types*) *assms cnj imp prv_imp_cnjI prv_imp_com prv_imp_mp prv_imp_trans*)
qed

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

shows
prv (*imp* (*imp* φ (*imp* χ ψ))
(*imp* χ (*imp* φ ψ)))
by (*metis* (*no_types*) *assms*
prv_prv_imp_trans prv_imp_com prv_imp_imp_triv prv_imp_trans imp)

lemma *prv_cnj_imp_monoR2*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla* **and** $\psi \in$ *fmla*
assumes *prv* (*imp* φ (*imp* χ ψ))
shows *prv* (*imp* (*cnj* φ χ) ψ)
proof −
have *prv* (*imp* (*cnj* φ χ) (*cnj* φ χ))
using *prv_imp_refl* **by** (*blast intro*: *assms*(*1*−*3*))
then have *prv* (*imp* (*imp* (*cnj* φ χ) (*imp* (*cnj* φ χ) ψ)) (*imp* (*cnj* φ χ) ψ))
by (*metis* (*no_types*) *cnj imp assms*(*1*−*3*) *prv_imp_com prv_imp_mp prv_imp_trans*)
then show *?thesis*
by (*metis* (*no_types*) *imp cnj assms prv_imp_cnjL prv_imp_cnjR prv_imp_com prv_imp_mp prv_prv_imp_trans*)
qed

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−
have *prv* (*imp* φ (*imp* (*imp* φ (*imp* χ ψ)) (*imp* χ ψ)))
by (*simp add*: *assms prv_imp_com prv_imp_refl*)
hence *prv* (*imp* φ (*imp* χ (*imp* (*imp* φ (*imp* χ ψ)) ψ)))
by (*metis* (*no_types*, *lifting*) *assms prv_prv_imp_trans prv_imp_imp_com imp*)
hence *prv* (*imp* (*cnj* φ χ)
(*imp* (*imp* φ (*imp* χ ψ)) ψ))
by (*simp add*: *assms prv_cnj_imp_monoR2*)
thus *?thesis* **using** *assms prv_imp_com prv_imp_mp* **by** (*meson cnj imp*)
qed

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)))
by (*metis* (*no_types*) *assms cnj prv_prv_imp_trans prv_imp_cnjI prv_imp_com prv_imp_trans2 imp*)

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))
using *assms prv_imp_cnj_imp prv_imp_mp* **by** (*meson cnj imp*)

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))))
by (*meson assms cnj imp prv_cnj_imp prv_cnj_imp_monoR2 prv_imp_cnj prv_imp_cnjL prv_imp_cnjR*)

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))
using *assms prv_imp_cnjL prv_imp_mp prv_imp_trans2*
by (*metis cnj imp*)

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))
using *prv_imp_cnjR prv_imp_mp prv_imp_trans2*
by (*metis assms cnj imp*)

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−
have *prv* (*imp* (*imp* (*cnj* $\varphi 1$ $\varphi 2$) $\chi 1$) (*imp* (*imp* (*cnj* $\varphi 1$ $\varphi 2$) $\chi 2$) (*imp* (*cnj* $\varphi 1$ $\varphi 2$) (*cnj* $\chi 1$ $\chi 2$))))
using *prv_cnj_imp_monoR*[*of cnj* $\varphi 1$ $\varphi 2$ $\chi 1$ $\chi 2$] *assms* **by** *auto*
hence *prv* (*imp* (*imp* $\varphi 1$ $\chi 1$) (*imp* (*imp* (*cnj* $\varphi 1$ $\varphi 2$) $\chi 2$) (*imp* (*cnj* $\varphi 1$ $\varphi 2$) (*cnj* $\chi 1$ $\chi 2$))))
by (*metis* (*no_types*) *imp cnj assms prv_imp_cnj3L prv_prv_imp_trans*)
hence *prv* (*imp* (*imp* (*cnj* $\varphi 1$ $\varphi 2$) $\chi 2$) (*imp* (*imp* $\varphi 1$ $\chi 1$) (*imp* (*cnj* $\varphi 1$ $\varphi 2$) (*cnj* $\chi 1$ $\chi 2$))))
using *prv_imp_com assms* **by** (*meson cnj imp*)
hence *prv* (*imp* (*imp* $\varphi 2$ $\chi 2$) (*imp* (*imp* $\varphi 1$ $\chi 1$) (*imp* (*cnj* $\varphi 1$ $\varphi 2$) (*cnj* $\chi 1$ $\chi 2$))))
using *prv_imp_cnj3R prv_imp_mp prv_imp_trans1*
by (*metis* (*no_types*) *assms cnj prv_prv_imp_trans imp*)
thus *?thesis* **using** *prv_imp_com assms*
by (*meson cnj imp*)
qed

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* $\chi 1$ $\chi 2$))
using *assms prv_cnj_imp_mono prv_imp_mp*
by (*metis* (*mono_tags*) *cnj prv_prv_imp_trans prv_imp_cnj prv_imp_cnjL prv_imp_cnjR*)

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* $\psi 1$ $\psi 2$)))))
by (*simp add*: *assms prv_cnj_imp prv_imp_cnj prv_imp_cnjL prv_imp_cnjR prv_cnj_imp_monoR2*)

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* $\psi 1$ $\psi 2$)))
by (*simp add*: *assms prv_cnj_imp prv_imp_cnj prv_cnj_imp_monoR2*)

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−
have *prv* (*imp* (*imp* $\varphi 1$ $\chi 1$) (*imp* (*imp* $\varphi 2$ $\chi 2$) (*imp* (*cnj* $\varphi 1$ $\varphi 2$) (*cnj* $\chi 1$ $\chi 2$))))
using *prv_cnj_imp_mono*[*of* $\varphi 1$ $\varphi 2$ $\chi 1$ $\chi 2$] *assms* **by** *auto*
hence *prv* (*imp* (*imp* $\varphi 1$ $\chi 1$) (*imp* (*cnj* $\varphi 1$ $\varphi 2$) (*imp* (*imp* $\varphi 2$ $\chi 2$) (*cnj* $\chi 1$ $\chi 2$))))
by (*metis* (*no_types*, *lifting*) *assms cnj imp prv_imp_imp_com prv_prv_imp_trans*)
hence *prv* (*imp* (*imp* $\varphi 1$ $\chi 1$) (*imp* $\varphi 1$ (*imp* $\varphi 2$ (*imp* (*imp* $\varphi 2$ $\chi 2$) (*cnj* $\chi 1$ $\chi 2$)))))
using *prv_imp_cnj_imp prv_imp_mp prv_imp_trans2*
by (*metis* (*mono_tags*) *assms cnj prv_prv_imp_trans imp*)
hence *1*: *prv* (*imp* (*imp* $\varphi 1$ $\chi 1$) (*imp* $\varphi 1$ (*imp* (*imp* $\varphi 2$ $\chi 2$) (*imp* $\varphi 2$ (*cnj* $\chi 1$ $\chi 2$)))))

using *prv_cnj_imp prv_imp_cnjR prv_imp_mp prv_imp_trans1*
by (*metis* (*no_types*) *assms cnj prv_cnj_imp_monoR prv_prv_imp_trans prv_imp_imp_triv imp*)
thus *?thesis*
by (*metis* (*no_types*, *lifting*) *assms cnj imp prv_prv_imp_trans prv_imp_imp_com*)
qed

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* $\chi 1$ $\chi 2$)))
by (*simp add*: *assms prv_cnj_imp prv_cnj_mono*)

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))
by (*simp add*: *assms prv_imp_cnj prv_imp_cnjL prv_imp_cnjR eqv_def*)

lemma *prv_eqv_eqv*:
assumes $\varphi \in fmla$ **and** $\chi \in fmla$
shows *prv* (*eqv* (*eqv* φ χ) (*eqv* χ φ))
using *assms prv_cnjI prv_eqv_imp eqv_def* **by** *auto*

lemma *prv_imp_eqvEL*:
$\varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow$ *prv* (*eqv* $\varphi 1$ $\varphi 2$) $\Longrightarrow$ *prv* (*imp* $\varphi 1$ $\varphi 2$)
unfolding *eqv_def* **by** (*meson cnj imp prv_imp_cnjL prv_imp_mp*)

lemma *prv_imp_eqvER*:
$\varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow$ *prv* (*eqv* $\varphi 1$ $\varphi 2$) $\Longrightarrow$ *prv* (*imp* $\varphi 2$ $\varphi 1$)
unfolding *eqv_def* **by** (*meson cnj imp prv_imp_cnjR prv_imp_mp*)

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−
have *prv* (*imp* (*eqv* φ χ) (*imp* (*imp* χ ψ) (*imp* φ ψ)))
using *assms prv_imp_cnjL prv_imp_mp prv_imp_trans2 eqv_def*
by (*metis prv_imp_cnj3L eqv imp*)
hence *prv* (*imp* (*eqv* φ χ) (*imp* (*eqv* χ ψ) (*imp* φ ψ)))
using *prv_imp_cnjL prv_imp_mp prv_imp_trans2 eqv_def*
by (*metis* (*no_types*) *assms prv_imp_cnj3L prv_imp_com eqv imp*)
hence *1*: *prv* (*imp* (*cnj* (*eqv* φ χ) (*eqv* χ ψ)) (*imp* φ ψ))
using *prv_cnj_imp_monoR2*
by (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*))
have *prv* (*imp* (*eqv* φ χ) (*imp* (*imp* ψ χ) (*imp* ψ φ)))
using *prv_imp_cnjR prv_imp_mp prv_imp_trans1 eqv_def*
by (*metis assms prv_cnj_imp_monoR2 prv_imp_triv imp*)
hence *prv* (*imp* (*eqv* φ χ) (*imp* (*eqv* χ ψ) (*imp* ψ φ)))
by (*metis assms cnj eqv_def imp prv_imp_cnj3R prv_prv_imp_trans*)
hence *2*: *prv* (*imp* (*cnj* (*eqv* φ χ) (*eqv* χ ψ)) (*imp* ψ φ))
using *prv_cnj_imp_monoR2*
by (*metis* (*no_types*, *lifting*) *assms eqv imp*)
have *prv* (*imp* (*cnj* (*eqv* φ χ) (*eqv* χ ψ)) (*eqv* φ ψ))
using *1 2* **using** *assms prv_imp_cnj* **by** (*auto simp*: *eqv_def*[*of* φ ψ])
thus *?thesis*
by (*simp add*: *assms prv_cnj_imp*)
qed

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))
by (*simp add*: *assms prv_eqv_imp_trans prv_cnj_imp_monoR2*)

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)
by (*simp add*: *assms eqv_def prv_cnjI*)

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)
by (*simp add*: *prv_cnjI prv_imp_refl eqv_def*)

lemma *prv_eqv_sym*:
assumes $\varphi \in fmla$ **and** $\chi \in fmla$
shows *prv* (*eqv* φ χ) $\Longrightarrow$ *prv* (*eqv* χ φ)
using *assms prv_cnjI prv_imp_cnjL prv_imp_cnjR prv_imp_mp eqv_def*
by (*meson prv_eqv_imp eqv*)

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)
using *assms prv_cnjI prv_cnj_imp_monoR2 prv_imp_mp prv_imp_trans1 prv_imp_imp_triv eqv_def*
by (*metis prv_prv_imp_trans prv_imp_cnjL prv_imp_cnjR eqv imp*)

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* $\varphi 1$ χ) (*imp* $\varphi 2$ χ)))
proof −
have *f*: *prv* (*imp* (*eqv* $\varphi 1$ $\varphi 2$) (*eqv* (*imp* $\varphi 1$ χ) (*imp* $\varphi 2$ χ)))
if *prv* (*imp* (*eqv* $\varphi 1$ $\varphi 2$) (*imp* (*imp* $\varphi 2$ χ) (*imp* $\varphi 1$ χ))) *prv* (*imp* (*eqv* $\varphi 1$ $\varphi 2$) (*imp* (*imp* $\varphi 1$ χ) (*imp* $\varphi 2$ χ)))
using *assms that prv_imp_cnj* **by** (*auto simp*: *eqv_def*)
moreover have (*prv* (*imp* (*eqv* $\varphi 1$ $\varphi 2$) (*imp* $\varphi 1$ $\varphi 2$)) ∧ *prv* (*imp* (*eqv* $\varphi 1$ $\varphi 2$) (*imp* $\varphi 2$ $\varphi 1$)))
by (*simp add*: *assms eqv_def prv_imp_cnjL prv_imp_cnjR*)
ultimately show *?thesis*
by (*metis* (*no_types*) *assms eqv imp prv_imp_trans2 prv_prv_imp_trans*)
qed

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* φ $\chi 1$) (*imp* φ $\chi 2$)))
by (*simp add*: *assms prv_cnj_mono prv_imp_trans1 eqv_def*)

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−
have *prv* (*imp* (*eqv* $\varphi 1$ $\varphi 2$) (*imp* (*eqv* $\chi 1$ $\chi 2$) (*cnj* (*eqv* (*imp* $\varphi 1$ $\chi 1$) (*imp* $\varphi 2$ $\chi 1$))
(*eqv* (*imp* $\varphi 2$ $\chi 1$) (*imp* $\varphi 2$ $\chi 2$)))))
using *prv_imp_cnj5*
[*OF* _ _ _ _ *imp_imp_compat_eqvL*[*of* $\varphi 1$ $\varphi 2$ $\chi 1$] *imp_imp_compat_eqvR*[*of* $\varphi 2$ $\chi 1$ $\chi 2$]] *assms*
by *auto*
hence *prv* (*imp* (*cnj* (*eqv* $\varphi 1$ $\varphi 2$) (*eqv* $\chi 1$ $\chi 2$)) (*cnj* (*eqv* (*imp* $\varphi 1$ $\chi 1$) (*imp* $\varphi 2$ $\chi 1$))
(*eqv* (*imp* $\varphi 2$ $\chi 1$) (*imp* $\varphi 2$ $\chi 2$))))

by(*simp add*: *assms prv_cnj_imp_monoR2*)
hence *prv* (*imp* (*cnj* (*eqv* $\varphi 1$ $\varphi 2$) (*eqv* $\chi 1$ $\chi 2$)) (*eqv* (*imp* $\varphi 1$ $\chi 1$) (*imp* $\varphi 2$ $\chi 2$)))
using *assms prv_eqv_cnj_trans*[*of imp* $\varphi 1$ $\chi 1$ *imp* $\varphi 2$ $\chi 1$ *imp* $\varphi 2$ $\chi 2$]
using *prv_imp_mp prv_imp_trans2*
by (*metis* (*no_types*) *cnj prv_prv_imp_trans eqv imp*)
thus *?thesis* **using** *assms prv_cnj_imp* **by** *auto*
qed

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* $\varphi 1$ χ) (*imp* $\varphi 2$ χ))
using *assms prv_imp_mp imp_imp_compat_eqvL* **by** (*meson eqv imp*)

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* φ $\chi 1$) (*imp* φ $\chi 2$))
using *assms prv_imp_mp imp_imp_compat_eqvR* **by** (*meson eqv imp*)

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* $\varphi 2$ $\chi 2$))
using *assms prv_imp_mp imp_imp_compat_eqv* **by** (*metis eqv imp*)

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* $\varphi 1$ χ) (*cnj* $\varphi 2$ χ)))
proof −
have *prv* (*imp* (*imp* (*imp* $\varphi 2$ $\varphi 1$) (*imp* (*cnj* $\varphi 2$ χ) (*cnj* $\varphi 1$ χ)))
(*imp* (*cnj* (*imp* $\varphi 1$ $\varphi 2$) (*imp* $\varphi 2$ $\varphi 1$)) (*cnj* (*imp* (*cnj* $\varphi 1$ χ) (*cnj* $\varphi 2$ χ))
(*imp* (*cnj* $\varphi 2$ χ) (*cnj* $\varphi 1$ χ)))))
by (*metis* (*no_types*) *imp cnj assms prv_imp_mp assms prv_cnj_imp_mono prv_imp_com prv_imp_refl*)
then show *?thesis*
by (*metis* (*no_types*) *imp cnj assms prv_imp_mp assms eqv_def prv_cnj_imp_mono prv_imp_com prv_imp_refl*)
qed

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* φ $\chi 1$) (*cnj* φ $\chi 2$)))
by (*simp add*: *assms prv_cnj_mono prv_imp_cnj3R prv_imp_cnj4 prv_imp_cnjL prv_imp_triv eqv_def*)

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−
have *prv* (*imp* (*eqv* $\varphi 1$ $\varphi 2$) (*imp* (*eqv* $\chi 1$ $\chi 2$) (*cnj* (*eqv* (*cnj* $\varphi 1$ $\chi 1$) (*cnj* $\varphi 2$ $\chi 1$))
(*eqv* (*cnj* $\varphi 2$ $\chi 1$) (*cnj* $\varphi 2$ $\chi 2$)))))
using *prv_imp_cnj5*
[*OF* _ _ _ _ *imp_cnj_compat_eqvL*[*of* $\varphi 1$ $\varphi 2$ $\chi 1$] *imp_cnj_compat_eqvR*[*of* $\varphi 2$ $\chi 1$ $\chi 2$]] *assms* **by** *auto*
hence *prv* (*imp* (*cnj* (*eqv* $\varphi 1$ $\varphi 2$) (*eqv* $\chi 1$ $\chi 2$)) (*cnj* (*eqv* (*cnj* $\varphi 1$ $\chi 1$) (*cnj* $\varphi 2$ $\chi 1$))
(*eqv* (*cnj* $\varphi 2$ $\chi 1$) (*cnj* $\varphi 2$ $\chi 2$))))

by(*simp add*: *assms prv_cnj_imp_monoR2*)
hence *prv* (*imp* (*cnj* (*eqv* $\varphi 1$ $\varphi 2$) (*eqv* $\chi 1$ $\chi 2$)) (*eqv* (*cnj* $\varphi 1$ $\chi 1$) (*cnj* $\varphi 2$ $\chi 2$)))
using *assms prv_eqv_cnj_trans*[*of cnj* $\varphi 1$ $\chi 1$ *cnj* $\varphi 2$ $\chi 1$ *cnj* $\varphi 2$ $\chi 2$]
using *prv_imp_mp prv_imp_trans2*
by (*metis* (*no_types*) *cnj prv_prv_imp_trans eqv*)
thus *?thesis* **using** *assms prv_cnj_imp* **by** *auto*
qed

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* $\varphi 1$ χ) (*cnj* $\varphi 2$ χ))
using *assms prv_imp_mp imp_cnj_compat_eqvL* **by** (*meson eqv cnj*)

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* φ $\chi 1$) (*cnj* φ $\chi 2$))
using *assms prv_imp_mp imp_cnj_compat_eqvR* **by** (*meson eqv cnj*)

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* $\varphi 2$ $\chi 2$))
using *assms prv_imp_mp imp_cnj_compat_eqv* **by** (*metis eqv imp cnj*)

lemma *prv_eqv_prv*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla*
assumes *prv* φ **and** *prv* (*eqv* φ χ)
shows *prv* χ
by (*metis assms prv_imp_cnjL prv_imp_mp eqv eqv_def imp*)

lemma *prv_eqv_prv_rev*:
assumes $\varphi \in$ *fmla* **and** $\chi \in$ *fmla*
assumes *prv* φ **and** *prv* (*eqv* χ φ)
shows *prv* χ
using *assms prv_eqv_prv prv_eqv_sym* **by** *blast*

lemma *prv_imp_eqv_transi*:
assumes $\varphi \in$ *fmla* **and** $\chi 1 \in$ *fmla* **and** $\chi 2 \in$ *fmla*
assumes *prv* (*imp* φ $\chi 1$) **and** *prv* (*eqv* $\chi 1$ $\chi 2$)
shows *prv* (*imp* φ $\chi 2$)
by (*meson assms imp imp_compat_eqvR prv_eqv_prv*)

lemma *prv_imp_eqv_transi_rev*:
assumes $\varphi \in$ *fmla* **and** $\chi 1 \in$ *fmla* **and** $\chi 2 \in$ *fmla*
assumes *prv* (*imp* φ $\chi 2$) **and** *prv* (*eqv* $\chi 1$ $\chi 2$)
shows *prv* (*imp* φ $\chi 1$)
by (*meson assms prv_eqv_sym prv_imp_eqv_transi*)

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* $\varphi 2$ χ)
shows *prv* (*imp* $\varphi 1$ χ)
by (*meson assms prv_imp_eqv_transi prv_imp_refl prv_prv_imp_trans*)

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* $\varphi 1$ χ)
shows *prv* (*imp* $\varphi 2$ χ)
by (*meson assms prv_eqv_imp_transi prv_eqv_sym*)

lemma *prv_imp_monoL*: $\varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
prv (*imp* χ ψ) $\Longrightarrow$ *prv* (*imp* (*imp* φ χ) (*imp* φ ψ))
by (*meson imp prv_imp_mp prv_imp_trans1*)

lemma *prv_imp_monoR*: $\varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
prv (*imp* ψ χ) $\Longrightarrow$ *prv* (*imp* (*imp* χ φ) (*imp* ψ φ))
by (*meson imp prv_imp_mp prv_imp_trans2*)

More properties involving conjunction:

lemma *prv_cnj_com_imp*:
assumes φ[*simp*]: $\varphi \in fmla$ **and** χ[*simp*]: $\chi \in fmla$
shows *prv* (*imp* (*cnj* φ χ) (*cnj* χ φ))
by (*simp add*: *prv_imp_cnj prv_imp_cnjL prv_imp_cnjR*)

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))
by (*simp add*: *prv_cnj_com_imp prv_eqvI*)

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))
by (*simp add*: *prv_cnj_imp_monoR2 prv_imp_cnj prv_imp_cnjL prv_imp_cnjR prv_imp_triv*)

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 −
have *prv* (*imp* (*cnj* φ χ) (*imp* ψ φ)) $\wedge$ *cnj* χ $\psi \in fmla \wedge$ *cnj* φ $\chi \in fmla$
by (*meson* χ φ ψ *cnj imp prv_cnj_imp_monoR2 prv_imp_imp_triv prv_prv_imp_trans*)
then show *?thesis*
using χ φ ψ *cnj imp prv_cnj_imp_monoR2 prv_imp_cnj4 prv_imp_cnjI prv_imp_triv* **by** *presburger*
qed

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))
by (*simp add*: *prv_cnj_assoc_imp1 prv_cnj_assoc_imp2 prv_eqvI*)

lemma *prv_cnj_com_imp3*:
assumes $\varphi 1 \in fmla$ $\varphi 2 \in fmla$ $\varphi 3 \in fmla$
shows *prv* (*imp* (*cnj* $\varphi 1$ (*cnj* $\varphi 2$ $\varphi 3$))
(*cnj* $\varphi 2$ (*cnj* $\varphi 1$ $\varphi 3$)))
by (*simp add*: *assms prv_cnj_imp_monoR2 prv_imp_cnj prv_imp_cnjL prv_imp_refl prv_imp_triv*)

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)
using *assms prv_all_inst prv_imp_mp* **by** (*meson subst all*)

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)
using *assms prv_exi_inst prv_imp_mp* **by** (*meson subst exi*)

lemma *prv_imp_imp_exi*:
assumes $x \in var$ **and** $\varphi \in fmla$ **and** $\chi \in fmla$
assumes $x \notin Fvars\ \varphi$
shows *prv* (*imp* (*exi* x (*imp* φ χ)) (*imp* φ (*exi* x χ)))
using *assms imp exi Fvars_exi Fvars_imp Un_iff assms prv_exi_imp_gen prv_exi_inst prv_imp_mp*
prv_imp_trans1 member_remove remove_def subst_same_Var **by** (*metis* (*full_types*) *Var*)

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))
using *assms prv_imp_imp_exi prv_imp_mp* **by** (*meson exi imp*)

lemma *prv_exi_imp*:
assumes *x*: $x \in var$ **and** $\varphi \in fmla$ **and** $\chi \in fmla$
assumes $x \notin Fvars\ \chi$ **and** *d*: *prv* (*all* x (*imp* φ χ))
shows *prv* (*imp* (*exi* x φ) χ)
proof−
have *prv* (*imp* φ χ) **using** *prv_allE*[*of* x _ *Var* x, *of imp* φ χ] *assms* **by** *simp*
thus *?thesis* **using** *assms prv_exi_imp_gen* **by** *blast*
qed

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* φ χ))
shows *prv* (*imp* φ (*all* x χ))
proof−
have *prv* (*imp* φ χ) **using** *prv_allE*[*of* x _ *Var* x, *of imp* φ χ] *assms* **by** *simp*
thus *?thesis* **using** *assms prv_all_imp_gen* **by** *blast*
qed

lemma *prv_exi_inst_same*:
assumes $\varphi \in fmla$ $\chi \in fmla$ $x \in var$
shows *prv* (*imp* φ (*exi* x φ))
proof−
have *0*: φ = *subst* φ (*Var* x) x **using** *assms* **by** *simp*
show *?thesis*
apply(*subst 0*)
using *assms* **by** (*intro prv_exi_inst*) *auto*
qed

lemma *prv_exi_cong*:
assumes $\varphi \in fmla$ $\chi \in fmla$ $x \in var$
and *prv* (*imp* φ χ)
shows *prv* (*imp* (*exi* x φ) (*exi* x χ))
proof−
have *0*: *prv* (*imp* χ (*exi* x χ)) **using** *assms prv_exi_inst_same* **by** *auto*
show *?thesis*
using *assms* **apply**(*intro prv_exi_imp_gen*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *0 exi prv_prv_imp_trans* **by** *blast* **.**

qed

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* $x\ \chi$)
by (*meson exi assms prv_exi_cong prv_imp_mp*)

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−
have *0*: *prv* (*imp* (*all* $x\ \varphi$) χ) **using** *assms*
by (*metis Var all prv_all_inst prv_prv_imp_trans subst_same_Var*)
show *?thesis* **by** (*simp add*: *0 assms prv_all_imp_gen*)
qed

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* $x\ \chi$)
by (*meson all assms prv_all_cong prv_imp_mp*)

Quantifiers versus free variables and substitution:

lemma *exists_no_Fvars*: $\exists\ \varphi.\ \varphi \in fmla \wedge prv\ \varphi \wedge Fvars\ \varphi = \{\}$
proof−
obtain n **where** [*simp*]: $n \in num$ **using** *numNE* **by** *blast*
show *?thesis*
by (*intro exI*[*of* _ *imp* (*eql n n*) (*eql n n*)]) (*simp add*: *prv_imp_refl*)
qed

lemma *prv_all_gen*:
assumes $x \in var$ **and** $\varphi \in fmla$
assumes *prv* φ **shows** *prv* (*all* $x\ \varphi$)
using *assms prv_all_imp_gen prv_imp_mp prv_imp_triv exists_no_Fvars* **by** *blast*

lemma *all_subst_rename_prv*:
$\varphi \in fmla \Longrightarrow x \in var \Longrightarrow y \in var \Longrightarrow$
$y \notin Fvars\ \varphi \Longrightarrow prv\ (all\ x\ \varphi) \Longrightarrow prv\ (all\ y\ (subst\ \varphi\ (Var\ y)\ x))$
by (*simp add*: *prv_allE prv_all_gen*)

lemma *allE_id*:
assumes $y \in var$ **and** $\varphi \in fmla$
assumes *prv* (*all* $y\ \varphi$)
shows *prv* φ
using *assms prv_allE* **by** (*metis Var subst_same_Var*)

lemma *prv_subst*:
assumes $x \in var$ **and** $\varphi \in fmla$ **and** $t \in trm$
shows *prv* $\varphi \Longrightarrow$ *prv* (*subst* $\varphi\ t\ x$)
by (*simp add*: *assms prv_allE prv_all_gen*)

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*)
using *assms* **apply** (*induct txs arbitrary*: φ)

```
  subgoal by auto
  subgoal for tx txs φ by (cases tx) (auto intro: prv_subst) .

lemma prv_psubst:
  assumes φ ∈ fmla and snd ‘ (set txs) ⊆ var and fst ‘ (set txs) ⊆ trm
    and prv φ
  shows prv (psubst φ txs)
proof−
  define us where us: us ≡ getFrN (map snd txs) (map fst txs) [φ] (length txs)
  have 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
    using assms unfolding us
    using getFrN_FvarsT[of map snd txs map fst txs [φ] _ length txs]
      getFrN_Fvars[of map snd txs map fst txs [φ] _ length txs]
      getFrN_var[of map snd txs map fst txs [φ] _ length txs]
      getFrN_length[of map snd txs map fst txs [φ] length txs]
      getFrN_length[of map snd txs map fst txs [φ] length txs]
        apply −
    subgoal by auto
    subgoal by fastforce
    subgoal by auto
    subgoal by (fastforce simp: image_iff)
    by auto
  show ?thesis using assms us_facts unfolding psubst_def
    by (auto simp: Let_def us[symmetric] intro!: prv_rawpsubst rawpsubst dest!: set_zip_D)
qed

lemma prv_eqv_rawpsubst:
  φ ∈ fmla ⟹ ψ ∈ fmla ⟹ snd ‘ set txs ⊆ var ⟹ fst ‘ set txs ⊆ trm ⟹ prv (eqv φ ψ) ⟹
 prv (eqv (rawpsubst φ txs) (rawpsubst ψ txs))
  by (metis eqv prv_rawpsubst rawpsubst_eqv)

lemma prv_eqv_psubst:
  φ ∈ fmla ⟹ ψ ∈ fmla ⟹ snd ‘ set txs ⊆ var ⟹ fst ‘ set txs ⊆ trm ⟹ prv (eqv φ ψ) ⟹
 distinct (map snd txs) ⟹
 prv (eqv (psubst φ txs) (psubst ψ txs))
  by (metis eqv prv_psubst psubst_eqv)

lemma prv_all_imp_trans:
  assumes x ∈ var and φ ∈ fmla and χ ∈ fmla and ψ ∈ fmla
  shows prv (all x (imp φ χ)) ⟹ prv (all x (imp χ ψ)) ⟹ prv (all x (imp φ ψ))
  by (metis Var assms prv_allE prv_all_gen prv_prv_imp_trans imp subst_same_Var)

lemma prv_all_imp_cnj:
  assumes x ∈ var and φ ∈ fmla and χ ∈ fmla and ψ ∈ fmla
  shows prv (all x (imp φ (imp ψ χ))) ⟹ prv (all x (imp (cnj ψ φ) χ))
 by (metis Var assms cnj prv_allE prv_all_gen prv_imp_com prv_cnj_imp_monoR2 imp subst_same_Var)

lemma prv_all_imp_cnj_rev:
  assumes x ∈ var and φ ∈ fmla and χ ∈ fmla and ψ ∈ fmla
  shows prv (all x (imp (cnj φ ψ) χ)) ⟹ prv (all x (imp φ (imp ψ χ)))
  by (metis (full_types) Var assms cnj prv_allE prv_all_gen prv_cnj_imp imp subst_same_Var)
```

3.1.3 Properties concerning equality

lemma *prv_eql_reflT*:
assumes *t*: $t \in trm$
shows *prv* (*eql t t*)
proof−
obtain *x* **where** *x*: $x \in var$ **using** *var_NE* **by** *auto*
show *?thesis* **using** *prv_subst*[*OF x _ t prv_eql_refl*[*OF x*]] *x t* **by** *simp*
qed

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−
have *finite* ($\{x\} \cup FvarsT\ t1 \cup FvarsT\ t2 \cup Fvars\ \varphi$) (**is** *finite ?A*)
by (*simp add*: *assms finite_FvarsT finite_Fvars*)
then obtain *y* **where** *y*: $y \in var$ **and** $y \notin ?A$
by (*meson finite_subset infinite_var subset_iff*)
hence *xy*: $x \neq y$ **and** *yt1*: $y \notin FvarsT\ t1$ **and** *yt2*: $y \notin FvarsT\ t2$ **and** $y\varphi$: $y \notin Fvars\ \varphi$ **by** *auto*
have *x*: $x \notin Fvars$ (*subst* φ (*Var y*) *x*) **using** *xy y assms* **by** *simp*
hence *1*: *prv* (*imp* (*eql t1* (*Var y*)) (*imp* (*subst* φ *t1 x*) (*subst* φ (*Var y*) *x*)))
using *xy y assms prv_subst*[*OF xx _ _ prv_eql_subst*[*OF xx y* φ]] **by** *simp*
have *yy*: $y \notin Fvars$ (*subst* φ *t1 x*) **using** *yt1 y*φ *assms* **by** *simp*
from *prv_subst*[*OF y _ _ 1*, *of t2*]
show *?thesis* **using** *xy yt1 yt2 y*φ *x xx y yy assms* **by** *auto*
qed

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*))
by (*meson assms eql imp local.subst prv_eql_subst_trm prv_imp_mp*)

lemma *prv_eql_sym*:
assumes [*simp*]: $t1 \in trm\ t2 \in trm$
shows *prv* (*imp* (*eql t1 t2*) (*eql t2 t1*))
proof−
obtain *x* **where** *x*[*simp*]: $x \in var\ x \notin FvarsT\ t1$
by (*meson assms finite_FvarsT finite_subset infinite_var subsetI*)
thus *?thesis* **using** *prv_eql_subst_trm*[*of x eql* (*Var x*) *t1 t1 t2*, *simplified*]
by (*meson assms eql imp prv_eql_reflT prv_imp_com prv_imp_mp*)
qed

lemma *prv_prv_eql_sym*: $t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow prv\ (eql\ t1\ t2) \Longrightarrow prv\ (eql\ t2\ t1)$
by (*meson eql prv_eql_sym prv_imp_mp*)

lemma *prv_all_eql*:
assumes $x \in var$ **and** $y \in var$ **and** $\varphi \in fmla$ **and** $t1 \in trm$ **and** $t2 \in trm$
shows *prv* (*all x* ((*imp* (*eql t1 t2*)
(*imp* (*subst* φ *t2 y*) (*subst* φ *t1 y*)))))
by (*meson subst assms prv_all_gen prv_prv_imp_trans prv_eql_subst_trm prv_eql_sym eql imp*)

lemma *prv_eql_subst_trm_rev*:
assumes $t1 \in trm$ **and** $t2 \in trm$ **and** $\varphi \in fmla$ **and** $y \in var$
shows
prv ((*imp* (*eql t1 t2*)
(*imp* (*subst* φ *t2 y*) (*subst* φ *t1 y*))))
using *assms prv_all_eql prv_all_inst prv_imp_mp subst_same_Var*

by (*meson subst prv_prv_imp_trans prv_eql_subst_trm prv_eql_sym eql imp*)

lemma *prv_eql_subst_trm_rev2*:
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 *t2 x*) (*subst* φ *t1 x*))
by (*meson assms eql imp local.subst prv_eql_subst_trm_rev prv_imp_mp*)

lemma *prv_eql_subst_trm_eqv*:
assumes $x \in var$ **and** $\varphi \in fmla$ **and** $t1 \in trm$ **and** $t2 \in trm$
assumes *prv* (*eql t1 t2*)
shows *prv* (*eqv* (*subst* φ *t1 x*) (*subst* φ *t2 x*))
using *assms prv_eql_subst_trm2*[*OF assms*] *prv_eql_subst_trm_rev2*[*OF assms*]
prv_eqvI **by** *auto*

lemma *prv_eql_subst_trm_id*:
assumes $y \in var$ $\varphi \in fmla$ **and** $n \in num$
shows *prv* (*imp* (*eql* (*Var y*) *n*) (*imp* φ (*subst* φ *n y*)))
using *assms prv_eql_subst_trm*
by (*metis Var in_num subst_same_Var*)

lemma *prv_eql_subst_trm_id_back*:
assumes $y \in var$ $\varphi \in fmla$ **and** $n \in num$
shows *prv* (*imp* (*eql* (*Var y*) *n*) (*imp* (*subst* φ *n y*) φ))
by (*metis Var assms in_num prv_eql_subst_trm_rev subst_same_Var*)

lemma *prv_eql_subst_trm_id_rev*:
assumes $y \in var$ $\varphi \in fmla$ **and** $n \in num$
shows *prv* (*imp* (*eql n* (*Var y*)) (*imp* φ (*subst* φ *n y*)))
using *assms prv_eql_subst_trm_rev* **by** (*metis Var in_num subst_same_Var*)

lemma *prv_eql_subst_trm_id_back_rev*:
assumes $y \in var$ $\varphi \in fmla$ **and** $n \in num$
shows *prv* (*imp* (*eql n* (*Var y*)) (*imp* (*subst* φ *n y*) φ))
by (*metis* (*full_types*) *Var assms in_num prv_eql_subst_trm subst_same_Var*)

lemma *prv_eql_imp_trans_rev*:
assumes *t1*[*simp*]: $t1 \in trm$ **and** *t2*[*simp*]: $t2 \in trm$ **and** *t3*[*simp*]: $t3 \in trm$
shows *prv* (*imp* (*eql t1 t2*) (*imp* (*eql t1 t3*) (*eql t2 t3*)))
proof−
obtain *y1* **where** *y1*[*simp*]: $y1 \in var$ **and** $y1 \notin FvarsT\ t1 \cup FvarsT\ t2 \cup FvarsT\ t3$
using *finite_FvarsT finite_subset infinite_var subsetI t1 t2 t3*
by (*metis* (*full_types*) *infinite_Un*)
hence [*simp*]: $y1 \notin FvarsT\ t1$ $y1 \notin FvarsT\ t2$ $y1 \notin FvarsT\ t3$ **by** *auto*
obtain *y2* **where** *y2*[*simp*]: $y2 \in var$ **and** $y2 \notin FvarsT\ t1 \cup FvarsT\ t2 \cup FvarsT\ t3 \cup \{y1\}$
using *finite_FvarsT finite_subset infinite_var subsetI t1 t2 t3*
by (*metis* (*full_types*) *finite_insert infinite_Un insert_is_Un*)
hence [*simp*]: $y2 \notin FvarsT\ t1$ $y2 \notin FvarsT\ t2$ $y2 \notin FvarsT\ t3$ $y1 \neq y2$ **by** *auto*
have *0*: *prv* (*imp* (*eql* (*Var y1*) (*Var y2*))
(*imp* (*eql* (*Var y1*) *t3*) (*eql* (*Var y2*) *t3*)))
using *prv_eql_subst*[*OF y1 y2*, *of eql* (*Var y1*) *t3*] **by** *simp*
note *1* = *prv_subst*[*OF y1 _ t1 0*, *simplified*]
show *?thesis* **using** *prv_subst*[*OF y2 _ t2 1*, *simplified*] **.**
qed

lemma *prv_eql_imp_trans*:
assumes *t1*[*simp*]: $t1 \in trm$ **and** *t2*[*simp*]: $t2 \in trm$ **and** *t3*[*simp*]: $t3 \in trm$
shows *prv* (*imp* (*eql t1 t2*) (*imp* (*eql t2 t3*) (*eql t1 t3*)))

by (*meson eql imp prv_eql_sym prv_eql_imp_trans_rev prv_prv_imp_trans t1 t2 t3*)

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*)
by (*meson assms cnj eql prv_cnjI prv_cnj_imp_monoR2 prv_eql_imp_trans prv_imp_mp*)

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*))
unfolding *softSubst_def* **by** (*rule prv_imp_exi*)
(*auto simp*: *prv_eql_reflT prv_imp_cnj prv_imp_refl prv_imp_triv subst_compose_same*
intro: *prv_exiI*[*of* _ _ *t*])

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*)
using *assms prv_subst_imp_softSubst*
by (*metis Var cnj eql exi subst prv_imp_mp softSubst_def*)

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*))
unfolding *softSubst_def* **apply**(*rule prv_exi_imp_gen*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*metis Var assms*(*1*−*3*) *eql subst prv_cnj_imp_monoR2 prv_eql_subst_trm subst_same_Var*)
.

