

Syntax-Independent Logic Infrastructure

Andrei Popescu Dmitriy Traytel

February 23, 2021

Abstract

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

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

Contents

1 Preliminaries	3
1.1 Trivia	3
1.2 Some Proof Infrastructure	4
2 Syntax	6
2.1 Generic Syntax	6
2.1.1 Instance Operator	8
2.1.2 Fresh Variables	9
2.1.3 Parallel Term Substitution	10
2.1.4 Parallel Formula Substitution	12
2.2 Adding Numerals to the Generic Syntax	25
2.3 Adding Connectives and Quantifiers	27
2.3.1 Iterated conjunction	38
2.3.2 Parallel substitution versus the new connectives	39
2.4 Adding Disjunction	40
2.4.1 Iterated disjunction	40
2.4.2 Parallel substitution versus the new connectives	41
2.5 Adding an Ordering-Like Formula	43
2.6 Allowing the Renaming of Quantified Variables	44
2.7 The Exists-Unique Quantifier	45
3 Deduction	47
3.1 Positive Logic Deduction	47
3.1.1 Properties of the propositional fragment	48
3.1.2 Properties involving quantifiers	56
3.1.3 Properties concerning equality	60
3.1.4 The equivalence between soft substitution and substitution	62
3.2 Deduction Considering False	62
3.2.1 Basic properties of False (fls)	63
3.2.2 Properties involving negation	63
3.2.3 Properties involving True (tru)	66
3.2.4 Property of set-based conjunctions	66
3.2.5 Consistency and ω -consistency	72
3.3 Deduction Considering False and Disjunction	74
3.3.1 Disjunction vs. disjunction	75
3.3.2 Disjunction vs. conjunction	76
3.3.3 Disjunction vs. True and False	76
3.3.4 Set-based disjunctions	77
3.4 Deduction with Quantified Variable Renaming Included	82
3.5 Deduction with PseudoOrder Axioms Included	82

4	Natural Deduction	84
4.1	Natural Deduction from the Hilbert System	84
4.2	Structural Rules for the Natural Deduction Relation	84
4.3	Back and Forth between Hilbert and Natural Deduction	85
4.4	More Structural Properties	85
4.5	Properties Involving Substitution	87
4.6	Introduction and Elimination Rules	88
4.7	Adding Lemmas of Various Shapes into the Proof Context	93
4.8	Rules for Equality	95
4.9	Other Rules	95
4.10	Natural Deduction for the Exists-Unique Quantifier	96
4.11	Eisbach Notation for Natural Deduction Proofs	98
5	Pseudo-Terms	100
5.1	Basic Setting	100
5.2	The \forall - \exists Equivalence	101
5.3	Instantiation	103
5.3.1	Instantiation with terms	103
5.3.2	Instantiation with pseudo-terms	104
5.3.3	Closure and compositionality properties of instantiation	104
5.4	Equality between Pseudo-Terms and Terms	108
6	Truth in a Standard Model	111
7	Arithmetic Constructs	114
7.1	Arithmetic Terms	116
7.2	The (Nonstrict and Strict) Order Relations	138
7.3	Bounded Quantification	141
8	Deduction in a System Embedding the Intuitionistic Robinson Arithmetic	143
8.1	Natural Deduction with the Bounded Quantifiers	143
8.2	Deduction with the Robinson Arithmetic Axioms	144
8.3	Properties Provable in Q	150
8.3.1	General properties, unconstrained by numerals	150
8.3.2	Representability properties	152
8.3.3	The "order-adequacy" properties	156
8.3.4	Verifying the abstract ordering assumptions	165

Chapter 1

Preliminaries

1.1 Trivia

abbreviation (*input*) *any* **where** *any* \equiv *undefined*

lemma *Un_Diff2*: $B \cap C = \{\} \implies A \cup B - C = A - C \cup B$ **by** *auto*

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

fun *first* :: *nat* \Rightarrow *nat list* **where**

first 0 = []

| *first* (*Suc* *n*) = *n* # *first* *n*

Facts about zipping lists:

lemma *fst_set_zip_map_fst*:

$length\ xs = length\ ys \implies fst\ '(set\ (zip\ (map\ fst\ xs)\ ys)) = fst\ '(set\ xs)$

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

lemma *snd_set_zip_map_snd*:

$length\ xs = length\ ys \implies snd\ '(set\ (zip\ xs\ (map\ snd\ ys))) = snd\ '(set\ ys)$

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

lemma *snd_set_zip*:

$length\ xs = length\ ys \implies snd\ '(set\ (zip\ xs\ ys)) = set\ ys$

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

lemma *set_zip_D*: $(x, y) \in set\ (zip\ xs\ ys) \implies x \in set\ xs \wedge y \in set\ ys$

using *set_zip_leftD* *set_zip_rightD* **by** *auto*

lemma *inj_on_set_zip_map*:

assumes *i*: *inj_on* *f* *X*

and *a*: $(f\ x1, y1) \in set\ (zip\ (map\ f\ xs)\ ys)$ $set\ xs \subseteq X$ $x1 \in X$ $length\ xs = length\ ys$

shows $(x1, y1) \in set\ (zip\ xs\ ys)$

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 $(u, x) \in set\ (zip\ us\ (map\ snd\ txs))$

and $(t, u) \in set\ (zip\ (map\ fst\ txs)\ us)$

and *distinct* (*map* *snd* *txs*)

and *distinct* *us* **and** $length\ us = length\ txs$

```

shows (t, x) ∈ 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) ∈ set (zip us (map snd txs))
and (t, x) ∈ set txs
and distinct (map snd txs)
and distinct us and length us = length txs
shows (t, u) ∈ 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) ∈ set (zip xs ys) and length xs = length ys
shows (f x1, y1) ∈ 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 ⇒ 'a list where
  asList A ≡ SOME as. set as = A

```

```

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

```

```

lemma triv_Un_imp_aux:
  (∧ a. φ ⇒ a ∉ A ⇒ a ∈ B ↔ a ∈ C) ⇒ φ → A ∪ B = A ∪ C
by auto

```

```

definition toN where toN n ≡ [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 \Rightarrow 'trm — trms include at least the variables

and *FvarsT* :: 'trm \Rightarrow 'var set — free variables for trms

and *substT* :: 'trm \Rightarrow 'trm \Rightarrow 'var \Rightarrow 'trm — substitution for trms

and *Fvars* :: 'fmla \Rightarrow 'var set — free variables for formulas

and *subst* :: 'fmla \Rightarrow 'trm \Rightarrow 'var \Rightarrow '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!]: $\bigwedge x. x \in var \Longrightarrow Var\ x \in trm$

and

inj_Var[simp]: $\bigwedge x\ y. x \in var \Longrightarrow y \in var \Longrightarrow (Var\ x = Var\ y \longleftrightarrow x = y)$

and

finite_FvarsT: $\bigwedge t. t \in trm \Longrightarrow finite\ (FvarsT\ t)$

and

FvarsT: $\bigwedge t. t \in trm \Longrightarrow FvarsT\ t \subseteq var$

and

substT[simp,intro]: $\bigwedge t1\ t\ x. t1 \in trm \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow substT\ t1\ t\ x \in trm$

and

FvarsT_Var[simp]: $\bigwedge x. x \in var \Longrightarrow FvarsT\ (Var\ x) = \{x\}$

and

substT_Var[simp]: $\bigwedge x\ t\ y. x \in var \Longrightarrow y \in var \Longrightarrow t \in trm \Longrightarrow$

substT (Var x) t y = (if x = y then t else Var x)

and

substT_notIn[simp]:

$\bigwedge t1\ t2\ x. x \in var \Longrightarrow t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow x \notin FvarsT\ t1 \Longrightarrow substT\ t1\ t2\ x = t1$

and

— Assumptions about the infrastructure (free vars and substitution) on formulas

finite_Fvars: $\bigwedge \varphi. \varphi \in fmla \Longrightarrow finite\ (Fvars\ \varphi)$

and