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*)
by (*metis Var assms cnj eql exi local.subst prv_imp_mp prv_softSubst_imp_subst softSubst_def*)

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*))
by (*metis Var assms cnj eql exi local.subst prv_eqvI prv_softSubst_imp_subst prv_subst_imp_softSubst*
softSubst_def)

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
using *assms prv_imp_mp* **by** *blast*

lemma *prv_cnjR_fls*: $\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*eqv* (*cnj fls* φ) *fls*)
by (*simp add*: *prv_eqvI prv_imp_cnjL*)

lemma *prv_cnjL_fls*: $\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*eqv* (*cnj* φ *fls*) *fls*)
by (*simp add*: *prv_eqvI prv_imp_cnjR*)

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*))
using *assms prv_imp_com prv_imp_refl neg_def* **by** *auto*

lemma *prv_neg_fls*:
assumes $\varphi \in$ *fmla*
assumes *prv* φ **and** *prv* (*neg* φ)
shows *prv fls*
by (*metis assms prv_imp_mp fls neg_def*)

lemma *prv_imp_neg_neg*:
assumes $\varphi \in$ *fmla*
shows *prv* (*imp* φ (*neg* (*neg* φ)))
using *assms prv_imp_neg_fls neg_def* **by** *auto*

lemma *prv_neg_neg*:
assumes $\varphi \in$ *fmla*
assumes *prv* φ
shows *prv* (*neg* (*neg* φ))
by (*metis assms prv_imp_mp prv_imp_neg_fls neg neg_def*)

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)))
unfolding *neg_def* **using** *prv_imp_trans2*[*OF assms fls*] **.**

lemma *prv_imp_neg_rev*:
assumes $\varphi \in fmla$ **and** $\chi \in fmla$
assumes *prv* (*imp* φ χ)
shows *prv* (*imp* (*neg* χ) (*neg* φ))
by (*meson assms imp neg prv_imp_imp_neg_rev prv_imp_mp*)

lemma *prv_eqv_neg_prv_fls*:
$\varphi \in fmla \Longrightarrow$
prv (*eqv* φ (*neg* φ)) $\Longrightarrow$ *prv fls*
by (*metis cnj fls neg neg_def prv_cnj_imp_monoR2 prv_eqv_imp_transi prv_imp_cnj prv_imp_eqvER prv_imp_mp prv_imp_neg_rev*)

lemma *prv_eqv_eqv_neg_prv_fls*:
$\varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow$
prv (*eqv* φ χ) $\Longrightarrow$ *prv* (*eqv* φ (*neg* χ))$\Longrightarrow$ *prv fls*
by (*meson neg prv_eqv_neg_prv_fls prv_eqv_sym prv_eqv_trans*)

lemma *prv_eqv_eqv_neg_prv_fls2*:
$\varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow$
prv (*eqv* φ χ) $\Longrightarrow$ *prv* (*eqv* χ (*neg* φ))$\Longrightarrow$ *prv fls*
by (*simp add*: *prv_eqv_eqv_neg_prv_fls prv_eqv_sym*)

lemma *prv_neg_imp_imp_trans*:
assumes $\varphi \in fmla$ $\chi \in fmla$ $\psi \in fmla$
and *prv* (*neg* φ)
and *prv* (*imp* χ (*imp* ψ φ))
shows *prv* (*imp* χ (*neg* ψ))
unfolding *neg_def*
by (*metis assms cnj fls neg_def prv_cnj_imp prv_cnj_imp_monoR2 prv_prv_imp_trans*)

lemma *prv_imp_neg_imp_neg_imp*:
assumes $\varphi \in fmla$ $\chi \in fmla$
and *prv* (*neg* φ)
shows *prv* ((*imp* χ (*neg* (*imp* χ φ))))
by (*metis assms fls imp neg_def prv_imp_com prv_imp_monoL*)

lemma *prv_prv_neg_imp_neg*:
assumes $\varphi \in fmla$ $\chi \in fmla$
and *prv* φ **and** *prv* χ
shows *prv* (*neg* (*imp* φ (*neg* χ)))
by (*meson assms imp neg prv_imp_mp prv_imp_neg_imp_neg_imp prv_neg_neg*)

lemma *prv_imp_neg_imp_cnjL*:
assumes $\varphi \in fmla$ $\varphi 1 \in fmla$ $\varphi 2 \in fmla$
and *prv* (*imp* φ (*neg* $\varphi 1$))
shows *prv* (*imp* φ (*neg* (*cnj* $\varphi 1$ $\varphi 2$)))
unfolding *neg_def* **by** (*metis assms cnj fls neg neg_def prv_imp_cnj3L prv_prv_imp_trans*)

lemma *prv_imp_neg_imp_cnjR*:
assumes $\varphi \in fmla$ $\varphi 1 \in fmla$ $\varphi 2 \in fmla$
and *prv* (*imp* φ (*neg* $\varphi 2$))
shows *prv* (*imp* φ (*neg* (*cnj* $\varphi 1$ $\varphi 2$)))
unfolding *neg_def* **by** (*metis assms cnj fls neg neg_def prv_imp_cnj3R prv_prv_imp_trans*)

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* φ)) (*neg* (*exi x* φ)))
proof−
  have *0*: *prv* (*imp* (*all x* (*neg* φ)) (*imp* φ *fls*))
    using *prv_all_inst*[*OF x*, *of neg* φ *Var x*,*simplified*] *assms* **by** (*auto simp*: *neg_def*)
  have *1*: *prv* (*imp* φ (*imp* (*all x* (*neg* φ)) *fls*))
    using *0* **by** (*simp add*: *assms prv_imp_com*)
  have *2*: *prv* (*imp* (*exi x* φ) (*imp* (*all x* (*neg* φ)) *fls*))
    using *1 assms* **by** (*intro prv_exi_imp_gen*) *auto*
  thus *?thesis* **by** (*simp add*: *assms neg_def prv_imp_com*)
qed

lemma *prv_neg_exi_imp_all_neg*:
  assumes *x*: $x \in var$ **and** φ: $\varphi \in fmla$
  shows *prv* (*imp* (*neg* (*exi x* φ)) (*all x* (*neg* φ)))
  using *assms*
  by (*intro prv_all_imp_gen*[*of x neg* (*exi x* φ)])
    (*auto simp*: *prv_exi_inst_same prv_imp_neg_rev*)

lemma *prv_neg_exi_eqv_all_neg*:
  assumes *x*: $x \in var$ **and** φ: $\varphi \in fmla$
  shows *prv* (*eqv* (*neg* (*exi x* φ)) (*all x* (*neg* φ)))
  by (*simp add*: φ *prv_all_neg_imp_neg_exi prv_eqvI prv_neg_exi_imp_all_neg x*)

lemma *prv_neg_exi_implies_all_neg*:
  assumes *x*: $x \in var$ **and** φ: $\varphi \in fmla$ **and** *prv* (*neg* (*exi x* φ))
  shows *prv* (*all x* (*neg* φ))
  by (*meson* φ *all assms*(*3*) *exi neg prv_imp_mp prv_neg_exi_imp_all_neg x*)

lemma *prv_neg_neg_exi*:
  assumes $x \in var$ $\varphi \in fmla$
    and *prv* (*neg* φ)
  shows *prv* (*neg* (*exi x* φ))
  using *assms neg_def prv_exi_imp_gen* **by** *auto*

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* φ) (*exi y* (*subst* φ (*Var y*) *x*)))
proof(*cases* $y = x$)
  case [*simp*]: *False* **hence** [*simp*]: $x \neq y$ **by** *blast*
  show *?thesis* **apply**(*rule prv_exi_imp_gen*)
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    using *yx*
    by (*auto intro*!: *prv_imp_exi prv_exiI*[*of _ _ Var x*]
      simp: *prv_imp_refl2*)
qed(*simp add*: *yx prv_imp_refl*)

lemma *prv_imp_neg_allI*:
  assumes $\varphi \in fmla$ $\chi \in fmla$ $t \in trm$ $x \in var$
    and *prv* (*imp* φ (*neg* (*subst* χ *t x*)))
  shows *prv* (*imp* φ (*neg* (*all x* χ)))
  by (*meson all assms subst neg prv_all_inst prv_imp_neg_rev prv_prv_imp_trans*)

lemma *prv_imp_neg_allWI*:
assumes $\chi \in fmla\ t \in trm\ x \in var$
and $prv\ (neg\ (subst\ \chi\ t\ x))$
shows $prv\ (neg\ (all\ x\ \chi))$
by (*metis all assms fls subst neg_def prv_all_inst prv_prv_imp_trans*)

3.2.3 Properties involving True (tru)

lemma *prv_imp_tru*: $\varphi \in fmla \Longrightarrow prv\ (imp\ \varphi\ tru)$
by (*simp add*: *neg_def prv_imp_triv tru_def*)

lemma *prv_tru_imp*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ (imp\ tru\ \varphi)\ \varphi)$
by (*metis imp neg_def prv_eqvI prv_fls prv_imp_com prv_imp_imp_triv prv_imp_mp prv_imp_refl tru tru_def*)

lemma *prv_fls_neg_tru*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ fls\ (neg\ tru))$
using *neg_def prv_eqvI prv_neg_neg tru_def* **by** *auto*

lemma *prv_tru_neg_fls*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ tru\ (neg\ fls))$
by (*simp add*: *prv_eqv_refl tru_def*)

lemma *prv_cnjR_tru*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ (cnj\ tru\ \varphi)\ \varphi)$
by (*simp add*: *prv_eqvI prv_imp_cnj prv_imp_cnjR prv_imp_refl prv_imp_tru*)

lemma *prv_cnjL_tru*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ (cnj\ \varphi\ tru)\ \varphi)$
by (*simp add*: *prv_eqvI prv_imp_cnj prv_imp_cnjL prv_imp_refl prv_imp_tru*)

3.2.4 Property of set-based conjunctions

These are based on properties of the auxiliary list conjunctions.

lemma *prv_lcnj_imp_in*:
assumes $set\ \varphi s \subseteq fmla$
and $\varphi \in set\ \varphi s$
shows $prv\ (imp\ (lcnj\ \varphi s)\ \varphi)$
proof−
have $\varphi \in fmla$ **using** *assms* **by** *auto*
thus *?thesis* **using** *assms*
by (*induct* φs *arbitrary*: φ)
(*auto simp* : *prv_imp_cnjL prv_cnj_imp_monoR2 prv_imp_triv*)
qed

lemma *prv_lcnj_imp*:
assumes $\chi \in fmla$ **and** $set\ \varphi s \subseteq fmla$
and $\varphi \in set\ \varphi s$ **and** $prv\ (imp\ \varphi\ \chi)$
shows $prv\ (imp\ (lcnj\ \varphi s)\ \chi)$
using *assms lcnj prv_lcnj_imp_in prv_prv_imp_trans* **by** *blast*

lemma *prv_imp_lcnj*:
assumes $\chi \in fmla$ **and** $set\ \varphi s \subseteq fmla$
and $\bigwedge\varphi.\ \varphi \in set\ \varphi s \Longrightarrow prv\ (imp\ \chi\ \varphi)$
shows $prv\ (imp\ \chi\ (lcnj\ \varphi s))$
using *assms*
by (*induct* φs *arbitrary*: χ) (*auto simp*: *prv_imp_tru prv_imp_com prv_imp_cnj*)

lemma *prv_lcnj_imp_inner*:
assumes $\varphi \in fmla\ set\ \varphi 1s \subseteq fmla\ set\ \varphi 2s \subseteq fmla$
shows $prv\ (imp\ (cnj\ \varphi\ (lcnj\ (\varphi 1s\ @\ \varphi 2s)))\ (lcnj\ (\varphi 1s\ @\ \varphi\ \#\ \varphi 2s)))$
using *assms* **proof**(*induction* $\varphi 1s$)

```
  case (Cons φ1 φ1s)
  have [intro!]: set (φ1s @ φ2s) ⊆ fmla set (φ1s @ φ # φ2s) ⊆ fmla  using Cons by auto
  have 0: prv (imp (cnj φ (cnj φ1 (lcnj (φ1s @ φ2s))))
               (cnj φ1 (cnj φ (lcnj (φ1s @ φ2s)))))
    using Cons by (intro prv_cnj_com_imp3) fastforce+
  have 1: prv (imp (cnj φ1 (cnj φ (lcnj (φ1s @ φ2s))))
               (cnj φ1 (lcnj (φ1s @ φ # φ2s))))
    using Cons by (intro prv_cnj_mono) (auto simp add: prv_imp_refl)
  show ?case using prv_prv_imp_trans[OF _ _ _ 0 1] Cons by auto
qed(simp add: prv_imp_refl)

lemma prv_lcnj_imp_remdups:
  assumes set φs ⊆ fmla
  shows prv (imp (lcnj (remdups φs)) (lcnj φs))
  using assms apply(induct φs)
  by (auto simp: prv_imp_cnj prv_lcnj_imp_in prv_cnj_mono prv_imp_refl)

lemma prv_lcnj_mono:
  assumes φ1s: set φ1s ⊆ fmla and set φ2s ⊆ set φ1s
  shows prv (imp (lcnj φ1s) (lcnj φ2s))
proof−
  define φ2s' where φ2s': φ2s' = remdups φ2s
  have A: set φ2s' ⊆ set φ1s distinct φ2s' unfolding  φ2s' using assms by auto
  have B: prv (imp (lcnj φ1s) (lcnj φ2s'))
    using φ1s A proof(induction φ1s arbitrary: φ2s')
    case (Cons φ1 φ1s φ2ss)
    show ?case proof(cases φ1 ∈ set φ2ss)
      case True
      then obtain φ2ss1 φ2ss2 where φ2ss: φ2ss = φ2ss1 @ φ1 # φ2ss2
        by (meson split_list)
      define φ2s where φ2s: φ2s ≡ φ2ss1 @ φ2ss2
      have nin: φ1 ∉ set φ2s using φ2ss ‹distinct φ2ss› unfolding φ2s by auto
      have [intro!]: set φ2s ⊆ fmla unfolding φ2s using φ2ss Cons by auto
      have 0: prv (imp (cnj φ1 (lcnj φ2s)) (lcnj φ2ss))
        unfolding φ2s φ2ss using Cons φ2ss
        by (intro prv_lcnj_imp_inner) auto
      have 1: prv (imp (lcnj φ1s) (lcnj φ2s))
        using nin Cons.prems True φ2s φ2ss by (intro Cons.IH) auto
      have 2: prv (imp (cnj φ1 (lcnj φ1s)) (cnj φ1 (lcnj φ2s)))
        using Cons φ2ss φ2s 1 by (intro prv_cnj_mono) (fastforce simp add: prv_imp_refl)+
      show ?thesis
        using Cons by (auto intro!: prv_prv_imp_trans[OF _ _ _ 2 0])
    next
      case False
      then show ?thesis
        by (meson Cons lcnj prv_imp_lcnj prv_lcnj_imp_in subset_iff)
    qed
  qed(simp add: prv_imp_refl)
  have C: prv (imp (lcnj φ2s') (lcnj φ2s))
    unfolding φ2s' using assms by (intro prv_lcnj_imp_remdups) auto
  show ?thesis using A assms by (intro prv_prv_imp_trans[OF _ _ _ B C]) auto
qed

lemma prv_lcnj_eqv:
  assumes set φ1s ⊆ fmla and set φ2s = set φ1s
  shows prv (eqv (lcnj φ1s) (lcnj φ2s))
  using assms prv_lcnj_mono  by (intro prv_eqvI) auto
```

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*))
using *assms* **apply**(*intro prv_imp_lcnj*)
subgoal by *auto*
subgoal by *auto*
subgoal using *prv_lcnj_imp* **by** *blast* **.**

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*)))
using *assms* **unfolding** *scnj_def* **by** (*fastforce intro*!: *prv_lcnj_eqv*)

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*)))
using *prv_subst_scnj*[*OF assms*] *assms* **by** (*intro prv_imp_eqvEL*) *auto*

lemma *prv_subst_scnj_imp*:
assumes *F ⊆ fmla finite F t ∈ trm x ∈ var*
shows *prv* (*imp* (*scnj* ((λφ. *subst φ t x*) ‘ *F*)) (*subst* (*scnj F*) *t x*))
using *prv_subst_scnj*[*OF assms*] *assms* **by** (*intro prv_imp_eqvER*) *auto*

lemma *prv_scnj_imp_in*:
assumes *F ⊆ fmla finite F*
and *φ ∈ F*
shows *prv* (*imp* (*scnj F*) *φ*)
unfolding *scnj_def* **using** *assms* **by** (*intro prv_lcnj_imp_in*) *auto*

lemma *prv_scnj_imp*:
assumes *χ ∈ fmla* **and** *F ⊆ fmla finite F*
and *φ ∈ F* **and** *prv* (*imp φ χ*)
shows *prv* (*imp* (*scnj F*) *χ*)
unfolding *scnj_def* **using** *assms* **by** (*intro prv_lcnj_imp*) *auto*

lemma *prv_imp_scnj*:
assumes *χ ∈ fmla* **and** *F ⊆ fmla finite F*
and ⋀*φ*. *φ ∈ F* ⟹ *prv* (*imp χ φ*)
shows *prv* (*imp χ* (*scnj F*))
unfolding *scnj_def* **using** *assms* **by** (*intro prv_imp_lcnj*) *auto*

lemma *prv_scnj_mono*:
assumes *F1 ⊆ fmla* **and** *F2 ⊆ F1* **and** *finite F1*
shows *prv* (*imp* (*scnj F1*) (*scnj F2*))
unfolding *scnj_def* **using** *assms* **apply** (*intro prv_lcnj_mono*)
subgoal by *auto*
subgoal by (*metis asList infinite_super*) **.**

lemma *prv_scnj_mono_imp*:
assumes *F1 ⊆ fmla F2 ⊆ fmla finite F1 finite F2*
and ∀ *φ2 ∈ F2*. ∃ *φ1 ∈ F1*. *prv* (*imp φ1 φ2*)
shows *prv* (*imp* (*scnj F1*) (*scnj F2*))
unfolding *scnj_def* **using** *assms* **by** (*intro prv_lcnj_mono_imp*) *auto*

Commutation with parallel substitution:

lemma *prv_imp_scnj_insert*:
assumes *F ⊆ fmla* **and** *finite F* **and** *φ ∈ fmla*

shows *prv* (*imp* (*scnj* (*insert* φ *F*)) (*cnj* φ (*scnj F*)))
using *assms* **apply** (*intro prv_imp_cnj*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto intro*: *prv_imp_refl prv_scnj_imp*)
subgoal by (*auto intro*: *prv_scnj_mono*) **.**

lemma *prv_implies_scnj_insert*:
assumes *F* $\subseteq$ *fmla* **and** *finite F* **and** $\varphi \in$ *fmla*
and *prv* (*scnj* (*insert* φ *F*))
shows *prv* (*cnj* φ (*scnj F*))
by (*meson assms cnj finite.insertI insert_subset prv_imp_mp prv_imp_scnj_insert scnj*)

lemma *prv_imp_cnj_scnj*:
assumes *F* $\subseteq$ *fmla* **and** *finite F* **and** $\varphi \in$ *fmla*
shows *prv* (*imp* (*cnj* φ (*scnj F*)) (*scnj* (*insert* φ *F*)))
using *assms*
by (*auto intro*!: *prv_imp_scnj prv_imp_cnjL*
simp: *prv_cnj_imp_monoR2 prv_imp_triv prv_scnj_imp_in subset_iff*)

lemma *prv_implies_cnj_scnj*:
assumes *F* $\subseteq$ *fmla* **and** *finite F* **and** $\varphi \in$ *fmla*
and *prv* (*cnj* φ (*scnj F*))
shows *prv* (*scnj* (*insert* φ *F*))
by (*meson assms cnj finite.insertI insert_subset prv_imp_cnj_scnj prv_imp_mp scnj*)

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*)))
by (*simp add*: *assms prv_eqvI prv_imp_cnj_scnj prv_imp_scnj_insert*)

lemma *prv_scnj1_imp*:
$\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* (*scnj* {φ}) φ)
using *prv_scnj_imp_in* **by** *auto*

lemma *prv_imp_scnj1*:
$\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* φ (*scnj* {φ}))
using *prv_imp_refl prv_imp_scnj* **by** *fastforce*

lemma *prv_scnj2_imp_cnj*:
$\varphi \in$ *fmla* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* (*scnj* {φ,ψ}) (*cnj* φ ψ))
using *prv_imp_cnj prv_scnj_imp_in* **by** *auto*

lemma *prv_cnj_imp_scnj2*:
$\varphi \in$ *fmla* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* (*cnj* φ ψ) (*scnj* {φ,ψ}))
using *prv_imp_cnjL prv_imp_cnjR prv_imp_scnj* **by** *fastforce*

lemma *prv_imp_imp_scnj2*:
$\varphi \in$ *fmla* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* φ (*imp* ψ (*scnj* {φ,ψ})))
using *prv_cnj_imp_scnj2 prv_cnj_imp* **by** *auto*

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*)))

```
  using assms proof(induction txs arbitrary: F)
  case (Cons tx txs)
  then obtain t x where tx[simp]: tx = (t,x) by (cases tx) auto
  hence [simp]: t ∈ trm x ∈ var using Cons.prems by auto
  have 0: (λφ. rawpsubst (subst φ t x) txs) ‘ F =
        (λφ. rawpsubst φ txs) ‘ ((λφ. subst φ t x) ‘ F)
    using Cons.prems by auto
  have prv (eqv (subst (scnj F) t x)
              (scnj ((λφ. subst φ t x) ‘ F)))
    using Cons.prems by (intro prv_subst_scnj) auto
  hence prv (eqv (rawpsubst (subst (scnj F) t x) txs)
               (rawpsubst (scnj ((λφ. subst φ t x) ‘ F)) txs))
    using Cons.prems by (intro prv_eqv_rawpsubst) auto
  moreover
  have prv (eqv (rawpsubst (scnj ((λφ. subst φ t x) ‘ F)) txs)
              (scnj ((λφ. rawpsubst (subst φ t x) txs) ‘ F)))
    unfolding 0 using Cons.prems by (intro Cons.IH) auto
  ultimately show ?case
    using Cons.prems apply − by (rule prv_eqv_trans) (auto intro!: rawpsubst)
qed(auto simp: image_def prv_eqv_refl)[]

lemma prv_psubst_scnj:
  assumes F ⊆ fmla finite F
    and snd ‘ (set txs) ⊆ var fst ‘ (set txs) ⊆ trm
    and distinct (map snd txs)
  shows prv (eqv (psubst (scnj F) txs) (scnj ((λφ. psubst φ txs) ‘ F)))
proof−
  define us where us: us ≡ getFrN (map snd txs) (map fst txs) [scnj F] (length txs)
  have us_facts: set us ⊆ var
    set us ∩ ⋃ (Fvars ‘ F) = {}
    set us ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
    set us ∩ snd ‘ (set txs) = {}
    length us = length txs
    distinct us
    using assms unfolding us
    using getFrN_Fvars[of map snd txs map fst txs [scnj F] _ length txs]
      getFrN_FvarsT[of map snd txs map fst txs [scnj F] _ length txs]
      getFrN_var[of map snd txs map fst txs [scnj F] _ length txs]
      getFrN_length[of map snd txs map fst txs [scnj F] length txs]
        apply −
    subgoal by auto
    subgoal by fastforce
    subgoal by (fastforce simp: image_iff)
    subgoal by (fastforce simp: image_iff)
    subgoal by (fastforce simp: image_iff)
    by auto

  define vs where vs: vs ≡ λ φ. getFrN (map snd txs) (map fst txs) [φ] (length txs)
  hence vss: ⋀φ. vs φ = getFrN (map snd txs) (map fst txs) [φ] (length txs) by auto
  {fix φ assume φ ∈ F hence φ ∈ fmla using assms by auto
    hence set (vs φ) ⊆ var ∧
    set (vs φ) ∩ Fvars φ = {} ∧
    set (vs φ) ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {} ∧
    set (vs φ) ∩ snd ‘ (set txs) = {} ∧
    length (vs φ) = length txs ∧
    distinct (vs φ)
      using assms unfolding vs
      using getFrN_Fvars[of map snd txs map fst txs [φ] _ length txs]
```

```
    getFrN_FvarsT[of map snd txs map fst txs [φ] _ length txs]
    getFrN_var[of map snd txs map fst txs [φ] _ length txs]
    getFrN_length[of map snd txs map fst txs [φ] length txs]
  apply (intro conjI)
  subgoal by auto
  subgoal by auto
  subgoal by fastforce
  subgoal by (fastforce simp: image_iff)
  by auto
} note vs_facts = this

have [simp]: ⋀ x f. f ∈ F ⟹ x ∈ set (vs f) ⟹ x ∈ var
  using vs_facts
  by (meson subsetD)

let ?tus = zip (map fst txs) us
let ?uxs = zip (map Var us) (map snd txs)
let ?tvs = λ φ. zip (map fst txs) (vs φ)
let ?vxs = λ φ. zip (map Var (vs φ)) (map snd txs)

let ?c = rawpsubst (scnj F) ?uxs
have c: prv (eqv ?c
                (scnj ((λφ. rawpsubst φ ?uxs) ` F)))
  using assms us_facts by (intro prv_rawpsubst_scnj) (auto intro!: rawpsubstT dest!: set_zip_D)
hence prv (eqv (rawpsubst ?c ?tus)
                (rawpsubst (scnj ((λφ. rawpsubst φ ?uxs) ` F)) ?tus))
  using assms us_facts
  by (intro prv_eqv_rawpsubst) (auto intro!: rawpsubst dest!: set_zip_D)
moreover
have prv (eqv (rawpsubst (scnj ((λφ. rawpsubst φ ?uxs) ` F)) ?tus)
              (scnj ((λφ. rawpsubst φ ?tus) ` ((λφ. rawpsubst φ ?uxs) ` F))))
  using assms us_facts
  by (intro prv_rawpsubst_scnj) (auto intro!: rawpsubst dest!: set_zip_D)
ultimately
have 0: prv (eqv (rawpsubst ?c ?tus)
                  (scnj ((λφ. rawpsubst φ ?tus) ` ((λφ. rawpsubst φ ?uxs) ` F))))
  using assms us_facts apply − by (rule prv_eqv_trans) (auto intro!: rawpsubst dest!: set_zip_D)
moreover
have prv (eqv (scnj ((λφ. rawpsubst φ ?tus) ` ((λφ. rawpsubst φ ?uxs) ` F)))
              (scnj ((λφ. rawpsubst (rawpsubst φ (?vxs φ)) (?tvs φ)) ` F)))
  using assms us_facts vs_facts apply(intro prv_eqvI)
  subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
  subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
  subgoal apply(rule prv_scnj_mono_imp)
    subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
    subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
    subgoal by auto
    subgoal by auto
    subgoal apply clarsimp
      subgoal for φ apply(rule bexI[of _ φ]) apply(rule prv_imp_refl2)
        subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
        subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
        subgoal by (rule rawpsubst_compose_freshVar2)
            (auto intro!: rawpsubst dest!: set_zip_D) . . .
  subgoal apply(rule prv_scnj_mono_imp)
    subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
    subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
    subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
```

subgoal by (*auto intro*!: *rawpsubst dest*!: *set_zip_D*)
subgoal apply *clarsimp*
subgoal for φ **apply**(*rule bexI*[*of* _ φ]) **apply**(*rule prv_imp_refl2*)
apply (*auto intro*!: *rawpsubst dest*!: *set_zip_D*)
apply(*rule rawpsubst_compose_freshVar2*)
apply (*auto intro*!: *rawpsubst dest*!: *set_zip_D*) . . . .
ultimately
have *prv* (*eqv* (*rawpsubst* (*rawpsubst* (*scnj F*) *?uxs*) *?tus*)
(*scnj* ((λφ. *rawpsubst* (*rawpsubst* φ (*?vxs* φ)) (*?tvs* φ)) ‘ *F*)))
using *assms us_facts*
apply − **by** (*rule prv_eqv_trans*) (*auto intro*!: *rawpsubst dest*!: *set_zip_D*)
thus *?thesis* **unfolding** *psubst_def* **by** (*simp add*: *Let_def us*[*symmetric*] *vss*)
qed

lemma *prv_imp_psubst_scnj*:
assumes *F* ⊆ *fmla finite F snd* ‘ *set txs* ⊆ *var fst* ‘ *set txs* ⊆ *trm*
and *distinct* (*map snd txs*)
shows *prv* (*imp* (*psubst* (*scnj F*) *txs*) (*scnj* ((λφ. *psubst* φ *txs*) ‘ *F*)))
using *prv_psubst_scnj*[*OF assms*] *assms* **apply**(*intro prv_imp_eqvEL*) **by** *auto*

lemma *prv_psubst_scnj_imp*:
assumes *F* ⊆ *fmla finite F snd* ‘ *set txs* ⊆ *var fst* ‘ *set txs* ⊆ *trm*
and *distinct* (*map snd txs*)
shows *prv* (*imp* (*scnj* ((λφ. *psubst* φ *txs*) ‘ *F*)) (*psubst* (*scnj F*) *txs*))
using *prv_psubst_scnj*[*OF assms*] *assms* **apply**(*intro prv_imp_eqvER*) **by** *auto*

3.2.5 Consistency and ω-consistency

definition *consistent* :: *bool* **where**
consistent ≡ ¬ *prv fls*

lemma *consistent_def2*: *consistent* ⟷ (∃ φ ∈ *fmla*. ¬ *prv* φ)
unfolding *consistent_def* **using** *prv_expl* **by** *blast*

lemma *consistent_def3*: *consistent* ⟷ (∀ φ ∈ *fmla*. ¬ (*prv* φ ∧ *prv* (*neg* φ)))
unfolding *consistent_def* **using** *prv_neg_fls neg_def* **by** *auto*

definition ω*consistent* :: *bool* **where**
ω*consistent* ≡
∀ φ ∈ *fmla*. ∀ *x* ∈ *var*. *Fvars* φ = {*x*} ⟶
(∀ *n* ∈ *num*. *prv* (*neg* (*subst* φ *n x*)))
⟶
¬ *prv* (*neg* (*neg* (*exi x* φ)))

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* ≡
∀ φ ∈ *fmla*. ∀ *x* ∈ *var*. *Fvars* φ = {*x*} ⟶
(∀ *n* ∈ *num*. *prv* (*neg* (*subst* φ *n x*))) ⟶ ¬ *prv* (*exi x* φ)

definition ω*consistentStd2* :: *bool* **where**
ω*consistentStd2* ≡
∀ φ ∈ *fmla*. ∀ *x* ∈ *var*. *Fvars* φ = {*x*} ⟶
(∀ *n* ∈ *num*. *prv* (*subst* φ *n x*)) ⟶ ¬ *prv* (*exi x* (*neg* φ))

lemma ω*consistent_impliesStd1*:

```
 ωconsistent ⟹
ωconsistentStd1
 unfolding ωconsistent_def ωconsistentStd1_def using prv_neg_neg by blast

lemma ωconsistent_impliesStd2:
 ωconsistent ⟹
ωconsistentStd2
 by (auto dest!: ωconsistent_impliesStd1 bspec[of _ _ neg _]
     simp: ωconsistentStd1_def ωconsistentStd2_def prv_neg_neg)
```

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

```
lemma ωconsistent_iffStd1:
 assumes ⋀ φ. φ ∈ fmla ⟹ prv (imp (neg (neg φ)) φ)
 shows ωconsistent ⟷ ωconsistentStd1
 apply standard
 subgoal using ωconsistent_impliesStd1 by auto
 subgoal unfolding ωconsistentStd1_def ωconsistent_def
   by (meson assms exi neg prv_imp_mp) .

lemma ωconsistent_iffStd2:
 assumes ⋀ φ. φ ∈ fmla ⟹ prv (imp (neg (neg φ)) φ)
 shows ωconsistent ⟷ ωconsistentStd2
 unfolding ωconsistent_iffStd1[OF assms, simplified]
  ωconsistentStd1_def ωconsistentStd2_def apply safe
 subgoal for φ x
   by (auto simp: prv_neg_neg dest: bspec[of _ _ neg _])
 subgoal for φ x
   using  prv_exi_congW prv_imp_neg_fls
   by (auto simp: neg_def prv_neg_neg dest!: bspec[of _ _ neg _]) .
```

ω-consistency implies consistency:

```
lemma ωconsistentStd1_implies_consistent:
 assumes ωconsistentStd1
 shows consistent
 unfolding consistent_def
proof safe
 assume pf: prv fls
 then obtain x where x: x ∈ var x ∉ Fvars fls
   using finite_Fvars getFresh by auto
 let ?fls = cnj (fls) (eql (Var x) (Var x))
 have 0: ∀ n ∈ num. prv (neg (subst ?fls n x)) and 1: prv (exi x fls)
   using x fls by (auto simp: pf prv_expl)
 have 2: ¬ prv (exi x ?fls) using 0 fls x assms
   unfolding ωconsistentStd1_def by simp
 show False using 1 2 consistent_def consistent_def2 pf x(1) by blast
qed

lemma ωconsistentStd2_implies_consistent:
 assumes ωconsistentStd2
 shows consistent
 unfolding consistent_def
proof safe
 assume pf: prv fls
 then obtain x where x: x ∈ var x ∉ Fvars fls
   using finite_Fvars getFresh by auto
 let ?fls = cnj (fls) (eql (Var x) (Var x))
 have 0: ∀ n ∈ num. prv (subst ?fls n x) and 1: prv (exi x (neg ?fls))
```

using *x fls* **by** (*auto simp*: *pf prv_expl*)
have *2*: ¬ *prv* (*exi x* (*neg ?fls*)) **using** *0 fls x assms*
unfolding *ωconsistentStd2_def* **by** *auto*
show *False* **using** *1 2* **by** *simp*
qed

corollary *ωconsistent_implies_consistent*:
assumes *ωconsistent*
shows *consistent*
by (*simp add*: *ωconsistentStd2_implies_consistent ωconsistent_impliesStd2 assms*)

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)
by (*metis assms dsj imp prv_imp_dsjE prv_imp_mp*)

lemma *prv_dsj_cases*:

assumes $\varphi 1 \in fmla\ \varphi 2 \in fmla\ \chi \in fmla$
and $prv\ (dsj\ \varphi 1\ \varphi 2)$ **and** $prv\ (imp\ \varphi 1\ \chi)$ **and** $prv\ (imp\ \varphi 2\ \chi)$
shows $prv\ \chi$
by (*meson assms dsj prv_imp_dsjEE prv_imp_mp*)