```

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

lemma var_NE:  $\text{var} \neq \{\}$ 
using infinite_var by auto

lemma Var_injD:  $\text{Var } x = \text{Var } y \implies x \in \text{var} \implies y \in \text{var} \implies x = y$ 
by auto

lemma FvarsT_VarD:  $x \in \text{FvarsT } (\text{Var } y) \implies y \in \text{var} \implies x = y$ 
by auto

lemma FvarsT':  $t \in \text{trm} \implies x \in \text{FvarsT } t \implies x \in \text{var}$ 
using FvarsT by auto

lemma Fvars':  $\varphi \in \text{fmla} \implies x \in \text{Fvars } \varphi \implies x \in \text{var}$ 
using Fvars by auto

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

lemma in_Fvars_substD:
 $y \in \text{Fvars } (\text{subst } \varphi \ t \ x) \implies \varphi \in \text{fmla} \implies t \in \text{trm} \implies x \in \text{var}$ 
 $\implies y \in (\text{Fvars } \varphi - \{x\}) \cup (\text{if } x \in \text{Fvars } \varphi \text{ then } \text{FvarsT } t \text{ else } \{\})$ 
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:
 $\bigwedge \varphi \ t1 \ t2 \ x. \varphi \in \text{fmla} \implies t1 \in \text{trm} \implies t2 \in \text{trm} \implies x \in \text{var} \implies$ 
 $\text{subst } (\text{subst } \varphi \ t1 \ x) \ t2 \ x = \text{subst } \varphi \ (\text{substT } t1 \ t2 \ x) \ x$ 
using subst_compose_eq_or by blast

lemma subst_subst[simp]:

```

assumes $\varphi[simp]$: $\varphi \in fmla$ **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 $subst_compose_eq_or$ [$OF \varphi _ t x y$, of $Var y$] **using** $subst_notIn yy$ **by** $simp$

lemma $subst_comp$:

$\bigwedge x y \varphi t. \varphi \in fmla \implies t \in trm \implies x \in var \implies y \in var \implies$
 $x \neq y \implies y \notin Fvars T t \implies$
 $subst (subst \varphi (Var x) y) t x = subst (subst \varphi t x) t y$
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 $inst[simp]$: $\varphi \in fmla \implies t \in trm \implies inst \varphi t \in fmla$

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

definition *getFresh* :: 'var set \Rightarrow 'var **where**
getFresh V = (SOME x. x \in var \wedge x \notin V)

lemma *getFresh*: finite V \implies *getFresh* V \in var \wedge *getFresh* V \notin V
by (*metis* (*no_types*, *lifting*) *finite_subset* *getFresh_def* *infinite_var* *someI_ex* *subsetI*)

definition *getFr* :: 'var list \Rightarrow 'trm list \Rightarrow 'fmla list \Rightarrow 'var **where**
getFr xs ts φ s =
getFresh (set xs \cup (\bigcup (FvarsT ' set ts)) \cup (\bigcup (Fvars ' set φ s)))

lemma *getFr_FvarsT_Fvars*:
assumes set xs \subseteq var set ts \subseteq trm **and** set φ s \subseteq fmla
shows *getFr* xs ts φ s \in var \wedge
getFr xs ts φ s \notin set xs \wedge
(t \in set ts \longrightarrow *getFr* xs ts φ s \notin FvarsT t) \wedge
(φ \in set φ s \longrightarrow *getFr* xs ts φ s \notin Fvars φ)

proof–

have finite (set xs \cup (\bigcup (FvarsT ' set ts)) \cup (\bigcup (Fvars ' set φ s)))
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 \subseteq var set ts \subseteq trm **and** set φ s \subseteq fmla
shows *getFr* xs ts φ s \in var
using *assms* *getFr_FvarsT_Fvars* **by** *auto*

lemma *getFr_var*:
assumes set xs \subseteq var set ts \subseteq trm **and** set φ s \subseteq fmla **and** t \in set ts
shows *getFr* xs ts φ s \notin set xs
using *assms* *getFr_FvarsT_Fvars* **by** *auto*

lemma *getFr_FvarsT*:
assumes set xs \subseteq var set ts \subseteq trm **and** set φ s \subseteq fmla **and** t \in set ts
shows *getFr* xs ts φ s \notin FvarsT t
using *assms* *getFr_FvarsT_Fvars* **by** *auto*

lemma *getFr_Fvars*:
assumes set xs \subseteq var set ts \subseteq trm **and** set φ s \subseteq fmla **and** φ \in set φ s
shows *getFr* xs ts φ s \notin Fvars φ
using *assms* *getFr_FvarsT_Fvars* **by** *auto*

2.1.2 Fresh Variables

fun *getFreshN* :: 'var set \Rightarrow nat \Rightarrow '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 \implies
set (*getFreshN* V n) \subseteq var \wedge set (*getFreshN* V n) \cap V = {} \wedge length (*getFreshN* V n) = n \wedge distinct
(*getFreshN* V n)
by (*induct* n *arbitrary*: V) (*auto simp: getFresh Let_def*)

definition *getFrN* :: 'var list \Rightarrow 'trm list \Rightarrow 'fmla list \Rightarrow nat \Rightarrow 'var list **where**
getFrN xs ts φ s n =
getFreshN (set xs \cup (\bigcup (FvarsT ' set ts)) \cup (\bigcup (Fvars ' set φ s))) n

lemma *getFrN_FvarsT_Fvars*:
assumes $set\ xs \subseteq var\ set\ ts \subseteq trm$ **and** $set\ \varphi s \subseteq fmla$
shows $set\ (getFrN\ xs\ ts\ \varphi s\ n) \subseteq var \wedge$
 $set\ (getFrN\ xs\ ts\ \varphi s\ n) \cap set\ xs = \{\} \wedge$
 $(t \in set\ ts \longrightarrow set\ (getFrN\ xs\ ts\ \varphi s\ n) \cap FvarsT\ t = \{\}) \wedge$
 $(\varphi \in set\ \varphi s \longrightarrow set\ (getFrN\ xs\ ts\ \varphi s\ n) \cap Fvars\ \varphi = \{\}) \wedge$
 $length\ (getFrN\ xs\ ts\ \varphi s\ n) = n \wedge$
 $distinct\ (getFrN\ xs\ ts\ \varphi s\ n)$
proof–
have $finite\ (set\ xs \cup (\bigcup (FvarsT\ 'set\ ts)) \cup (\bigcup (Fvars\ 'set\ \varphi s)))$
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 $set\ xs \subseteq var\ set\ ts \subseteq trm$ **and** $set\ \varphi s \subseteq fmla$
shows $set\ (getFrN\ xs\ ts\ \varphi s\ n) \subseteq var$
using *assms getFrN_FvarsT_Fvars* **by** *auto*

lemma *getFrN_var*:
assumes $set\ xs \subseteq var\ set\ ts \subseteq trm$ **and** $set\ \varphi s \subseteq fmla$ **and** $t \in set\ ts$
shows $set\ (getFrN\ xs\ ts\ \varphi s\ n) \cap set\ xs = \{\}$
using *assms getFrN_FvarsT_Fvars* **by** *auto*

lemma *getFrN_FvarsT*:
assumes $set\ xs \subseteq var\ set\ ts \subseteq trm$ **and** $set\ \varphi s \subseteq fmla$ **and** $t \in set\ ts$
shows $set\ (getFrN\ xs\ ts\ \varphi s\ n) \cap FvarsT\ t = \{\}$
using *assms getFrN_FvarsT_Fvars* **by** *auto*

lemma *getFrN_Fvars*:
assumes $set\ xs \subseteq var\ set\ ts \subseteq trm$ **and** $set\ \varphi s \subseteq fmla$ **and** $\varphi \in set\ \varphi s$
shows $set\ (getFrN\ xs\ ts\ \varphi s\ n) \cap Fvars\ \varphi = \{\}$
using *assms getFrN_FvarsT_Fvars* **by** *auto*

lemma *getFrN_length*:
assumes $set\ xs \subseteq var\ set\ ts \subseteq trm$ **and** $set\ \varphi s \subseteq fmla$
shows $length\ (getFrN\ xs\ ts\ \varphi s\ n) = n$
using *assms getFrN_FvarsT_Fvars* **by** *auto*

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

2.1.3 Parallel Term Substitution

fun *rawpsubstT* :: $'trm \Rightarrow ('trm \times 'var)\ list \Rightarrow 'trm$ **where**
 $rawpsubstT\ t\ [] = t$
 $| rawpsubstT\ t\ ((t1,x1) \# txs) = rawpsubstT\ (substT\ t\ t1\ x1)\ txs$

lemma *rawpsubstT[simp]*:
assumes $t \in trm$ **and** $snd\ ' (set\ txs) \subseteq var$ **and** $fst\ ' (set\ txs) \subseteq trm$
shows $rawpsubstT\ t\ txs \in trm$
using *assms* **by** *(induct txs arbitrary: t) fastforce+*

definition *psubstT* :: $'trm \Rightarrow ('trm \times 'var)\ list \Rightarrow 'trm$ **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 $t \in \text{trm}$ **and** $\text{snd} \text{ ' (set txs) } \subseteq \text{var}$ **and** $\text{fst} \text{ ' (set txs) } \subseteq \text{trm}$
shows $\text{psubstT } t \text{ txs} \in \text{trm}$

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

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, hide_lams) 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 $x \in \text{var}$ $\text{snd} \text{ ' (set txs) } \subseteq \text{var}$ $\text{fst} \text{ ' (set txs) } \subseteq \text{trm}$

and $x \notin \text{snd} \text{ ' (set txs)}$

shows $\text{rawpsubstT} (Var x) \text{ txs} = Var x$

using assms **by** (induct txs) auto

lemma psubstT_Var_not[simp]:

assumes $x \in \text{var}$ $\text{snd} \text{ ' (set txs) } \subseteq \text{var}$ $\text{fst} \text{ ' (set txs) } \subseteq \text{trm}$

and $x \notin \text{snd} \text{ ' (set txs)}$

shows $\text{psubstT} (Var x) \text{ txs} = Var x$

proof–

define us **where** us: us = getFrN (map snd txs) (Var x # map fst txs) [] (length txs)

have us_facts: set us \subseteq var

$x \notin$ set us

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 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 $x \in \text{var } \text{snd} \text{ ' (set txs) } \subseteq \text{var } \text{fst} \text{ ' (set txs) } \subseteq \text{trm } t \in \text{trm}$
and $\text{FvarsT } t \cap \text{snd} \text{ ' (set txs) } = \{\}$
shows $\text{rawpsubstT } t \text{ txs} = t$
using *assms* **by** (*induct txs*) *auto*

lemma *psubstT_notIn*[simp]:
assumes $x \in \text{var } \text{snd} \text{ ' (set txs) } \subseteq \text{var } \text{fst} \text{ ' (set txs) } \subseteq \text{trm } t \in \text{trm}$
and $\text{FvarsT } t \cap \text{snd} \text{ ' (set txs) } = \{\}$
shows $\text{psubstT } t \text{ txs} = t$

proof–

define *us* **where** *us*: $us = \text{getFrN } (\text{map } \text{snd } \text{txs}) (t \# \text{map } \text{fst } \text{txs}) [] (\text{length } \text{txs})$
have *us_facts*: $\text{set } us \subseteq \text{var}$
 $\text{set } us \cap \text{FvarsT } t = \{\}$
 $\text{set } us \cap \bigcup (\text{FvarsT } \text{ ' (fst } \text{ ' (set txs))}) = \{\}$
 $\text{set } us \cap \text{snd} \text{ ' (set txs) } = \{\}$
 $\text{length } us = \text{length } \text{txs}$
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*]: $\text{rawpsubstT } t (\text{zip } (\text{map } \text{Var } us) (\text{map } \text{snd } \text{txs})) = t$
using *assms us_facts*
by(*intro rawpsubstT_notIn*) (*auto 0 3 dest: set_zip_rightD set_zip_leftD*)
have [*simp*]: $\text{rawpsubstT } t (\text{zip } (\text{map } \text{fst } \text{txs}) us) = t$
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* :: $\text{'fmla} \Rightarrow (\text{'trm} \times \text{'var}) \text{ list} \Rightarrow \text{'fmla}$ **where**
 $\text{rawpsubst } \varphi [] = \varphi$
 $|\ \text{rawpsubst } \varphi ((t1,x1) \# \text{txs}) = \text{rawpsubst } (\text{subst } \varphi t1 x1) \text{txs}$