3.3.1 Disjunction vs. disjunction

lemma *prv_dsj_com_imp*:
assumes $\varphi[simp]: \varphi \in fmla$ **and** $\chi[simp]: \chi \in fmla$
shows $prv\ (imp\ (dsj\ \varphi\ \chi)\ (dsj\ \chi\ \varphi))$
by (*metis* $\chi\ \varphi$ *dsj imp prv_dsj_impL prv_dsj_impR prv_imp_dsjE prv_imp_mp*)

lemma *prv_dsj_com*:
assumes $\varphi[simp]: \varphi \in fmla$ **and** $\chi[simp]: \chi \in fmla$
shows $prv\ (eqv\ (dsj\ \varphi\ \chi)\ (dsj\ \chi\ \varphi))$
by (*simp add*: *prv_dsj_com_imp prv_eqvI*)

lemma *prv_dsj_assoc_imp1*:
assumes $\varphi[simp]: \varphi \in fmla$ **and** $\chi[simp]: \chi \in fmla$ **and** $\psi[simp]: \psi \in fmla$
shows $prv\ (imp\ (dsj\ \varphi\ (dsj\ \chi\ \psi))\ (dsj\ (dsj\ \varphi\ \chi)\ \psi))$
proof −
have *f1*: $\bigwedge f\ fa\ fb.\ f \notin fmla \vee \neg\ prv\ (imp\ fa\ fb) \vee fb \notin fmla \vee fa \notin fmla \vee prv\ (imp\ fa\ (dsj\ fb\ f))$
by (*meson dsj prv_dsj_impL prv_prv_imp_trans*)
have $prv\ (imp\ \varphi\ (dsj\ \varphi\ \chi))$
by (*simp add*: *prv_dsj_impL*)
then show *?thesis*
using *f1* $\chi\ \varphi\ \psi$ *dsj prv_dsj_impR prv_imp_dsjEE* **by** *presburger*
qed

lemma *prv_dsj_assoc_imp2*:
assumes $\varphi[simp]: \varphi \in fmla$ **and** $\chi[simp]: \chi \in fmla$ **and** $\psi[simp]: \psi \in fmla$
shows $prv\ (imp\ (dsj\ (dsj\ \varphi\ \chi)\ \psi)\ (dsj\ \varphi\ (dsj\ \chi\ \psi)))$
proof −
have *f1*: $\forall f\ fa\ fb.\ (((prv\ (imp\ f\ (dsj\ fa\ fb)) \vee \neg\ prv\ (imp\ f\ (dsj\ fb\ fa))) \vee f \notin fmla) \vee fa \notin fmla) \vee fb \notin fmla$
by (*meson dsj prv_dsj_com_imp prv_prv_imp_trans*)
have $\forall f\ fa\ fb.\ (((prv\ (imp\ f\ (dsj\ fa\ fb)) \vee \neg\ prv\ (imp\ f\ fa)) \vee fa \notin fmla) \vee f \notin fmla) \vee fb \notin fmla$
by (*meson dsj prv_dsj_impL prv_prv_imp_trans*)
then show *?thesis*
using *f1* **by** (*metis* $\chi\ \varphi\ \psi$ *dsj prv_dsj_impR prv_imp_dsjEE*)
qed

lemma *prv_dsj_assoc*:
assumes $\varphi[simp]: \varphi \in fmla$ **and** $\chi[simp]: \chi \in fmla$ **and** $\psi[simp]: \psi \in fmla$
shows $prv\ (eqv\ (dsj\ \varphi\ (dsj\ \chi\ \psi))\ (dsj\ (dsj\ \varphi\ \chi)\ \psi))$
by (*simp add*: *prv_dsj_assoc_imp1 prv_dsj_assoc_imp2 prv_eqvI*)

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 −
have $\forall f\ fa\ fb.\ (((prv\ (imp\ f\ (dsj\ fb\ fa)) \vee \neg\ prv\ (imp\ f\ fa)) \vee fa \notin fmla) \vee f \notin fmla) \vee fb \notin fmla$
by (*meson dsj prv_dsj_impR prv_prv_imp_trans*)
then show *?thesis*
by (*meson assms*(*1*) *assms*(*2*) *assms*(*3*) *dsj prv_dsj_impL prv_dsj_impR prv_imp_dsjEE*)
qed

lemma *prv_dsj_mono*:
$\varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \chi 1 \in fmla \Longrightarrow \chi 2 \in fmla \Longrightarrow$
$prv\ (imp\ \varphi 1\ \chi 1) \Longrightarrow prv\ (imp\ \varphi 2\ \chi 2) \Longrightarrow prv\ (imp\ (dsj\ \varphi 1\ \varphi 2)\ (dsj\ \chi 1\ \chi 2))$
by (*meson dsj prv_dsj_impL prv_dsj_impR prv_imp_dsjEE prv_prv_imp_trans*)

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 $prv\ (imp\ (cnj\ \varphi\ (dsj\ \chi 1\ \chi 2))\ (dsj\ (cnj\ \varphi\ \chi 1)\ (cnj\ \varphi\ \chi 2)))$
by (*simp add*: *prv_cnj_imp prv_cnj_imp_monoR2 prv_dsj_impL prv_dsj_impR prv_imp_com prv_imp_dsjEE*)

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 $prv\ (imp\ (dsj\ (cnj\ \varphi\ \chi 1)\ (cnj\ \varphi\ \chi 2))\ (cnj\ \varphi\ (dsj\ \chi 1\ \chi 2)))$
by (*simp add*: *prv_cnj_mono prv_dsj_impL prv_dsj_impR prv_imp_dsjEE prv_imp_refl*)

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 $prv\ (eqv\ (cnj\ \varphi\ (dsj\ \chi 1\ \chi 2))\ (dsj\ (cnj\ \varphi\ \chi 1)\ (cnj\ \varphi\ \chi 2)))$
by (*simp add*: *prv_cnj_dsj_distrib1 prv_cnj_dsj_distrib2 prv_eqvI*)

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 $prv\ (imp\ (dsj\ \varphi\ (cnj\ \chi 1\ \chi 2))\ (cnj\ (dsj\ \varphi\ \chi 1)\ (dsj\ \varphi\ \chi 2)))$
by (*simp add*: *prv_cnj_mono prv_dsj_impL prv_dsj_impR prv_imp_cnj prv_imp_dsjEE*)

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 $prv\ (imp\ (cnj\ (dsj\ \varphi\ \chi 1)\ (dsj\ \varphi\ \chi 2))\ (dsj\ \varphi\ (cnj\ \chi 1\ \chi 2)))$
proof −
have $\forall f\ fa\ fb.\ (((prv\ (imp\ f\ (imp\ fb\ fa)) \vee \neg\ prv\ (imp\ f\ fa)) \vee fa \notin fmla) \vee f \notin fmla) \vee fb \notin fmla$
by (*meson imp prv_imp_imp_triv prv_prv_imp_trans*)
then show *?thesis*
by (*metis* $\chi 1$ $\chi 2$ φ *cnj dsj imp prv_cnj_imp prv_cnj_imp_monoR2 prv_dsj_impL prv_dsj_impR prv_imp_com prv_imp_dsjEE*)
qed

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 $prv\ (eqv\ (dsj\ \varphi\ (cnj\ \chi 1\ \chi 2))\ (cnj\ (dsj\ \varphi\ \chi 1)\ (dsj\ \varphi\ \chi 2)))$
by (*simp add*: *prv_dsj_cnj_distrib1 prv_dsj_cnj_distrib2 prv_eqvI*)

3.3.3 Disjunction vs. True and False

lemma *prv_dsjR_fls*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ (dsj\ fls\ \varphi)\ \varphi)$
by (*simp add*: *prv_dsj_impR prv_eqvI prv_imp_dsjEE prv_imp_refl*)

lemma *prv_dsjL_fls*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ (dsj\ \varphi\ fls)\ \varphi)$
by (*simp add*: *prv_dsj_impL prv_eqvI prv_imp_dsjEE prv_imp_refl*)

lemma *prv_dsjR_tru*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ (dsj\ tru\ \varphi)\ tru)$
by (*simp add*: *prv_dsj_impL prv_eqvI prv_imp_tru*)

lemma *prv_dsjL_tru*: $\varphi \in fmla \Longrightarrow prv\ (eqv\ (dsj\ \varphi\ tru)\ tru)$
by (*simp add*: *prv_dsj_impR prv_eqvI prv_imp_tru*)

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 φs ⊆ fmla*
and *φ ∈ set φs*
shows *prv (imp φ (ldsj φs))*
proof−
have *φ ∈ fmla* **using** *assms* **by** *auto*
thus *?thesis*
using *assms* **apply**(*induct φs arbitrary*: *φ*)
subgoal by *auto*
subgoal by (*simp add*: *prv_dsj_impL*)
(*meson dsj ldsj prv_dsj_impL prv_dsj_impR prv_prv_imp_trans*) **.**
qed

lemma *prv_imp_ldsj*:
assumes *χ ∈ fmla* **and** *set φs ⊆ fmla*
and *φ ∈ set φs* **and** *prv (imp χ φ)*
shows *prv (imp χ (ldsj φs))*
using *assms ldsj prv_imp_ldsj_in prv_prv_imp_trans* **by** *blast*

lemma *prv_ldsj_imp*:
assumes *χ ∈ fmla* **and** *set φs ⊆ fmla*
and *⋀φ. φ ∈ set φs ⟹ prv (imp φ χ)*
shows *prv (imp (ldsj φs) χ)*
using *assms*
by (*induct φs arbitrary*: *χ*)
(*auto simp add*: *prv_imp_tru prv_imp_com prv_imp_dsjEE*)

lemma *prv_ldsj_imp_inner*:
assumes *φ ∈ fmla set φ1s ⊆ fmla set φ2s ⊆ fmla*
shows *prv (imp (ldsj (φ1s @ φ # φ2s)) (dsj φ (ldsj (φ1s @ φ2s))))*
using *assms* **proof**(*induction φ1s*)
case (*Cons φ1 φ1s*)
have [*intro*!]: *set (φ1s @ φ2s) ⊆ fmla set (φ1s @ φ # φ2s) ⊆ fmla* **using** *Cons* **by** *auto*
have *0*: *prv (imp (dsj φ1 (dsj φ (ldsj (φ1s @ φ2s))))*
(dsj φ (dsj φ1 (ldsj (φ1s @ φ2s)))))
using *Cons* **by** (*intro prv_dsj_com_imp3*) *fastforce*+
have *1*: *prv (imp (dsj φ1 (ldsj (φ1s @ φ # φ2s)))*
(dsj φ1 (dsj φ (ldsj (φ1s @ φ2s)))))
using *Cons* **by** (*intro prv_dsj_mono*) (*auto simp add*: *prv_imp_refl*)
show *?case* **using** *prv_prv_imp_trans*[*OF _ _ _ 1 0*] *Cons* **by** *auto*
qed(*simp add*: *prv_imp_refl*)

lemma *prv_ldsj_imp_remdups*:
assumes *set φs ⊆ fmla*
shows *prv (imp (ldsj φs) (ldsj (remdups φs)))*
using *assms* **apply**(*induct φs*)
subgoal by *auto*
subgoal by (*metis ldsj prv_imp_ldsj_in prv_ldsj_imp set_remdups*) **.**

lemma *prv_ldsj_mono*:
assumes *φ2s*: *set φ2s ⊆ fmla* **and** *set φ1s ⊆ set φ2s*
shows *prv (imp (ldsj φ1s) (ldsj φ2s))*
proof−
define *φ1s′* **where** *φ1s′*: *φ1s′ = remdups φ1s*
have *A*: *set φ1s′ ⊆ set φ2s distinct φ1s′* **unfolding** *φ1s′* **using** *assms* **by** *auto*

```
  have B: prv (imp (ldsj φ1s') (ldsj φ2s))
  using φ2s A proof(induction φ2s arbitrary: φ1s')
    case (Cons φ2 φ2s φ1ss)
    show ?case proof(cases φ2 ∈ set φ1ss)
      case True
      then obtain φ1ss1 φ1ss2 where φ1ss: φ1ss = φ1ss1 @ φ2 # φ1ss2
      by (meson split_list)
      define φ1s where φ1s: φ1s ≡ φ1ss1 @ φ1ss2
      have nin: φ2 ∉ set φ1s using φ1ss ‹distinct φ1ss› unfolding φ1s by auto
      have [intro!,simp]: set φ1s ⊆ fmla unfolding φ1s using φ1ss Cons by auto
      have 0: prv (imp (ldsj φ1ss) (dsj φ2 (ldsj φ1s)))
        unfolding φ1s φ1ss
        apply(rule prv_ldsj_imp_inner) using Cons φ1ss by auto
      have 1: prv (imp (ldsj φ1s) (ldsj φ2s)) apply(rule Cons.IH) using nin Cons.prems True
      using φ1s φ1ss by  auto
      have 2: prv (imp (dsj φ2 (ldsj φ1s)) (dsj φ2 (ldsj φ2s)))
      using Cons φ1ss φ1s 1  apply(intro prv_dsj_mono)
      using prv_imp_refl by auto blast+
      show ?thesis using Cons by (auto intro!: prv_prv_imp_trans[OF _ _ _ 0 2])
    next
      case False
      then show ?thesis using Cons
      by (meson ldsj order.trans prv_imp_ldsj_in prv_ldsj_imp subset_code(1))
    qed
  qed(simp add: prv_imp_refl)
  have C: prv (imp (ldsj φ1s) (ldsj φ1s'))
    unfolding φ1s' using assms by (intro prv_ldsj_imp_remdups) auto
  show ?thesis  using A assms by (intro prv_prv_imp_trans[OF _ _ _ C B]) auto
qed

lemma prv_ldsj_eqv:
assumes set φ1s ⊆ fmla and set φ2s = set φ1s
shows prv (eqv (ldsj φ1s) (ldsj φ2s))
  using assms prv_ldsj_mono by (intro prv_eqvI) auto

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))
  using assms apply(intro prv_ldsj_imp)
  subgoal by auto
  subgoal by auto
  subgoal using prv_imp_ldsj by blast .
```

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 ((λφ. subst φ t x) ‘ F)))
unfolding sdsj_def by (fastforce intro!: prv_ldsj_eqv)

lemma prv_imp_sdsj_in:
  assumes φ ∈ fmla and F ⊆ fmla finite F
  and φ ∈ F
  shows prv (imp φ (sdsj F))
  unfolding sdsj_def using assms by (intro prv_imp_ldsj_in) auto

lemma prv_imp_sdsj:
assumes χ ∈ fmla and F ⊆ fmla finite F
```

and $\varphi \in F$ **and** *prv* (*imp* χ φ)
shows *prv* (*imp* χ (*sdsj F*))
unfolding *sdsj_def* **using** *assms* **by** (*intro prv_imp_ldsj*) *auto*

lemma *prv_sdsj_imp*:
assumes $\chi \in$ *fmla* **and** $F \subseteq$ *fmla finite F*
and $\bigwedge\varphi.\ \varphi \in F \Longrightarrow$ *prv* (*imp* φ χ)
shows *prv* (*imp* (*sdsj F*) χ)
unfolding *sdsj_def* **using** *assms* **by** (*intro prv_ldsj_imp*) *auto*

lemma *prv_sdsj_mono*:
assumes *F2* $\subseteq$ *fmla* **and** *F1* $\subseteq$ *F2* **and** *finite F2*
shows *prv* (*imp* (*sdsj F1*) (*sdsj F2*))
unfolding *sdsj_def* **using** *assms* **apply**(*intro prv_ldsj_mono*)
subgoal by *auto*
subgoal by (*metis asList infinite_super*) **.**

lemma *prv_sdsj_mono_imp*:
assumes *F1* $\subseteq$ *fmla F2* $\subseteq$ *fmla finite F1 finite F2*
and $\forall$ *φ1* $\in$ *F1*. $\exists$ *φ2* $\in$ *F2*. *prv* (*imp φ1 φ2*)
shows *prv* (*imp* (*sdsj F1*) (*sdsj F2*))
unfolding *sdsj_def* **using** *assms* **by** (*intro prv_ldsj_mono_imp*) *auto*

lemma *prv_sdsj_cases*:
assumes $F \subseteq$ *fmla finite F* $\psi \in$ *fmla*
and *prv* (*sdsj F*) **and** $\bigwedge\varphi.\ \varphi \in F \Longrightarrow$ *prv* (*imp* φ ψ)
shows *prv* ψ
by (*meson assms prv_imp_mp prv_sdsj_imp sdsj*)

lemma *prv_sdsj1_imp*:
$\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* (*sdsj* {φ}) φ)
using *prv_imp_refl prv_sdsj_imp* **by** *fastforce*

lemma *prv_imp_sdsj1*:
$\varphi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* φ (*sdsj* {φ}))
using *prv_imp_refl prv_imp_sdsj* **by** *fastforce*

lemma *prv_sdsj2_imp_dsj*:
$\varphi \in$ *fmla* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* (*sdsj* {φ,ψ}) (*dsj* φ ψ))
using *prv_dsj_impL prv_dsj_impR prv_sdsj_imp* **by** *fastforce*

lemma *prv_dsj_imp_sdsj2*:
$\varphi \in$ *fmla* $\Longrightarrow$ $\psi \in$ *fmla* $\Longrightarrow$ *prv* (*imp* (*dsj* φ ψ) (*sdsj* {φ,ψ}))
by (*simp add*: *prv_imp_dsjEE prv_imp_sdsj_in*)

Commutation with parallel substitution:

lemma *prv_rawpsubst_sdsj*:
assumes $F \subseteq$ *fmla finite F*
and *snd* ‘ (*set txs*) $\subseteq$ *var fst* ‘ (*set txs*) $\subseteq$ *trm*
shows *prv* (*eqv* (*rawpsubst* (*sdsj F*) *txs*) (*sdsj* ((λφ. *rawpsubst* φ *txs*) ‘ *F*)))
using *assms* **proof**(*induction txs arbitrary*: *F*)
case (*Cons tx txs*)
then obtain *t x* **where** *tx*[*simp*]: *tx* = (*t*,*x*) **by** (*cases tx*) *auto*
hence [*simp*]: *t* $\in$ *trm x* $\in$ *var* **using** *Cons.prems* **by** *auto*
have *0*: (λφ. *rawpsubst* (*subst* φ *t x*) *txs*) ‘ *F* =
(λφ. *rawpsubst* φ *txs*) ‘ ((λφ. *subst* φ *t x*) ‘ *F*)
using *Cons.prems* **by** *auto*
have *prv* (*eqv* (*subst* (*sdsj F*) *t x*)

```
        (sdsj ((λφ. subst φ t x) ‘ F)))
  using Cons.prems by (intro prv_subst_sdsj) auto
  hence prv (eqv (rawpsubst (subst (sdsj F) t x) txs)
          (rawpsubst (sdsj ((λφ. subst φ t x) ‘ F)) txs))
  using Cons.prems by (intro prv_eqv_rawpsubst) auto
  moreover
  have prv (eqv (rawpsubst (sdsj ((λφ. subst φ t x) ‘ F)) txs)
          (sdsj ((λφ. rawpsubst (subst φ t x) txs) ‘ F)))
  unfolding 0 using Cons.prems by (intro Cons.IH) auto
  ultimately show ?case
  using Cons.prems apply − by (rule prv_eqv_trans) (auto intro!: rawpsubst)
qed(auto simp: image_def prv_eqv_refl)[]

lemma prv_psubst_sdsj:
assumes F ⊆ fmla finite F
and snd ‘ (set txs) ⊆ var fst ‘ (set txs) ⊆ trm
and distinct (map snd txs)
shows prv (eqv (psubst (sdsj F) txs) (sdsj ((λφ. psubst φ txs) ‘ F)))
proof−
  define us where us: us ≡ getFrN (map snd txs) (map fst txs) [sdsj F] (length txs)
  have us_facts: set us ⊆ var
  set us ∩ ⋃ (Fvars ‘ F) = {}
  set us ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
  set us ∩ snd ‘ (set txs) = {}
  length us = length txs
  distinct us
  using assms unfolding us
  using getFrN_Fvars[of map snd txs map fst txs [sdsj F] _ length txs]
      getFrN_FvarsT[of map snd txs map fst txs [sdsj F] _ length txs]
      getFrN_var[of map snd txs map fst txs [sdsj F] _ length txs]
      getFrN_length[of map snd txs map fst txs [sdsj F] length txs]
  apply −
   subgoal by auto
   subgoal by fastforce
   subgoal by (fastforce simp: image_iff)
   subgoal by (fastforce simp: image_iff)
   subgoal by (fastforce simp: image_iff)
   by auto

  define vs where vs: vs ≡ λ φ. getFrN (map snd txs) (map fst txs) [φ] (length txs)
  hence vss: ⋀φ. vs φ = getFrN (map snd txs) (map fst txs) [φ] (length txs) by auto
  {fix φ assume φ ∈ F hence φ ∈ fmla using assms by auto
   hence set (vs φ) ⊆ var ∧
    set (vs φ) ∩ Fvars φ = {} ∧
    set (vs φ) ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {} ∧
    set (vs φ) ∩ snd ‘ (set txs) = {} ∧
    length (vs φ) = length txs ∧
    distinct (vs φ)
   using assms unfolding vs
   using getFrN_Fvars[of map snd txs map fst txs [φ] _ length txs]
       getFrN_FvarsT[of map snd txs map fst txs [φ] _ length txs]
       getFrN_var[of map snd txs map fst txs [φ] _ length txs]
       getFrN_length[of map snd txs map fst txs [φ] length txs]
   apply(intro conjI)
    subgoal by auto
    subgoal by fastforce
    subgoal by (fastforce simp: image_iff)
    subgoal by (fastforce simp: image_iff)
```

```
  subgoal by (fastforce simp: image_iff)
  by auto
} note vs_facts = this

have [simp]: ⋀ x f. f ∈ F ⟹ x ∈ set (vs f) ⟹ x ∈ var
using vs_facts by (meson subsetD)

let ?tus = zip (map fst txs) us
let ?uxs = zip (map Var us) (map snd txs)
let ?tvs = λ φ. zip (map fst txs) (vs φ)
let ?vxs = λ φ. zip (map Var (vs φ)) (map snd txs)

let ?c = rawpsubst (sdsj F) ?uxs
have c: prv (eqv ?c
               (sdsj ((λφ. rawpsubst φ ?uxs) ` F)))
using assms us_facts
by (intro prv_rawpsubst_sdsj) (auto intro!: rawpsubstT dest!: set_zip_D)
hence prv (eqv (rawpsubst ?c ?tus)
              (rawpsubst (sdsj ((λφ. rawpsubst φ ?uxs) ` F)) ?tus))
using assms us_facts by (intro prv_eqv_rawpsubst) (auto intro!: rawpsubst dest!: set_zip_D)
moreover
have prv (eqv (rawpsubst (sdsj ((λφ. rawpsubst φ ?uxs) ` F)) ?tus)
             (sdsj ((λφ. rawpsubst φ ?tus) ` ((λφ. rawpsubst φ ?uxs) ` F))))
using assms us_facts by (intro prv_rawpsubst_sdsj) (auto intro!: rawpsubst dest!: set_zip_D)
ultimately
have 0: prv (eqv (rawpsubst ?c ?tus)
                (sdsj ((λφ. rawpsubst φ ?tus) ` ((λφ. rawpsubst φ ?uxs) ` F))))
using assms us_facts apply− by (rule prv_eqv_trans) (auto intro!: rawpsubst dest!: set_zip_D)
moreover
have prv (eqv (sdsj ((λφ. rawpsubst φ ?tus) ` ((λφ. rawpsubst φ ?uxs) ` F)))
             (sdsj ((λφ. rawpsubst (rawpsubst φ (?vxs φ)) (?tvs φ)) ` F)))
using assms us_facts vs_facts apply(intro prv_eqvI)
subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
subgoal apply(rule prv_sdsj_mono_imp)
  subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
  subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
  subgoal by auto
  subgoal by auto
  subgoal apply clarsimp
    subgoal for φ apply(rule bexI[of _ φ]) apply(rule prv_imp_refl2)
    subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
    subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
    subgoal by (rule rawpsubst_compose_freshVar2)
              (auto intro!: rawpsubst dest!: set_zip_D)  . . .
subgoal apply(rule prv_sdsj_mono_imp)
  subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
  subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
  subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
  subgoal by (auto intro!: rawpsubst dest!: set_zip_D)
  subgoal apply clarsimp
    subgoal for φ apply(rule bexI[of _ φ]) apply(rule prv_imp_refl2)
    apply (auto intro!: rawpsubst dest!: set_zip_D)
    apply(rule rawpsubst_compose_freshVar2)
    apply (auto intro!: rawpsubst dest!: set_zip_D) . . . .
ultimately
have prv (eqv (rawpsubst (rawpsubst (sdsj F) ?uxs) ?tus)
        (sdsj ((λφ. rawpsubst (rawpsubst φ (?vxs φ)) (?tvs φ)) ` F)))
```

using *assms us_facts*
apply− **by** (*rule prv_eqv_trans*) (*auto intro*!: *rawpsubst dest*!: *set_zip_D*)
thus *?thesis* **unfolding** *psubst_def* **by** (*simp add*: *Let_def us*[*symmetric*] *vss*)
qed

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))
using *assms Lq_num* **unfolding** *LLq_def* **by** *auto*

lemma *LLq_num2*:
assumes *p* ∈ *num*
shows ∃ *P* ⊆ *num*. *finite P* ∧ *prv* (*dsj* (*sdsj* {*eql* (*Var yy*) *r* | *r*. *r* ∈ *P*}) (*LLq p* (*Var yy*)))
using *assms Lq_num2* **unfolding** *LLq_def* **by** *auto*

end — context *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*
unfolding *nprv_def*
by (*simp add*: *prv_scnj_imp_in subset_iff*)

4.2 Structural Rules for the Natural Deduction Relation

lemma *prv_nprv0I*: *prv φ* ⟹ *φ* ∈ *fmla* ⟹ *nprv* {} *φ*
unfolding *nprv_def* **by** (*simp add*: *prv_imp_triv*)

lemma *prv_nprv_emp*: *φ* ∈ *fmla* ⟹ *prv φ* ⟷ *nprv* {} *φ*
using *prv_nprv0I* **unfolding** *nprv_def*
by (*metis asList eqv finite.simps insert_not_empty lcnj.simps*(*1*) *ldsj.cases*
list.simps(*15*) *prv_eqvI prv_imp_mp prv_imp_tru scnj_def tru*)

lemma *nprv_mono*:
assumes *nprv G φ*
and *F* ⊆ *fmla finite F G* ⊆ *F φ* ∈ *fmla*
shows *nprv F φ*
using *assms* **unfolding** *nprv_def*
by (*meson order_trans prv_prv_imp_trans prv_scnj_mono rev_finite_subset scnj*)

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*
using *assms* **unfolding** *nprv_def*
by (*metis* (*full_types*) *cnj finite.insertI*
insert_subset prv_imp_cnj prv_imp_cnj_scnj prv_imp_refl prv_prv_imp_trans scnj)

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
by (*meson finite.insertI insert_subsetI nprv_cut*)

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*
by (*meson finite.insertI insert_subsetI nprv_mono nprv_strong_cut2 subset_insertI*)

Useful for fine control of the eigenformula:

lemma *nprv_insertShiftI*:
nprv (*insert φ1* (*insert φ2 F*)) *ψ* ⟹ *nprv* (*insert φ2* (*insert φ1 F*)) *ψ*
by (*simp add*: *insert_commute*)

lemma *nprv_insertShift2I*:
nprv (*insert φ3* (*insert φ1* (*insert φ2 F*))) *ψ* ⟹ *nprv* (*insert φ1* (*insert φ2* (*insert φ3 F*))) *ψ*
by (*simp add*: *insert_commute*)

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*
using *prv_nprv0I*
by (*simp add*: *nprv_def prv_imp_triv*)

thm *prv_nprv0I*

lemma *prv_nprv1I*:
assumes *φ* ∈ *fmla ψ* ∈ *fmla* **and** *prv* (*imp φ ψ*)
shows *nprv* {*φ*} *ψ*
using *assms* **unfolding** *nprv_def* **by** (*simp add*: *prv_scnj_imp*)

lemma *prv_nprv2I*:
assumes *prv* (*imp φ1* (*imp φ2 ψ*)) *φ1* ∈ *fmla φ2* ∈ *fmla ψ* ∈ *fmla*
shows *nprv* {*φ1*,*φ2*} *ψ*
using *assms* **unfolding** *nprv_def*
by (*meson cnj empty_subsetI finite.simps insert_subsetI prv_cnj_imp_monoR2 prv_prv_imp_trans prv_scnj2_imp_cnj scnj*)

lemma *nprv_prvI*: *nprv* {} *φ* ⟹ *φ* ∈ *fmla* ⟹ *prv φ*
using *prv_nprv_emp* **by** *auto*

4.4 More Structural Properties

lemma *nprv_clear*: *nprv* {} *φ* ⟹ *F* ⊆ *fmla* ⟹ *finite F* ⟹ *φ* ∈ *fmla* ⟹ *nprv F φ*
by (*rule nprv_mono*) *auto*

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*
using *G F n1 n2* **proof**(*induction arbitrary*: *F χ*)
case (*insert ψ G F χ*)
hence *0*: *nprv F ψ* **by** *auto*

have *1*: *nprv* (*insert* ψ *F*) χ
using *insert.prems* **apply**− **apply**(*rule insert.IH*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*meson finite.simps insert_subset nprv_mono subsetD subset_insertI*)
by *auto*
show *?case* **using** *insert.prems* **by** (*intro nprv_cut*[*OF 0 1*]) *auto*
qed(*insert nprv_clear*, *auto*)

lemma *nprv_clear2_1*:
nprv {$\varphi 2$} $\psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
nprv {$\varphi 1$,$\varphi 2$} ψ
by (*rule nprv_mono*) *auto*

lemma *nprv_clear2_2*:
nprv {$\varphi 1$} $\psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
nprv {$\varphi 1$,$\varphi 2$} ψ
by (*rule nprv_mono*) *auto*

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 {$\varphi 1$,$\varphi 2$,$\varphi 3$} ψ
by (*rule nprv_mono*) *auto*

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 {$\varphi 1$,$\varphi 2$,$\varphi 3$} ψ
by (*rule nprv_mono*) *auto*

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 {$\varphi 1$,$\varphi 2$,$\varphi 3$} ψ
by (*rule nprv_mono*) *auto*

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 {$\varphi 1$,$\varphi 2$,$\varphi 3$,$\varphi 4$} ψ
by (*rule nprv_mono*) *auto*

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 {$\varphi 1$,$\varphi 2$,$\varphi 3$,$\varphi 4$} ψ
by (*rule nprv_mono*) *auto*

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 {$\varphi 1$,$\varphi 2$,$\varphi 3$,$\varphi 4$} ψ
by (*rule nprv_mono*) *auto*

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 {$\varphi 1$,$\varphi 2$,$\varphi 3$,$\varphi 4$} ψ
by (*rule nprv_mono*) *auto*

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

$\Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4,\varphi 5\}\ \psi$
by (*rule nprv_mono*) *auto*

lemma *nprv_clear5_2*:
$nprv\ \{\varphi 1,\varphi 3,\varphi 4,\varphi 5\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \varphi 4 \in fmla \Longrightarrow \varphi 5 \in fmla$
$\Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4,\varphi 5\}\ \psi$
by (*rule nprv_mono*) *auto*

lemma *nprv_clear5_3*:
$nprv\ \{\varphi 1,\varphi 2,\varphi 4,\varphi 5\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \varphi 4 \in fmla \Longrightarrow \varphi 5 \in fmla$
$\Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4,\varphi 5\}\ \psi$
by (*rule nprv_mono*) *auto*

lemma *nprv_clear5_4*:
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 5\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \varphi 4 \in fmla \Longrightarrow \varphi 5 \in fmla$
$\Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4,\varphi 5\}\ \psi$
by (*rule nprv_mono*) *auto*

lemma *nprv_clear5_5*:
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4\}\ \psi \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \varphi 3 \in fmla \Longrightarrow \varphi 4 \in fmla \Longrightarrow \varphi 5 \in fmla$
$\Longrightarrow \psi \in fmla \Longrightarrow$
$nprv\ \{\varphi 1,\varphi 2,\varphi 3,\varphi 4,\varphi 5\}\ \psi$
by (*rule nprv_mono*) *auto*

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*: $nprv\ F\ \psi$
shows $nprv\ ((\lambda\varphi.\ subst\ \varphi\ t\ x)\ `\ F)\ (subst\ \psi\ t\ x)$
using *assms* **using** *prv_subst*[*OF* _ _ _ *1*[*unfolded nprv_def*]] **unfolding** *nprv_def*
by (*intro prv_prv_imp_trans*[*OF* _ _ _ *prv_subst_scnj_imp*]) *auto*

lemma *nprv_subst_fresh*:
assumes *0*: $x \in var\ t \in trm\ \psi \in fmla\ finite\ F\ F \subseteq fmla$
$nprv\ F\ \psi$ **and** *1*: $x \notin \bigcup\ (Fvars\ `\ F)$
shows $nprv\ F\ (subst\ \psi\ t\ x)$
proof−
  have *2*: $(\lambda\varphi.\ subst\ \varphi\ t\ x)\ `\ F = F$ **unfolding** *image_def* **using** *assms* **by** *force*
  show *?thesis* **using** *nprv_subst*[*OF 0*] **unfolding** *2* **.**
qed

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 $nprv\ F\ \psi$
proof−
  have *0*: $subst\ (subst\ \psi\ (Var\ y)\ x)\ (Var\ x)\ y = \psi$
  using *assms* **by** (*auto simp*: *subst_compose_eq_or*)
  have $nprv\ ((\lambda\varphi.\ subst\ \varphi\ (Var\ x)\ y)\ `\ (\lambda\varphi.\ subst\ \varphi\ (Var\ y)\ x)\ `\ F)\ \psi$
  using *assms* **apply**(*subst 0*[*symmetric*]) **by** (*rule nprv_subst*) *auto*
  moreover
  have $prv\ (imp\ (scnj\ F)$

```
        (scnj ((λφ. subst φ (Var x) y) ‘ (λφ. subst φ (Var y) x) ‘ F)))
  using assms apply(intro prv_scnj_mono_imp)
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal apply clarify
    subgoal for _ _ φ
    by (auto simp: subst_compose_eq_or intro!: bexI[of _ φ] prv_imp_refl2) . .
  ultimately show ?thesis
  unfolding nprv_def using assms
  apply− by(rule prv_prv_imp_trans) auto
qed

lemma nprv_psubst:
assumes 0: snd ‘ set txs ⊆ var fst ‘ set txs ⊆ trm ψ ∈ fmla finite F F ⊆ fmla
distinct (map snd txs)
and 1: nprv F ψ
shows nprv ((λφ. psubst φ txs) ‘ F) (psubst ψ txs)
using assms  unfolding nprv_def
apply(intro prv_prv_imp_trans[OF _ _ _ prv_psubst_scnj_imp])
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal using prv_psubst[OF _ _ _ 1[unfolded nprv_def]]
 by (metis imp psubst_imp scnj) .
```

4.6 Introduction and Elimination Rules

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