lemma *rawpsubst*[simp]:
assumes $\varphi \in \text{fmla}$ **and** $\text{snd} \text{ ' (set txs) } \subseteq \text{var}$ **and** $\text{fst} \text{ ' (set txs) } \subseteq \text{trm}$
shows $\text{rawpsubst } \varphi \text{ txs} \in \text{fmla}$
using *assms* **by** (*induct txs arbitrary: \varphi*) *fastforce*+

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

The *psubst* versions of the *subst* axioms.

lemma *psubst*[*simp,intro*]:
assumes $\varphi \in \text{fmla}$ **and** $\text{snd} \text{ ' (set txs) } \subseteq \text{var}$ **and** $\text{fst} \text{ ' (set txs) } \subseteq \text{trm}$
shows $\text{psubst } \varphi \text{ txs} \in \text{fmla}$

proof–

define *us* **where** *us*: $us = \text{getFrN } (\text{map } \text{snd } \text{txs}) (\text{map } \text{fst } \text{txs}) [\varphi] (\text{length } \text{txs})$
have *us_facts*: $\text{set } us \subseteq \text{var}$
 $\text{set } us \cap \text{Fvars } \varphi = \{\}$
 $\text{set } us \cap \bigcup (\text{FvarsT } \text{ ' (fst } \text{ ' (set txs))}) = \{\}$
 $\text{set } us \cap \text{snd} \text{ ' (set txs) } = \{\}$

$length\ us = length\ txs$
 $distinct\ us$
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] $\varphi\ length\ txs$]
 $getFrN_var$ [*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 $\varphi \in fmla$ **and** $snd\ '(set\ txs) \subseteq var$ **and** $fst\ '(set\ txs) \subseteq trm$
shows $Fvars\ (rawpsubst\ \varphi\ txs) \subseteq$
 $(Fvars\ \varphi - snd\ '(set\ txs)) \cup (\bigcup\ \{FvarsT\ t\ |\ t\ x.\ (t,x) \in set\ txs\})$
using *assms* **proof**(*induction txs arbitrary: \varphi*)
case (*Cons tx txs \varphi*)
then obtain *t x* **where** $tx = (t,x)$ **by** *force*
have $t \in trm$ **and** $x \in var$ **using** *Cons.prem*s **unfolding** *tx* **by** *auto*
define χ **where** $\chi = subst\ \varphi\ t\ x$
have $0: Fvars\ \chi = Fvars\ \varphi - \{x\} \cup (if\ x \in Fvars\ \varphi\ then\ FvarsT\ t\ else\ \{\})$
using *Cons.prem*s **unfolding** χ_def **by** (*auto simp: tx t*)
have $\chi: \chi \in fmla$ **unfolding** χ_def **using** *Cons.prem*s *t x* **by** *auto*
have $Fvars\ (rawpsubst\ \chi\ txs) \subseteq$
 $(Fvars\ \chi - snd\ '(set\ txs)) \cup$
 $(\bigcup\ \{FvarsT\ t\ |\ t\ x.\ (t,x) \in set\ txs\})$
using *Cons.prem*s χ **by** (*intro Cons.IH*) *auto*
also have $\dots \subseteq Fvars\ \varphi - insert\ x\ (snd\ '(set\ txs)) \cup \bigcup\ \{FvarsT\ ta\ |\ ta.\ \exists xa.\ ta = t \wedge xa = x \vee (ta, xa) \in set\ txs\}$
(is $_ \subseteq ?R$) **by**(*auto simp: 0 tx Cons.prem*s)
finally have $1: Fvars\ (rawpsubst\ \chi\ txs) \subseteq ?R$.
have $2: Fvars\ \chi = Fvars\ \varphi - \{x\} \cup (if\ x \in Fvars\ \varphi\ then\ FvarsT\ t\ else\ \{\})$
using *Cons.prem*s *t x* **unfolding** χ_def **using** *Fvars_subst* **by** *auto*
show *?case* **using** 1 **by** (*simp add: tx \chi_def[symmetric] 2*)
qed *auto*

lemma *in_Fvars_rawpsubst_imp*:
assumes $y \in Fvars\ (rawpsubst\ \varphi\ txs)$
and $\varphi \in fmla$ **and** $snd\ '(set\ txs) \subseteq var$ **and** $fst\ '(set\ txs) \subseteq trm$
shows $(y \in Fvars\ \varphi - snd\ '(set\ txs)) \vee$
 $(y \in \bigcup\ \{FvarsT\ t\ |\ t\ x.\ (t,x) \in set\ txs\})$
using *Fvars_rawpsubst_su*[*OF assms(2-4)*]
using *assms(1)* **by** *blast*

lemma *Fvars_rawpsubst*:
assumes $\varphi \in fmla$ **and** $snd\ '(set\ txs) \subseteq var$ **and** $fst\ '(set\ txs) \subseteq trm$
and $distinct\ (map\ snd\ txs)$ **and** $\forall x \in snd\ '(set\ txs).\ \forall t \in fst\ '(set\ txs).\ x \notin FvarsT\ t$
shows $Fvars\ (rawpsubst\ \varphi\ txs) =$
 $(Fvars\ \varphi - snd\ '(set\ txs)) \cup$
 $(\bigcup\ \{if\ x \in Fvars\ \varphi\ then\ FvarsT\ t\ else\ \{\}\ |\ t\ x.\ (t,x) \in set\ txs\})$
using *assms* **proof**(*induction txs arbitrary: \varphi*)
case (*Cons a txs \varphi*)
then obtain *t x* **where** $a = (t,x)$ **by** *force*
have $t \in trm$ **and** $x \in var$ **using** *Cons.prem*s **unfolding** *a* **by** *auto*
have $x\ txs: \bigwedge ta\ xa.\ (ta, xa) \in set\ txs \implies x \neq xa$ **using** $distinct\ (map\ snd\ (a\ \#\ txs))$
unfolding *a* **by** (*auto simp: rev_image_eqI*)

have $xt: x \notin \text{Fvars}T\ t \wedge \text{snd} \text{ ' set } txs \cap \text{Fvars}T\ t = \{\}$ **using** *Cons.prem*s **unfolding** *a* **by** *auto*
hence $0: \text{Fvars } \varphi - \{x\} \cup \text{Fvars}T\ t - \text{snd} \text{ ' set } txs =$
 $\text{Fvars } \varphi - \text{insert } x (\text{snd} \text{ ' set } txs) \cup \text{Fvars}T\ t$

by *auto*

define χ **where** $\chi_def: \chi = \text{subst } \varphi\ t\ x$

have $\chi: \chi \in \text{fm}la$ **unfolding** χ_def **using** *Cons.prem*s $t\ x$ **by** *auto*

have $1: \text{Fvars} (\text{rawpsubst } \chi\ txs) =$

$(\text{Fvars } \chi - \text{snd} \text{ ' (set } txs)) \cup$

$(\bigcup \{if\ x \in \text{Fvars } \chi\ \text{then } \text{Fvars}T\ t\ \text{else } \{\} \mid t\ x . (t,x) \in \text{set } txs\})$

using *Cons.prem*s χ **by** (*intro Cons.IH*) *auto*

have $2: \text{Fvars } \chi = \text{Fvars } \varphi - \{x\} \cup (if\ x \in \text{Fvars } \varphi\ \text{then } \text{Fvars}T\ t\ \text{else } \{\})$

using *Cons.prem*s $t\ x$ **unfolding** χ_def **using** *Fvars_subst* **by** *auto*

define f **where** $f \equiv \lambda ta\ xa. if\ xa \in \text{Fvars } \varphi\ \text{then } \text{Fvars}T\ ta\ \text{else } \{\}$

have $3: \bigcup \{f\ ta\ xa \mid ta\ xa. (ta, xa) \in \text{set } ((t, x) \# txs)\} =$

$f\ t\ x \cup (\bigcup \{f\ ta\ xa \mid ta\ xa. (ta, xa) \in \text{set } txs\})$ **by** *auto*

have $4: \text{snd} \text{ ' set } ((t, x) \# txs) = \{x\} \cup \text{snd} \text{ ' set } txs$ **by** *auto*

have $5: f\ t\ x \cap \text{snd} \text{ ' set } txs = \{\}$ **unfolding** f_def **using** xt **by** *auto*

have $6: \bigcup \{if\ xa \in \text{Fvars } \varphi - \{x\} \cup f\ t\ x\ \text{then } \text{Fvars}T\ ta\ \text{else } \{\} \mid ta\ xa. (ta, xa) \in \text{set } txs\}$

$= (\bigcup \{f\ ta\ xa \mid ta\ xa. (ta, xa) \in \text{set } txs\})$

unfolding f_def **using** $xt\ x_txs$ **by** (*fastforce split: if_splits*)

have $\text{Fvars } \varphi - \{x\} \cup f\ t\ x - \text{snd} \text{ ' set } txs \cup$

$\bigcup \{if\ xa \in \text{Fvars } \varphi - \{x\} \cup f\ t\ x\ \text{then } \text{Fvars}T\ ta\ \text{else } \{\}$

$\mid ta\ xa. (ta, xa) \in \text{set } txs\} =$

$\text{Fvars } \varphi - \text{snd} \text{ ' set } ((t, x) \# txs) \cup$

$\bigcup \{f\ ta\ xa \mid ta\ xa. (ta, xa) \in \text{set } ((t, x) \# txs)\}$

unfolding $3\ 4\ 6$ **unfolding** $Un_Diff2[OF\ 5]$ Un_assoc **unfolding** $Diff_Diff_Un\ ..$

thus *?case* **unfolding** *a* *rawpsubst.simps 1 2* $\chi_def[symmetric]$ f_def **by** *simp*

qed *auto*

lemma *in_Fvars_rawpsubstD*:

assumes $y \in \text{Fvars} (\text{rawpsubst } \varphi\ txs)$

and $\varphi \in \text{fm}la$ **and** $\text{snd} \text{ ' (set } txs) \subseteq \text{var}$ **and** $\text{fst} \text{ ' (set } txs) \subseteq \text{trm}$

and *distinct* (*map snd txs*) **and** $\forall x \in \text{snd} \text{ ' (set } txs). \forall t \in \text{fst} \text{ ' (set } txs). x \notin \text{Fvars}T\ t$

shows $(y \in \text{Fvars } \varphi - \text{snd} \text{ ' (set } txs)) \vee$

$(y \in \bigcup \{if\ x \in \text{Fvars } \varphi\ \text{then } \text{Fvars}T\ t\ \text{else } \{\} \mid t\ x . (t,x) \in \text{set } txs\})$

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

lemma *Fvars_psubst*:

assumes $\varphi \in \text{fm}la$ **and** $\text{snd} \text{ ' (set } txs) \subseteq \text{var}$ **and** $\text{fst} \text{ ' (set } txs) \subseteq \text{trm}$

and *distinct* (*map snd txs*)

shows $\text{Fvars} (\text{psubst } \varphi\ txs) =$

$(\text{Fvars } \varphi - \text{snd} \text{ ' (set } txs)) \cup$

$(\bigcup \{if\ x \in \text{Fvars } \varphi\ \text{then } \text{Fvars}T\ t\ \text{else } \{\} \mid t\ x . (t,x) \in \text{set } txs\})$

proof–

define us **where** $us: us = \text{getFrN} (\text{map } \text{snd } txs) (\text{map } \text{fst } txs) [\varphi] (\text{length } txs)$

have $us_facts: \text{set } us \subseteq \text{var}$

$\text{set } us \cap \text{Fvars } \varphi = \{\}$

$\text{set } us \cap \bigcup (\text{Fvars}T \text{ ' (fst ' (set } txs))) = \{\}$

$\text{set } us \cap \text{snd} \text{ ' (set } txs) = \{\}$

$\text{length } us = \text{length } txs$

distinct us

using *assms* **unfolding** us

using $\text{getFrN_Fvars}T[\text{of } \text{map } \text{snd } txs\ \text{map } \text{fst } txs\ [\varphi]\ _ \text{length } txs]$

$\text{getFrN_Fvars}[\text{of } \text{map } \text{snd } txs\ \text{map } \text{fst } txs\ [\varphi]\ _ \text{length } txs]$


```

    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, hide_lams) 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_snd 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  $xx \in \bigcup \{ \text{if } x \in \text{Fvars } \varphi \text{ then } \text{FvarsT } t \text{ else } \{ \} \mid t \text{ x. } (t, x) \in \text{set } \text{txs} \}$ 
proof(cases  $y \in \text{Fvars } \chi$ )
  case True note  $y = \text{True}$ 
    hence  $xx: xx \in \text{FvarsT } tt$  using  $xx$  by simp
    obtain  $x$  where  $x\varphi: x \in \text{Fvars } \varphi$ 
      and  $yx: (Var y, x) \in \text{set } (\text{zip } (\text{map } Var \text{us}) (\text{map } \text{snd } \text{txs}))$ 
      using  $y$  notin unfolding  $0$  by auto (metis empty_iff insert_iff)
    have  $yx: (y, x) \in \text{set } (\text{zip } \text{us } (\text{map } \text{snd } \text{txs}))$ 
    using  $yvar$   $\text{us\_facts}$  by (intro inj_on_set_zip_map[OF inj_on_Var yx]) auto
    have  $(tt, x) \in \text{set } \text{txs}$  apply(rule set_zip_map_fst_snd[OF yx ty])
      using  $\langle \text{distinct } (\text{map } \text{snd } \text{txs}) \rangle$   $\text{us\_facts}$  by auto
    thus ?thesis using  $xx$   $x\varphi$  by auto
  qed(insert xx, auto)
next
  fix  $y$   $tt$   $xx$ 
  assume  $y: y \in (\text{if } xx \in \text{Fvars } \varphi \text{ then } \text{FvarsT } tt \text{ else } \{ \})$ 
    and  $tx: (tt, xx) \in \text{set } \text{txs}$ 
    hence  $xx\text{snd}: xx \in \text{snd } ' \text{set } \text{txs}$  by force
    obtain  $u$  where  $uin: u \in \text{set } \text{us}$  and  $u\text{xx}: (u, xx) \in \text{set } (\text{zip } \text{us } (\text{map } \text{snd } \text{txs}))$ 
      by (metis xxsnd_in_set_impl_in_set_zip2 length_map set_map set_zip_leftD us_facts(5))
    hence  $uvar: u \in \text{var}$  using  $\text{us\_facts}$  by auto
    show  $y \in \bigcup \{ \text{if } u \in \text{Fvars } \chi \text{ then } \text{FvarsT } t \text{ else } \{ \} \mid t \text{ u. } (t, u) \in \text{set } (\text{zip } (\text{map } \text{fst } \text{txs}) \text{us}) \}$ 
    proof(cases  $xx \in \text{Fvars } \varphi$ )
      case True note  $xx = \text{True}$ 
        hence  $y: y \in \text{FvarsT } tt$  using  $y$  by auto
        have  $(Var u, xx) \in \text{set } (\text{zip } (\text{map } Var \text{us}) (\text{map } \text{snd } \text{txs}))$ 
        using  $\text{us\_facts}$  by (intro set_zip_length_map[OF uxx]) auto
        hence  $u\chi: u \in \text{Fvars } \chi$  using  $uin$   $xx$   $uvar$  unfolding  $0$  by auto
        have  $ttu: (tt, u) \in \text{set } (\text{zip } (\text{map } \text{fst } \text{txs}) \text{us})$ 
        using  $\text{assms}$   $\text{us\_facts}$  by (intro set_zip_map_fst_snd2[OF uxx tx]) auto
        show ?thesis using  $u\chi$   $ttu$   $y$  by auto
      qed(insert y, auto)
    qed
    show ?thesis
    by (simp add: psubst_def Let_def us[symmetric] \chi_def[symmetric] 1 2 3)
  qed

lemma in_Fvars_psubstD:
  assumes  $y \in \text{Fvars } (\text{psubst } \varphi \text{ txs})$ 
    and  $\varphi \in \text{fmla}$  and  $\text{snd } ' (\text{set } \text{txs}) \subseteq \text{var}$  and  $\text{fst } ' (\text{set } \text{txs}) \subseteq \text{trm}$ 
    and distinct ( $\text{map } \text{snd } \text{txs}$ )
  shows  $y \in (\text{Fvars } \varphi - \text{snd } ' (\text{set } \text{txs})) \cup$ 
    ( $\bigcup \{ \text{if } x \in \text{Fvars } \varphi \text{ then } \text{FvarsT } t \text{ else } \{ \} \mid t \text{ x. } (t, x) \in \text{set } \text{txs} \}$ )
  using  $\text{assms}$  Fvars_psubst by auto

lemma subst2_fresh_switch:
  assumes  $\varphi \in \text{fmla}$   $t \in \text{trm}$   $s \in \text{trm}$   $x \in \text{var}$   $y \in \text{var}$ 
    and  $x \neq y$   $x \notin \text{FvarsT } s$   $y \notin \text{FvarsT } t$ 
  shows  $\text{subst } (\text{subst } \varphi \text{ s } y) \text{ t } x = \text{subst } (\text{subst } \varphi \text{ t } x) \text{ s } y$  (is  $?L = ?R$ )
  using  $\text{assms}$  by (simp add: subst_compose_diff[of \varphi s t y x])

lemma rawpsubst2_fresh_switch:
  assumes  $\varphi \in \text{fmla}$   $t \in \text{trm}$   $s \in \text{trm}$   $x \in \text{var}$   $y \in \text{var}$ 
    and  $x \neq y$   $x \notin \text{FvarsT } s$   $y \notin \text{FvarsT } t$ 
  shows  $\text{rawpsubst } \varphi ((s, y), (t, x)) = \text{rawpsubst } \varphi ((t, x), (s, y))$ 
  using  $\text{assms}$  by (simp add: subst2_fresh_switch)