```
lemma nprv_impI:
nprv (insert φ F) ψ ⟹
 F ⊆ fmla ⟹ finite F ⟹ φ ∈ fmla ⟹ ψ ∈ fmla ⟹
 nprv F (imp φ ψ)
unfolding nprv_def
by (metis cnj finite.insertI insert_subset prv_cnj_imp prv_imp_cnj_scnj prv_imp_com prv_prv_imp_trans
scnj)

lemma nprv_impI_rev:
assumes nprv F (imp φ ψ)
and F ⊆ fmla and finite F and φ ∈ fmla and ψ ∈ fmla
shows nprv (insert φ F) ψ
using assms unfolding nprv_def
by (metis cnj finite.insertI insert_subset prv_cnj_imp_monoR2 prv_eqv_imp_transi
    prv_eqv_scnj_insert prv_imp_com scnj)

lemma nprv_impI_rev2:
assumes nprv F (imp φ ψ) and G: insert φ F ⊆ G
and G ⊆ fmla and finite G and φ ∈ fmla and ψ ∈ fmla
shows nprv G ψ
using assms apply− apply(rule nprv_mono[of insert φ F])
subgoal by (meson nprv_impI_rev order_trans rev_finite_subset subset_insertI)
```

by *auto*

lemma *nprv_mp*:
nprv F (imp φ ψ) ⟹ nprv F φ ⟹
F ⊆ fmla ⟹ finite F ⟹ φ ∈ fmla ⟹ ψ ∈ fmla ⟹
nprv F ψ
unfolding *nprv_def*
by (*metis* (*full_types*) *cnj prv_cnj_imp_monoR2 prv_imp_cnj prv_imp_refl prv_prv_imp_trans scnj*)

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
using *nprv_cut nprv_mp* **by** *blast*

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)
unfolding *nprv_def* **by** (*simp add*: *prv_imp_cnj*)

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
unfolding *nprv_def*
by (*metis cnj nprv_cut2 nprv_def prv_imp_cnjL prv_imp_cnjR prv_prv_imp_trans scnj*)

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)
unfolding *nprv_def* **by** (*meson dsj prv_dsj_impL prv_prv_imp_trans scnj*)

lemma *nprv_dsjIR*:
nprv F ψ ⟹
F ⊆ fmla ⟹ finite F ⟹ φ ∈ fmla ⟹ ψ ∈ fmla ⟹
nprv F (dsj φ ψ)
unfolding *nprv_def* **by** (*meson dsj prv_dsj_impR prv_prv_imp_trans scnj*)

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−

have *nprv F (imp (dsj φ ψ) χ)*
by (*meson assms dsj imp nprv_def nprv_impI prv_imp_com prv_imp_dsjEE scnj*)
hence *nprv (insert (dsj φ ψ) F) χ* **using** *assms*
by (*simp add*: *nprv_impI_rev*)
thus *?thesis* **using** *assms* **by** (*meson dsj nprv_cut*)
qed

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* ⟹ *F* ⊆ *fmla* ⟹ *finite F* ⟹ *φ* ∈ *fmla* ⟹ *nprv F φ*
unfolding *nprv_def* **using** *prv_prv_imp_trans scnj* **by** *blast*

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

lemma *nprv_truI*: *F* ⊆ *fmla* ⟹ *finite F* ⟹ *nprv F tru*
unfolding *nprv_def* **by** (*simp add*: *prv_imp_tru*)

lemma *nprv_negI*:
nprv (*insert φ F*) *fls* ⟹
F ⊆ *fmla* ⟹ *finite F* ⟹ *φ* ∈ *fmla* ⟹
nprv F (*neg φ*)
unfolding *neg_def* **by** (*auto intro*: *nprv_impI*)

lemma *nprv_neg_fls*:
nprv F (*neg φ*) ⟹ *nprv F φ* ⟹
F ⊆ *fmla* ⟹ *finite F* ⟹ *φ* ∈ *fmla* ⟹ *ψ* ∈ *fmla* ⟹
nprv F fls
unfolding *neg_def* **using** *nprv_mp* **by** *blast*

lemma *nprv_negE*:
nprv F (*neg φ*) ⟹ *nprv F φ* ⟹
F ⊆ *fmla* ⟹ *finite F* ⟹ *φ* ∈ *fmla* ⟹ *ψ* ∈ *fmla* ⟹
nprv F ψ
using *nprv_flsE nprv_neg_fls* **by** *blast*

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*:
(⋀ *ψ*. *ψ* ∈ *G* ⟹ *nprv F ψ*) ⟹
F ⊆ *fmla* ⟹ *finite F* ⟹ *G* ⊆ *fmla* ⟹ *finite G* ⟹
nprv F (*scnj G*)
unfolding *nprv_def* **by** (*simp add*: *prv_imp_scnj*)

lemma *nprv_scnjE*:
nprv F (*scnj G*) ⟹ *nprv* (*G* ∪ *F*) *ψ* ⟹
F ⊆ *fmla* ⟹ *finite F* ⟹ *G* ⊆ *fmla* ⟹ *finite G* ⟹ *ψ* ∈ *fmla* ⟹
nprv F ψ
apply(*rule nprv_cut_set*[*of _ G*])
subgoal by *auto*
subgoal by *auto*

subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*meson in_mono nprv_def prv_prv_imp_trans prv_scnj_imp_in scnj*) **.**

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*:
(⋀ ψ. $\psi \in$ *set* $\psi s \Longrightarrow$ *nprv* F ψ) ⟹
$F \subseteq$ *fmla* ⟹ *finite* $F \Longrightarrow$ *set* $\psi s \subseteq$ *fmla* ⟹
nprv F (*lcnj* ψs)
unfolding *nprv_def* **by** (*simp add*: *prv_imp_lcnj*)

lemma *nprv_lcnjE*:
nprv F (*lcnj* φs) ⟹ *nprv* (*set* $\varphi s \cup F$) ψ ⟹
$F \subseteq$ *fmla* ⟹ *finite* $F \Longrightarrow$ *set* $\varphi s \subseteq$ *fmla* ⟹ $\psi \in$ *fmla* ⟹
nprv F ψ
apply(*rule nprv_cut_set*[*of* _ *set* $\varphi s \cup F$])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
apply (*elim UnE*)
apply (*meson lcnj nprv_def prv_lcnj_imp_in prv_prv_imp_trans scnj subset_code*(*1*))
by *auto*
subgoal by *auto* **.**

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 φ ⟹
$F \subseteq$ *fmla* ⟹ *finite* $F \Longrightarrow$ $G \subseteq$ *fmla* ⟹ *finite* $G \Longrightarrow$ $\varphi \in G$ ⟹
nprv F (*sdsj* G)
unfolding *nprv_def* **by** (*simp add*: *prv_imp_sdsj*)

lemma *nprv_sdsjE*:
assumes *nprv* F (*sdsj* G)
and ⋀ ψ. $\psi \in G \Longrightarrow$ *nprv* (*insert* ψ F) χ
and $F \subseteq$ *fmla finite* F $G \subseteq$ *fmla finite* G $\chi \in$ *fmla*
shows *nprv* F χ
proof−
have *0*: *prv* (*imp* (*sdsj* G) (*imp* (*scnj* F) χ))
using *assms* **apply**(*intro prv_sdsj_imp*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*meson nprv_def nprv_impI prv_imp_com scnj set_rev_mp*) **.**
hence *nprv* F (*imp* (*sdsj* G) χ)
by (*simp add*: *0 assms nprv_def prv_imp_com*)
thus *?thesis* **using** *assms nprv_mp* **by** *blast*
qed

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* φs)
unfolding *nprv_def* **by**(*simp add*: *prv_imp_ldsj*)

lemma *nprv_ldsjE*:
assumes *nprv F* (*ldsj* ψs)
and $\bigwedge \psi$. $\psi \in$ *set* $\psi s \Longrightarrow$ *nprv* (*insert* ψ *F*) χ
and *F* $\subseteq$ *fmla finite F set* $\psi s \subseteq$ *fmla* $\chi \in$ *fmla*
shows *nprv F* χ
proof−
have *0*: *prv* (*imp* (*ldsj* ψs) (*imp* (*scnj F*) χ))
using *assms* **apply**(*intro prv_ldsj_imp*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*meson nprv_def nprv_impI prv_imp_com scnj set_rev_mp*) **.**
hence *nprv F* (*imp* (*ldsj* ψs) χ)
by (*simp add*: *0 assms nprv_def prv_imp_com*)
thus *?thesis* **using** *assms nprv_mp* **by** *blast*
qed

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* φ)
unfolding *nprv_def* **by** (*rule prv_all_imp_gen*) *auto*

lemma *nprv_allE*:
assumes *nprv F* (*all x* φ) *nprv* (*insert* (*subst* φ *t x*) *F*) ψ
F $\subseteq$ *fmla finite F* $\varphi \in$ *fmla t* $\in$ *trm x* $\in$ *var* $\psi \in$ *fmla*
shows *nprv F* ψ
proof−
have *nprv F* (*subst* φ *t x*)
using *assms* **unfolding** *nprv_def* **by** (*meson all subst prv_all_inst prv_prv_imp_trans scnj*)
thus *?thesis* **by** (*meson assms local.subst nprv_cut*)
qed

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* $\Longrightarrow$ $\varphi \in$ *fmla* $\Longrightarrow$ *t* $\in$ *trm* $\Longrightarrow$ *x* $\in$ *var* $\Longrightarrow$
nprv F (*exi x* φ)
unfolding *nprv_def* **by** (*meson exi local.subst prv_exi_inst prv_prv_imp_trans scnj*)

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* $\varphi \in$ *fmla x* $\in$ *var* $\psi \in$ *fmla*
and *x*: *x* $\notin \bigcup$ (*Fvars* ' *F*) *x* $\notin$ *Fvars* ψ
shows *nprv F* ψ
proof−
have *nprv F* (*imp* (*exi x* φ) ψ) **unfolding** *nprv_def* **apply**(*rule prv_imp_com*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal apply(*rule prv_exi_imp_gen*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *x* **by** *auto*
subgoal apply(*rule prv_imp_com*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *assms*(*3*−*5*) *assms*(*7*) *nn nprv_def nprv_impI* **by** *blast* . . .
thus *?thesis* **using** *n assms nprv_mp* **by** *blast*
qed

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 $\varphi \in$ *fmla* $\psi \in$ *fmla* **and** *F* $\subseteq$ *fmla* **and** *finite F*
shows *nprv F* ψ
using *assms nprv_cut prv_nprvI* **by** *blast*

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

lemma *nprv_addImpLemmaI*:
assumes *prv* (*imp* φ*1* φ*2*)
and *F* $\subseteq$ *fmla finite F* φ*1* $\in$ *fmla* φ*2* $\in$ *fmla*
and *nprv F* φ*1*
shows *nprv F* φ*2*
by (*meson assms nprv_def prv_prv_imp_trans scnj*)

lemma *nprv_addImpLemmaE*:
assumes *prv* (*imp* φ*1* φ*2*) **and** *nprv F* φ*1* **and** *nprv* ((*insert* φ*2*) *F*) ψ
and *F* $\subseteq$ *fmla finite F* φ*1* $\in$ *fmla* φ*2* $\in$ *fmla* $\psi \in$ *fmla*
shows *nprv F* ψ
using *assms nprv_addImpLemmaI nprv_cut* **by** *blast*

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* φ*1* (*imp* φ*2* φ*3*))
and *F* $\subseteq$ *fmla finite F* φ*1* $\in$ *fmla* φ*2* $\in$ *fmla* φ*3* $\in$ *fmla*

and *nprv F φ1 nprv F φ2*
shows *nprv F φ3*
by (*meson assms imp nprv_addImpLemmaI nprv_mp*)

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* ⊆ *fmla finite F φ1* ∈ *fmla φ2* ∈ *fmla φ3* ∈ *fmla ψ* ∈ *fmla*
shows *nprv F ψ*
by (*meson assms nprv_addImp2LemmaI nprv_cut*)

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*
by (*meson assms imp nprv_addImpLemmaI nprv_mp*)

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*
by (*meson assms nprv_addImp3LemmaI nprv_cut*)

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*

by (*meson assms dsj nprv_clear nprv_dsjE prv_nprv0I*)

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)
by (*simp add*: *prv_eql_reflT prv_nprvI*)

lemma *nprv_eq_eqlI*: $t1 = t2 \Longrightarrow F \subseteq fmla \Longrightarrow$ *finite* $F \Longrightarrow t1 \in trm \Longrightarrow$ *nprv* F (*eql t1 t2*)
by (*simp add*: *prv_eql_reflT prv_nprvI*)

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* $\varphi 1$ (*insert* $\varphi 2$ F)) $\psi \Longrightarrow$
$F \subseteq fmla \Longrightarrow$ *finite* $F \Longrightarrow \varphi 1 \in fmla \Longrightarrow \varphi 2 \in fmla \Longrightarrow \psi \in fmla \Longrightarrow$
nprv (*insert* (*cnj* $\varphi 1$ $\varphi 2$) F) ψ
apply(*rule nprv_cut2*[*of* _ $\varphi 1$ $\varphi 2$])

subgoal by (*auto simp*: *nprv_impI_rev prv_imp_cnjL prv_imp_cnjR prv_nprvI*)
subgoal by (*auto simp*: *nprv_impI_rev prv_imp_cnjL prv_imp_cnjR prv_nprvI*)
subgoal by (*auto simp*: *nprv_impI_rev prv_imp_cnjL prv_imp_cnjR prv_nprvI*)
subgoal by (*auto simp*: *nprv_impI_rev prv_imp_cnjL prv_imp_cnjR prv_nprvI*)
subgoal by (*auto simp*: *nprv_impI_rev prv_imp_cnjL prv_imp_cnjR prv_nprvI*)
subgoal by (*auto simp*: *nprv_impI_rev prv_imp_cnjL prv_imp_cnjR prv_nprvI*)
subgoal by (*auto simp*: *nprv_impI_rev prv_imp_cnjL prv_imp_cnjR prv_nprvI*)
by (*meson cnj finite.insertI insert_iff insert_subset nprv_mono subset_insertI*)

lemma *nprv_exi_commute*:
assumes [*simp*]: $x \in var$ $y \in var$ $\varphi \in fmla$
shows *nprv* {*exi x* (*exi y* φ)} (*exi y* (*exi x* φ))
apply(*rule nprv_exiE0*[*of x exi y* φ], *auto*)
apply(*rule nprv_clear2_2*, *auto*)
apply(*rule nprv_exiE0*[*of y* φ], *auto*)
apply(*rule nprv_clear2_2*, *auto*)
apply(*rule nprv_exiI*[*of _ _ Var y*], *auto*)
by (*rule nprv_exiI*[*of _ _ Var x*], *auto*)

lemma *prv_exi_commute*:
assumes [*simp*]: $x \in var$ $y \in var$ $\varphi \in fmla$
shows *prv* (*imp* (*exi x* (*exi y* φ)) (*exi y* (*exi x* φ)))
apply(*rule nprv_prvI*, *auto*)
apply(*rule nprv_impI*, *auto*)
by (*rule nprv_exi_commute*, *auto*)

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* φ *t x*) **and** *n2*: *nprv* (*insert* φ *F*) (*eql* (*Var x*) *t*)
and *i*[*simp*]: $F \subseteq fmla$ *finite F* $\varphi \in fmla$ $t \in trm$ $x \in var$ $x \notin FvarsT\ t$
and *u*: $x \notin (\bigcup \varphi \in F.\ Fvars\ \varphi)$
shows *nprv F* (*exu x* φ)
proof−
  define *z* **where** $z \equiv getFr\ [x]\ [t]\ [\varphi]$
  have *z_facts*[*simp*]: $z \in var$ $z \neq x$ $x \neq z$ $z \notin FvarsT\ t$ $z \notin Fvars\ \varphi$
  using *getFr_FvarsT_Fvars*[*of* [*x*] [*t*] [φ]] **unfolding** *z_def*[*symmetric*] **by** *auto*
  have *0*: *exu x* φ = *cnj* (*exi x* φ) (*exi z* (*all x* (*imp* φ (*eql* (*Var x*) (*Var z*)))))
  by (*simp add*: *exu_def_var*[*of _ z*])
  show *?thesis*
  unfolding *0*
    apply(*rule nprv_cnjI*, *auto*)
    apply(*rule nprv_exiI*[*of _ _ t*], *auto*)
    apply(*rule n1*)

    apply(*rule nprv_exiI*[*of _ _ t*], *auto*)
    apply(*rule nprv_allI*, *insert u*, *auto*)
    apply(*rule nprv_impI*, *insert n2*, *auto*)
  done
qed

lemma *nprv_exuI_var*:
assumes *n1*: *nprv F* (*subst* φ *t x*) **and** *n2*: *nprv* (*insert* (*subst* φ (*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* φ
shows *nprv* F (*exu* x φ)
apply(*subst exu_rename2*[*of* _ _ *y*])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *yx* **by** *auto*
subgoal apply(*intro nprv_exuI*[*of* _ _ *t*])
subgoal by (*metis i*(*3*) *i*(*4*) *i*(*5*) *i*(*6*) *n1 subst_same_Var subst_subst yx*)
using *n2 u* **by** *auto* **.**

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

lemma *nprv_exuI_exi*:
assumes *n1*: *nprv* F (*exi* x φ) **and** *n2*: *nprv* (*insert* (*subst* φ (*Var* y) x) (*insert* φ 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* φ
and *u*: $x \notin (\bigcup \varphi \in F.$ *Fvars* $\varphi)$ $y \notin (\bigcup \varphi \in F.$ *Fvars* $\varphi)$
shows *nprv* F (*exu* x φ)
proof−
have *e*: *nprv* (*insert* φ F) (*exu* x φ)
apply(*rule nprv_exuI_var*[*of* _ _ *Var x* _ *y*])
using *n2 u* **by** *auto*
show *?thesis*
apply(*rule nprv_cut*[*OF n1*], *auto*)
apply(*rule nprv_exiE0*, *insert u*, *auto*)
apply(*rule nprv_mono*[*OF e*], *auto*) **.**
qed

lemma *prv_exu_imp_exi*:
assumes [*simp*]: $\varphi \in$ *fmla* $x \in$ *var*
shows *prv* (*imp* (*exu* x φ) (*exi* x φ))
proof−
define z **where** z: $z \equiv$ *getFr* [x] [] [φ]
have *z_facts*[*simp*]: $z \in$ *var* $z \neq x$ $x \neq z$ $z \notin$ *Fvars* φ
using *getFr_FvarsT_Fvars*[*of* [x] [] [φ]] **unfolding** z **by** *auto*
show *?thesis* **unfolding** *exu_def*
by (*simp add*: *Let_def z*[*symmetric*] *prv_imp_cnjL*)
qed

lemma *prv_exu_exi*:
assumes $x \in$ *var* $\varphi \in$ *fmla prv* (*exu* x φ)
shows *prv* (*exi* x φ)
by (*meson assms exi exu prv_exu_imp_exi prv_imp_mp*)

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

lemma *nprv_exuE_exi*:
assumes *n1*: *nprv* F (*exu* x φ) **and** *n2*: *nprv* (*insert* φ F) ψ
and *i*[*simp*]: $F \subseteq$ *fmla finite* F $\varphi \in$ *fmla* $x \in$ *var* $\psi \in$ *fmla* $x \notin$ *Fvars* ψ
and *u*: $x \notin (\bigcup \varphi \in F.$ *Fvars* $\varphi)$
shows *nprv* F ψ
using *assms* **apply**− **apply**(*rule nprv_exiE*[*of* _ x φ])
subgoal by (*rule nprv_addImpLemmaI*[*OF prv_exu_imp_exi*[*of* φ x]]) *auto*
by *auto*

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

and *u*: $x \notin (\bigcup \varphi \in F.\ Fvars\ \varphi)$
shows *nprv F ψ*
using *assms nprv_exuE_exi nprv_hyp* **by** *metis*

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−
define *z* **where** *z*: $z \equiv getFr\ [x]\ [t1,t2]\ [\varphi]$
have *z_facts*[*simp*]: $z \in var\ z \neq x\ x \neq z\ z \notin Fvars\ \varphi\ \ z \notin FvarsT\ t1\ z \notin FvarsT\ t2$
using *getFr_FvarsT_Fvars*[*of* [*x*] [*t1*,*t2*] [φ]] **unfolding** *z* **by** *auto*
show *?thesis*
apply(*rule nprv_prvI*, *auto*)
apply(*rule nprv_impI*, *auto*)
apply(*simp add*: *exu_def_var*[*of* _ *z*])
apply(*rule nprv_cnjE0*, *auto*)
apply(*rule nprv_clear3_1*, *auto*)
apply(*rule nprv_clear2_2*, *auto*)
apply(*rule nprv_exiE0*, *auto*)
apply(*rule nprv_clear2_2*, *auto*)
apply(*rule nprv_allE0*[*of* _ _ _ *t1*], *auto*)
apply(*rule nprv_allE0*[*of* _ _ _ *t2*], *auto*)
apply(*rule nprv_clear3_3*, *auto*)
apply(*rule nprv_impI*, *auto*)
apply(*rule nprv_impI*, *auto*)
apply(*rule nprv_impE01*, *auto*)
apply(*rule nprv_clear5_2*, *auto*)
apply(*rule nprv_clear4_3*, *auto*)
apply(*rule nprv_impE01*, *auto*)
apply(*rule nprv_clear4_3*, *auto*)
apply(*rule nprv_clear3_3*, *auto*)
apply(*rule nprv_eql_symE0*[*of t2 Var z*], *auto*)
apply(*rule nprv_eql_transE012*, *auto*) **.**
qed

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*
by *auto*

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*
unfolding *ptrm_def* **by** *auto*

lemma *ptrm_1_Fvars*[*simp*]: *σ* ∈ *ptrm* (*Suc 0*) ⟹ *Fvars σ* = {*out*,*inp*}
unfolding *ptrm_def* **by** *auto*

lemma *ptrm_prv_exu*: $\sigma \in ptrm\ n \Longrightarrow prv\ (exu\ out\ \sigma)$
unfolding *ptrm_def* **by** *auto*

lemma *ptrm_prv_exi*: $\sigma \in ptrm\ n \Longrightarrow prv\ (exi\ out\ \sigma)$
by (*simp add*: *ptrm_prv_exu prv_exu_exi*)

lemma *nprv_ptrmE_exi*:
$\sigma \in ptrm\ n \Longrightarrow nprv\ (insert\ \sigma\ F)\ \psi \Longrightarrow$
$F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow$
$\psi \in fmla \Longrightarrow out \notin Fvars\ \psi \Longrightarrow out \notin \bigcup\ (Fvars\ `\ F) \Longrightarrow nprv\ F\ \psi$
apply (*frule ptrm_prv_exu*, *drule ptrm*)
apply(*rule nprv_exuE_exi*[*of* _ *out* σ])
by (*auto intro*!: *prv_nprvI*)

lemma *nprv_ptrmE_uni*:
$\sigma \in ptrm\ n \Longrightarrow nprv\ F\ (subst\ \sigma\ t1\ out) \Longrightarrow nprv\ F\ (subst\ \sigma\ t2\ out) \Longrightarrow$
$nprv\ (insert\ (eql\ t1\ t2)\ F)\ \psi \Longrightarrow$
$F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow \psi \in fmla \Longrightarrow t1 \in trm \Longrightarrow t2 \in trm$
$\Longrightarrow nprv\ F\ \psi$
apply (*frule ptrm_prv_exu*, *drule ptrm*)
apply(*rule nprv_exuE_uni*[*of* _ *out* σ *t1 t2*])
by (*auto intro*!: *prv_nprvI*)

lemma *nprv_ptrmE_uni0*:
$\sigma \in ptrm\ n \Longrightarrow nprv\ F\ \sigma \Longrightarrow nprv\ F\ (subst\ \sigma\ t\ out) \Longrightarrow$
$nprv\ (insert\ (eql\ (Var\ out)\ t)\ F)\ \psi \Longrightarrow$
$F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow \psi \in fmla \Longrightarrow t \in trm$
$\Longrightarrow nprv\ F\ \psi$
by (*rule nprv_ptrmE_uni*[*of* σ _ _ *Var out t*]) *auto*

lemma *nprv_ptrmE0_uni0*:
$\sigma \in ptrm\ n \Longrightarrow \sigma \in F \Longrightarrow nprv\ F\ (subst\ \sigma\ t\ out) \Longrightarrow$
$nprv\ (insert\ (eql\ (Var\ out)\ t)\ F)\ \psi \Longrightarrow$
$F \subseteq fmla \Longrightarrow finite\ F \Longrightarrow \psi \in fmla \Longrightarrow t \in trm$
$\Longrightarrow nprv\ F\ \psi$
by (*rule nprv_ptrmE_uni0*[*of* σ *n* _ *t*]) *auto*

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−
have [*simp*]: $\sigma \in fmla$ **using** σ **by** *simp*
define z **where** $z \equiv getFr\ [out]\ []\ [\sigma,\varphi]$
have *z_facts*[*simp*]: $z \in var\ z \neq out\ z \notin Fvars\ \sigma\ z \notin Fvars\ \varphi$
using *getFr_FvarsT_Fvars*[*of* [*out*] [] [σ,φ]] **unfolding** *z_def*[*symmetric*] **by** *auto*
have *0*: $exi\ out\ (cnj\ \sigma\ \varphi) = exi\ z\ (subst\ (cnj\ \sigma\ \varphi)\ (Var\ z)\ out)$
by(*rule exi_rename*, *auto*)
show *?thesis*
unfolding *0*

```
    apply(nrule r: nprv_exiE0[of z subst (cnj σ φ) (Var z) out])
    apply(nrule2 r: nprv_ptrmE0_uni0[OF σ, of _ Var z])
     subgoal by (nrule r: nprv_cnjE0)
     subgoal
      apply(nrule r: nprv_clear4_4)
      apply(nrule r: nprv_clear3_3)
      apply (nrule r: nprv_cnjE0)
      apply(nrule r: nprv_clear4_4)
      apply(nrule r: nprv_clear3_1)
      apply(nrule r: nprv_eql_substE012[of Var out Var z _ φ out φ]) . .
qed

lemma ptrm_nprv_exi_all:
  assumes σ: σ ∈ ptrm n and [simp]: φ ∈ fmla
  shows nprv {exi out (cnj σ φ)} (all out (imp σ φ))
proof−
  have [simp]: σ ∈ fmla using σ by simp
  show ?thesis
  apply(nrule r: nprv_allI)
  apply(nrule r: nprv_impI)
  apply(nrule r: ptrm_nprv_exi[OF σ]) .
qed

lemma ptrm_prv_exi_imp_all:
  assumes σ: σ ∈ ptrm n and [simp]: φ ∈ fmla
  shows prv (imp (exi out (cnj σ φ)) (all out (imp σ φ)))
proof−
  have [simp]: σ ∈ fmla using σ by simp
  show ?thesis
  apply(nrule r: nprv_prvI)
  apply(nrule r: nprv_impI)
  apply(nrule r: ptrm_nprv_exi_all[OF σ]) .
qed

lemma ptrm_nprv_all_imp_exi:
  assumes σ: σ ∈ ptrm n and [simp]: φ ∈ fmla
  shows nprv {all out (imp σ φ)} (exi out (cnj σ φ))
proof−
  have [simp]: σ ∈ fmla using σ by simp
  define z where z ≡ getFr [out] [] [σ,φ]
  have z_facts[simp]: z ∈ var z ≠ out z ∉ Fvars σ z ∉ Fvars φ
  using getFr_FvarsT_Fvars[of [out] [] [σ,φ]] unfolding z_def[symmetric] by auto
  show ?thesis
    apply(nrule r: nprv_ptrmE_exi[OF σ])
    apply(nrule r: nprv_exiI[of _ cnj σ φ Var out out])
    apply(nrule r: nprv_allE0[of out imp σ φ _ Var out])
    apply(nrule r: nprv_clear3_3)
    apply(nrule r: nprv_cnjI)
    apply(nrule r: nprv_impE01) .
qed

lemma ptrm_prv_all_imp_exi:
  assumes σ: σ ∈ ptrm n and [simp]: φ ∈ fmla
  shows prv (imp (all out (imp σ φ)) (exi out (cnj σ φ)))
proof−
  have [simp]: σ ∈ fmla using σ by simp
  define z where z ≡ getFr [out] [] [σ,φ]
  have z_facts[simp]: z ∈ var z ≠ out z ∉ Fvars σ z ∉ Fvars φ
```

using *getFr_FvarsT_Fvars*[*of* [*out*] [] [σ,φ]] **unfolding** *z_def*[*symmetric*] **by** *auto*
show *?thesis*
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_impI*)
apply(*nrule r*: *ptrm_nprv_all_imp_exi*[*OF* σ]) .
qed

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* $\Rightarrow$ *'trm* $\Rightarrow$ *'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*
using *assms instInp_def* **by** *auto*

lemma *Fvars_instInp*[*simp*,*intro*]:
assumes $\varphi \in$ *fmla* **and** *t* $\in$ *trm Fvars* $\varphi = \{inp\}$
shows *Fvars* (*instInp* φ *t*) = *FvarsT t*
using *assms instInp_def* **by** *auto*

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*)
using *assms instInp_def* **by** *auto*

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−
note *Let_def*[*simp*]
have [*simp*]: $\tau \in$ *fmla Fvars* $\tau = \{out,inp\}$
using *assms* **unfolding** *ptrm_def* **by** *auto*
have [*simp*]: *Fvars* (*instInp* τ *t*) = *insert out* (*FvarsT t*)
using τ **by** (*subst Fvars_instInp_ptrm_1*) *auto*
have *0*: *exu out* (*instInp* τ *t*) = *subst* (*exu out* τ) *t inp*

```
    unfolding instInp_def by (subst subst_exu) auto
  have 1: prv (exu out τ) using τ unfolding ptrm_def by auto
  have prv (exu out (instInp τ t))
  unfolding 0 by (rule prv_subst[OF _ _ _ 1], auto)
  thus ?thesis using assms unfolding ptrm_def[of n] by auto
qed

lemma instInp_0:
assumes τ: τ ∈ ptrm (Suc 0) and t ∈ trm and FvarsT t = {}
shows instInp τ t ∈ ptrm 0
using assms by (intro instInp) auto

lemma instInp_1:
assumes τ: τ ∈ ptrm (Suc 0) and t ∈ trm and FvarsT t = {inp}
shows instInp τ t ∈ ptrm (Suc 0)
using assms by (intro instInp) auto
```

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* φ n τ 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 ⇒ nat ⇒ 'fmla ⇒ 'fmla where
instInpP φ n τ ≡ let zz = Variable (Suc (Suc n)) in
   exi zz (cnj (subst τ (Var zz) out) (subst φ (Var zz) inp))

lemma instInpP_fmla[simp, intro]:
 assumes φ ∈ fmla and τ ∈ fmla
 shows instInpP φ n τ ∈ fmla
 using assms unfolding instInpP_def by (auto simp: Let_def)

lemma Fvars_instInpP[simp]:
assumes φ ∈ fmla and τ: τ ∈ ptrm n  Variable (Suc (Suc n)) ∉ Fvars φ
shows Fvars (instInpP φ n τ) = Fvars φ − {inp} ∪ Variable ‘ {(Suc 0)..n}
using assms unfolding instInpP_def Let_def ptrm_def by auto

lemma Fvars_instInpP2[simp]:
assumes φ ∈ fmla and τ: τ ∈ ptrm n and Fvars φ ⊆ {inp}
shows Fvars (instInpP φ n τ) = Fvars φ − {inp} ∪ Variable ‘ {(Suc 0)..n}
using assms by (subst Fvars_instInpP) auto
```

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: σ ∈ ptrm (Suc 0) and τ: τ ∈ ptrm n
shows instInpP σ n τ ∈ ptrm n
proof−
  note Let_def[simp]
  have [simp]: σ ∈ fmla τ ∈ fmla Fvars σ = {out,inp}
   Fvars τ = Variable ‘ {0..n}
    using assms unfolding ptrm_def by auto
  define zz where zz ≡ Variable (Suc (Suc n))
  have zz_facts[simp]: zz ∈ var ⋀i. i ≤ n ⟹ Variable i ≠ zz ∧ zz ≠ Variable i
    out ≠ zz zz ≠ out inp ≠ zz zz ≠ inp
   unfolding zz_def by auto
```

define x **where** $x \equiv getFr\ [out,inp,zz]\ []\ [\sigma,\tau]$
have $x\_facts[simp]$: $x \in var\ x \neq out\ x \neq inp$
$x \neq zz\ zz \neq x\ x \notin Fvars\ \sigma\ x \notin Fvars\ \tau$
using $getFr\_FvarsT\_Fvars[of\ [out,inp,zz]\ []\ [\sigma,\tau]]$ **unfolding** $x\_def[symmetric]$ **by** $auto$
have $[simp]$: $x \neq Variable\ (Suc\ (Suc\ n))$
using $x\_facts(4)\ zz\_def$ **by** $auto$
define z **where** $z \equiv getFr\ [out,inp,zz,x]\ []\ [\sigma,\tau]$
have $z\_facts[simp]$: $z \in var\ z \neq out\ z \neq inp\ z \neq x\ z \neq zz\ z \notin Fvars\ \sigma\ z \notin Fvars\ \tau$
using $getFr\_FvarsT\_Fvars[of\ [out,inp,zz,x]\ []\ [\sigma,\tau]]$ **unfolding** $z\_def[symmetric]$ **by** $auto$

have $[simp]$: $\bigwedge i.\ z = Variable\ i \Longrightarrow \neg\ i \leq n$
and $[simp]$: $\bigwedge i.\ x = Variable\ i \Longrightarrow \neg\ i \leq n$
using $\langle Fvars\ \tau = Variable\ `\ \{0..n\}\rangle\ atLeastAtMost\_iff\ z\_facts(7)\ x\_facts(7)$
by $blast+$

have $[simp]$: $Fvars\ (instInpP\ \sigma\ n\ \tau) = Variable\ `\ \{0..n\}$
unfolding $instInpP\_def$ **by** $auto$
have tt: $exi\ out\ \tau = exi\ zz\ (subst\ \tau\ (Var\ zz)\ out)$
by $(rule\ exi\_rename)\ auto$

have exi_σ: $prv\ (exi\ out\ \sigma)$ **and** exi_τ: $prv\ (exi\ zz\ (subst\ \tau\ (Var\ zz)\ out))$
using $\sigma\ \tau\ ptrm\_prv\_exi\ tt$ **by** $fastforce+$
have exi_σ: $prv\ (exi\ out\ (subst\ \sigma\ (Var\ zz)\ inp))$
using $prv\_subst[OF\ \_\ \_\ \_\ exi\_\sigma,\ of\ inp\ Var\ zz]$ **by** $auto$

have exu_σ: $prv\ (exu\ out\ \sigma)$
using $\sigma\ ptrm\_prv\_exu$ **by** $blast$
have exu_σ: $prv\ (exu\ out\ (subst\ \sigma\ (Var\ zz)\ inp))$
using $prv\_subst[OF\ \_\ \_\ \_\ exu\_\sigma,\ of\ inp\ Var\ zz]$ **by** $auto$

have zz_z: $exi\ zz\ (cnj\ (subst\ \tau\ (Var\ zz)\ out)\ (subst\ \sigma\ (Var\ zz)\ inp)) =$
$exi\ z\ (cnj\ (subst\ \tau\ (Var\ z)\ out)\ (subst\ \sigma\ (Var\ z)\ inp))$
by $(variousSubsts2\ auto\ s1$: $exi\_rename[of\ \_\ zz\ z]\ s2$: $subst\_subst)$

have 0: $prv\ (exu\ out\ (instInpP\ \sigma\ n\ \tau))$
apply$(nrule\ r$: $nprv\_prvI)$
apply$(nrule2\ r$: $nprv\_exuI\_exi[of\ \_\ \_\ \_\ x])$
subgoal unfolding $instInpP\_def\ Let\_def$
apply$(nrule\ r$: $nprv\_addImpLemmaI[OF\ prv\_exi\_commute])$
apply$(nrule\ r$: $nprv\_addLemmaE[OF\ exi\_\tau])$
apply$(nrule\ r$: $nprv\_exiE[of\ \_\ zz\ subst\ \tau\ (Var\ zz)\ out])$
apply$(nrule\ r$: $nprv\_clear2\_2)$
apply$(nrule\ r$: $nprv\_exiI[of\ \_\ \_\ Var\ zz])$
apply$(nrule\ r$: $nprv\_addLemmaE[OF\ exi\_\sigma])$
apply$(nrule\ r$: $nprv\_exiE[of\ \_\ out\ subst\ \sigma\ (Var\ zz)\ inp])$
apply$(nrule\ r$: $nprv\_clear3\_2)$
apply$(nrule\ r$: $nprv\_exiI[of\ \_\ \_\ Var\ out])$
apply$(variousSubsts1\ auto\ s1$: $subst\_subst)$
apply$(nrule\ r$: $nprv\_cnjI)$ **.**
subgoal
unfolding $instInpP\_def\ Let\_def\ zz\_def[symmetric]$
apply$(nrule\ r$: $nprv\_exiE0[of\ zz])$
apply$(nrule\ r$: $nprv\_clear3\_2)$
apply$(nrule\ r$: $nprv\_cnjE0)$
apply$(nrule\ r$: $nprv\_clear4\_3)$
unfolding zz_z
apply$(nrule\ r$: $nprv\_exiE0[of\ z])$

```
    apply(nrule r: nprv_clear4_4)
    apply(nrule r: nprv_cnjE0)
    apply(nrule r: nprv_clear5_3)
    apply(nrule r: nprv_cut[of _ eql (Var z) (Var zz)])
    subgoal by (nprover3 r1: nprv_clear4_2 r2: nprv_clear3_3
                r3: nprv_ptrmE_uni[OF τ, of _ Var z Var zz])
    subgoal
      apply(nrule r: nprv_clear5_2)
      apply(nrule r: nprv_clear4_3)
      apply(nrule2 r: nprv_eql_substE[of _ Var zz Var z σ inp])
      subgoal by (nrule r: nprv_eql_symE01)
      subgoal
        apply(nrule r: nprv_clear4_2)
        apply(nrule r: nprv_clear3_2)
        apply(nrule r: nprv_addLemmaE[OF exu_σ])
        apply(nrule r: nprv_exuE_uni[of _ out subst σ (Var zz) inp Var out Var x])
        apply(nrule r: nprv_eql_symE01) . . . .
 show ?thesis using 0 unfolding ptrm_def instInpP_def Let_def by auto
qed
```

Term and pseudo-term instantiation compose smoothly:

```
lemma instInp_instInpP:
assumes φ: φ ∈ fmla Fvars φ ⊆ {inp} and τ: τ ∈ ptrm (Suc 0)
and t ∈ trm and FvarsT t = {}
shows instInp (instInpP φ (Suc 0) τ) t = instInpP φ 0 (instInp τ t)
proof−
  define x1 and x2 where
   x12: x1 ≡ Variable (Suc (Suc 0))
        x2 ≡ Variable (Suc (Suc (Suc 0)))
  have x_facts[simp]: x1 ∈ var x2 ∈ var x1 ≠ inp x2 ≠ inp
   x1 ≠ out out ≠ x1 x2 ≠ out out ≠ x2 x1 ≠ x2 x2 ≠ x1
   unfolding x12 by auto
  show ?thesis
  using assms unfolding instInp_def instInpP_def Let_def x12[symmetric]

  apply(subst subst_exi)
  apply(TUF simp)

  apply(variousSubsts5 auto
   s1: subst_compose_same
   s2: subst_compose_diff
   s3: exi_rename[of _ x1 x2]
   s4: subst_comp
   s5: subst_notIn[of φ _ x1]
  ) .
qed
```

Pseudo-term instantiation also composes smoothly with itself:

```
lemma nprv_instInpP_compose:
assumes [simp]: χ ∈ fmla Fvars χ = {inp}
and σ[simp]: σ ∈ ptrm (Suc 0) and τ[simp]: τ ∈ ptrm 0
shows nprv {instInpP (instInpP χ (Suc 0) σ) 0 τ}
           (instInpP χ 0 (instInpP σ 0 τ)) (is ?A)
and
      nprv {instInpP χ 0 (instInpP σ 0 τ)}
           (instInpP (instInpP χ (Suc 0) σ) 0 τ) (is ?B)
proof−
  define χσ and στ where χσ_def: χσ ≡ instInpP χ (Suc 0) σ and στ_def: στ ≡ instInpP σ 0 τ
```

have [*simp*]: $\sigma \in$ *fmla Fvars* $\sigma = \{out,inp\}$ $\tau \in$ *fmla Fvars* $\tau = \{out\}$
using σ τ **unfolding** *ptrm_def* **by** *auto*
have $\chi\sigma$[*simp*]: $\chi\sigma \in$ *fmla Fvars* $\chi\sigma = \{inp\}$
unfolding $\chi\sigma$*_def* **by** *auto*
have $\sigma\tau$[*simp*]: $\sigma\tau \in$ *ptrm 0* $\sigma\tau \in$ *fmla Fvars* $\sigma\tau = \{out\}$ **unfolding** $\sigma\tau$*_def*
by *auto*
define *z* **where** $z \equiv$ *Variable* (*Suc* (*Suc 0*))
have *z_facts*[*simp*]: $z \in$ *var*
$out \neq z \wedge z \neq out\ inp \neq z \wedge z \neq inp\ z \notin$ *Fvars* $\chi\ z \notin$ *Fvars* $\sigma\ z \notin$ *Fvars* τ
unfolding *z_def* **by** *auto*
define *zz* **where** $zz \equiv$ *Variable* (*Suc* (*Suc* (*Suc 0*)))
have *zz_facts*[*simp*]: $zz \in$ *var*
$out \neq zz \wedge zz \neq out\ inp \neq zz \wedge zz \neq inp\ z \neq zz \wedge zz \neq z$
$zz \notin$ *Fvars* $\chi\ zz \notin$ *Fvars* $\sigma\ zz \notin$ *Fvars* τ
unfolding *zz_def z_def* **by** *auto*
define z' **where** $z' \equiv$ *getFr* [*out*,*inp*,*z*,*zz*] [] [χ,σ,τ]
have z'*_facts*[*simp*]: $z' \in$ *var* $z' \neq out\ z' \neq inp\ z' \neq z\ z \neq z'\ z' \neq zz\ zz \neq z'$
$z' \notin$ *Fvars* $\chi z' \notin$ *Fvars* $\sigma\ z' \notin$ *Fvars* τ
using *getFr_FvarsT_Fvars*[*of* [*out*,*inp*,*z*,*zz*] [] [χ,σ,τ]] **unfolding** z'*_def*[*symmetric*] **by** *auto*

have $\chi\sigma'$: *instInpP* $\chi\sigma$ *0* τ = *exi* z' (*cnj* (*subst* τ (*Var* z') *out*) (*subst* $\chi\sigma$ (*Var* z') *inp*))
unfolding *instInpP_def Let_def z_def*[*symmetric*]
by (*auto simp*: *exi_rename*[*of* _ *z z'*])
have $\chi\sigma z'$: *subst* $\chi\sigma$ (*Var* z') *inp* =
exi zz (*cnj* (*subst* (*subst* σ (*Var zz*) *out*) (*Var* z') *inp*) (*subst* χ (*Var zz*) *inp*))
unfolding $\chi\sigma$*_def instInpP_def Let_def zz_def*[*symmetric*]
by (*auto simp*: *subst_compose_same*)
have $\sigma\tau zz$: *subst* $\sigma\tau$ (*Var zz*) *out* =
exi z (*cnj* (*subst* τ (*Var z*) *out*) (*subst* (*subst* σ (*Var zz*) *out*) (*Var z*) *inp*))
unfolding $\sigma\tau$*_def instInpP_def Let_def z_def*[*symmetric*]
by (*variousSubsts2 auto s1*: *subst_compose_same s2*: *subst_compose_diff*)

have *nprv* {*instInpP* $\chi\sigma$ *0* τ} (*instInpP* χ *0* $\sigma\tau$)
unfolding $\chi\sigma'$
apply(*nrule r*: *nprv_exiE0*)
apply(*nrule r*: *nprv_clear2_2*)
apply(*nrule r*: *nprv_cnjE0*)
apply(*nrule r*: *nprv_clear3_3*)
unfolding $\chi\sigma z'$
apply(*nrule r*: *nprv_exiE0*)
apply(*nrule r*: *nprv_clear3_3*)
apply(*nrule r*: *nprv_cnjE0*)
apply(*nrule r*: *nprv_clear4_3*)
unfolding *instInpP_def Let_def z_def*[*symmetric*]
apply(*nrule r*: *nprv_exiI*[*of* _ _ *Var zz*])
apply(*nrule r*: *nprv_cnjI*)
apply(*nrule r*: *nprv_clear3_2*)
unfolding $\sigma\tau zz$
apply(*nrule r*: *nprv_exiI*[*of* _ _ *Var z'*])
apply(*nrule r*: *nprv_cnjI*) **.**
thus *?A* **unfolding** $\chi\sigma$*_def* $\sigma\tau$*_def* **.**

have $\chi\sigma\tau$: *instInpP* χ *0* $\sigma\tau$ = *exi* z' (*cnj* (*subst* $\sigma\tau$ (*Var* z') *out*) (*subst* χ (*Var* z') *inp*))
unfolding *instInpP_def Let_def z_def*[*symmetric*]
by (*auto simp*: *exi_rename*[*of* _ *z z'*])

have $\sigma\tau z'$: *subst* $\sigma\tau$ (*Var* z') *out* =

```
  exi z (cnj (subst τ (Var z) out) (subst (subst σ (Var z) inp) (Var z') out))
 unfolding στ_def instInpP_def Let_def z_def[symmetric]
 by (auto simp: subst_compose_same)

 have χσz: subst χσ (Var z) inp =
   exi zz (cnj (subst (subst σ (Var z) inp) (Var zz) out) (subst χ (Var zz) inp))
 unfolding χσ_def instInpP_def Let_def zz_def[symmetric]
 by (variousSubsts2 auto s1: subst_compose_same s2: subst_compose_diff)

 have nprv {instInpP χ 0 στ} (instInpP χσ 0 τ)
 unfolding χστ
 apply(nrule r: nprv_exiE0)
 apply(nrule r: nprv_clear2_2)
 apply(nrule r: nprv_cnjE0)
 apply(nrule r: nprv_clear3_3)
 unfolding στz'
 apply(nrule r: nprv_exiE0)
 apply(nrule r: nprv_clear3_2)
 apply(nrule r: nprv_cnjE0)
 apply(nrule r: nprv_clear4_3)
 unfolding instInpP_def Let_def z_def[symmetric]
 apply(nrule r: nprv_exiI[of _ _ Var z])
 apply(nrule r: nprv_cnjI)
 unfolding χσz
 apply(nrule r: nprv_exiI[of _ _ Var z'])
 apply(nrule r: nprv_cnjI) .
 thus ?B unfolding χσ_def στ_def .
qed

lemma prv_instInpP_compose:
assumes [simp]: χ ∈ fmla Fvars χ = {inp}
and σ[simp]: σ ∈ ptrm (Suc 0) and τ[simp]: τ ∈ 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−
 have [simp]: σ ∈ fmla Fvars σ = {out,inp} τ ∈ fmla Fvars τ = {out}
 using σ τ unfolding ptrm_def by auto
 show ?A ?B by (intro nprv_prvI nprv_impI nprv_instInpP_compose, auto)+
 thus ?C by (intro prv_eqvI) auto
qed
```

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*]: $t \in trm\ FvarsT\ t = \{\}$
and τt: $prveqlPT\ \tau\ t$
shows $nprv\ \{instInpP\ \varphi\ 0\ \tau\}\ (instInp\ \varphi\ t)$
proof−
have [*simp*]: $\tau \in fmla\ Fvars\ \tau = \{out\}$ **using** τ **unfolding** *ptrm_def* **by** *auto*
define *zz* **where** $zz \equiv Variable\ (Suc\ (Suc\ 0))$
have *zz_facts*[*simp*]: $zz \in var$
$out \neq zz \wedge zz \neq out\ inp \neq zz \wedge zz \neq inp\ zz \notin Fvars\ \tau\ zz \notin Fvars\ \varphi$
unfolding *zz_def* **using** f **by** *auto*

note *lemma1* = *nprv_addLemmaE*[*OF* τt[*unfolded prveqlPT_def*]]

show *?thesis* **unfolding** *instInpP_def Let_def zz_def*[*symmetric*] *instInp_def*
apply(*nrule r*: *lemma1*)
apply(*nrule r*: *nprv_exiE0*[*of zz*])
apply(*nrule r*: *nprv_clear3_3*)
apply(*nrule r*: *nprv_cnjE0*)
apply(*nrule r*: *nprv_clear4_3*)
apply(*nrule r*: *nprv_ptrmE_uni*[*OF* τ, *of* _ *t Var zz*])
apply(*nrule r*: *nprv_clear4_2*)
apply(*nrule r*: *nprv_clear3_3*)
apply(*nrule r*: *nprv_eql_substE012*[*of t Var zz* _ φ *inp*]) **.**
qed

lemma *prveqlPT_prv_instInp_instInpP*:
assumes$\varphi \in fmla$ **and** f: $Fvars\ \varphi \subseteq \{inp\}$ **and** τ: $\tau \in ptrm\ 0$
and $t \in trm\ FvarsT\ t = \{\}$
and τt: $prveqlPT\ \tau\ t$
shows $prv\ (imp\ (instInpP\ \varphi\ 0\ \tau)\ (instInp\ \varphi\ t))$
using *assms* **by** (*intro nprv_prvI nprv_impI prveqlPT_nprv_instInp_instInpP*) *auto*

lemma *prveqlPT_nprv_instInpP_instInp*:
assumes[*simp*]: $\varphi \in fmla$ **and** f: $Fvars\ \varphi \subseteq \{inp\}$ **and** τ: $\tau \in ptrm\ 0$
and [*simp*]: $t \in trm\ FvarsT\ t = \{\}$
and τt: $prveqlPT\ \tau\ t$
shows $nprv\ \{instInp\ \varphi\ t\}\ (instInpP\ \varphi\ 0\ \tau)$
proof−
have [*simp*]: $\tau \in fmla\ Fvars\ \tau = \{out\}$ **using** τ **unfolding** *ptrm_def* **by** *auto*
define *zz* **where** $zz \equiv Variable\ (Suc\ (Suc\ 0))$
have *zz_facts*[*simp*]: $zz \in var$
$out \neq zz \wedge zz \neq out\ inp \neq zz \wedge zz \neq inp\ zz \notin Fvars\ \tau\ zz \notin Fvars\ \varphi$
unfolding *zz_def* **using** f **by** *auto*

note *lemma1* = *nprv_addLemmaE*[*OF* τt[*unfolded prveqlPT_def*]]

show *?thesis* **unfolding** *instInpP_def Let_def zz_def*[*symmetric*] *instInp_def*
by (*nprover3 r1*: *lemma1 r2*: *nprv_exiI*[*of* _ _ *t*] *r3*: *nprv_cnjI*)
qed

lemma *prveqlPT_prv_instInpP_instInp*:
assumes$\varphi \in fmla$ **and** f: $Fvars\ \varphi \subseteq \{inp\}$ **and** τ: $\tau \in ptrm\ 0$
and $t \in trm\ FvarsT\ t = \{\}$
and τt: $prveqlPT\ \tau\ t$
shows $prv\ (imp\ (instInp\ \varphi\ t)\ (instInpP\ \varphi\ 0\ \tau))$
using *assms* **by** (*intro nprv_prvI nprv_impI prveqlPT_nprv_instInpP_instInp*) *auto*

lemma *prveqlPT_prv_instInp_eqv_instInpP*:
assumes$\varphi \in fmla$ **and** f: $Fvars\ \varphi \subseteq \{inp\}$ **and** τ: $\tau \in ptrm\ 0$
and $t \in trm\ FvarsT\ t = \{\}$
and τt: $prveqlPT\ \tau\ t$
shows $prv\ (eqv\ (instInpP\ \varphi\ 0\ \tau)\ (instInp\ \varphi\ t))$
using *assms prveqlPT_prv_instInp_instInpP prveqlPT_prv_instInpP_instInp*
by (*intro prv_eqvI*) *auto*

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*:

$\varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow$
$isTrue\ \varphi \Longrightarrow isTrue\ (neg\ \varphi) \Longrightarrow False$
using *isTrue_imp not_isTrue_fls* **unfolding** *neg_def* **by** *auto*

lemma *isTrue_neg_neg*:
assumes $\varphi \in fmla\ Fvars\ \varphi = \{\}$
and *isTrue* (*neg* (*neg* φ))
shows *isTrue* φ
using *assms isTrue_neg isTrue_neg_excl* **by** *fastforce*

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*
unfolding *consistent_def* **using** *not_isTrue_fls sound_isTrue* **by** *blast*

lemma *prv_neg_excl*:
$\varphi \in fmla \Longrightarrow Fvars\ \varphi = \{\} \Longrightarrow prv\ \varphi \Longrightarrow prv\ (neg\ \varphi) \Longrightarrow False$
using *isTrue_neg_excl*[*of* φ] *sound_isTrue* **by** *auto*

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−
have *isTrue* (*imp* $\varphi\ \chi$) **using** *p* **by** (*intro sound_isTrue*) *auto*

thus *?thesis* **using** *assms isTrue_imp* **by** *blast*
qed

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

lemma *ωconsistent*: *ωconsistent*
unfolding *ωconsistent_def* **proof** (*safe del*: *notI*)
fix φ *x* **assume** *0*[*simp*,*intro*]: $\varphi \in$ *fmla* $x \in$ *var* **and** *1*: *Fvars* $\varphi = \{x\}$
and *00*: $\forall\, n{\in}$*num*. *prv* (*neg* (*subst* φ *n x*))
hence $\forall\, n{\in}$*num*. *isTrue* (*neg* (*subst* φ *n x*))
using *00 1* **by** (*auto intro*!: *sound_isTrue*)
hence *isTrue* (*all x* (*neg* φ)) **by** (*simp add*: *1 isTrue_all*)
moreover
{have *prv* (*imp* (*all x* (*neg* φ)) (*neg* (*exi x* φ)))
using *prv_all_neg_imp_neg_exi* **by** *blast*
hence *isTrue* (*imp* (*all x* (*neg* φ)) (*neg* (*exi x* φ)))
by (*simp add*: *1 sound_isTrue*)
}
ultimately have *isTrue* (*neg* (*exi x* φ))
by (*metis 0 1 Diff_insert_absorb Fvars_all Fvars_exi Fvars_neg all*
exi insert_absorb insert_not_empty isTrue_imp neg)
hence ¬ *isTrue* (*neg* (*neg* (*exi x* φ)))
using *1 isTrue_neg_excl* **by** *force*
thus ¬ *prv* (*neg* (*neg* (*exi x* φ)))
using *1 sound_isTrue* **by** *auto*
qed

lemma *ωconsistentStd1*: *ωconsistentStd1*
using *ωconsistent ωconsistent_impliesStd1* **by** *blast*

lemma *ωconsistentStd2*: *ωconsistentStd2*
using *ωconsistent ωconsistent_impliesStd2* **by** *blast*

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*
using *num_Num* **by** *auto*

lemma *FvarsT_Num*[*simp*]: *FvarsT* (*Num n*) = {}
by *auto*

lemma *substT_Num*[*simp*]: *x* ∈ *var* ⟹ *t* ∈ *trm* ⟹ *substT* (*Num n*) *t x* = *Num n*
by *auto*

lemma *zer*[*simp*,*intro*!]: $zer \in num$
and *suc_num*[*simp*]: $\bigwedge n.\ n \in num \Longrightarrow suc\ n \in num$
by (*metis Num Num.simps*(*1*), *metis Num Num.simps*(*2*) *imageE num_Num*)

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** $t \in trm$
using *assms* **by** *induct auto*

lemma *atrm_trm*: $atrm \subseteq trm$
using *atrm_imp_trm* **by** *auto*

lemma *zer_atrm*[*simp*]: $zer \in atrm$ **by** *auto*

lemma *Num_atrm*[*simp*]: $Num\ n \in atrm$
by *auto*

lemma *substT_atrm*[*simp*]:
assumes $r \in atrm$ **and** $x \in var$ **and** $t \in atrm$
shows $substT\ r\ t\ x \in atrm$
using *assms* **by** (*induct*) *auto*

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 $FvarsT\ (substT\ s\ t\ x) = (FvarsT\ s - \{x\}) \cup (if\ x \in FvarsT\ s\ then\ FvarsT\ t\ else\ \{\})$
using *assms* **by** *induct auto*

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 $substT\ (substT\ s\ t1\ x1)\ t2\ x2 = substT\ s\ (substT\ t1\ t2\ x2)\ x1$
using *assms* **apply** *induct*
subgoal by *auto*
subgoal by *auto*
subgoal by (*metis FvarsT_suc atrm_imp_trm substT substT_suc*)
subgoal by (*metis Fvars_pls UnCI atrm_imp_trm substT substT_pls*)
subgoal by (*metis Fvars_tms UnCI atrm_imp_trm substT substT_tms*) **.**

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 $substT\ (substT\ s\ t1\ x1)\ t2\ x2 = substT\ (substT\ s\ t2\ x2)\ (substT\ t1\ t2\ x2)\ x1$
using *assms* **apply** *induct*
subgoal by *auto*

subgoal by *auto*
subgoal by (*metis atrm_imp_trm substT substT_suc*)
subgoal by (*metis atrm_imp_trm substT substT_pls*)
subgoal by (*metis atrm_imp_trm substT substT_tms*) **.**

lemma *substT_same_Var*[*simp*]:
assumes $s \in atrm$ $x \in var$
shows *substT s* (*Var x*) *x* = *s*
using *assms* **by** *induct auto*

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

lemma *in_FvarsT_substTD*:
$y \in FvarsT\ (substT\ r\ t\ x) \Longrightarrow r \in atrm \Longrightarrow t \in trm \Longrightarrow x \in var$
$\Longrightarrow y \in (FvarsT\ r - \{x\}) \cup (if\ x \in FvarsT\ r\ then\ FvarsT\ t\ else\ \{\})$
using *FvarsT_substT* **by** *auto*

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*
using *substT_compose_eq_or* **by** *blast*

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*
using *substT_compose_eq_or*[*OF s _ t x y*, *of Var y*] **using** *subst_notIn yy* **by** *simp*

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*
by (*simp add*: *substT_compose_diff*)

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*
using *assms* **by** (*induct txs arbitrary*: *r*) *auto*

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−
have *txs_trm*: *fst* ‘ (*set txs*) $\subseteq$ *trm* **using** *assms atrm_trm* **by** *auto*
define *us* **where** *us*: *us* ≡ *getFrN* (*map snd txs*) (*r* # *map fst txs*) [] (*length txs*)
have *us_facts*: *set us* $\subseteq$ *var*
set us ∩ *FvarsT r* = {}
set us ∩ $\bigcup$ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set us ∩ *snd* ‘ (*set txs*) = {}
length us = *length txs*
distinct us
using *assms*(*1*,*2*) *txs_trm* **unfolding** *us*
using *getFrN_FvarsT*[*of map snd txs r # map fst txs* [] _ *length txs*]
getFrN_Fvars[*of map snd txs r # map fst txs* [] _ *length txs*]
getFrN_var[*of map snd txs r # map fst txs* [] _ *length txs*]
getFrN_length[*of map snd txs r # map fst txs* [] *length txs*]
getFrN_distinct[*of map snd txs r # map fst txs* [] *length txs*]
apply −

```
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by force
    by auto

  show ?thesis using assms us_facts unfolding psubstT_def
    by (force simp: Let_def us[symmetric]
      intro!: rawpsubstT_atrm[of _ zip (map fst txs) us] dest!: set_zip_D)
qed

lemma Fvars_rawpsubst_su:
assumes r ∈ atrm and snd ‘ (set txs) ⊆ var and fst ‘ (set txs) ⊆ atrm
shows FvarsT (rawpsubstT r txs) ⊆
      (FvarsT r − snd ‘ (set txs)) ∪ (⋃ {FvarsT t | t x . (t,x) ∈ set txs})
using assms proof(induction txs arbitrary: r)
  case (Cons tx txs r)
  then obtain t x where tx: tx = (t,x) by force
  have t: t ∈ trm and x: x ∈ var using Cons.prems unfolding tx by auto
  define χ where χ ≡ substT r t x
  have 0: FvarsT χ =  FvarsT r − {x} ∪ (if x ∈ FvarsT r then FvarsT t else {})
  using Cons.prems unfolding χ_def by (auto simp: tx t FvarsT_substT)
  have χ: χ ∈ trm χ ∈ atrm unfolding χ_def using Cons.prems t x  by (auto simp add: tx)
  have FvarsT (rawpsubstT χ txs) ⊆
      (FvarsT χ − snd ‘ (set txs)) ∪
      (⋃ {FvarsT t | t x . (t,x) ∈ set txs})
    using Cons.prems χ by (intro Cons.IH) auto
  also have ... ⊆ FvarsT r − insert x (snd ‘ set txs) ∪ ⋃{FvarsT ta |ta. ∃xa. ta = t ∧ xa = x ∨ (ta,
xa) ∈ set txs}
  (is _ ⊆ ?R) by(auto simp: 0 tx Cons.prems)
  finally have 1: FvarsT (rawpsubstT χ txs) ⊆ ?R .
  have 2: FvarsT χ = FvarsT r − {x} ∪ (if x ∈ FvarsT r then FvarsT t else {})
    using Cons.prems t x unfolding χ_def using FvarsT_substT by auto
  show ?case using 1 by (simp add: tx χ_def[symmetric] 2)
qed auto

lemma in_FvarsT_rawpsubstT_imp:
  assumes y ∈ FvarsT (rawpsubstT r txs)
and r ∈ atrm and snd ‘ (set txs) ⊆ var and fst ‘ (set txs) ⊆ atrm
shows (y ∈ FvarsT r − snd ‘ (set txs)) ∨
    (y ∈ ⋃ { FvarsT t | t x . (t,x) ∈ set txs})
using Fvars_rawpsubst_su[OF assms(2−4)]
using assms(1) by blast

lemma FvarsT_rawpsubstT:
assumes r ∈ atrm and snd ‘ (set txs) ⊆ var and fst ‘ (set txs) ⊆ atrm
and distinct (map snd txs) and ∀ x ∈ snd ‘ (set txs). ∀ t ∈ fst ‘ (set txs). x ∉ FvarsT t
shows FvarsT (rawpsubstT r txs) =
      (FvarsT r − snd ‘ (set txs)) ∪
      (⋃ {if x ∈ FvarsT r then FvarsT t else {} | t x . (t,x) ∈ set txs})
using assms proof(induction txs arbitrary: r)
  case (Cons a txs r)
  then obtain t x where a: a = (t,x) by force
  have t: t ∈ trm and x: x ∈ var using Cons.prems unfolding a by auto
  have xt: x ∉ FvarsT t ∧ snd ‘ set txs ∩ FvarsT t = {} using Cons.prems unfolding a by auto
  hence 0: FvarsT r − {x} ∪ FvarsT t − snd ‘ set txs = FvarsT r − insert x (snd ‘ set txs) ∪ FvarsT t
  by auto
  have x_txs: ⋀ta xa. (ta, xa) ∈ set txs ⟹ x ≠ xa using ‹distinct (map snd (a # txs))›
```

unfolding *a* **by** (*auto simp*: *rev_image_eqI*)

define χ **where** χ_*def*: $\chi \equiv$ *substT r t x*
have χ: $\chi \in$ *trm* $\chi \in$ *atrm* **unfolding** χ_*def* **using** *Cons.prems t x* **by** (*auto simp*: *a*)
have *1*: *FvarsT* (*rawpsubstT* χ *txs*) =
(*FvarsT* χ − *snd* ‘ (*set txs*)) ∪
(⋃ {*if x* ∈ *FvarsT* χ *then FvarsT t else* {} | *t x* . (*t*,*x*) ∈ *set txs*})
using *Cons.prems* χ **by** (*intro Cons.IH*) *auto*
have *2*: *FvarsT* χ = *FvarsT r* − {*x*} ∪ (*if x* ∈ *FvarsT r then FvarsT t else* {})
using *Cons.prems t x* **unfolding** χ_*def* **using** *FvarsT_substT* **by** *auto*

define *f* **where** *f* ≡ λ*ta xa*. *if xa* ∈ *FvarsT r then FvarsT ta else* {}

have *3*: ⋃ {*f ta xa* |*ta xa*. (*ta*, *xa*) ∈ *set* ((*t*, *x*) # *txs*)} =
f t x ∪ (⋃ {*f ta xa* |*ta xa*. (*ta*, *xa*) ∈ *set txs*}) **by** *auto*
have *4*: *snd* ‘ *set* ((*t*, *x*) # *txs*) = {*x*} ∪ *snd* ‘ *set txs* **by** *auto*
have *5*: *f t x* ∩ *snd* ‘ *set txs* = {} **unfolding** *f_def* **using** *xt* **by** *auto*
have *6*: ⋃ {*if xa* ∈ *FvarsT r* − {*x*} ∪ *f t x then FvarsT ta else* {} | *ta xa*. (*ta*, *xa*) ∈ *set txs*}
= (⋃ {*f ta xa* | *ta xa*. (*ta*, *xa*) ∈ *set txs*})
unfolding *f_def* **using** *xt x_txs* **by** (*fastforce split*: *if_splits*)

have *FvarsT r* − {*x*} ∪ *f t x* − *snd* ‘ *set txs* ∪
⋃ {*if xa* ∈ *FvarsT r* − {*x*} ∪ *f t x then FvarsT ta else* {}
| *ta xa*. (*ta*, *xa*) ∈ *set txs*} =
FvarsT r − *snd* ‘ *set* ((*t*, *x*) # *txs*) ∪
⋃ {*f ta xa* |*ta xa*. (*ta*, *xa*) ∈ *set* ((*t*, *x*) # *txs*)}
unfolding *3 4 6* **unfolding** *Un_Diff2*[*OF 5*] *Un_assoc* **unfolding** *Diff_Diff_Un* **..**

thus *?case* **unfolding** *a rawpsubstT.simps 1 2* χ_*def*[*symmetric*] *f_def* **by** *simp*
qed *auto*

lemma *in_FvarsT_rawpsubstTD*:
assumes *y* ∈ *FvarsT* (*rawpsubstT r txs*)
and *r* ∈ *atrm* **and** *snd* ‘ (*set txs*) ⊆ *var* **and** *fst* ‘ (*set txs*) ⊆ *atrm*
and *distinct* (*map snd txs*) **and** ∀ *x* ∈ *snd* ‘ (*set txs*). ∀ *t* ∈ *fst* ‘ (*set txs*). *x* ∉ *FvarsT t*
shows (*y* ∈ *FvarsT r* − *snd* ‘ (*set txs*)) ∨
(*y* ∈ ⋃ {*if x* ∈ *FvarsT r then FvarsT t else* {} | *t x* . (*t*,*x*) ∈ *set txs*})
using *FvarsT_rawpsubstT assms* **by** *auto*

lemma *FvarsT_psubstT*:
assumes *r* ∈ *atrm* **and** *snd* ‘ (*set txs*) ⊆ *var* **and** *fst* ‘ (*set txs*) ⊆ *atrm*
and *distinct* (*map snd txs*)
shows *FvarsT* (*psubstT r txs*) =
(*FvarsT r* − *snd* ‘ (*set txs*)) ∪
(⋃ {*if x* ∈ *FvarsT r then FvarsT t else* {} | *t x* . (*t*,*x*) ∈ *set txs*})
proof−
have *txs_trm*: *fst* ‘ (*set txs*) ⊆ *trm* **using** *assms* **by** *auto*
define *us* **where** *us*: *us* ≡ *getFrN* (*map snd txs*) (*r* # *map fst txs*) [] (*length txs*)
have *us_facts*: *set us* ⊆ *var*
set us ∩ *FvarsT r* = {}
set us ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set us ∩ *snd* ‘ (*set txs*) = {}
length us = *length txs*
distinct us
using *assms*(*1*,*2*) *txs_trm* **unfolding** *us*
using *getFrN_FvarsT*[*of map snd txs r* # *map fst txs* [] _ *length txs*]
getFrN_Fvars[*of map snd txs r* # *map fst txs* [] _ *length txs*]
getFrN_var[*of map snd txs r* # *map fst txs* [] _ *length txs*]

```
      getFrN_length[of map snd txs r # map fst txs [] length txs]
      getFrN_length[of map snd txs r # map fst txs [] length txs]
  apply −
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by force
    by auto
  have [simp]: ⋀ aa b. b ∈ set (map snd txs) ⟹
     aa ∈ set (map Var us) ⟹ b ∉ FvarsT aa
  using us_facts by (fastforce simp: image_def Int_def)
  have [simp]:
   ⋀b ac bc. b ∈ set us ⟹ b ∈ FvarsT ac ⟹ (ac, bc) ∉ set txs
   using us_facts(3) by (fastforce simp: image_def Int_def)

  define χ where χ_def: χ ≡ rawpsubstT r (zip (map Var us) (map snd txs))
  have χ: χ ∈ atrm unfolding χ_def
  using assms using us_facts by (intro rawpsubstT_atrm) (force dest!: set_zip_D)+

  hence χ ∈ trm by auto
  note χ = χ this
  have set_us: set us = snd ‘ (set (zip (map fst txs) us))
    using us_facts by (intro snd_set_zip[symmetric]) auto
  have set_txs: snd ‘ set txs = snd ‘ (set (zip (map Var us) (map snd txs)))
    using us_facts by (intro snd_set_zip_map_snd[symmetric]) auto
  have ⋀ t x. (t, x) ∈ set (zip (map Var us) (map snd txs)) ⟹ ∃ u. t = Var u
    using us_facts set_zip_leftD by fastforce
  hence 00: ⋀ t x. (t, x) ∈ set (zip (map Var us) (map snd txs))
     ⟷ (∃ u ∈ var. t = Var u ∧ (Var u, x) ∈ set (zip (map Var us) (map snd txs)))
    using us_facts set_zip_leftD by fastforce
  have FvarsT χ =
       FvarsT r − snd ‘ set txs ∪
       ⋃{if x ∈ FvarsT r then FvarsT t else {} |t x.
            (t, x) ∈ set (zip (map Var us) (map snd txs))}
    unfolding χ_def set_txs using assms us_facts set_txs
    by (intro FvarsT_rawpsubstT) (force dest!: set_zip_D)+
  also have ... =
     FvarsT r − snd ‘ set txs ∪
     ⋃{if x ∈ FvarsT r then {u} else {} |u x. u ∈ var ∧ (Var u, x) ∈ set (zip (map Var us) (map snd
txs))}
  (is ... = ?R)
  apply(subst 00)
  by (metis (no_types, opaque_lifting) FvarsT_Var)
  finally have 0: FvarsT χ = ?R .
  have 1: FvarsT (rawpsubstT χ (zip (map fst txs) us)) =
       (FvarsT χ − set us) ∪
       (⋃ {if u ∈ FvarsT χ then FvarsT t else {} | t u . (t,u) ∈ set (zip (map fst txs) us)})
  unfolding us_facts set_us using assms χ apply (intro FvarsT_rawpsubstT)
  subgoal by auto
  subgoal using us_facts by (auto dest!: set_zip_D)
  subgoal using us_facts by (auto dest!: set_zip_D)
  subgoal using us_facts by (auto dest!: set_zip_D)
  subgoal by (auto dest!: set_zip_D) .

  have 2: FvarsT χ − set us = FvarsT r − snd ‘ set txs
  unfolding 0 apply auto
  using set_zip_leftD us_facts(1) apply fastforce
  using set_zip_leftD us_facts(1) apply fastforce
```

```
  using us_facts(2) by auto
  have 3:
  (⋃ {if u ∈ FvarsT χ then FvarsT t else {} | t u . (t,u) ∈ set (zip (map fst txs) us)}) =
   (⋃ {if x ∈ FvarsT r then FvarsT t else {} | t x . (t,x) ∈ set txs})
  proof safe
    fix xx tt y
    assume xx: xx ∈ (if y ∈ FvarsT χ then FvarsT tt else {})
    and ty: (tt, y) ∈ set (zip (map fst txs) us)
    have ttin: tt ∈ fst ‘ set txs using ty using set_zip_leftD by fastforce
    have yin: y ∈ set us using ty by (meson set_zip_D)
    have yvar: y ∈ var using us_facts yin by auto
    have ynotin: y ∉ snd ‘ set txs y ∉ FvarsT r using yin us_facts by auto
    show xx ∈ ⋃{if x ∈ FvarsT r then FvarsT t else {} |t x. (t, x) ∈ set txs}
    proof(cases y ∈ FvarsT χ)
      case True note y = True
      hence xx: xx ∈ FvarsT tt using xx by simp
      obtain x where xr: x ∈ FvarsT r
      and yx: (Var y, x) ∈ set (zip (map Var us) (map snd txs))
        using y ynotin unfolding 0 by (auto split: if_splits)
      have yx: (y, x) ∈ set (zip us (map snd txs))
      using yvar us_facts by (intro inj_on_set_zip_map[OF inj_on_Var yx]) auto
      have (tt, x) ∈ set txs apply(rule set_zip_map_fst_snd[OF yx ty])
      using ‹distinct (map snd txs)› us_facts by auto
      thus ?thesis using xx xr by auto
    qed(insert xx, auto)
  next
    fix y tt xx
    assume y: y ∈ (if xx ∈ FvarsT r then FvarsT tt else {})
    and tx: (tt, xx) ∈ set txs
    hence xxsnd: xx ∈ snd ‘ set txs by force
    obtain u where uin: u ∈ set us and uxx: (u, xx) ∈ set (zip us (map snd txs))
      by (metis xxsnd in_set_impl_in_set_zip2 length_map set_map set_zip_leftD us_facts(5))
    hence uvar: u ∈ var using us_facts by auto
    show y ∈ ⋃{if u ∈ FvarsT χ then FvarsT t else {} |t u. (t, u) ∈ set (zip (map fst txs) us)}
    proof(cases xx ∈ FvarsT r)
      case True note xx = True
      hence y: y ∈ FvarsT tt using y by auto
      have (Var u, xx) ∈ set (zip (map Var us) (map snd txs))
      apply(rule set_zip_length_map[OF uxx]) using us_facts by auto
      hence uχ: u ∈ FvarsT χ using uin xx uvar unfolding 0 by auto
      have ttu: (tt, u) ∈ set (zip (map fst txs) us)
      apply(rule set_zip_map_fst_snd2[OF uxx tx]) using assms us_facts by auto
      show ?thesis using uχ ttu y by auto
    qed(insert y, auto)
  qed
  show ?thesis
  by (simp add: psubstT_def Let_def us[symmetric] χ_def[symmetric] 1 2 3)
qed


lemma in_FvarsT_psubstTD:
assumes y ∈ FvarsT (psubstT r txs)
and r ∈ atrm and snd ‘ (set txs) ⊆ var and fst ‘ (set txs) ⊆ atrm
and distinct (map snd txs)
shows y ∈ (FvarsT r − snd ‘ (set txs)) ∪
        (⋃ {if x ∈ FvarsT r then FvarsT t else {} | t x . (t,x) ∈ set txs})
using assms FvarsT_psubstT by auto
```

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*)
using *assms* **by** (*simp add*: *substT_compose_diff*[*of r s t y x*])

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 $rawpsubstT\ r\ ([(s,y),(t,x)]) = rawpsubstT\ r\ ([(t,x),(s,y)])$
using *assms* **by** (*simp add*: *substT2_fresh_switch*)

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 $rawpsubstT\ (rawpsubstT\ t\ txs1)\ txs2 = rawpsubstT\ t\ (txs1\ @\ txs2)$
using *assms* **apply** (*induct txs1 arbitrary*: *txs2 t*)
subgoal by *simp*
subgoal for *tx1 txs1 txs2 t* **apply** (*cases tx1*) **by** *auto* **.**

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 $rawpsubstT\ (substT\ r\ s\ y)\ txs = rawpsubstT\ r\ (txs\ @\ [(s,y)])$
using *assms* **proof**(*induction txs arbitrary*: *r s y*)
case (*Cons tx txs*)
obtain *t x* **where** *tx*[*simp*]: $tx = (t,x)$ **by** *force*
have *x*: $x \in var$ **and** *t*: $t \in trm$ **using** *Cons* **unfolding** *tx* **by** *auto*
have $rawpsubstT\ r\ ((s,\ y)\ \#\ (t,\ x)\ \#\ txs) = rawpsubstT\ r\ ([(s,\ y),(t,\ x)]\ @\ txs)$ **by** *simp*
also have $\ldots = rawpsubstT\ (rawpsubstT\ r\ [(s,\ y),(t,\ x)])\ txs$
using *Cons* **by** *auto*
also have $rawpsubstT\ r\ [(s,\ y),(t,\ x)] = rawpsubstT\ r\ [(t,\ x),(s,\ y)]$
using *Cons* **by** (*intro rawpsubst2_fresh_switch*) *auto*
also have $rawpsubstT\ (rawpsubstT\ r\ [(t,\ x),(s,\ y)])\ txs = rawpsubstT\ r\ ([(t,\ x),(s,\ y)]\ @\ txs)$
using *Cons* **by** (*intro rawpsubstT_compose*) *auto*
also have $\ldots = rawpsubstT\ (substT\ r\ t\ x)\ (txs\ @\ [(s,y)])$ **using** *Cons* **by** *auto*
also have $\ldots = rawpsubstT\ r\ (((t,\ x)\ \#\ txs)\ @\ [(s,\ y)])$ **by** *simp*
finally show *?case* **unfolding** *tx* **by** *auto*
qed *auto*

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 $substT\ (rawpsubstT\ r\ txs)\ s\ y = rawpsubstT\ r\ ((s,y)\ \#\ txs)$
using *assms* **proof**(*induction txs arbitrary*: *r s y*)
case (*Cons tx txs*)
obtain *t x* **where** *tx*[*simp*]: $tx = (t,x)$ **by** *force*
have *x*: $x \in var$ **and** *t*: $t \in trm$ **using** *Cons* **unfolding** *tx* **by** *auto*
have $substT\ (rawpsubstT\ (substT\ r\ t\ x)\ txs)\ s\ y = rawpsubstT\ (substT\ r\ t\ x)\ ((s,y)\ \#\ txs)$
using *Cons.prems* **by** (*intro Cons.IH*) *auto*
also have $\ldots = rawpsubstT\ (rawpsubstT\ r\ [(t,x)])\ ((s,y)\ \#\ txs)$ **by** *simp*