lemma rawpsubst_compose:

```

assumes $\varphi \in \text{fm}la$ **and** $\text{snd} \text{ ' (set } txs1) \subseteq \text{var}$ **and** $\text{fst} \text{ ' (set } txs1) \subseteq \text{tr}m$
and $\text{snd} \text{ ' (set } txs2) \subseteq \text{var}$ **and** $\text{fst} \text{ ' (set } txs2) \subseteq \text{tr}m$
shows $\text{rawpsubst} (\text{rawpsubst } \varphi \text{ } txs1) \text{ } txs2 = \text{rawpsubst } \varphi (txs1 @ txs2)$
using *assms* **by** (*induct* $txs1$ *arbitrary:* $txs2$ φ) *auto*

lemma *rawpsubst_subst_fresh_switch*:

assumes $\varphi \in \text{fm}la$ $\text{snd} \text{ ' (set } txs) \subseteq \text{var}$ **and** $\text{fst} \text{ ' (set } txs) \subseteq \text{tr}m$
and $\forall x \in \text{snd} \text{ ' (set } txs). x \notin \text{FvarsT } s$
and $\forall t \in \text{fst} \text{ ' (set } txs). y \notin \text{FvarsT } t$
and *distinct* (*map* $\text{snd } txs$)
and $s \in \text{tr}m$ **and** $y \in \text{var}$ $y \notin \text{snd} \text{ ' (set } txs)$
shows $\text{rawpsubst} (\text{subst } \varphi \text{ } s \text{ } y) \text{ } txs = \text{rawpsubst } \varphi (txs @ [(s,y)])$
using *assms* **proof**(*induction* txs *arbitrary:* $\varphi \text{ } s \text{ } y$)
case (*Cons* $tx \text{ } txs$)
obtain $t \text{ } x$ **where** $tx[\text{simp}]$: $tx = (t,x)$ **by** *force*
have x : $x \in \text{var}$ **and** t : $t \in \text{tr}m$ **using** *Cons* **unfolding** tx **by** *auto*
have $\text{rawpsubst } \varphi ((s, y) \# (t, x) \# txs) = \text{rawpsubst } \varphi ((s, y), (t, x)) @ txs$ **by** *simp*
also **have** $\dots = \text{rawpsubst} (\text{rawpsubst } \varphi [(s, y), (t, x)]) \text{ } txs$
using *Cons* **by** *auto*
also **have** $\text{rawpsubst } \varphi [(s, y), (t, x)] = \text{rawpsubst } \varphi [(t, x), (s, y)]$
using *Cons* **by** (*intro* *rawpsubst2_fresh_switch*) *auto*
also **have** $\text{rawpsubst} (\text{rawpsubst } \varphi [(t, x), (s, y)]) \text{ } txs = \text{rawpsubst } \varphi (((t, x), (s, y)) @ txs)$
using *Cons* **by** *auto*
also **have** $\dots = \text{rawpsubst} (\text{subst } \varphi \text{ } t \text{ } x) (txs @ [(s,y)])$ **using** *Cons* **by** *auto*
also **have** $\dots = \text{rawpsubst } \varphi (((t, x) \# txs) @ [(s, y)])$ **by** *simp*
finally **show** *?case* **unfolding** tx **by** *auto*
qed *auto*

lemma *subst_rawpsubst_fresh_switch*:

assumes $\varphi \in \text{fm}la$ $\text{snd} \text{ ' (set } txs) \subseteq \text{var}$ **and** $\text{fst} \text{ ' (set } txs) \subseteq \text{tr}m$
and $\forall x \in \text{snd} \text{ ' (set } txs). x \notin \text{FvarsT } s$
and $\forall t \in \text{fst} \text{ ' (set } txs). y \notin \text{FvarsT } t$
and *distinct* (*map* $\text{snd } txs$)
and $s \in \text{tr}m$ **and** $y \in \text{var}$ $y \notin \text{snd} \text{ ' (set } txs)$
shows $\text{subst} (\text{rawpsubst } \varphi \text{ } txs) \text{ } s \text{ } y = \text{rawpsubst } \varphi ((s,y) \# txs)$
using *assms* **proof**(*induction* txs *arbitrary:* $\varphi \text{ } s \text{ } y$)
case (*Cons* $tx \text{ } txs$)
obtain $t \text{ } x$ **where** $tx[\text{simp}]$: $tx = (t,x)$ **by** *force*
have x : $x \in \text{var}$ **and** t : $t \in \text{tr}m$ **using** *Cons* **unfolding** tx **by** *auto*
have $\text{subst} (\text{rawpsubst} (\text{subst } \varphi \text{ } t \text{ } x) \text{ } txs) \text{ } s \text{ } y = \text{rawpsubst} (\text{subst } \varphi \text{ } t \text{ } x) ((s,y) \# txs)$
using *Cons.prem*s **by** (*intro* *Cons.IH*) *auto*
also **have** $\dots = \text{rawpsubst} (\text{rawpsubst } \varphi [(t,x)]) ((s,y) \# txs)$ **by** *simp*
also **have** $\dots = \text{rawpsubst } \varphi (((t,x)) @ ((s,y) \# txs))$
using *Cons.prem*s **by** *auto*
also **have** $\dots = \text{rawpsubst } \varphi (((t,x), (s,y)) @ txs)$ **by** *simp*
also **have** $\dots = \text{rawpsubst} (\text{rawpsubst } \varphi [(t,x), (s,y)]) \text{ } txs$
using *Cons.prem*s **by** *auto*
also **have** $\text{rawpsubst } \varphi [(t,x), (s,y)] = \text{rawpsubst } \varphi [(s,y), (t,x)]$
using *Cons.prem*s **by** (*intro* *rawpsubst2_fresh_switch*) *auto*
also **have** $\text{rawpsubst} (\text{rawpsubst } \varphi [(s,y), (t,x)]) \text{ } txs = \text{rawpsubst } \varphi (((s,y), (t,x)) @ txs)$
using *Cons.prem*s **by** *auto*
finally **show** *?case* **by** *simp*
qed *auto*

lemma *rawpsubst_compose_freshVar*:

assumes $\varphi \in \text{fm}la$ $\text{snd} \text{ ' (set } txs) \subseteq \text{var}$ **and** $\text{fst} \text{ ' (set } txs) \subseteq \text{tr}m$
and *distinct* (*map* $\text{snd } txs$)
and $\bigwedge i \text{ } j. i < j \implies j < \text{length } txs \implies \text{snd} (txs!j) \notin \text{FvarsT} (\text{fst} (txs!i))$

```

and us_facts: set us  $\subseteq$  var
set us  $\cap$  Fvars  $\varphi = \{\}$ 
set us  $\cap \bigcup (FvarsT \text{ ' } (fst \text{ ' } (set \text{ } txs))) = \{\}$ 
set us  $\cap$  snd  $\text{ ' } (set \text{ } txs) = \{\}$ 
length us = length txs
distinct us
shows rawpsubst (rawpsubst  $\varphi$  (zip (map Var us) (map snd txs))) (zip (map fst txs) us) = rawpsubst  $\varphi$ 
txs
using assms proof(induction txs arbitrary: us  $\varphi$ )
case (Cons tx txs uus  $\varphi$ )
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  $\subseteq$  var
set us  $\cap$  Fvars  $\varphi = \{\}$ 
set us  $\cap \bigcup (FvarsT \text{ ' } (fst \text{ ' } (set \text{ } txs))) = \{\}$ 
set us  $\cap$  snd  $\text{ ' } (set \text{ } txs) = \{\}$ 
length us = length txs
distinct us and u_facts: u  $\in$  var u  $\notin$  Fvars  $\varphi$ 
u  $\notin \bigcup (FvarsT \text{ ' } (fst \text{ ' } (set \text{ } txs)))$ 
u  $\notin$  snd  $\text{ ' } (set \text{ } txs)$  u  $\notin$  set us
using Cons by auto
let ?uux = zip (map Var us) (map snd txs)
have 1: rawpsubst (subst  $\varphi$  (Var u) x) ?uux = rawpsubst  $\varphi$  (?uux @ [(Var u,x)])
using u_facts Cons.prems
by (intro rawpsubst_subst_fresh_switch) (auto simp: subsetD dest!: set_zip_D)
let ?uux = zip (map Var uus) (map snd (tx # txs))
let ?tus = zip (map fst txs) us let ?txs = zip (map fst (tx # txs)) uus
have 2: u  $\in$  Fvars (rawpsubst  $\varphi$  (zip (map Var us) (map snd txs)))  $\implies$  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= $\lambda x. x \in set us \longrightarrow (\forall y \in set txs. x \neq snd y)$ ,
THEN mp[OF _ nth_mem], THEN bspec[OF _ nth_mem]])
subgoal
by (auto simp: in_set_zip subset_eq split: if_splits) .