```
  also have ... = rawpsubstT r ([(t,x)] @ ((s,y) # txs))
    using Cons.prems by (intro rawpsubstT_compose) auto
  also have ... = rawpsubstT r ([(t,x),(s,y)] @ txs) by simp
  also have ... = rawpsubstT (rawpsubstT r [(t,x),(s,y)]) txs
    using Cons.prems by (intro rawpsubstT_compose[symmetric]) auto
  also have rawpsubstT r [(t,x),(s,y)] = rawpsubstT r [(s,y),(t,x)]
    using Cons.prems by (intro rawpsubst2_fresh_switch) auto
  also have rawpsubstT (rawpsubstT r [(s,y),(t,x)]) txs = rawpsubstT r ([(s,y),(t,x)] @ txs)
    using Cons.prems by (intro rawpsubstT_compose) auto
  finally show ?case by simp
qed auto

lemma rawpsubstT_compose_freshVar:
assumes r ∈ atrm snd ‘ (set txs) ⊆ var and fst ‘ (set txs) ⊆ atrm
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 ∩ FvarsT r = {}
  set us ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
  set us ∩ 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
using assms proof(induction txs arbitrary: us r)
  case (Cons tx txs uus r)
  obtain t x where tx[simp]: tx = (t,x) by force
  obtain u us where uus[simp]: uus = u # us using Cons by (cases uus) auto
  have us_facts: set us ⊆ var
  set us ∩ FvarsT r = {}
  set us ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
  set us ∩ snd ‘ (set txs) = {}
  length us = length txs
  distinct us and u_facts: u ∈ var u ∉ FvarsT r
  u ∉ ⋃ (FvarsT ‘ (fst ‘ (set txs)))
  u ∉ snd ‘ (set txs) u ∉ set us
    using Cons by auto
  have [simp]: ⋀ bb xaa ab. bb ∈ FvarsT (Var xaa) ⟹
        (ab, bb) ∈ set txs ⟹ xaa ∉ set us
  using us_facts(1,4) by force

  let ?uxs = zip (map Var us) (map snd txs)
  have 1: rawpsubstT (substT r (Var u) x) ?uxs = rawpsubstT r (?uxs @ [(Var u,x)])
  using Cons.prems u_facts apply(intro rawpsubstT_subst_fresh_switch)
  subgoal by auto
  subgoal by (auto dest!: set_zip_D)
  subgoal by (fastforce dest!: set_zip_D)
  subgoal by (auto dest!: set_zip_D)
  subgoal by (fastforce dest!: set_zip_D)
  by (auto dest!: set_zip_D)

  let ?uuxs = zip (map Var uus) (map snd (tx # txs))
  let ?tus = zip (map fst txs) us  let ?ttxs = zip (map fst (tx # txs)) uus
  have 2: u ∈ FvarsT (rawpsubstT r (zip (map Var us) (map snd txs))) ⟹ False
  apply(drule in_FvarsT_rawpsubstTD) apply−
  subgoal using Cons.prems by auto
  subgoal using Cons.prems by (auto dest!: set_zip_D)
  subgoal using Cons.prems  by (force dest!: set_zip_D)
```

```
  subgoal using Cons.prems by (auto dest!: set_zip_D)
  subgoal by (auto dest!: set_zip_D)
  subgoal using us_facts(1,4,5) Cons.prems(7)
    by(fastforce dest!: set_zip_D split: if_splits simp: u_facts(5)) .

  have 3: (tt, xx) ∉ set txs if xx ∈ FvarsT t for tt xx
    unfolding set_conv_nth mem_Collect_eq
  proof safe
    fix i
    assume (tt, xx) = txs ! i i < length txs
    then show False
      using that Cons.prems(4) Cons.prems(5)[of 0 Suc i] tx
      by (auto simp: nth_Cons' split: if_splits dest: sym)
  qed

  have 00: rawpsubstT (rawpsubstT r ?uuxs) ?ttxs = rawpsubstT (substT (rawpsubstT r (?uxs @ [(Var
u, x)])) t u) ?tus
    by (simp add: 1)

  have rawpsubstT r (?uxs @ [(Var u, x)]) = rawpsubstT (rawpsubstT r ?uxs) [(Var u, x)]
    using Cons.prems
    by (intro rawpsubstT_compose[symmetric]) (auto 0 3 dest!: set_zip_D)
  also have rawpsubstT (rawpsubstT r ?uxs) [(Var u, x)] = substT (rawpsubstT r ?uxs) (Var u) x by
simp
  finally have substT (rawpsubstT r (?uxs @ [(Var u, x)])) t u =
              substT (substT (rawpsubstT r ?uxs) (Var u) x) t u by simp
  also have ... = substT (rawpsubstT r ?uxs) t x
    using Cons 2 by (intro substT_substT) (auto 0 3 intro!: rawpsubstT_atrm[of r] dest!: set_zip_D)
  also have ... = rawpsubstT r ((t,x) # ?uxs)
    using Cons.prems 3
    by (intro substT_rawpsubstT_fresh_switch) (auto 0 3 dest!: set_zip_D FvarsT_VarD)
  also have ... =  rawpsubstT r ([(t,x)] @ ?uxs) by simp
  also have ... = rawpsubstT (rawpsubstT r [(t,x)]) ?uxs
    using Cons.prems by (intro rawpsubstT_compose[symmetric]) (auto 0 3 dest!: set_zip_D)
  finally have rawpsubstT (substT (rawpsubstT r (?uxs @ [(Var u, x)])) t u) ?tus =
              rawpsubstT (rawpsubstT (rawpsubstT r [(t,x)]) ?uxs) ?tus by auto
  hence rawpsubstT (rawpsubstT r ?uuxs) ?ttxs = rawpsubstT (rawpsubstT (rawpsubstT r [(t,x)]) ?uxs)
?tus
    using 00 by auto
  also have ... = rawpsubstT (rawpsubstT r [(t,x)]) txs
  using Cons.prems apply(intro Cons.IH)
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by (metis Suc_leI le_imp_less_Suc length_Cons nth_Cons_Suc)
  subgoal by auto
  subgoal by (auto intro!: rawpsubstT dest!: set_zip_D in_FvarsT_substTD
          split: if_splits)
  by auto
  finally show ?case by simp
qed auto

lemma rawpsubstT_compose_freshVar2_aux:
assumes r[simp]: r ∈ atrm
and ts: set ts ⊆ atrm
and xs: set xs ⊆ var  distinct xs
and us_facts: set us ⊆ var  distinct us
```

```
  set us ∩ FvarsT r = {}
  set us ∩ ⋃ (FvarsT ‘ (set ts)) = {}
  set us ∩ set xs = {}
and vs_facts: set vs ⊆ var  distinct vs
  set vs ∩ FvarsT r = {}
  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 rawpsubstT (rawpsubstT r (zip (map Var us) xs)) (zip ts us) =
      rawpsubstT (rawpsubstT r (zip (map Var vs) xs)) (zip ts vs)
using assms proof(induction xs arbitrary: r ts us vs)
  case (Cons x xs r tts uus vvs)
  obtain t ts u us v vs where tts[simp]: tts = t # ts and lts[simp]: length ts = length xs
  and uus[simp]: uus = u # us and lus[simp]: length us = length xs
  and vvs[simp]: vvs = v # vs and lvs[simp]: length vs = length xs
  using ‹length uus = length (x # xs)› ‹length vvs = length (x # xs)› ‹length tts = length (x # xs)›
  apply(cases tts)
  subgoal by auto
  subgoal apply(cases uus)
    subgoal by auto
    subgoal by (cases vvs) auto . .

  let ?rux = substT r (Var u) x   let ?rvx = substT r (Var v) x

  have 0: rawpsubstT (rawpsubstT ?rux (zip (map Var us) xs)) (zip ts us) =
          rawpsubstT (rawpsubstT ?rux (zip (map Var vs) xs)) (zip ts vs)
  using Cons.prems by (intro Cons.IH) (auto intro!: rawpsubstT dest!: set_zip_D simp: FvarsT_substT)

  have 1: rawpsubstT ?rux (zip (map Var vs) xs) =
          substT (rawpsubstT r (zip (map Var vs) xs)) (Var u) x
  using Cons.prems
  by (intro substT_rawpsubstT_fresh_switch[simplified,symmetric])
     (auto intro!: rawpsubstT dest!: set_zip_D simp: subset_eq)

  have 11: rawpsubstT ?rvx (zip (map Var vs) xs) =
          substT (rawpsubstT r (zip (map Var vs) xs)) (Var v) x
  using Cons.prems
  by (intro substT_rawpsubstT_fresh_switch[simplified,symmetric])
     (auto intro!: rawpsubstT dest!: set_zip_D simp: subset_eq)

  have substT (substT (rawpsubstT r (zip (map Var vs) xs)) (Var u) x) t u =
       substT (rawpsubstT r (zip (map Var vs) xs)) t x
  using Cons.prems
  by (intro substT_substT)
     (auto 0 3 intro!: rawpsubstT_atrm[of r]
          dest!: set_zip_D in_FvarsT_rawpsubstT_imp FvarsT_VarD simp: FvarsT_rawpsubstT)
  also have ... = substT (substT (rawpsubstT r (zip (map Var vs) xs)) (Var v) x) t v
  using Cons.prems
  by (intro substT_substT[symmetric])
    (auto 0 3 intro!: rawpsubstT_atrm[of r] dest!: set_zip_D in_FvarsT_rawpsubstT_imp FvarsT_VarD
          simp: FvarsT_rawpsubstT)
  finally have
  2: substT (substT (rawpsubstT r (zip (map Var vs) xs)) (Var u) x) t u =
     substT (substT (rawpsubstT r (zip (map Var vs) xs)) (Var v) x) t v  .

  have rawpsubstT (substT (rawpsubstT ?rux (zip (map Var us) xs)) t u) (zip ts us) =
       substT (rawpsubstT (rawpsubstT ?rux (zip (map Var us) xs)) (zip ts us)) t u
```

using *Cons.prems*
by (*intro substT_rawpsubstT_fresh_switch*[*simplified*,*symmetric*])
(*auto 0 3 intro*!: *rawpsubstT_atrm*[*of ?rux*] *substT_atrm dest*!: *set_zip_D*)
also have ... = *substT* (*rawpsubstT* (*rawpsubstT ?rux* (*zip* (*map Var vs*) *xs*)) (*zip ts vs*)) *t u*
unfolding *0* **..**
also have ... = *rawpsubstT* (*substT* (*rawpsubstT ?rux* (*zip* (*map Var vs*) *xs*)) *t u*) (*zip ts vs*)
using *Cons.prems*
by (*intro substT_rawpsubstT_fresh_switch*[*simplified*])
(*auto 0 3 intro*!: *rawpsubstT_atrm*[*of ?rux*] *dest*!: *set_zip_D*)
also have ... = *rawpsubstT* (*substT* (*substT* (*rawpsubstT r* (*zip* (*map Var vs*) *xs*)) (*Var u*) *x*) *t u*) (*zip ts vs*)
unfolding *1* **..**
also have ... = *rawpsubstT* (*substT* (*substT* (*rawpsubstT r* (*zip* (*map Var vs*) *xs*)) (*Var v*) *x*) *t v*) (*zip ts vs*)
unfolding *2* **..**
also have ... = *rawpsubstT* (*substT* (*rawpsubstT ?rvx* (*zip* (*map Var vs*) *xs*)) *t v*) (*zip ts vs*)
unfolding *11* **..**
finally have *rawpsubstT* (*substT* (*rawpsubstT ?rux* (*zip* (*map Var us*) *xs*)) *t u*) (*zip ts us*) =
rawpsubstT (*substT* (*rawpsubstT ?rvx* (*zip* (*map Var vs*) *xs*)) *t v*) (*zip ts vs*) **.**
thus *?case* **by** *simp*
qed *auto*

lemma *rawpsubstT_compose_freshVar2*:
assumes *r*[*simp*]: *r* ∈ *atrm*
and *ts*: *set ts* ⊆ *atrm*
and *xs*: *set xs* ⊆ *var distinct xs*
and *us_facts*: *set us* ⊆ *var distinct us*
set us ∩ *FvarsT r* = {}
set us ∩ ⋃ (*FvarsT* ‘ (*set ts*)) = {}
set us ∩ *set xs* = {}
and *vs_facts*: *set vs* ⊆ *var distinct vs*
set vs ∩ *FvarsT r* = {}
set vs ∩ ⋃ (*FvarsT* ‘ (*set ts*)) = {}
set vs ∩ *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−
have *ts_trm*: *set ts* ⊆ *trm* **using** *ts* **by** *auto*
define *ws* **where** *ws* = *getFrN* (*xs* @ *us* @ *vs*) (*r* # *ts*) [] (*length xs*)
have *ws_facts*: *set ws* ⊆ *var distinct ws*
set ws ∩ *FvarsT r* = {}
set ws ∩ ⋃ (*FvarsT* ‘ (*set ts*)) = {}
set ws ∩ *set xs* = {} *set ws* ∩ *set us* = {} *set ws* ∩ *set vs* = {}
length ws = *length xs* **using** *assms*(*1*) *ts_trm assms*(*3*−*17*) **unfolding** *ws_def*
using *getFrN_Fvars*[*of xs* @ *us* @ *vs r* # *ts* [] _ *length xs*]
getFrN_FvarsT[*of xs* @ *us* @ *vs r* # *ts* [] _ *length xs*]
getFrN_var[*of xs* @ *us* @ *vs r* # *ts* [] _ *length xs*]
getFrN_length[*of xs* @ *us* @ *vs r* # *ts* [] *length xs*]
apply −
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *force*
subgoal by *force*
subgoal by *force*
subgoal by *force*

by *auto*
have *?L* = *rawpsubstT* (*rawpsubstT r* (*zip* (*map Var ws*) *xs*)) (*zip ts ws*)
using *assms ws_facts* **by** (*intro rawpsubstT_compose_freshVar2_aux*) *auto*
also have ... = *?R*
using *assms ws_facts* **by** (*intro rawpsubstT_compose_freshVar2_aux*) *auto*
finally show *?thesis* **.**
qed

lemma *in_fst_image*: *a* ∈ *fst* ‘ *AB* ⟷ (∃ *b*. (*a*,*b*) ∈ *AB*) **by** *force*

lemma *psubstT_eq_rawpsubstT*:
assumes *r* ∈ *atrm snd* ‘ (*set txs*) ⊆ *var* **and** *fst* ‘ (*set txs*) ⊆ *atrm*
and *distinct* (*map snd txs*)

and ⋀ *i j*. *i* < *j* ⟹ *j* < *length txs* ⟹ *snd* (*txs*!*j*) ∉ *FvarsT* (*fst* (*txs*!*i*))
shows *psubstT r txs* = *rawpsubstT r txs*
proof−
have *txs_trm*: *r* ∈ *trm fst* ‘ (*set txs*) ⊆ *trm* **using** *assms* **by** *auto*

note *frt* = *getFrN_FvarsT*[*of map snd txs r* # *map fst txs* [] _ *length txs*]
and *fr* = *getFrN_Fvars*[*of map snd txs r* # *map fst txs* [] _ *length txs*]
and *var* = *getFrN_var*[*of map snd txs r* # *map fst txs* [] _ *length txs*]
and *l* = *getFrN_length*[*of map snd txs r* # *map fst txs* [] *length txs*]
define *us* **where** *us*: *us* ≡ *getFrN* (*map snd txs*) (*r* # *map fst txs*) [] (*length txs*)
have *us_facts*: *set us* ⊆ *var*
set us ∩ *FvarsT r* = {}
set us ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set us ∩ *snd* ‘ (*set txs*) = {}
length us = *length txs*
distinct us
using *assms*(*2*,*4*,*5*) *txs_trm* **unfolding** *us*

apply −
subgoal by *auto*
subgoal using *frt* **by** *auto*
subgoal using *frt* **by** (*simp add*: *in_fst_image Int_def*) (*metis prod.collapse*)
subgoal using *var* **by** (*simp add*: *in_fst_image Int_def*) (*metis*)
subgoal using *l* **by** *auto*
subgoal by *auto* **.**

show *?thesis*
using *rawpsubstT_compose_freshVar assms us_facts*
by (*simp add*: *psubstT_def Let_def us*[*symmetric*])
qed

lemma *psubstT_eq_substT*:
assumes *r* ∈ *atrm x* ∈ *var* **and** *t* ∈ *atrm*
shows *psubstT r* [(*t*,*x*)] = *substT r t x*
proof−
have *psubstT r* [(*t*,*x*)] = *rawpsubstT r* [(*t*,*x*)]
using *assms* **by** (*intro psubstT_eq_rawpsubstT*) *auto*
thus *?thesis* **by** *auto*
qed

lemma *psubstT_eq_rawpsubst2*:
assumes *r* ∈ *atrm x1* ∈ *var x2* ∈ *var t1* ∈ *atrm t2* ∈ *atrm*

and *x1* ≠ *x2 x2* ∉ *FvarsT t1*
shows *psubstT r* [(*t1*,*x1*),(*t2*,*x2*)] = *rawpsubstT r* [(*t1*,*x1*),(*t2*,*x2*)]
using *assms* **using** *less_SucE* **by** (*intro psubstT_eq_rawpsubstT*) *force*+

lemma *psubstT_eq_rawpsubst3*:
assumes *r* ∈ *atrm x1* ∈ *var x2* ∈ *var x3* ∈ *var t1* ∈ *atrm t2* ∈ *atrm t3* ∈ *atrm*
and *x1* ≠ *x2 x1* ≠ *x3 x2* ≠ *x3*
x2 ∉ *FvarsT t1 x3* ∉ *FvarsT t1 x3* ∉ *FvarsT t2*
shows *psubstT r* [(*t1*,*x1*),(*t2*,*x2*),(*t3*,*x3*)] = *rawpsubstT r* [(*t1*,*x1*),(*t2*,*x2*),(*t3*,*x3*)]
using *assms less_SucE less_Suc_eq_0_disj*
by (*intro psubstT_eq_rawpsubstT*) *auto*

lemma *psubstT_eq_rawpsubst4*:
assumes *r* ∈ *atrm x1* ∈ *var x2* ∈ *var x3* ∈ *var x4* ∈ *var*
t1 ∈ *atrm t2* ∈ *atrm t3* ∈ *atrm t4* ∈ *atrm*
and *x1* ≠ *x2 x1* ≠ *x3 x2* ≠ *x3 x1* ≠ *x4 x2* ≠ *x4 x3* ≠ *x4*
x2 ∉ *FvarsT t1 x3* ∉ *FvarsT t1 x3* ∉ *FvarsT t2 x4* ∉ *FvarsT t1 x4* ∉ *FvarsT t2 x4* ∉ *FvarsT t3*
shows *psubstT r* [(*t1*,*x1*),(*t2*,*x2*),(*t3*,*x3*),(*t4*,*x4*)] = *rawpsubstT r* [(*t1*,*x1*),(*t2*,*x2*),(*t3*,*x3*),(*t4*,*x4*)]
using *assms less_SucE less_Suc_eq_0_disj*
by (*intro psubstT_eq_rawpsubstT*) *auto*

lemma *rawpsubstT_same_Var*[*simp*]:
assumes *r* ∈ *atrm set xs* ⊆ *var*
shows *rawpsubstT r* (*map* (λ*x*. (*Var x*,*x*)) *xs*) = *r*
using *assms* **by** (*induct xs*) *auto*

lemma *psubstT_same_Var*[*simp*]:
assumes *r* ∈ *atrm set xs* ⊆ *var* **and** *distinct xs*
shows *psubstT r* (*map* (λ*x*. (*Var x*,*x*)) *xs*) = *r*
proof−
have *psubstT r* (*map* (λ*x*. (*Var x*,*x*)) *xs*) = *rawpsubstT r* (*map* (λ*x*. (*Var x*,*x*)) *xs*)
using *assms FvarsT_Var*[*of xs* ! _] *nth_mem*[*of* _ *xs*]
by (*intro psubstT_eq_rawpsubstT*)
(*auto simp*: *o_def distinct_conv_nth dest*!: *FvarsT_VarD*)
thus *?thesis* **using** *assms* **by** *auto*
qed

thm *psubstT_notIn*

lemma *rawpsubst_eql*:
assumes *t1* ∈ *trm t2* ∈ *trm*
and *snd* ‘ (*set txs*) ⊆ *var fst* ‘ (*set txs*) ⊆ *trm*
shows *rawpsubst* (*eql t1 t2*) *txs* = *eql* (*rawpsubstT t1 txs*) (*rawpsubstT t2 txs*)
using *assms* **apply** (*induct txs arbitrary*: *t1 t2*)
subgoal by *auto*
subgoal for *tx txs t1 t2* **by** (*cases tx*) *auto* **.**

lemma *psubst_eql*[*simp*]:
assumes *t1* ∈ *atrm t2* ∈ *atrm*
and *snd* ‘ (*set txs*) ⊆ *var fst* ‘ (*set txs*) ⊆ *atrm*
and *distinct* (*map snd txs*)
shows *psubst* (*eql t1 t2*) *txs* = *eql* (*psubstT t1 txs*) (*psubstT t2 txs*)
proof−

have *t12*: *fst ‘ (set txs) ⊆ trm* **using** *assms* **by** *auto*
define *us* **where** *us*: *us ≡ getFrN (map snd txs) (map fst txs) [eql t1 t2] (length txs)*
have *us_facts*: *set us ⊆ var*
set us ∩ FvarsT t1 = {}
set us ∩ FvarsT t2 = {}
set us ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
set us ∩ snd ‘ (set txs) = {}
length us = length txs
distinct us
using *assms(1−3) t12* **unfolding** *us*
using *getFrN_Fvars[of map snd txs map fst txs [eql t1 t2] _ length txs]*
getFrN_FvarsT[of map snd txs map fst txs [eql t1 t2] _ length txs]
getFrN_var[of map snd txs map fst txs [eql t1 t2] _ length txs]
getFrN_length[of map snd txs map fst txs [eql t1 t2] length txs]
apply −
subgoal by *auto*
subgoal by *force*
subgoal by *force*
subgoal by *fastforce*
subgoal by (*fastforce simp*: *image_iff*)
by *auto*

define *vs1* **where** *vs1*: *vs1 ≡ getFrN (map snd txs) (t1 # map fst txs) [] (length txs)*
have *vs1_facts*: *set vs1 ⊆ var*
set vs1 ∩ FvarsT t1 = {}
set vs1 ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
set vs1 ∩ snd ‘ (set txs) = {}
length vs1 = length txs
distinct vs1
using *assms(1−3) t12* **unfolding** *vs1*
using *getFrN_Fvars[of map snd txs t1 # map fst txs [] _ length txs]*
getFrN_FvarsT[of map snd txs t1 # map fst txs [] _ length txs]
getFrN_var[of map snd txs t1 # map fst txs [] _ length txs]
getFrN_length[of map snd txs t1 # map fst txs [] length txs]
apply −
subgoal by *auto*
subgoal by *force*
subgoal by *auto*
subgoal by *force*
subgoal by (*fastforce simp*: *image_iff*)
by *auto*

define *vs2* **where** *vs2*: *vs2 ≡ getFrN (map snd txs) (t2 # map fst txs) [] (length txs)*
have *vs2_facts*: *set vs2 ⊆ var*
set vs2 ∩ FvarsT t2 = {}
set vs2 ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
set vs2 ∩ snd ‘ (set txs) = {}
length vs2 = length txs
distinct vs2
using *assms(1−3) t12* **unfolding** *vs2*
using *getFrN_Fvars[of map snd txs t2 # map fst txs [] _ length txs]*
getFrN_FvarsT[of map snd txs t2 # map fst txs [] _ length txs]
getFrN_var[of map snd txs t2 # map fst txs [] _ length txs]
getFrN_length[of map snd txs t2 # map fst txs [] length txs]
apply −
subgoal by *auto*
subgoal by *force*
subgoal by *auto*

subgoal by *force*
subgoal by (*fastforce simp*: *image_iff*)
by *auto*

let *?tus* = *zip* (*map fst txs*) *us*
let *?uxs* = *zip* (*map Var us*) (*map snd txs*)
let *?e* = *rawpsubst* (*eql t1 t2*) *?uxs*
have *e*: *?e* = *eql* (*rawpsubstT t1 ?uxs*) (*rawpsubstT t2 ?uxs*)
apply(*rule rawpsubst_eql*) **using** *assms us_facts* **apply** *auto*
apply(*drule set_zip_rightD*) **apply** *simp* **apply** *blast*
apply(*drule set_zip_leftD*) **apply** *simp* **apply** *blast* **.**
have *0*: *rawpsubst ?e ?tus* =
eql (*rawpsubstT* (*rawpsubstT t1 ?uxs*) *?tus*) (*rawpsubstT* (*rawpsubstT t2 ?uxs*) *?tus*)
unfolding *e* **using** *assms us_facts* **apply**(*intro rawpsubst_eql*)
subgoal by (*auto intro*!: *rawpsubstT dest*!: *set_zip_D*)
subgoal by (*auto intro*!: *rawpsubstT dest*!: *set_zip_D*)
subgoal by (*auto intro*!: *rawpsubstT dest*!: *set_zip_D*)
subgoal by (*fastforce intro*!: *rawpsubstT dest*!: *set_zip_D*) **.**
have *1*: *rawpsubstT* (*rawpsubstT t1 ?uxs*) *?tus* =
rawpsubstT (*rawpsubstT t1* (*zip* (*map Var vs1*) (*map snd txs*))) (*zip* (*map fst txs*) *vs1*)
using *assms us_facts vs1_facts*
by (*intro rawpsubstT_compose_freshVar2*) *auto*
have *2*: *rawpsubstT* (*rawpsubstT t2 ?uxs*) *?tus* =
rawpsubstT (*rawpsubstT t2* (*zip* (*map Var vs2*) (*map snd txs*))) (*zip* (*map fst txs*) *vs2*)
using *assms us_facts vs2_facts*
by (*intro rawpsubstT_compose_freshVar2*) *auto*
show *?thesis* **unfolding** *psubstT_def psubst_def*
by (*simp add*: *Let_def us*[*symmetric*] *vs1*[*symmetric*] *vs2*[*symmetric*] *0 1 2*)
qed

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−
have *f*: *fst* ‘ *set txs* $\subseteq$ *trm* **using** *assms* **by** (*meson atrm_trm subset_trans*)
note *assms1* = *assms*(*1*−*3*) *assms*(*5*−*7*) *f*
define *u* **where** *u*: $u \equiv$ *getFr* (*x* # *map snd txs*) (*map fst txs*) [φ]
have *u_facts*: $u \in var\ u \neq x$
$u \notin$ *snd* ‘ *set txs* $u \notin (\bigcup t \in fst$ ‘ *set txs*. *FvarsT t*) $u \notin Fvars\ \varphi$
unfolding *u* **using** *f getFr_FvarsT_Fvars*[*of x* # *map snd txs map fst txs* [φ]] **by** (*auto simp*: *assms*)
hence [*simp*]: *psubst* (*subst* φ (*Var u*) *x*) *txs* = *subst* (*psubst* φ *txs*) (*Var u*) *x*
using *assms* **apply**(*intro psubst_subst_fresh_switch f*) **by** *auto*
show *?thesis* **using** *f assms u_facts*
by (*subst exu_def_var*[*of* _ *u psubst* φ *txs*])
(*auto dest*!: *in_Fvars_psubstD split*: *if_splits simp*: *exu_def_var*[*of* _ *u*])
qed

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*
using *assms* **proof**(*induction txs*)
case (*Cons tx txs*)
obtain *t x* **where** *tx*[*simp*]: *tx* = (*t*,*x*) **by** (*cases tx*) *auto*

have *00*: *FvarsT t* ∩ *snd* ‘ *set txs* = {}
using *Cons.prems*(*5*)[*of 0 Suc* _] **by** (*auto simp*: *set_conv_nth*)

have *rawpsubstT* (*substT* (*Var y*) *t x*) *txs* = *s*
proof(*cases y* = *x*)
case [*simp*]:*True* **hence** [*simp*]: *s* = *t* **using** ‹*distinct* (*map snd* (*tx* # *txs*))›
‹(*s*, *y*) ∈ *set* (*tx* # *txs*)› **using** *image_iff* **by** *fastforce*
show *?thesis* **using** *Cons.prems 00* **by** *auto*
next
case *False*
hence [*simp*]: *substT* (*Var y*) *t x* = *Var y*
using *Cons.prems* **by** (*intro substT_notIn*) *auto*
have *rawpsubstT* (*Var y*) *txs* = *s*
using *Cons.prems* **apply**(*intro Cons.IH*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *False* **by** *auto*
subgoal by (*metis length_Cons less_Suc_eq_0_disj nth_Cons_Suc*) **.**
thus *?thesis* **by** *simp*
qed
thus *?case* **by** *simp*
qed *auto*

lemma *psubstT_Var_in*:
assumes *y* ∈ *var snd* ‘ (*set txs*) ⊆ *var fst* ‘ (*set txs*) ⊆ *trm*
and *distinct* (*map snd txs*) **and** (*s*,*y*) ∈ *set txs*
shows *psubstT* (*Var y*) *txs* = *s*
proof−
define *us* **where** *us*: *us* ≡ *getFrN* (*map snd txs*) (*Var y* # *map fst txs*) [] (*length txs*)
have *us_facts*: *set us* ⊆ *var*
set us ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
y ∉ *set us*
set us ∩ *snd* ‘ (*set txs*) = {}
length us = *length txs*
distinct us
using *assms* **unfolding** *us*
using *getFrN_FvarsT*[*of map snd txs Var y* # *map fst txs* [] _ *length txs*]
getFrN_var[*of map snd txs Var y* # *map fst txs* [] _ *length txs*]
getFrN_length[*of map snd txs Var y* # *map fst txs* [] *length txs*]
apply −
subgoal by *auto*
subgoal by *auto*
subgoal by *force*
subgoal by *force*
by *auto*
obtain *i* **where** *i*[*simp*]: *i* < *length txs txs*!*i* = (*s*,*y*) **using** ‹(*s*,*y*) ∈ *set txs*›
by (*metis in_set_conv_nth*)
hence *00*[*simp*]: $\bigwedge j.\ j < length\ txs \Longrightarrow txs\ !\ j = txs\ !\ i \Longrightarrow j = i$
using ‹*distinct* (*map snd txs*)› *distinct_Ex1 nth_mem* **by** *fastforce*
have *000*[*simp*]: $\bigwedge j\ ia.\ j < length\ txs \Longrightarrow ia < length\ txs \Longrightarrow snd\ (txs\ !\ j) \neq us\ !\ ia$
using *assms us_facts*

```
  by (metis IntI empty_iff length_map list.set_map nth_map nth_mem)
 have [simp]: ⋀ii jj. ii < jj ⟹ jj < length txs ⟹ us ! ii ∈ var
  using nth_mem us_facts(1) us_facts(5) by auto
 have [simp]: ⋀i j. i < j ⟹ j < length txs ⟹ us ! j ∉ FvarsT (fst (txs ! i))
  using us_facts(2,5) by (auto simp: Int_def)

 have 0: rawpsubstT (Var y) (zip (map Var us) (map snd txs)) = Var (us!i)
 using assms us_facts
 by (intro rawpsubstT_Var_in)
    (auto dest!: set_zip_D simp: in_set_conv_nth intro!: exI[of _ i])

 have rawpsubstT (rawpsubstT (Var y) (zip (map Var us) (map snd txs))) (zip (map fst txs) us) = s
 unfolding 0 using assms us_facts
 by (intro rawpsubstT_Var_in)
    (auto dest!: set_zip_D simp: in_set_conv_nth intro!: exI[of _ i])
 thus ?thesis unfolding psubstT_def by (simp add: Let_def us[symmetric])
qed

lemma psubstT_Var_Cons_aux:
assumes y ∈ var x ∈ var t ∈ atrm
snd ‘ set txs ⊆ var fst ‘ set txs ⊆ atrm x ∉ snd ‘ set txs
distinct (map snd txs)  y ≠ x
shows psubstT (Var y) ((t, x) # txs) = psubstT (Var y) txs
proof−
 have txs_trm: t ∈ trm fst ‘ set txs ⊆ trm using assms by auto
 note assms1 = assms(1,2) assms(4) assms(6−8) txs_trm

 note fvt = getFrN_FvarsT[of x # map snd txs Var y # t # map fst txs [] _ Suc (length txs)]
 and var = getFrN_var[of x # map snd txs Var y # t # map fst txs [] _ Suc (length txs)]
 and l = getFrN_length[of x # map snd txs Var y # t # map fst txs [] Suc (length txs)]
 define uus where uus:
 uus ≡ getFrN (x # map snd txs) (Var y # t # map fst txs) [] (Suc (length txs))
 have uus_facts: set uus ⊆ var
 set uus ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
 set uus ∩ snd ‘ (set txs) = {}
 set uus ∩ FvarsT t = {}
 x ∉ set uus
 y ∉ set uus
 length uus = Suc (length txs)
 distinct uus
 using assms1 unfolding uus
 apply −
 subgoal by auto
 subgoal using fvt by (simp add: in_fst_image Int_def) (metis prod.collapse)
 subgoal using var by (force simp add: in_fst_image Int_def)
 subgoal using fvt by auto
 subgoal using var by (fastforce simp: in_fst_image Int_def)
 subgoal using fvt by (force simp: in_fst_image Int_def)
 subgoal using l by auto
 subgoal by auto .

 obtain u us where uus_us[simp]: uus = u # us using uus_facts by (cases uus) auto

 have us_facts: set us ⊆ var
 set us ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
 set us ∩ snd ‘ (set txs) = {}
 set us ∩ FvarsT t = {}
 x ∉ set us
```

```
  y ∉ set us
  length us = length txs
  distinct us
  and u_facts: u ∈ var
  u ∉ ⋃ (FvarsT ‘ (fst ‘ (set txs)))
  u ∉ snd ‘ (set txs)
  u ∉FvarsT t
  u ≠ x
  u ≠ y
  u ∉ set us
  using uus_facts by auto

  note fvt = getFrN_FvarsT[of map snd txs Var y # map fst txs [] _ length txs]
  and var = getFrN_var[of map snd txs Var y # map fst txs [] _ length txs]
  and l = getFrN_length[of map snd txs Var y # map fst txs [] length txs]
  define vs where vs: vs ≡ getFrN (map snd txs) (Var y #  map fst txs) [] (length txs)
  have vs_facts: set vs ⊆ var
  set vs ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
  y ∉ set vs
  set vs ∩ snd ‘ (set txs) = {}
  length vs = length txs
  distinct vs
  using assms1 unfolding vs
  apply −
  subgoal by auto
  subgoal using fvt by (simp add: in_fst_image Int_def) (metis prod.collapse)
  subgoal using fvt l by fastforce
  subgoal using var by (force simp: Int_def in_fst_image)
  subgoal using l by auto
  subgoal by auto .

  have 0: substT (Var y) (Var u) x = Var y
    using assms u_facts by auto
  have 1: substT (rawpsubstT (Var y) (zip (map Var us) (map snd txs))) t u =
          rawpsubstT (Var y) (zip (map Var us) (map snd txs))
  using assms u_facts us_facts
  by (intro substT_notIn)
     (auto 0 3 intro!: rawpsubstT dest!: set_zip_D in_FvarsT_rawpsubstT_imp FvarsT_VarD)

  have rawpsubstT (rawpsubstT (Var y) (zip (map Var us) (map snd txs))) (zip (map fst txs) us) =
    rawpsubstT (rawpsubstT (Var y) (zip (map Var vs) (map snd txs))) (zip (map fst txs) vs)
  using assms vs_facts us_facts by (intro rawpsubstT_compose_freshVar2) auto
  thus ?thesis unfolding psubstT_def
  by (simp add: Let_def uus[symmetric] vs[symmetric] 0 1)
qed
```

Simplification rules for parallel substitution:

```
lemma psubstT_Var_Cons[simp]:
y ∈ var ⟹ x ∈ var ⟹ t ∈ atrm ⟹
 snd ‘ set txs ⊆ var ⟹ fst ‘ set txs ⊆ atrm ⟹ distinct (map snd txs) ⟹ x ∉ snd ‘ set txs ⟹
 psubstT (Var y) ((t,x) # txs) = (if y = x then t else psubstT (Var y) txs)
apply(cases y = x)
subgoal by (rule psubstT_Var_in) auto
subgoal by (auto intro!: psubstT_Var_Cons_aux) .

lemma psubstT_zer[simp]:
assumes snd ‘ (set txs) ⊆ var and fst ‘ (set txs) ⊆ trm
shows psubstT zer txs = zer
```

using *assms* **by** (*intro psubstT_num*) *auto*

lemma *rawpsubstT_suc*:
assumes $r \in trm$ **and** $snd\ `\ (set\ txs) \subseteq var$ **and** $fst\ `\ (set\ txs) \subseteq trm$
shows $rawpsubstT\ (suc\ r)\ txs = suc\ (rawpsubstT\ r\ txs)$
using *assms* **apply**(*induct txs arbitrary*: *r*)
subgoal by *simp*
subgoal for *tx txs r* **by** (*cases tx*) *auto* **.**

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−
have *000*: $r \in trm\ fst\ `\ (set\ txs) \subseteq trm$ **using** *assms* **by** *auto*
define *us* **where** *us*: $us \equiv getFrN\ (map\ snd\ txs)\ (suc\ r\ \#\ map\ fst\ txs)\ []\ (length\ txs)$
have *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
using *assms*(*2*) *000* **unfolding** *us*
using *getFrN_FvarsT*[*of map snd txs suc r # map fst txs [] _ length txs*]
getFrN_Fvars[*of map snd txs suc r # map fst txs [] _ length txs*]
getFrN_var[*of map snd txs suc r # map fst txs [] _ length txs*]
getFrN_length[*of map snd txs suc r # map fst txs [] length txs*]
getFrN_length[*of map snd txs suc r # map fst txs [] length txs*]
apply −
subgoal by *auto*
subgoal by *force*
subgoal by *auto*
subgoal by *force*
by *auto*
define *vs* **where** *vs*: $vs \equiv getFrN\ (map\ snd\ txs)\ (r\ \#\ map\ fst\ txs)\ []\ (length\ txs)$
have *vs_facts*: $set\ vs \subseteq var$
$set\ vs \cap FvarsT\ r = \{\}$
$set\ vs \cap \bigcup\ (FvarsT\ `\ (fst\ `\ (set\ txs))) = \{\}$
$set\ vs \cap snd\ `\ (set\ txs) = \{\}$
$length\ vs = length\ txs$
distinct vs
using *assms*(*2*) *000* **unfolding** *vs*
using *getFrN_FvarsT*[*of map snd txs r # map fst txs [] _ length txs*]
getFrN_Fvars[*of map snd txs r # map fst txs [] _ length txs*]
getFrN_var[*of map snd txs r # map fst txs [] _ length txs*]
getFrN_length[*of map snd txs r # map fst txs [] length txs*]
getFrN_length[*of map snd txs r # map fst txs [] length txs*]
apply −
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *force*
by *auto*
have *0*: $rawpsubstT\ (suc\ r)\ (zip\ (map\ Var\ vs)\ (map\ snd\ txs)) =$
$suc\ (rawpsubstT\ r\ (zip\ (map\ Var\ vs)\ (map\ snd\ txs)))$
using *assms vs_facts* **by** (*intro rawpsubstT_suc*) (*auto dest*!: *set_zip_D*)

have $rawpsubstT\ (rawpsubstT\ (suc\ r)\ (zip\ (map\ Var\ us)\ (map\ snd\ txs)))\ (zip\ (map\ fst\ txs)\ us) =$

```
      rawpsubstT (rawpsubstT (suc r) (zip (map Var vs) (map snd txs))) (zip (map fst txs) vs)
  using assms us_facts vs_facts by (intro rawpsubstT_compose_freshVar2) auto
  also have ... = suc (rawpsubstT (rawpsubstT r (zip (map Var vs) (map snd txs))) (zip (map fst txs)
vs))
  unfolding 0 using assms vs_facts apply(intro rawpsubstT_suc)
  subgoal by (auto dest!: set_zip_D intro!: rawpsubstT)
  subgoal by (auto dest!: set_zip_D)
  subgoal by (fastforce dest!: set_zip_D simp: Int_def) .
  finally show ?thesis
  by (simp add: Let_def us[symmetric] vs[symmetric] psubstT_def)
qed

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)
using assms apply(induct txs arbitrary: r1 r2)
subgoal by simp
subgoal for tx txs r by (cases tx) auto .

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−
  have 000: fst ‘ (set txs) ⊆ trm using assms by auto
  define us where us: us ≡ getFrN (map snd txs) (pls r1 r2 # map fst txs) [] (length txs)
  have us_facts: set us ⊆ var
  set us ∩ FvarsT r1 = {}
  set us ∩ FvarsT r2 = {}
  set us ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
  set us ∩ snd ‘ (set txs) = {}
  length us = length txs
  distinct us
  using assms(1−3) 000 unfolding us
  using getFrN_FvarsT[of map snd txs pls r1 r2 # map fst txs [] _ length txs]
      getFrN_Fvars[of map snd txs pls r1 r2 # map fst txs [] _ length txs]
      getFrN_var[of map snd txs pls r1 r2 # map fst txs [] _ length txs]
      getFrN_length[of map snd txs pls r1 r2 # map fst txs [] length txs]
      getFrN_length[of map snd txs pls r1 r2 # map fst txs [] length txs]
  apply −
  subgoal by auto
  subgoal by force
  subgoal by force
  subgoal by auto
  subgoal by force
  by auto
  define vs1 where vs1: vs1 ≡ getFrN (map snd txs) (r1 # map fst txs) [] (length txs)
  have vs1_facts: set vs1 ⊆ var
  set vs1 ∩ FvarsT r1 = {}
  set vs1 ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
  set vs1 ∩ snd ‘ (set txs) = {}
  length vs1 = length txs
  distinct vs1
  using assms(1−3) 000 unfolding vs1
  using getFrN_FvarsT[of map snd txs r1 # map fst txs [] _ length txs]
      getFrN_Fvars[of map snd txs r1 # map fst txs [] _ length txs]
      getFrN_var[of map snd txs r1 # map fst txs [] _ length txs]
      getFrN_length[of map snd txs r1 # map fst txs [] length txs]
```

```
      getFrN_length[of map snd txs r1 # map fst txs [] length txs]
  apply −
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by force
  by auto
  define vs2 where vs2: vs2 ≡ getFrN (map snd txs) (r2 # map fst txs) [] (length txs)
  have vs2_facts: set vs2 ⊆ var
  set vs2 ∩ FvarsT r2 = {}
  set vs2 ∩ ⋃ (FvarsT ‘ (fst ‘ (set txs))) = {}
  set vs2 ∩ snd ‘ (set txs) = {}
  length vs2 = length txs
  distinct vs2
  using assms(1−3) 000 unfolding vs2
  using getFrN_FvarsT[of map snd txs r2 # map fst txs [] _ length txs]
      getFrN_Fvars[of map snd txs r2 # map fst txs [] _ length txs]
      getFrN_var[of map snd txs r2 # map fst txs [] _ length txs]
      getFrN_length[of map snd txs r2 # map fst txs [] length txs]
      getFrN_length[of map snd txs r2 # map fst txs [] length txs]
  apply −
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by force
  by auto
  have 0: rawpsubstT (pls r1 r2) (zip (map Var us) (map snd txs)) =
        pls (rawpsubstT r1 (zip (map Var us) (map snd txs)))
          (rawpsubstT r2 (zip (map Var us) (map snd txs)))
  using assms us_facts by (intro rawpsubstT_pls) (auto dest!: set_zip_D)

  have 1: rawpsubstT (rawpsubstT r1 (zip (map Var us) (map snd txs))) (zip (map fst txs) us) =
        rawpsubstT (rawpsubstT r1 (zip (map Var vs1) (map snd txs))) (zip (map fst txs) vs1)
  using assms us_facts vs1_facts by (intro rawpsubstT_compose_freshVar2) auto

  have 2: rawpsubstT (rawpsubstT r2 (zip (map Var us) (map snd txs))) (zip (map fst txs) us) =
        rawpsubstT (rawpsubstT r2 (zip (map Var vs2) (map snd txs))) (zip (map fst txs) vs2)
  using assms us_facts vs2_facts by (intro rawpsubstT_compose_freshVar2) auto

  have 3: rawpsubstT (rawpsubstT (pls r1 r2) (zip (map Var us) (map snd txs))) (zip (map fst txs) us)
=
    pls (rawpsubstT (rawpsubstT r1 (zip (map Var us) (map snd txs))) (zip (map fst txs) us))
     (rawpsubstT (rawpsubstT r2 (zip (map Var us) (map snd txs))) (zip (map fst txs) us))
  unfolding 0 using assms us_facts apply(intro rawpsubstT_pls)
  subgoal by (auto dest!: set_zip_D intro!: rawpsubstT)
  subgoal by (force dest!: set_zip_D intro!: rawpsubstT simp: Int_def)
  subgoal by (auto dest!: set_zip_D intro!: rawpsubstT)
  subgoal by (fastforce dest!: set_zip_D intro!: rawpsubstT simp: Int_def) .
  show ?thesis unfolding psubstT_def
  by (simp add: Let_def us[symmetric] vs1[symmetric] vs2[symmetric] 1 2 3)
qed

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)
using assms apply(induct txs arbitrary: r1 r2)
subgoal by simp
subgoal for tx txs r by (cases tx) auto .
```

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−
have *000*: *fst* ‘ (*set txs*) ⊆ *trm* **using** *assms* **by** *auto*
define *us* **where** *us*: *us* ≡ *getFrN* (*map snd txs*) (*tms r1 r2* # *map fst txs*) [] (*length txs*)
have *us_facts*: *set us* ⊆ *var*
set us ∩ *FvarsT r1* = {}
set us ∩ *FvarsT r2* = {}
set us ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set us ∩ *snd* ‘ (*set txs*) = {}
length us = *length txs*
distinct us
using *assms*(*1*−*3*) *000* **unfolding** *us*
using *getFrN_FvarsT*[*of map snd txs tms r1 r2* # *map fst txs* [] _ *length txs*]
getFrN_Fvars[*of map snd txs tms r1 r2* # *map fst txs* [] _ *length txs*]
getFrN_var[*of map snd txs tms r1 r2* # *map fst txs* [] _ *length txs*]
getFrN_length[*of map snd txs tms r1 r2* # *map fst txs* [] *length txs*]
getFrN_length[*of map snd txs tms r1 r2* # *map fst txs* [] *length txs*]
apply −
subgoal by *auto*
subgoal by *force*
subgoal by *force*
subgoal by *auto*
subgoal by *force*
by *auto*
define *vs1* **where** *vs1*: *vs1* ≡ *getFrN* (*map snd txs*) (*r1* # *map fst txs*) [] (*length txs*)
have *vs1_facts*: *set vs1* ⊆ *var*
set vs1 ∩ *FvarsT r1* = {}
set vs1 ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set vs1 ∩ *snd* ‘ (*set txs*) = {}
length vs1 = *length txs*
distinct vs1
using *assms*(*1*−*3*) *000* **unfolding** *vs1*
using *getFrN_FvarsT*[*of map snd txs r1* # *map fst txs* [] _ *length txs*]
getFrN_Fvars[*of map snd txs r1* # *map fst txs* [] _ *length txs*]
getFrN_var[*of map snd txs r1* # *map fst txs* [] _ *length txs*]
getFrN_length[*of map snd txs r1* # *map fst txs* [] *length txs*]
getFrN_length[*of map snd txs r1* # *map fst txs* [] *length txs*]
apply −
subgoal by *auto*
subgoal by *force*
subgoal by *auto*
subgoal by *force*
subgoal by *force*
by *auto*
define *vs2* **where** *vs2*: *vs2* ≡ *getFrN* (*map snd txs*) (*r2* # *map fst txs*) [] (*length txs*)
have *vs2_facts*: *set vs2* ⊆ *var*
set vs2 ∩ *FvarsT r2* = {}
set vs2 ∩ ⋃ (*FvarsT* ‘ (*fst* ‘ (*set txs*))) = {}
set vs2 ∩ *snd* ‘ (*set txs*) = {}
length vs2 = *length txs*
distinct vs2
using *assms*(*1*−*3*) *000* **unfolding** *vs2*
using *getFrN_FvarsT*[*of map snd txs r2* # *map fst txs* [] _ *length txs*]
getFrN_Fvars[*of map snd txs r2* # *map fst txs* [] _ *length txs*]

```
    getFrN_var[of map snd txs r2 # map fst txs [] _ length txs]
    getFrN_length[of map snd txs r2 # map fst txs [] length txs]
    getFrN_length[of map snd txs r2 # map fst txs [] length txs]
  apply −
  subgoal by auto
  subgoal by force
  subgoal by auto
  subgoal by force
  subgoal by force
  by auto
  have 0: rawpsubstT (tms r1 r2) (zip (map Var us) (map snd txs)) =
        tms (rawpsubstT r1 (zip (map Var us) (map snd txs)))
          (rawpsubstT r2 (zip (map Var us) (map snd txs)))
  using assms us_facts by (intro rawpsubstT_tms) (auto dest!: set_zip_D)

  have 1: rawpsubstT (rawpsubstT r1 (zip (map Var us) (map snd txs))) (zip (map fst txs) us) =
        rawpsubstT (rawpsubstT r1 (zip (map Var vs1) (map snd txs))) (zip (map fst txs) vs1)
  using assms us_facts vs1_facts by (intro rawpsubstT_compose_freshVar2) auto

  have 2: rawpsubstT (rawpsubstT r2 (zip (map Var us) (map snd txs))) (zip (map fst txs) us) =
        rawpsubstT (rawpsubstT r2 (zip (map Var vs2) (map snd txs))) (zip (map fst txs) vs2)
  using assms us_facts vs2_facts by (intro rawpsubstT_compose_freshVar2) auto

  have 3: rawpsubstT (rawpsubstT (tms r1 r2) (zip (map Var us) (map snd txs))) (zip (map fst txs) us)
=
    tms (rawpsubstT (rawpsubstT r1 (zip (map Var us) (map snd txs))) (zip (map fst txs) us))
     (rawpsubstT (rawpsubstT r2 (zip (map Var us) (map snd txs))) (zip (map fst txs) us))
  unfolding 0 using assms us_facts apply(intro rawpsubstT_tms)
  subgoal by (auto dest!: set_zip_D intro!: rawpsubstT)
  subgoal by (force dest!: set_zip_D intro!: rawpsubstT simp: Int_def)
  subgoal by (auto dest!: set_zip_D intro!: rawpsubstT)
  subgoal by (fastforce dest!: set_zip_D intro!: rawpsubstT simp: Int_def) .

  show ?thesis unfolding psubstT_def
  by (simp add: Let_def us[symmetric] vs1[symmetric] vs2[symmetric] 1 2 3)
qed
```

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)))
unfolding Lq_def using exi_rename[of eql (Var yy) (pls (Var zz) (Var xx)) zz z] by auto

lemma Lq[simp,intro!]: Lq ∈ fmla
unfolding Lq_def by auto

lemma Fvars_Lq[simp]: Fvars Lq = {xx,yy}
unfolding Lq_def by auto
```

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*
using *assms* **unfolding** *LLq_def* **by** *auto*

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

lemma *Fvars_LLq*[*simp*]: *t1* ∈ *trm* ⟹ *t2* ∈ *trm* ⟹
Fvars (*LLq t1 t2*) = *FvarsT t1* ∪ *FvarsT t2*
unfolding *LLq_def* **apply**(*subst Fvars_psubst*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal apply *safe*
subgoal by *auto*
subgoal by *auto*
subgoal by *force*
subgoal by *force*
subgoal by *force*
subgoal by *force* . .

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−
define *z'* **where** *z'* ≡ *getFr* [*xx*,*yy*,*z*] [*t1*,*t2*] []
have *z_facts*[*simp*]: *z'* ∈ *var z'* ≠ *yy z'* ≠ *xx z'* ≠ *z z* ≠ *z' z'* ∉ *FvarsT t1 z'* ∉ *FvarsT t2*
using *getFr_FvarsT_Fvars*[*of* [*xx*,*yy*,*z*] [*t1*,*t2*] []] **unfolding** *z'_def* **by** *auto*
have *LLq t1 t2* = *exi z'* (*eql t2* (*pls* (*Var z'*) *t1*))
by (*simp add*: *LLq_def Lq_def2*[*of z'*])
also have . . . = *exi z* (*eql t2* (*pls* (*Var z*) *t1*))
using *exi_rename*[*of eql t2* (*pls* (*Var z'*) *t1*) *z' z*, *simplified*] .
finally show *?thesis* .
qed

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*))
using *assms* **by** (*intro LLq_pls*) *auto*

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−
define *z* **where** *z* ≡ *getFr* [*xx*,*yy*,*x*] [*t1*,*t2*,*s*] []
have *z_facts*[*simp*]: *z* ∈ *var z* ≠ *xx z* ≠ *yy z* ≠ *x z* ∉ *FvarsT t1 z* ∉ *FvarsT t2 z* ∉ *FvarsT s*
using *getFr_FvarsT_Fvars*[*of* [*xx*,*yy*,*x*] [*t1*,*t2*,*s*] []] **unfolding** *z_def* **by** *auto*
show *?thesis*
by(*simp add*: *FvarsT_substT LLq_pls*[*of* _ _ *z*] *subst2_fresh_switch Lq_def*)
qed

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−
  have 0: t1 ∈ trm t2 ∈ trm fst ‘ set txs ⊆ trm using 1 by auto
  define z where z: z ≡ getFr ([xx,yy] @ map snd txs) ([t1,t2] @ map fst txs) []
  have us_facts: z ∈ var  z ≠ xx z ≠ yy
  z ∉ FvarsT t1 z ∉ FvarsT t2
  z ∉ ⋃ (FvarsT ‘ (fst ‘ (set txs)))
  z ∉ snd ‘ (set txs)
  using 0 2 unfolding z
  using getFr_FvarsT[of [xx,yy] @  map snd txs [t1,t2] @ map fst txs [] ]
        getFr_Fvars[of [xx,yy] @  map snd txs [t1,t2] @ map fst txs [] ]
        getFr_var[of [xx,yy] @  map snd txs [t1,t2] @ map fst txs []]
  apply −
  subgoal by auto
  subgoal by force
  subgoal by force
  subgoal by force
  subgoal by force
  subgoal by auto
  subgoal by (force simp: image_iff) .

  note in_FvarsT_psubstTD[dest!]
  note if_splits[split]
  show ?thesis
  using assms 0 us_facts apply(subst LLq_pls[of _ _ z])
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal apply(subst LLq_pls[of _ _ z]) by auto .
qed
```

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
unfolding Ls_def by auto

lemma Fvars_Ls[simp]: Fvars Ls = {xx,yy}
unfolding Ls_def by auto

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
  using assms unfolding LLs_def by auto

lemma LLs2[simp,intro!]:
n ∈ num ⟹ LLs n (Var yy') ∈ fmla
  by auto

lemma Fvars_LLs[simp]: t1 ∈ trm ⟹ t2 ∈ trm ⟹
```

Fvars (LLs t1 t2) = FvarsT t1 ∪ FvarsT t2
unfolding *LLs_def* **apply**(*subst Fvars_psubst*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal apply *safe*
subgoal by *auto*
subgoal by *auto*
subgoal by *force*
subgoal by *force*
subgoal by *force*
subgoal by *force* **. .**

The working definition of LLs:

lemma *LLs_LLq*:
t1 ∈ atrm ⟹ t2 ∈ atrm ⟹
LLs t1 t2 = cnj (LLq t1 t2) (neg (eql t1 t2))
by (*simp add*: *LLs_def Ls_def LLq_def*)

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)*
by(*simp add*: *LLs_LLq subst2_fresh_switch Ls_def*)

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−
have *0*: *t1 ∈ trm t2 ∈ trm fst ‘ set txs ⊆ trm* **using** *1* **by** *auto*
define *z* **where** *z*: *z ≡ getFr* ([*xx*,*yy*] @ *map snd txs*) ([*t1*,*t2*] @ *map fst txs*) []
have *us_facts*: *z ∈ var z ≠ xx z ≠ yy*
z ∉ FvarsT t1 z ∉ FvarsT t2
z ∉ ⋃ (FvarsT ‘ (fst ‘ (set txs)))
z ∉ snd ‘ (set txs)
using *0 2* **unfolding** *z*
using *getFr_FvarsT*[*of* [*xx*,*yy*] @ *map snd txs* [*t1*,*t2*] @ *map fst txs* []]
getFr_Fvars[*of* [*xx*,*yy*] @ *map snd txs* [*t1*,*t2*] @ *map fst txs* []]
getFr_var[*of* [*xx*,*yy*] @ *map snd txs* [*t1*,*t2*] @ *map fst txs* []]
apply −
subgoal by *auto*
subgoal by *force*
subgoal by *force*
subgoal by *force*
subgoal by *force*
subgoal by *auto*
subgoal by (*force simp*: *image_iff*) **.**
show *?thesis*
using *assms 0 us_facts* **by** (*simp add*: *LLs_LLq*)
qed

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*
unfolding *ball_def* **by** *auto*

lemma *Fvars_ball*[*simp*]:
x ∈ var ⟹ φ ∈ fmla ⟹ t ∈ trm ⟹ Fvars (ball x t φ) = (Fvars φ ∪ FvarsT t) − {x}
unfolding *ball_def* **by** *auto*

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)
unfolding *ball_def* **by** *simp*

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)
unfolding *ball_def* **by** *simp*

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*
unfolding *bexi_def* **by** *auto*

lemma *Fvars_bexi*[*simp*]:
x ∈ var ⟹ φ ∈ fmla ⟹ t ∈ trm ⟹ Fvars (bexi x t φ) = (Fvars φ ∪ FvarsT t) − {x}
unfolding *bexi_def* **by** *auto*

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)
unfolding *bexi_def* **by** *simp*

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 φ) txs = bexi y (psubstT t txs) (psubst φ txs)
unfolding *bexi_def* **by** *simp*

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)
unfolding *ball_def*
by(*nprover2 r1*: *nprv_allI r2*: *nprv_impI*)

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*)
unfolding *ball_def*
by(*nprover3 r1*: *nprv_allE*[*of* _ *x imp* (*LLq* (*Var x*) *t*) φ *t1*]
r2: *nprv_impE0*[*of LLq t1 t subst* φ *t1 x*]
r3: *nprv_mono*[*of F*])

lemma *nprv_ballE*:
nprv F (*ball x t* φ) $\Longrightarrow$ *nprv F* (*LLq t1 t*) $\Longrightarrow$ *nprv* (*insert* (*subst* φ *t1 x*) *F*) $\psi \Longrightarrow$
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
by (*meson atrm_trm local.subst nprv_ballE_aux nprv_cut rev_subsetD*)

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)
unfolding *bexi_def*
by (*nprover2 r1*: *nprv_exiI*[*of* _ _ *t1*] *r2*: *nprv_cnjI*)

lemma *nprv_bexiE*:
nprv F (*bexi x t* φ) $\Longrightarrow$ *nprv* (*insert* (*LLq* (*Var x*) *t*) (*insert* φ *F*)) $\psi \Longrightarrow$
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* $\psi \Longrightarrow$ *x* $\notin$ *FvarsT t* $\Longrightarrow$
nprv F ψ
unfolding *bexi_def*
by (*nprover2 r1*: *nprv_exiE*[*of* _ *x cnj* (*LLq* (*Var x*) *t*) φ] *r2*: *nprv_cnjH*)

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−
have [*simp*]: $\bigwedge\varphi.\ \varphi \in F \Longrightarrow y \notin$ *Fvars* φ **using** *u* **by** *auto*
show *?thesis*
apply(*subst all_rename2*[*of* _ _ *y*])
subgoal by *auto*

subgoal by *auto*
subgoal by *auto*
subgoal using *yx* **by** *auto*
subgoal by (*nrule r*: *nprv_allI*) **.**
qed

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* ⊆ *fmla finite F φ* ∈ *fmla x* ∈ *var y* ∈ *var ψ* ∈ *fmla*
and *yx*: *y* = *x* ∨ *y* ∉ *Fvars φ y* ∉ ⋃ (*Fvars* ‘ *F*) *y* ∉ *Fvars ψ*
shows *nprv F ψ*
proof−
have *e*: *exi x φ* = *exi y* (*subst φ* (*Var y*) *x*)
using *0 yx n nn* **by** (*subst exi_rename2*[*of* _ _ *y*]) (*auto simp*: *0*)
show *?thesis*
using *assms* **unfolding** *e*
by (*auto intro*: *nprv_exiE*[*of* _ *y subst φ* (*Var y*) *x*])
qed

lemma *prv_neg_zer_suc*:
assumes [*simp*]: *t* ∈ *atrm* **shows** *prv* (*neg* (*eql zer* (*suc t*)))
using *prv_psubst*[*OF* _ _ _ *prv_neg_zer_suc_var*, *of* [(*t*,*xx*)]]
by *simp*

lemma *prv_neg_suc_zer*:
assumes *t* ∈ *atrm* **shows** *prv* (*neg* (*eql* (*suc t*) *zer*))
by (*metis assms atrm.simps atrm_imp_trm eql fls neg_def prv_eql_sym*
prv_neg_zer_suc prv_prv_imp_trans zer_atrm)

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*) ⟹ *nprv F* (*eql t* (*suc t1*)) ⟹
finite F ⟹ *F* ⊆ *fmla* ⟹ *t* ∈ *atrm* ⟹ *t1* ∈ *atrm* ⟹ *φ* ∈ *fmla* ⟹
nprv F φ
using *nprv_eql_transI*[*OF nprv_eql_symI*] *nprv_zer_suc_contrE*
by (*meson atrm_imp_trm suc zer_atrm*)

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 ∈ *atrm* ⟹ *t′* ∈ *atrm* ⟹
prv (*imp* (*eql* (*suc t*) (*suc t′*))
(*eql t t′*))

using *prv_psubst*[*OF* _ _ _ *prv_inj_suc_var*, *of* [(*t*,*xx*),(*t'*,*yy*)]]
by *simp*

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*)
(*exi x* (*eql t* (*suc* (*Var x*)))))
proof−
define x' **where** x': $x' \equiv getFr\ [x,yy]\ [t]\ []$
have x'*_facts*[*simp*]: $x' \in var\ x' \neq x\ \ x' \neq yy\ x' \notin FvarsT\ t$ **unfolding** x'
using *getFr_FvarsT_Fvars*[*of* [*x*,*yy*] [*t*] []] **by** *auto*

have *prv* (*imp* (*exi xx* (*eql* (*Var yy*) (*suc* (*Var xx*)))) (*exi x'* (*eql* (*Var yy*) (*suc* (*Var x'*)))))
by (*auto intro*!: *prv_exi_imp prv_all_gen*
simp: *prv_exi_inst*[*of x' eql* (*Var yy*) (*suc* (*Var x'*)) *Var xx*, *simplified*])
with *prv_zer_dsj_suc_var*
have *0*: *prv* (*dsj* (*eql* (*Var yy*) *zer*) (*exi x'* (*eql* (*Var yy*) (*suc* (*Var x'*)))))
by (*elim prv_dsj_cases*[*rotated 3*])
(*auto intro*: *prv_dsj_impL prv_dsj_impR elim*!: *prv_prv_imp_trans*[*rotated 3*])

note *1* = *prv_psubst*[*OF* _ _ _ *0*, *of* [(*t*,*yy*)], *simplified*]
moreover have *prv* (*imp* (*exi x'* (*eql t* (*suc* (*Var x'*)))) (*exi x* (*eql t* (*suc* (*Var x*)))))
by (*auto intro*!: *prv_exi_imp prv_all_gen simp*: *prv_exi_inst*[*of x eql t* (*suc* (*Var x*)) *Var x'*, *simplified*])
ultimately show *?thesis*
by (*elim prv_dsj_cases*[*rotated 3*])
(*auto intro*: *prv_dsj_impL prv_dsj_impR elim*!: *prv_prv_imp_trans*[*rotated 3*])
qed

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 φ
by (*nprover3 r1*: *nprv_addDsjLemmaE*[*OF prv_zer_dsj_suc*]
r2: *nprv_exiE0*[*of x eql t* (*suc* (*Var x*))]
r3: *nprv_mono*[*of insert* (*eql* _ (*suc* _)) _])

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*]: $t \in atrm$ **shows** *prv* (*eql* (*pls t zer*) *t*)
using *prv_psubst*[*OF* _ _ _ *prv_pls_zer_var*, *of* [(*t*,*xx*)]]
by *simp*

lemma *prv_pls_suc*:

$t \in atrm \Longrightarrow t' \in atrm \Longrightarrow$
$prv\ (eql\ (pls\ t\ (suc\ t'))$
$\quad (suc\ (pls\ t\ t')))$
using *prv_psubst*[*OF* _ _ _ *prv_pls_suc_var*, *of* [(*t*,*xx*),(*t'*,*yy*)]]
by *simp*

lemma *prv_tms_zer*:
assumes [*simp*]: $t \in atrm$ **shows** $prv\ (eql\ (tms\ t\ zer)\ zer)$
using *prv_psubst*[*OF* _ _ _ *prv_tms_zer_var*, *of* [(*t*,*xx*)]]
by *simp*

lemma *prv_tms_suc*:
$t \in atrm \Longrightarrow t' \in atrm \Longrightarrow$
$prv\ (eql\ (tms\ t\ (suc\ t'))$
$\quad (pls\ (tms\ t\ t')\ t))$
using *prv_psubst*[*OF* _ _ _ *prv_tms_suc_var*, *of* [(*t*,*xx*),(*t'*,*yy*)]]
by *simp*

lemma *prv_suc_imp_cong*:
assumes *t1*[*simp*]: $t1 \in atrm$ **and** *t2*[*simp*]: $t2 \in atrm$
shows $prv\ (imp\ (eql\ t1\ t2)$
$\quad (eql\ (suc\ t1)\ (suc\ t2)))$
proof−
define *z* **where** $z \equiv getFr\ [xx,yy,zz]\ [t1,t2]\ []$
have *z_facts*[*simp*]: $z \in var\ z \neq xx\ z \neq yy\ z \neq zz\ zz \neq z\ z \notin FvarsT\ t1\ \ z \notin FvarsT\ t2$
using *getFr_FvarsT_Fvars*[*of* [*xx*,*yy*,*zz*] [*t1*,*t2*] []] **unfolding** *z_def*[*symmetric*] **by** *auto*
show *?thesis*
by (*nprover4 r1*: *nprv_prvI r2*: *nprv_impI*
r3: *nprv_eql_substE02*[*of t1 t2* _ *eql* (*suc* (*Var z*)) (*suc t2*) *z*]
r4: *nprv_eq_eqlI*)
qed

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*]: $t1 \in atrm$ **and** *t2*[*simp*]: $t2 \in atrm$
assumes $prv\ (eql\ t1\ t2)$
shows $prv\ (eql\ (suc\ t1)\ (suc\ t2))$
by (*meson assms atrm_suc atrm_imp_trm eql prv_imp_mp prv_suc_imp_cong t1 t2*)

lemma *prv_pls_imp_cong*:
assumes *t1*[*simp*]: $t1 \in atrm$ **and** *t1'*[*simp*]: $t1' \in atrm$
and *t2*[*simp*]: $t2 \in atrm$ **and** *t2'*[*simp*]: $t2' \in atrm$
shows $prv\ (imp\ (eql\ t1\ t1')$
$\quad (imp\ (eql\ t2\ t2')\ (eql\ (pls\ t1\ t2)\ (pls\ t1'\ t2'))))$
proof−
define *z* **where** $z \equiv getFr\ [xx,yy,zz]\ [t1,t1',t2,t2']\ []$
have *z_facts*[*simp*]: $z \in var\ z \neq xx\ z \neq yy\ z \neq zz\ zz \neq z$
$z \notin FvarsT\ t1\ z \notin FvarsT\ t1'\ z \notin FvarsT\ t2\ z \notin FvarsT\ t2'$

using *getFr_FvarsT_Fvars*[*of* [*xx*,*yy*,*zz*] [*t1*,*t1'*,*t2*,*t2'*] []] **unfolding** *z_def*[*symmetric*] **by** *auto*
show *?thesis*
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_impI*)
apply(*nrule r*: *nprv_impI*)
apply(*nrule r*: *nprv_eql_substE02*[*of t1 t1'* _ *eql* (*pls* (*Var z*) *t2*) (*pls t1' t2'*) *z*])
apply(*nrule r*: *nprv_eql_substE02*[*of t2 t2'* _ *eql* (*pls t1'* (*Var z*)) (*pls t1' t2'*) *z*])
apply(*nrule r*: *nprv_eq_eqlI*) **.**
qed

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'*))
by (*metis assms atrm_imp_trm cnj eql pls prv_cnjI prv_cnj_imp_monoR2 prv_imp_mp prv_pls_imp_cong*)

lemma *prv_pls_congL*:
t1 ∈ *atrm* ⟹ *t1'* ∈ *atrm* ⟹ *t2* ∈ *atrm* ⟹
prv (*eql t1 t1'*) ⟹ *prv* (*eql* (*pls t1 t2*) (*pls t1' t2*))
by (*rule prv_pls_cong*[*OF* _ _ _ _ _ *prv_eql_reflT*]) *auto*

lemma *prv_pls_congR*:
t1 ∈ *atrm* ⟹ *t2* ∈ *atrm* ⟹ *t2'* ∈ *atrm* ⟹
prv (*eql t2 t2'*) ⟹ *prv* (*eql* (*pls t1 t2*) (*pls t1 t2'*))
by (*rule prv_pls_cong*[*OF* _ _ _ _ *prv_eql_reflT*]) *auto*

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'*))
unfolding *nprv_def*
by (*auto intro*!: *prv_prv_imp_trans*[*OF* _ _ _ *prv_scnj2_imp_cnj*] *prv_cnj_imp_monoR2 prv_pls_imp_cong*)

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−
define *z* **where** *z* ≡ *getFr* [*xx*,*yy*,*zz*] [*t1*,*t1'*,*t2*,*t2'*] []
have *z_facts*[*simp*]: *z* ∈ *var z* ≠ *xx z* ≠ *yy z* ≠ *zz zz* ≠ *z*
z ∉ *FvarsT t1 z* ∉ *FvarsT t1' z* ∉ *FvarsT t2 z* ∉ *FvarsT t2'*
using *getFr_FvarsT_Fvars*[*of* [*xx*,*yy*,*zz*] [*t1*,*t1'*,*t2*,*t2'*] []] **unfolding** *z_def*[*symmetric*] **by** *auto*
show *?thesis*
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_impI*)
apply(*nrule r*: *nprv_impI*)

apply(*nrule r*: *nprv_eql_substE02*[*of t1 t1′ _ eql* (*tms* (*Var z*) *t2*) (*tms t1′ t2′*) *z*])
apply(*nrule r*: *nprv_eql_substE02*[*of t2 t2′ _ eql* (*tms t1′* (*Var z*)) (*tms t1′ t2′*) *z*])
apply(*nrule r*: *nprv_eq_eqlI*) **.**
qed

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′*))
by (*metis assms atrm_imp_trm cnj eql tms prv_cnjI prv_cnj_imp_monoR2 prv_imp_mp prv_tms_imp_cong*)

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′*))
unfolding *nprv_def*
by (*auto intro*!: *prv_prv_imp_trans*[*OF _ _ _ prv_scnj2_imp_cnj*] *prv_cnj_imp_monoR2 prv_tms_imp_cong*)

lemma *prv_tms_congL*:
t1 ∈ *atrm* ⟹ *t1′* ∈ *atrm* ⟹ *t2* ∈ *atrm* ⟹
prv (*eql t1 t1′*) ⟹ *prv* (*eql* (*tms t1 t2*) (*tms t1′ t2*))
by (*rule prv_tms_cong*[*OF _ _ _ _ _ prv_eql_reflT*]) *auto*

lemma *prv_tms_congR*:
t1 ∈ *atrm* ⟹ *t2* ∈ *atrm* ⟹ *t2′* ∈ *atrm* ⟹
prv (*eql t2 t2′*) ⟹ *prv* (*eql* (*tms t1 t2*) (*tms t1 t2′*))
by (*rule prv_tms_cong*[*OF _ _ _ _ prv_eql_reflT*]) *auto*

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*))
by (*metis* (*no_types*, *opaque_lifting*) *atrm.atrm_pls atrm_imp_trm*
pls prv_eql_trans prv_pls_suc prv_pls_zer prv_suc_cong suc zer_atrm)

lemma *prv_LLq_suc_imp*:
assumes [*simp*]: *t1* ∈ *atrm t2* ∈ *atrm*
shows *prv* (*imp* (*LLq* (*suc t1*) (*suc t2*)) (*LLq t1 t2*))
proof− **define** *z* **where** *z* ≡ *getFr* [*xx*,*yy*,*zz*] [*t1*,*t2*] []
have *z_facts*[*simp*]: *z* ∈ *var z* ≠ *xx z* ≠ *yy z* ≠ *zz zz* ≠ *z z* ∉ *FvarsT t1 z* ∉ *FvarsT t2*
using *getFr_FvarsT_Fvars*[*of* [*xx*,*yy*,*zz*] [*t1*,*t2*] []] **unfolding** *z_def*[*symmetric*] **by** *auto*
note *LLq_pls*[*of _ _ z*,*simp*]
show *?thesis*
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_impI*)

apply(*nrule r*: *nprv_exiE0*)
apply(*nrule r*: *nprv_addLemmaE*[*OF prv_pls_suc*[*of Var z t1*]])
apply(*nrule r*: *nprv_clear3_3*)
apply(*nrule r*: *nprv_eql_transE01*[*of suc t2 pls* (*Var z*) (*suc t1*) _ *suc* (*pls* (*Var z*) *t1*)])
apply(*nrule r*: *nprv_eql_sucE0*[*of t2 pls* (*Var z*) *t1*])
apply(*nrule r*: *nprv_exiI*[*of* _ _ *Var z z*]) .
qed

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*))
by (*simp add*: *LLs_LLq prv_imp_cnjL*)

lemma *prv_LLq_refl*:
prv (*LLq zer zer*)
by (*auto simp*: *LLq_pls_zz prv_pls_zer prv_prv_eql_sym intro*!: *prv_exiI*[*of zz* _ *zer*])

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−
have *assms1*: *t1* ∈ *trm t2* ∈ *trm* **using** *assms* **by** *auto*
define *z* **where** *z* ≡ *getFr* [*xx*,*yy*,*zz*] [*t1*,*t2*] []
have *z_facts*[*simp*]: *z* ∈ *var z* ≠ *xx z* ≠ *yy z* ≠ *zz zz* ≠ *z z* ∉ *FvarsT t1 z* ∉ *FvarsT t2*
using *getFr_FvarsT_Fvars*[*of* [*xx*,*yy*,*zz*] [*t1*,*t2*] []] **using** *assms1* **unfolding** *z_def*[*symmetric*] **by** *auto*
define *x* **where** *x* ≡ *getFr* [*xx*,*yy*,*zz*,*z*] [*t1*,*t2*] []
have *x_facts*[*simp*]: *x* ∈ *var x* ≠ *xx x* ≠ *yy x* ≠ *zz zz* ≠ *x x* ≠ *z z* ≠ *x x* ∉ *FvarsT t1x* ∉ *FvarsT t2*
using *getFr_FvarsT_Fvars*[*of* [*xx*,*yy*,*zz*,*z*] [*t1*,*t2*] []] **using** *assms1* **unfolding** *x_def*[*symmetric*] **by** *auto*
note *assms*[*simp*]
note *LLq_pls*[*of* _ _ *z*, *simp*]
show *?thesis*
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_impI*)
apply(*nrule r*: *nprv_exiE0*[*of z eql t2* (*pls* (*Var z*) *t1*)])
apply(*nrule r*: *nprv_clear2_2*)
apply(*nrule r*: *nprv_exiI*[*of* _ _ *Var z*])
apply(*nrule r*: *nprv_addLemmaE*[*OF prv_pls_suc*[*of Var z t1*]])
apply(*nrule r*: *nprv_eql_substE*[*of* _
pls (*Var z*) (*suc t1*) *suc* (*pls* (*Var z*) *t1*)
eql (*suc t2*) (*Var x*) *x*])
apply(*nrule r*: *nprv_clear2_1*)
apply(*nrule r*: *nprv_suc_congI*) .
qed