have 3:  $\bigwedge xx \text{ } tt. xx \in FvarsT \text{ } t \implies (tt, xx) \notin set \text{ } txs$ 
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  $\varphi$  ?uux) ?txs = rawpsubst (subst (rawpsubst  $\varphi$  (?uux @ [(Var u, x)]))
t u) ?tus
by (simp add: 1)

have rawpsubst  $\varphi$  (?uux @ [(Var u, x)]) = rawpsubst (rawpsubst  $\varphi$  ?uux) [(Var u, x)]
using Cons.prems by (intro rawpsubst_compose[symmetric]) (auto intro!: rawpsubst dest!: set_zip_D)
also have rawpsubst (rawpsubst  $\varphi$  ?uux) [(Var u, x)] = subst (rawpsubst  $\varphi$  ?uux) (Var u) x by simp
finally have subst (rawpsubst  $\varphi$  (?uux @ [(Var u, x)])) t u =
subst (subst (rawpsubst  $\varphi$  ?uux) (Var u) x) t u by simp
also have ... = subst (rawpsubst  $\varphi$  ?uux) t x
using Cons 2 by (intro subst_subst) (auto intro!: rawpsubst dest!: set_zip_D)
also have ... = rawpsubst  $\varphi$  ((t,x) # ?uux)
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  $\varphi$  [(t,x)] @ ?uxs by simp
also have ... = rawpsubst (rawpsubst  $\varphi$  [(t,x)]) ?uxs
using Cons.prem by (intro rawpsubst_compose[symmetric]) (auto dest!: set_zip_D)
finally have rawpsubst (subst (rawpsubst  $\varphi$  (?uxs @ [(Var u, x]))) t u) ?tus =
  rawpsubst (rawpsubst (rawpsubst  $\varphi$  [(t,x)]) ?uxs) ?tus by auto
hence rawpsubst (rawpsubst  $\varphi$  ?uuxs) ?ttxs = rawpsubst (rawpsubst (rawpsubst  $\varphi$  [(t,x)]) ?uxs) ?tus
using 00 by auto
also have ... = rawpsubst (rawpsubst  $\varphi$  [(t,x)]) txs
using Cons.prem 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  $\varphi$  [(t,x)] @ txs
using Cons.prem by (intro rawpsubst_compose) (auto dest!: set_zip_D)
finally show ?case by simp
qed auto

```

lemma `rawpsubst_compose_freshVar2_aux`:

```

assumes  $\varphi$ [simp]:  $\varphi \in \text{fnla}$ 
and ts:  $\text{set } ts \subseteq \text{trm}$ 
and xs:  $\text{set } xs \subseteq \text{var}$  distinct xs
and us_facts:  $\text{set } us \subseteq \text{var}$  distinct us
 $\text{set } us \cap \text{Fvars } \varphi = \{\}$ 
 $\text{set } us \cap \bigcup (\text{FvarsT } ' (\text{set } ts)) = \{\}$ 
 $\text{set } us \cap \text{set } xs = \{\}$ 
and vs_facts:  $\text{set } vs \subseteq \text{var}$  distinct vs
 $\text{set } vs \cap \text{Fvars } \varphi = \{\}$ 
 $\text{set } vs \cap \bigcup (\text{FvarsT } ' (\text{set } ts)) = \{\}$ 
 $\text{set } vs \cap \text{set } xs = \{\}$ 
and l:  $\text{length } us = \text{length } xs$   $\text{length } vs = \text{length } xs$   $\text{length } ts = \text{length } xs$ 
and d:  $\text{set } us \cap \text{set } vs = \{\}$ 
shows rawpsubst (rawpsubst  $\varphi$  (zip (map Var us) xs)) (zip ts us) =
  rawpsubst (rawpsubst  $\varphi$  (zip (map Var vs) xs)) (zip ts vs)
using assms proof(induction xs arbitrary:  $\varphi$  ts us vs)
case (Cons x xs  $\varphi$  tts uus vvs)
obtain t ts u us v vs where tts[simp]:  $tts = t \# ts$  and lts[simp]:  $\text{length } ts = \text{length } xs$ 
and uus[simp]:  $uus = u \# us$  and lus[simp]:  $\text{length } us = \text{length } xs$ 
and vvs[simp]:  $vvs = v \# vs$  and lvs[simp]:  $\text{length } vs = \text{length } xs$ 
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 $?\varphi ux = \text{subst } \varphi (\text{Var } u) x$ let $?\varphi vx = \text{subst } \varphi (\text{Var } v) x$

```

have 0: rawpsubst (rawpsubst ? $\varphi$ ux (zip (map Var us) xs)) (zip ts us) =
  rawpsubst (rawpsubst ? $\varphi$ ux (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 ? $\varphi$ ux (zip (map Var vs) xs) =
  subst (rawpsubst  $\varphi$  (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 ? $\varphi$ vx (zip (map Var vs) xs) =
  subst (rawpsubst  $\varphi$  (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  $\varphi$  (zip (map Var vs) xs)) (Var u) x) t u =
  subst (rawpsubst  $\varphi$  (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  $\varphi$  (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  $\varphi$  (zip (map Var vs) xs)) (Var u) x) t u =
    subst (subst (rawpsubst  $\varphi$  (zip (map Var vs) xs)) (Var v) x) t v .

have rawpsubst (subst (rawpsubst ? $\varphi$ ux (zip (map Var us) xs)) t u) (zip ts us) =
  subst (rawpsubst (rawpsubst ? $\varphi$ ux (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 ? $\varphi$ ux (zip (map Var vs) xs)) (zip ts vs)) t u
  unfolding 0 ..
also have ... = rawpsubst (subst (rawpsubst ? $\varphi$ ux (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  $\varphi$  (zip (map Var vs) xs)) (Var u) x) t u) (zip ts vs)
  unfolding 1 ..
also have ... = rawpsubst (subst (subst (rawpsubst  $\varphi$  (zip (map Var vs) xs)) (Var v) x) t v) (zip ts vs)
  unfolding 2 ..
also have ... = rawpsubst (subst (rawpsubst ? $\varphi$ vx (zip (map Var vs) xs)) t v) (zip ts vs)
  unfolding 11 ..
finally have rawpsubst (subst (rawpsubst ? $\varphi$ ux (zip (map Var us) xs)) t u) (zip ts us) =
  rawpsubst (subst (rawpsubst ? $\varphi$ vx (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_fresh Var2:
assumes  $\varphi$ [simp]:  $\varphi \in fmla$ 
and ts: set ts  $\subseteq$  trm
and xs: set xs  $\subseteq$  var distinct xs
and us_facts: set us  $\subseteq$  var distinct us
  set us  $\cap$  Fvars  $\varphi = \{\}$ 
  set us  $\cap \bigcup (FvarsT \text{ ` (set ts) }) = \{\}$ 
  set us  $\cap$  set xs =  $\{\}$ 