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))
apply(*induct n*)
apply (*metis Num Num.simps(1) Num_atrm eql fls in_num neg_def prv_eql_sym prv_neg_zer_suc prv_prv_imp_trans suc*)
by (*metis Num_atrm atrm_imp_trm eql fls neg_def prv_eql_sym prv_neg_zer_suc prv_prv_imp_trans suc zer_atrm*)

lemma *diff_prv_eql_Num*:
assumes $m \neq n$
shows *prv (neg (eql (Num m) (Num n)))*
using *assms* **proof**(*induct m arbitrary*: *n*)
case *0*
then obtain n' **where** *n*: $n = Suc\ n'$ **by** (*cases n*) *auto*
thus *?case* **unfolding** *n* **by** (*simp add*: *prv_neg_zer_suc*)
next
case (*Suc m n*) **note** $s = Suc$
show *?case*
proof(*cases n*)
case *0*
thus *?thesis* **by** (*simp add*: *prv_neg_eql_suc_Num_zer*)
next
case (*Suc* n') **note** $n = Suc$
thus *?thesis* **using** *s*
by *simp* (*meson Num Num_atrm eql in_num neg prv_imp_mp prv_imp_neg_rev prv_inj_suc suc*)
qed
qed

lemma *consistent_prv_eql_Num_equal*:
assumes *consistent* **and** *prv (eql (Num m) (Num n))*
shows $m = n$
using *assms consistent_def3 diff_prv_eql_Num in_num* **by** *blast*

Representability of addition

lemma *prv_pls_zer_zer*:
prv (eql (pls zer zer) zer)
by (*simp add*: *prv_pls_zer*)

lemma *prv_eql_pls_plus*:
prv (eql (pls (Num m) (Num n))
(Num (m+n)))
proof(*induct n*)
case (*Suc n*)
note *0 = prv_pls_suc*[*of Num m Num n, simplified*]
show *?case*
by (*auto intro*: *prv_eql_trans*[*OF _ _ _ 0 prv_suc_cong*[*OF _ _ Suc*]])
qed(*simp add*: *prv_pls_zer*)

lemma *not_plus_prv_neg_eql_pls*:
assumes $m + n \neq k$

shows *prv* (*neg* (*eql* (*pls* (*Num m*) (*Num n*)) (*Num k*)))
using *assms* **proof**(*induction n arbitrary*: *k*)
case *0* **hence** *m*: $m \neq k$ **by** *simp*
note *diff_prv_eql_Num*[*OF m*, *simp*]
show *?case*
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_addLemmaE*[*OF prv_pls_zer*, *of Num m*])
apply(*nrule r*: *nprv_eql_substE*
[*of* _ *pls* (*Num m*) *zer Num m neg* (*eql* (*Var xx*) (*Num k*)) *xx*])
apply(*nrule r*: *prv_nprvI*) **.**
next
case (*Suc n*)
have *0*: $\bigwedge k'.\ k = Suc\ k' \Longrightarrow$
prv (*neg* (*eql* (*pls* (*Num m*) (*Num n*)) (*Num k'*))) $\wedge\ m + n \neq k'$
using *Suc.IH Suc.prems* **by** *auto*
show *?case*
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_addLemmaE*[*OF prv_pls_suc*, *of Num m Num n*])
apply(*nrule r*: *nprv_eql_substE*[*of* _ *pls* (*Num m*) (*suc* (*Num n*))
suc (*pls* (*Num m*) (*Num n*)) *neg* (*eql* (*Var xx*) (*Num k*)) *xx*])
apply(*nrule r*: *nprv_clear*)
apply(*cases k*)
subgoal by (*nprover2 r1*: *prv_nprvI r2*: *prv_neg_suc_zer*)
subgoal for k' **apply**(*frule 0*)
by (*nprover4 r1*: *nprv_addLemmaE r2*: *nprv_negI*
r3: *nprv_negE0 r4*: *nprv_eql_sucI*) **.**
qed

lemma *consistent_prv_eql_pls_plus_rev*:
assumes *consistent prv* (*eql* (*pls* (*Num m*) (*Num n*)) (*Num k*))
shows $k = m + n$
by (*metis Num assms consistent_def eql not_plus_prv_neg_eql_pls num pls prv_neg_fls subsetCE*)

Representability of multiplication

lemma *prv_tms_Num_zer*:
prv (*eql* (*tms* (*Num n*) *zer*) *zer*)
by(*auto simp*: *prv_tms_zer*)

lemma *prv_eql_tms_times*:
prv (*eql* (*tms* (*Num m*) (*Num n*)) (*Num* $(m * n)$))
proof(*induct n*)
case (*Suc n*)
note *0* = *prv_pls_congL*[*OF* _ _ _ *Suc*, *of Num m*, *simplified*]
thm *prv_pls_cong*[*no_vars*]
note *add.commute*[*simp*]
show *?case*
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_addLemmaE*[*OF 0*])
apply(*nrule r*: *nprv_addLemmaE*[*OF prv_tms_suc*[*of Num m Num n*, *simplified*]])
apply(*nrule r*: *nprv_eql_transE01*[*of*
tms (*Num m*) (*suc* (*Num n*))
pls (*tms* (*Num m*) (*Num n*)) (*Num m*) _
pls (*Num* $(m * n)$) (*Num m*)])
apply(*nrule r*: *nprv_clear3_2*)
apply(*nrule r*: *nprv_clear2_2*)
apply(*nrule r*: *nprv_addLemmaE*[*OF prv_eql_pls_plus*[*of* $m * n$ *m*]])
apply(*nrule r*: *nprv_eql_transE01*[*of*
tms (*Num m*) (*suc* (*Num n*))

```
      pls (Num (m * n)) (Num m) _
      Num (m * n + m)]) .
qed(auto simp: prv_tms_zer)

lemma ge_prv_neg_eql_pls_Num_zer:
assumes [simp]: t ∈ atrm and m: m > k
shows prv (neg (eql (pls t (Num m)) (Num k)))
proof−
  define z where z ≡ getFr [xx,yy,zz] [t] []
  have z_facts[simp]: z ∈ var z ≠ xx z ≠ yy z ≠ zz zz ≠ z z ∉ FvarsT t
  using getFr_FvarsT_Fvars[of [xx,yy,zz] [t] []] using assms unfolding z_def[symmetric] by auto
  show ?thesis using m proof(induction k arbitrary: m)
    case (0 m)
    show ?case
    apply(cases m)
    subgoal using 0 by auto
    subgoal for n
      apply(nrule r: nprv_prvI)
      apply(nrule r: nprv_addLemmaE[OF prv_neg_suc_zer[of pls t (Num n)]])
      apply(nrule r: nprv_negI)
      apply(nrule r: nprv_negE0)
      apply(nrule r: nprv_clear2_2)
      apply(nrule r: nprv_eql_symE0)
      apply(nrule r: nprv_eql_substE[of _ zer pls t (suc (Num n)) eql (suc (pls t (Num n))) (Var z) z])
      apply(nrule r: nprv_clear)
      apply(nrule r: nprv_eql_symI)
      apply(nrule r: prv_nprvI)
      apply(nrule r: prv_pls_suc) . .
  next
    case (Suc k mm)
    then obtain m where mm[simp]: mm = Suc m and k: k < m by (cases mm) auto
    show ?case unfolding mm Num.simps
    apply(nrule r: nprv_prvI)
    apply(nrule r: nprv_addLemmaE[OF Suc.IH[OF k]])
    apply(nrule r: nprv_negI)
    apply(nrule r: nprv_negE0)
    apply(nrule r: nprv_clear2_2)
    apply(nrule r: nprv_impI_rev)
    apply(nrule r: nprv_addLemmaE[OF prv_pls_suc[of t Num m]])
    apply(nrule r: nprv_eql_substE[of _ pls t (suc (Num m)) suc (pls t (Num m))
      imp (eql (Var z) (suc (Num k))) (eql (pls t (Num m)) (Num k)) z])
    apply(nrule r: nprv_clear)
    apply(nrule r: nprv_impI)
    apply(nrule r: nprv_eql_sucI) .
   qed
qed

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−
  define z where z ≡ getFr [xx,yy] [t1,t2] []
  have z_facts[simp]: z ∈ var z ≠ xx z ≠ yy z ∉ FvarsT t1 z ∉ FvarsT t2
  using getFr_FvarsT_Fvars[of [xx,yy] [t1,t2] []] unfolding z_def[symmetric] by auto
  show ?thesis proof(induction m)
    case 0
    show ?case unfolding Num.simps
```

```
    apply(nrule r: nprv_prvI)
    apply(nrule r: nprv_addLemmaE[OF prv_pls_zer[of t1]])
    apply(nrule r: nprv_eql_substE[of _ pls t1 zer t1 imp (eql (Var z) (pls t2 zer)) (eql t1 t2) z])
    apply(nrule r: nprv_clear)
    apply(nrule r: nprv_addLemmaE[OF prv_pls_zer[of t2]])
    apply(nrule r: nprv_eql_substE[of _ pls t2 zer t2 imp (eql t1 (Var z)) (eql t1 t2) z])
    apply(nrule r: nprv_impI) .
  next
    case (Suc m)
    note Suc.IH[simp]
    show ?case
    apply(nrule r: nprv_prvI)
    apply(nrule r: nprv_addLemmaE[OF prv_pls_suc[of t1 Num m]])
    apply(nrule r: nprv_eql_substE[of _ pls t1 (suc (Num m)) suc (pls t1 (Num m))
     imp (eql (Var z) (pls t2 (suc (Num m)))) (eql t1 t2) z])
    apply(nrule r: nprv_clear)
    apply(nrule r: nprv_addLemmaE[OF prv_pls_suc[of t2 Num m]])
    apply(nrule r: nprv_eql_substE[of _ pls t2 (suc (Num m)) suc (pls t2 (Num m))
       imp (eql (suc (pls t1 (Num m))) (Var z)) (eql t1 t2) z])
    apply(nrule r: nprv_clear)
    apply(nrule r: nprv_impI)
    apply(nrule r: nprv_eql_sucE0)
    apply(nrule r: nprv_clear2_2)
    apply(nrule r: prv_nprv1I) .
  qed
qed


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 ≠ k
shows prv (neg (eql (tms (Num m) (Num n)) (Num k)))
using assms proof(induction n arbitrary: k)
  case 0 hence m: 0 ≠ k by simp have zer: zer = Num 0 by simp
  have [simp]: prv (neg (eql zer (Num k))) by (subst zer, rule diff_prv_eql_Num[OF m])
  show ?case
  apply(nrule r: nprv_prvI)
  apply(nrule r: nprv_addLemmaE[OF prv_tms_zer, of Num m])
  apply(nrule r: nprv_eql_substE[of _ tms (Num m) zer zer neg (eql (Var xx) (Num k)) xx])
  apply(nrule r: prv_nprvI) .
next
  case (Suc n)
  have [simp]: nprv {} (neg (eql (pls (tms (Num m) (Num n)) (Num m)) (Num k)))
  proof(cases k < m)
    case [simp]: True
    thus ?thesis apply− by (nprover2 r1: prv_nprvI r2: ge_prv_neg_eql_pls_Num_zer)
  next
    case False
    define k' where k' ≡ k − m
```

```
    with False have k: k = k' + m by auto
    hence mm: m * n ≠ k' using False Suc.prems by auto
    note IH = Suc.IH[OF mm]
    show ?thesis unfolding k
    apply(nrule r: nprv_negI)
    apply(nrule r: nprv_addLemmaE[OF prv_prv_eql_sym[OF _ _ prv_eql_pls_plus[of k' m]]])
    apply(nrule r: nprv_eql_transE01[of _ Num (k' + m)])
    apply(nrule r: nprv_clear3_2)
    apply(nrule r: nprv_clear2_2)
    apply(nrule r: nprv_pls_Num_injectE0)
    apply(nrule r: nprv_clear2_2)
    apply(nrule r: nprv_addLemmaE[OF IH])
    apply(nrule r: nprv_negE0) .
  qed
  show ?case
  apply(nrule r: nprv_prvI)
  apply(nrule r: nprv_addLemmaE[OF prv_tms_suc, of Num m Num n])
  apply(nrule r: nprv_eql_substE[of _ tms (Num m) (suc (Num n)) pls (tms (Num m) (Num n)) (Num
m)
     neg (eql (Var xx) (Num k)) xx])
  apply(nrule r: nprv_clear) .
qed

lemma consistent_prv_eql_tms_times_rev:
assumes consistent prv (eql (tms (Num m) (Num n)) (Num k))
shows k = m * n
by (metis Num assms consistent_def eql not_times_prv_neg_eql_tms num tms prv_neg_fls subsetCE)
```

Representability of the order

```
lemma leq_prv_LLq_Num:
assumes m ≤ n
shows prv (LLq (Num m) (Num n))
proof−
  obtain i where n: n = i + m using assms add.commute le_Suc_ex by blast
  note prv_eql_pls_plus[simp]
  have prv (exi zz (eql (Num (i + m)) (pls (Var zz) (Num m))))
  by(nprover2 r1: prv_exiI[of _ _ Num i] r2: prv_prv_eql_sym)
  thus ?thesis unfolding n by (simp add: LLq_pls_zz)
qed
```

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 ∈ atrm
shows prv (LLq zer t)
proof−
  define z where z ≡ getFr [xx,yy] [t] []
  have z_facts[simp]: z ∈ var z ≠ xx z ≠ yy z ∉ FvarsT t
  using getFr_FvarsT_Fvars[of [xx,yy] [t] []] unfolding z_def[symmetric] by auto
  have prv (exi z (eql t (pls (Var z) zer)))
  apply(nrule r: nprv_prvI)
  apply(nrule r: nprv_exiI[of _ _ t])
  apply(nrule r: nprv_eql_symI)
  apply(nrule r: prv_nprvI)
  apply(nrule r: prv_pls_zer) .
```

```
  thus ?thesis by (simp add: LLq_pls[of _ _ z])
qed

lemmas Q1 = prv_LLq_zer

lemma prv_LLq_zer_imp_eql:
assumes [simp]: t ∈ atrm
shows prv (imp (LLq t zer) (eql t zer))
proof−
  define y where y ≡ getFr [] [t] []
  have y_facts[simp]: y ∈ var y ∉ FvarsT t
  using getFr_FvarsT_Fvars[of [] [t] []] unfolding y_def[symmetric] by auto
  define z where z ≡ getFr [y] [t] []
  have z_facts[simp]: z ∈ var z ≠ y z ∉ FvarsT t
  using getFr_FvarsT_Fvars[of [y] [t] []] unfolding z_def[symmetric] by auto
  define x where x ≡ getFr [y,z] [t] []
  have x_facts[simp]: x ∈ var x ≠ y x ≠ z x ∉ FvarsT t
  using getFr_FvarsT_Fvars[of [y,z] [t] []] unfolding x_def by auto
  note LLq_pls[of _ _ z,simp]
  show ?thesis
  apply(nrule r: nprv_prvI)
  apply(nrule r: nprv_impI)
  apply(nrule r: nprv_zer_suc_casesE[of t _ _ y])
  apply(nrule r: nprv_exiE0[of z eql zer (pls (Var z) t)])
  apply(nrule r: nprv_clear3_3)
  apply(nrule r: nprv_eql_symE0[of t])
  apply(nrule r: nprv_eql_substE01[of suc (Var y) t _ eql zer (pls (Var z) (Var x)) x])
  apply(nrule r: nprv_addLemmaE[OF prv_pls_suc[of Var z Var y,simplified]])
  apply(nrule r: nprv_eql_transE01[of zer pls (Var z) (suc (Var y)) _ suc (pls (Var z) (Var y))])
  apply(nrule r: nprv_zer_suc_contrE0[of pls (Var z) (Var y)]) .
qed


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 ∈ atrm
shows prv (imp (ldsj (map (λi. eql t (Num i)) (toN n))) (LLq t (Num n)))
proof−
  define z where z ≡ getFr [xx,yy] [t] []
  have z_facts[simp]: z ∈ var z ≠ xx z ≠ yy z ∉ FvarsT t
  using getFr_FvarsT_Fvars[of [xx,yy] [t] []] unfolding z_def[symmetric] by auto
  note imp[rule del, intro!]  note dsj[intro!]
  show ?thesis
  apply(nrule r: nprv_prvI)
  apply(nrule r: nprv_impI)
  apply(nrule r: nprv_ldsjE0)
  subgoal for i
  apply(nrule r: nprv_eql_substE[of _ t Num i LLq (Var z) (Num n) z])
  subgoal by (nrule r: nprv_hyp)
  subgoal by (nprover3 r1: nprv_addLemmaE[OF leq_prv_LLq_Num])
  subgoal by (nrule r: nprv_hyp)  . .
```

qed

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* ∈ *atrm*
shows *prv* (*imp* (*LLq t* (*Num n*)) (*ldsj* (*map* (λ*i*. *eql t* (*Num i*)) (*toN n*))))
using *assms* **proof**(*induction n arbitrary*: *t*)
  case (*0 t*) **note** *0*[*simp*]
  note *prv_LLq_zer_imp_eql*[*OF 0*,*simp*]
  show *?case*
  by (*nprover4 r1*: *nprv_prvI r2*: *nprv_impI r3*: *nprv_ldsjI r4*: *prv_nprv1I*)
next
  case (*Suc n*) **note** *t*[*simp*] = ‹*t* ∈ *atrm*›
  define *z* **where** *z* ≡ *getFr* [*xx*,*yy*,*zz*] [*t*] []
  have *z_facts*[*simp*]: *z* ∈ *var z* ≠ *xx z* ≠ *yy z* ≠ *zz zz* ≠ *z z* ∉ *FvarsT t*
  using *getFr_FvarsT_Fvars*[*of* [*xx*,*yy*,*zz*] [*t*] []] **unfolding** *z_def*[*symmetric*] **by** *auto*
  note *subset_eq*[*simp*]
  have [*simp*]: *eql t zer* ∈ (λ*x*. *eql t* (*Num x*)) ‘ {*0*..*Suc n*} **by** (*force simp*: *image_def*)
  have [*simp*]: ⋀*i*. *i* ≤ *n* ⟹
    eql (*suc* (*Var z*)) (*suc* (*Num i*)) ∈ (λ*x*. *eql* (*suc* (*Var z*)) (*Num x*)) ‘ {*0*..*Suc n*}
  by (*auto simp*: *image_def intro*!: *bexI*[*of _ Suc _*])
  show *?case*
  apply(*nrule r*: *nprv_prvI*)
  apply(*nrule2 r*: *nprv_zer_suc_casesE*[*of t _ _ z*])
  subgoal by (*nprover3 r1*: *nprv_impI r2*: *nprv_clear2_1 r3*: *nprv_ldsjI*)
  subgoal
    apply(*nrule r*: *nprv_eql_substE*[*of _ t suc* (*Var z*)
        imp (*LLq* (*Var xx*) (*suc* (*Num n*))) (*ldsj* (*map* (λ*i*. *eql* (*Var xx*) (*Num i*)) (*toN* (*Suc n*)))) *xx*])
    apply(*nrule r*: *nprv_clear*)
    apply(*nrule r*: *nprv_impI*)
    apply(*nrule r*: *nprv_LLq_sucE0*)
    apply(*nrule r*: *nprv_addImpLemmaE*[*OF Suc.IH*[*of Var z*, *simplified*]])
    apply(*nrule r*: *nprv_ldsjE0*)
    subgoal for *i* **apply**(*nrule r*: *nprv_ldsjI*[*of _ eql* (*suc* (*Var z*)) (*suc* (*Num i*))])
    apply(*nrule r*: *nprv_suc_congI*) . . .
qed

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 ¬ $m \leq n$
shows *prv* (*neg* (*LLq* (*Num m*) (*Num n*)))
proof−
  have [*simp*]: $\bigwedge i.\ i \leq n \Longrightarrow$ *prv* (*imp* (*eql* (*Num m*) (*Num i*)) *fls*)
  unfolding *neg_def*[*symmetric*]
  using *assms* **by** (*intro diff_prv_eql_Num*) *simp*
  show *?thesis*
  apply(*nrule r*: *nprv_prvI*)
  apply(*nrule r*: *nprv_negI*)
  apply(*nrule r*: *O3E0*)
  apply(*nrule r*: *nprv_ldsjE0*)
  apply(*nrule r*: *nprv_clear3_2*)
  apply(*nrule r*: *nprv_clear2_2*)
  apply(*nrule r*: *prv_nprv1I*) **.**
qed

lemma *consistent_prv_LLq_Num_leq*:
assumes *consistent prv* (*LLq* (*Num m*) (*Num n*))
shows $m \leq n$
by (*metis Num assms consistent_def LLq not_leq_prv_neg_LLq_Num num prv_neg_fls subsetCE*)

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)
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_ballI*)
apply(*nrule r*: *O3E0*)
apply(*nrule r*: *nprv_clear2_2*)
apply(*nrule r*: *nprv_ldsjE0*)
apply(*nrule r*: *nprv_clear2_2*)
apply(*nrule r*: *nprv_eql_substE*[*of _ Var x Num _* φ *x*])
apply(*nrule r*: *prv_nprvI*) **.**

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* φ (*Num i*) *x*)
shows *prv* (*bexi x* (*Num n*) φ)
proof−
  note *leq_prv_LLq_Num*[*simp*]
  show *?thesis*
  by (*nprover4 r1*: *nprv_prvI r2*: *nprv_bexiI*[*of _ _ Num i*] *r3*: *prv_nprvI r4*: *prv_nprvI*)

qed

lemmas *O5* = *prv_bexi_NumI*

lemma *prv_LLq_Num_imp_Suc*: — O6
assumes [*simp*]: *t* ∈ *atrm*
shows *prv* (*imp* (*LLq t* (*Num n*)) (*LLq t* (*suc* (*Num n*))))
proof−
have [*simp*]: ⋀*i*. *i* ≤ *n* ⟹ *prv* (*LLq* (*Num i*) (*suc* (*Num n*)))
apply(*subst Num.simps*(*2*)[*symmetric*])
by (*rule leq_prv_LLq_Num*) *simp*
show *?thesis*
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_impI*)
apply(*nrule r*: *O3E0*)
apply(*nrule r*: *nprv_clear2_2*)
apply(*nrule r*: *nprv_ldsjE0*)
apply(*nrule r*: *nprv_clear2_2*)
apply(*nrule r*: *nprv_eql_substE*[*of* _ *t Num* _ *LLq* (*Var xx*) (*suc* (*Num n*)) *xx*])
apply(*nrule r*: *prv_nprvI*) .
qed

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* ∈ *atrm*
shows *prv* (*LLq* (*suc* (*Num n*)) (*pls* (*suc t*) (*Num n*)))
proof−
define *z* **where** *z* ≡ *getFr* [*xx*,*yy*,*zz*] [*t*] []
have *z_facts*[*simp*]: *z* ∈ *var z* ≠ *xx z* ≠ *yy z* ≠ *zz zz* ≠ *z z* ∉ *FvarsT t*
using *getFr_FvarsT_Fvars*[*of* [*xx*,*yy*,*zz*] [*t*] []] **unfolding** *z_def*[*symmetric*] **by** *auto*
show *?thesis*
proof(*induction n*)
case *0*
have *prv* (*exi z* (*eql* (*pls* (*suc t*) *zer*) (*pls* (*Var z*) (*suc zer*))))
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_exiI*[*of* _ _ *t*])
apply(*nrule r*: *nprv_addLemmaE*[*OF prv_pls_zer*[*of suc t*]])
apply(*nrule r*: *nprv_eql_substE*[*of* _ *pls* (*suc t*) *zer suc t eql* (*Var z*) (*pls t* (*suc zer*)) *z*])
apply(*nrule r*: *nprv_clear*)
apply(*nrule r*: *nprv_eql_symI*)
apply(*nrule r*: *prv_nprvI*)
apply(*nrule r*: *prv_pls_suc_zer*) .

```
    thus ?case by (simp add: LLq_pls[of _ _ z])
  next
    case (Suc n)
    have nn: suc (Num n) = suc (Num n) by simp
    note Suc.IH[simp]
    show ?case
    apply(nrule r: nprv_prvI)
    apply(nrule r: nprv_addLemmaE[OF prv_pls_suc[of suc t Num n]])
    apply(nrule r: nprv_eql_substE[of _ pls (suc t) (suc (Num n)) suc (pls (suc t) (Num n))
      LLq (suc (suc (Num n))) (Var z) z])
    apply(nrule r: nprv_clear)
    apply(nrule r: nprv_suc_mono_LLqI)
    apply(nrule r: prv_nprvI) .
  qed
qed

lemma prv_Num_LLq_imp_eql_suc: — O7
assumes [simp]: t ∈ atrm
shows prv (imp (LLq (Num n) t)
               (dsj (eql (Num n) t)
                    (LLq (suc (Num n)) t)))
proof−
  define z where z ≡ getFr [xx,yy,zz] [t] []
  have z_facts[simp]: z ∈ var z ≠ xx z ≠ yy z ≠ zz zz ≠ z z ∉ FvarsT t
  using getFr_FvarsT_Fvars[of [xx,yy,zz] [t] []] unfolding z_def[symmetric] by auto
  define x where x ≡ getFr [xx,yy,zz,z] [t] []
  have x_facts[simp]: x ∈ var x ≠ xx x ≠ yy x ≠ zz zz ≠ x x ≠ z z ≠ x x ∉ FvarsT t
  using getFr_FvarsT_Fvars[of [xx,yy,zz,z] [t] []] unfolding x_def[symmetric] by auto
  define y where y ≡ getFr [x,z] [t] []
  have y_facts[simp]: y ∈ var y ∉ FvarsT t x ≠ y y ≠ x z ≠ y y ≠ z
  using getFr_FvarsT_Fvars[of [x,z] [t] []] unfolding y_def[symmetric] by auto
  have [simp]: prv (eql (pls zer (Num n)) (Num n))
  by (subst Num.simps(1)[symmetric]) (metis plus_nat.add_0 prv_eql_pls_plus)
  have [simp]: prv (LLq (suc (Num n)) (pls (suc (Var x)) (Num n)))
  by (simp add: prv_LLq_suc_Num_pls_Num)

  note LLq_pls[of Num n t z, simplified, simp]
  show ?thesis
  apply(nrule r: nprv_prvI)
  apply(nrule r: nprv_impI)
  apply(nrule r: nprv_exiE0)
  apply(nrule r: nprv_clear2_2)
  apply(nrule r: nprv_zer_suc_casesE[of Var z _ _ x])
  subgoal
    apply(nrule r: nprv_dsjIL)
    apply(nrule r: nprv_impI_rev2[of {eql (Var z) zer} eql t (pls (Var z) (Num n))])
    apply(nrule r: nprv_eql_substE
      [of _ Var z zer imp (eql t (pls (Var y) (Num n))) (eql (Num n) t) y])
    apply(nrule r: nprv_clear)
    apply(nrule r: nprv_impI)
    apply(nrule r: nprv_eql_substE[of _ t pls zer (Num n) eql (Num n) (Var y) y])
    apply(nrule r: nprv_clear)
    apply(nrule r: prv_nprvI)
    apply(nrule r: prv_prv_eql_sym) .
  subgoal
    apply(nrule r: nprv_dsjIR)
    apply(nrule r: nprv_impI_rev2[of {eql (Var z) (suc (Var x))} eql t (pls (Var z) (Num n))])
    apply(nrule r: nprv_eql_substE
```

[*of* _ *Var z suc* (*Var x*) *imp* (*eql t* (*pls* (*Var y*) (*Num n*))) (*LLq* (*suc* (*Num n*)) *t*) *y*])

apply(*nrule r*: *nprv_clear*)
apply(*nrule r*: *nprv_impI*)
apply(*nrule r*: *nprv_eql_substE*
[*of* _ *t pls* (*suc* (*Var x*)) (*Num n*) *LLq* (*suc* (*Num n*)) (*Var y*) *y*])
apply(*nrule r*: *prv_nprvI*) . .
qed

lemma *prv_Num_LLq_eql_sucE*:
nprv F (*LLq* (*Num n*) *t*) ⟹
nprv (*insert* (*eql* (*Num n*) *t*) *F*) *ψ* ⟹
nprv (*insert* (*LLq* (*suc* (*Num n*)) *t*) *F*) *ψ* ⟹
t ∈ *atrm* ⟹ *F* ⊆ *fmla* ⟹ *finite F* ⟹ *ψ* ∈ *fmla* ⟹
nprv F ψ
apply(*nrule r*: *nprv_addImpLemmaE*[*OF prv_Num_LLq_imp_eql_suc*])
apply(*nrule2 r*: *nprv_dsjE0*[*of eql* (*Num n*) *t LLq* (*suc* (*Num n*)) *t*])
subgoal by (*nrule r*: *nprv_mono*[*of insert* (*eql* (*Num n*) *t*) *F*])
subgoal by (*nrule r*: *nprv_mono*[*of insert* (*LLq* (*suc* (*Num n*)) *t*) *F*]) .

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* ∈ *atrm*
shows *prv* (*dsj* (*eql t* (*Num n*)) (*neg* (*eql t* (*Num n*))))
using *assms* **proof**(*induction n arbitrary*: *t*)
case [*simp*]:(*0 t*)
define *z* **where** *z* ≡ *getFr* [*xx*,*yy*,*zz*] [*t*] []
have *z_facts*[*simp*]: *z* ∈ *var z* ≠ *xx z* ≠ *yy z* ≠ *zz zz* ≠ *z z* ∉ *FvarsT t*
using *getFr_FvarsT_Fvars*[*of* [*xx*,*yy*,*zz*] [*t*] []] **unfolding** *z_def*[*symmetric*] **by** *auto*
show *?case*
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_zer_suc_casesE*[*of t* _ _ *z*])
subgoal by (*nrule r*: *nprv_dsjIL*)
subgoal by (*nprover3 r1*: *nprv_dsjIR r2*: *nprv_negI r3*: *nprv_zer_suc_2contrE01*) .
next
case (*Suc n*) **note** ‹*t* ∈ *atrm*›[*simp*]
define *z* **where** *z* ≡ *getFr* [*xx*,*yy*,*zz*] [*t*] []
have *z_facts*[*simp*]: *z* ∈ *var z* ≠ *xx z* ≠ *yy z* ≠ *zz zz* ≠ *z z* ∉ *FvarsT t*
using *getFr_FvarsT_Fvars*[*of* [*xx*,*yy*,*zz*] [*t*] []] **unfolding** *z_def*[*symmetric*] **by** *auto*
show *?case*
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_zer_suc_casesE*[*of t* _ _ *z*])
subgoal by (*nprover3 r1*: *nprv_dsjIR r2*: *nprv_negI r3*: *nprv_zer_suc_2contrE01*)

```
  subgoal
    apply(nrule r: nprv_eql_substE [of _ t suc (Var z)
      dsj (eql (Var z) (suc (Num n))) (neg (eql (Var z) (suc (Num n)))) z])
    apply(nrule r: nprv_clear)
    apply(nrule r: nprv_addLemmaE[OF Suc.IH[of Var z]])
    apply(nrule r: nprv_dsjE0)
    subgoal by (nprover2 r1: nprv_dsjIL r2: nprv_suc_congI)
    subgoal by (nprover4 r1: nprv_dsjIR r2: nprv_negI r3: nprv_negE0 r4: nprv_eql_sucI) . .
qed


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 ∈ atrm
shows prv (dsj (LLq t (Num n)) (LLq (Num n) t))
proof(induction n)
  case 0
  note prv_LLq_zer[simp]
  show ?case by (nprover3 r1: nprv_prvI r2: nprv_dsjIR r3: prv_nprvI)
next
  case (Suc n)
  have nn: suc (Num n) = Num (Suc n) by simp
  have [simp]: prv (LLq (Num n) (suc (Num n)))
  apply(subst nn) by (rule leq_prv_LLq_Num) simp
  show ?case
  apply(nrule r: nprv_prvI)
  apply(nrule r: nprv_addLemmaE[OF Suc.IH])
  apply(nrule r: nprv_dsjE0)
  subgoal by (nprover2 r1: nprv_dsjIL r2: O6I)
  subgoal
    apply(nrule r: nprv_clear2_2)
    apply(nrule2 r: nprv_eql_Num_casesE[of t n])
    subgoal by (nprover3 r1: nprv_dsjIL
      r2: nprv_eql_substE[of _ t Num n LLq (Var xx) (suc (Num n)) xx]
      r3: prv_nprvI)
    subgoal
      apply(nrule r: O7E0[of n t])
      subgoal by (nprover2 r1: nprv_eql_symE0 r2: nprv_negE01 )
      subgoal by (nrule r: nprv_dsjIR) . . .
qed


lemma prv_LLq_Num_casesE:
nprv (insert (LLq t (Num n)) F) ψ ⟹
 nprv (insert (LLq (Num n) t) F) ψ ⟹
 t ∈ atrm ⟹ F ⊆ fmla ⟹ finite F ⟹ ψ ∈ fmla ⟹
 nprv F ψ
by (rule nprv_addDsjLemmaE[OF prv_LLq_Num_dsj]) auto

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* ∈ *atrm*
shows *prv* (*imp* (*LLq t* (*suc* (*Num n*)))
(*imp* ((*neg* (*eql t* (*suc* (*Num n*)))))
(*LLq t* (*Num n*))))
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_impI*)
apply(*nrule r*: *nprv_impI*)
apply(*nrule r*: *O3E0*[*of t Suc n*])
apply(*nrule r*: *nprv_clear3_3*)
apply(*nrule r*: *nprv_ldsjE0*)
subgoal for *i*
apply(*nrule r*: *nprv_clear3_2*)
apply(*nrule r*: *nprv_impI_rev2*[*of* {*eql t* (*Num i*)} *neg* (*eql t* (*suc* (*Num n*)))])
apply(*nrule r*: *nprv_eql_substE*[*of* _ *t Num i*
imp (*neg* (*eql* (*Var xx*) (*suc* (*Num n*)))) (*LLq* (*Var xx*) (*Num n*)) *xx*])
apply(*nrule r*: *nprv_clear*)
apply(*nrule r*: *nprv_impI*)
apply(*cases i* = *Suc n*)
subgoal by (*nprover2 r1*: *nprv_negE0 r2*: *nprv_eql_reflI*)
subgoal by (*nprover2 r1*: *prv_nprvI r2*: *leq_prv_LLq_Num*) . .

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* ∈ *var* φ ∈ *fmla*
shows *prv* (*imp* (*ball x* (*Num n*) φ)
(*ball x* (*suc* (*Num n*)) (*imp* (*neg* (*eql* (*Var x*) (*suc* (*Num n*)))) φ)))
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *nprv_impI*)
apply(*nrule r*: *nprv_ballI*)
apply(*nrule r*: *nprv_impI*)
apply(*nrule r*: *nprv_LLq_neg_Num_sucE01*)
apply(*nrule r*: *nprv_clear4_2*)
apply(*nrule r*: *nprv_clear3_2*)
apply(*nrule r*: *nprv_ballE0*[*of x Num n* φ _ *Var x*]) .

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*]: *φ* ∈ *fmla* *Fvars φ* = {*zz*} **and** *q*: *q* ∈ *num* **and** *p*: ∀ *p* ∈ *num*. *prv* (*subst φ p zz*)
shows *prv* (*all zz* (*imp* (*LLq* (*Var zz*) *q*) *φ*))
proof−
obtain *n* **where** *q*: *q* = *Num n* **using** *q num_Num* **by** *auto*
— NB: We did not need the whole strength of the assumption p – we only needed that to hold for numerals smaller than n. However, the abstract framework allowed us to make this strong assumption, and did not need to even assume an order on the numerals.
show *?thesis* **unfolding** *q ball_def*[*symmetric*] **using** *p p num_Num* **by** (*intro O4*) *auto*
qed

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−
obtain *n* **where** *q*[*simp*]: *p* = *Num n* **using** *assms num_Num* **by** *auto*
have [*simp*]: {*eql* (*Var yy*) *r* |*r*. ∃ *i*. *r* = *Num i* ∧ *i* ≤ *n*} ⊆ *fmla* **by** *auto*
show *?thesis*
apply(*nrule r*: *exI*[*of _* {*Num i* | *i* . *i* ≤ *n*}])
apply(*nrule r*: *nprv_prvI*)
apply(*nrule r*: *O8E*[*of Var yy n*])
subgoal
apply(*nrule r*: *nprv_dsjIL*)
apply(*nrule r*: *O3E0*)
apply(*nrule r*: *nprv_ldsjE0*)
apply(*nrule r*: *nprv_sdsjI*[*of _ eql* (*Var yy*) (*Num _*)])
apply(*nrule r*: *nprv_hyp*) **.**
subgoal by (*nrule r*: *nprv_dsjIR*) **.**
qed

end — context *Deduct_Q*

sublocale *Deduct_Q* < *lab*: *Deduct_with_PseudoOrder* **where** *Lq* = *LLq* (*Var zz*) (*Var yy*)
apply *standard* **apply** *auto*[] **using** *Fvars_Lq* **apply** *auto*[]
using *LLq_num LLq_num2* **apply** *auto*
done

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.