```

and vs_facts : $set\ vs \subseteq var$ $distinct\ vs$
 $set\ vs \cap Fvars\ \varphi = \{\}$
 $set\ vs \cap \bigcup (FvarsT\ ' (set\ ts)) = \{\}$
 $set\ vs \cap set\ xs = \{\}$
and l : $length\ us = length\ xs$ $length\ vs = length\ xs$ $length\ ts = length\ xs$
shows $rawpsubst\ (rawpsubst\ \varphi\ (zip\ (map\ Var\ us)\ xs))\ (zip\ ts\ us) =$
 $rawpsubst\ (rawpsubst\ \varphi\ (zip\ (map\ Var\ vs)\ xs))\ (zip\ ts\ vs)$ **(is ?L = ?R)**
proof–
define ws **where** $ws = getFrN\ (xs\ @\ us\ @\ vs)\ ts\ [\varphi]\ (length\ xs)$
note $fv = getFrN_Fvars[of\ xs\ @\ us\ @\ vs\ ts\ [\varphi]\ _]\ length\ xs$
and $fv_t = getFrN_FvarsT[of\ xs\ @\ us\ @\ vs\ ts\ [\varphi]\ _]\ length\ xs$
and $var = getFrN_var[of\ xs\ @\ us\ @\ vs\ ts\ [\varphi]\ _]\ length\ xs$
and $l = getFrN_length[of\ xs\ @\ us\ @\ vs\ ts\ [\varphi]\ length\ xs]$
have ws_facts : $set\ ws \subseteq var$ $distinct\ ws$
 $set\ ws \cap Fvars\ \varphi = \{\}$
 $set\ ws \cap \bigcup (FvarsT\ ' (set\ ts)) = \{\}$
 $set\ ws \cap set\ xs = \{\}$ $set\ ws \cap set\ us = \{\}$ $set\ ws \cap set\ vs = \{\}$
 $length\ ws = length\ xs$ **using** $assms$ **unfolding** ws_def
apply –
subgoal **by** $auto$
subgoal **by** $auto$
subgoal **using** fv **by** $auto$
subgoal **using** fv_t $IntI$ $empty_iff$ **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 $?L = rawpsubst\ (rawpsubst\ \varphi\ (zip\ (map\ Var\ ws)\ xs))\ (zip\ ts\ ws)$
apply($rule\ rawpsubst_compose_freshVar2_aux$) **using** $assms\ ws_facts$ **by** $auto$
also **have** $\dots = ?R$
apply($rule\ rawpsubst_compose_freshVar2_aux$) **using** $assms\ ws_facts$ **by** $auto$
finally **show** $?thesis$.
qed

lemma $psubst_subst_fresh_switch$:
assumes $\varphi \in fmla\ snd\ ' set\ txs \subseteq var\ fst\ ' set\ txs \subseteq trm$
and $\forall x \in snd\ ' set\ txs.\ x \notin FvarsT\ s\ \forall t \in fst\ ' set\ txs.\ y \notin FvarsT\ t$
and $distinct\ (map\ snd\ txs)$
and $s \in trm\ y \in var\ y \notin snd\ ' set\ txs$
shows $psubst\ (subst\ \varphi\ s\ y)\ txs = subst\ (psubst\ \varphi\ txs)\ s\ y$
proof–
define us **where** $us = getFrN\ (map\ snd\ txs)\ (map\ fst\ txs)\ [\varphi]\ (length\ txs)$
note $fv_t = getFrN_FvarsT[of\ map\ snd\ txs\ map\ fst\ txs\ [\varphi]\ _]\ length\ txs$
and $fv = getFrN_Fvars[of\ map\ snd\ txs\ map\ fst\ txs\ [\varphi]\ _]\ length\ txs$
and $var = getFrN_var[of\ map\ snd\ txs\ map\ fst\ txs\ [\varphi]\ _]\ length\ txs$
and $l = getFrN_length[of\ map\ snd\ txs\ map\ fst\ txs\ [\varphi]\ length\ txs]$

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 **apply** –
subgoal **by** $auto$
subgoal **using** fv **by** ($cases\ txs,\ auto$)
subgoal **using** fv_t **by** ($cases\ txs,\ auto$)
subgoal **using** var **by** ($cases\ txs,\ auto$)

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

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

have vs_facts : $set \text{ vs } \subseteq \text{var}$
 $set \text{ vs } \cap \text{Fvars } (\text{subst } \varphi s y) = \{\}$
 $set \text{ vs } \cap \bigcup (\text{FvarsT } '(\text{fst } '(set \text{ txs}))) = \{\}$
 $set \text{ vs } \cap \text{snd } '(set \text{ txs}) = \{\}$
 $\text{length } vs = \text{length } txs$
 $\text{distinct } vs$

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 : $ws = \text{getFrN } (y \# \text{map } \text{snd } txs) (s \# \text{map } \text{fst } txs) [\varphi] (\text{length } txs)$
note $\text{fvt} = \text{getFrN_FvarsT}[of y \# \text{map } \text{snd } txs s \# \text{map } \text{fst } txs [\varphi] _ \text{length } txs]$
and $\text{fv} = \text{getFrN_Fvars}[of y \# \text{map } \text{snd } txs s \# \text{map } \text{fst } txs [\varphi] _ \text{length } txs]$
and $\text{var} = \text{getFrN_var}[of y \# \text{map } \text{snd } txs s \# \text{map } \text{fst } txs [\varphi] _ \text{length } txs]$
and $l = \text{getFrN_length}[of y \# \text{map } \text{snd } txs s \# \text{map } \text{fst } txs [\varphi] \text{length } txs]$

have ws_facts : $set \text{ ws } \subseteq \text{var}$
 $set \text{ ws } \cap \text{Fvars } \varphi = \{\} \ y \notin set \text{ ws } \ set \text{ ws } \cap \text{FvarsT } s = \{\}$
 $set \text{ ws } \cap \bigcup (\text{FvarsT } '(\text{fst } '(set \text{ txs}))) = \{\}$
 $set \text{ ws } \cap \text{snd } '(set \text{ txs}) = \{\}$
 $\text{length } ws = \text{length } txs$
 $\text{distinct } ws$

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 $?vxs = \text{zip } (\text{map } \text{Var } vs) (\text{map } \text{snd } txs)$
let $?tvs = (\text{zip } (\text{map } \text{fst } txs) vs)$
let $?uxs = \text{zip } (\text{map } \text{Var } us) (\text{map } \text{snd } txs)$
let $?tus = (\text{zip } (\text{map } \text{fst } txs) us)$
let $?wxs = \text{zip } (\text{map } \text{Var } ws) (\text{map } \text{snd } txs)$
let $?tws = (\text{zip } (\text{map } \text{fst } txs) ws)$

have 0 : $\text{rawpsubst } (\text{subst } \varphi s y) ?wxs = \text{subst } (\text{rawpsubst } \varphi ?wxs) s y$
apply($\text{subst } \text{rawpsubst_compose}[of \varphi ?wxs [(s,y)],\text{simplified}]$)
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  $\varphi$  ?wxs) ?tus = rawpsubst (rawpsubst  $\varphi$  ?uxs) ?tus
  using assms ws_facts us_facts by (intro rawpsubst_compose_freshVar2) (auto simp: subset_eq)

have rawpsubst (rawpsubst (subst  $\varphi$  s y) ?wxs) ?tus =
  rawpsubst (rawpsubst (subst  $\varphi$  s y) ?wxs) ?tus
  using assms ws_facts vs_facts
  by (intro rawpsubst_compose_freshVar2) (auto simp: subset_eq)
also have ... = rawpsubst (subst (rawpsubst  $\varphi$  ?wxs) s y) ?tus unfolding 0 ..
also have ... = subst (rawpsubst (rawpsubst  $\varphi$  ?wxs) ?tus) s y
apply(subst rawpsubst_compose[of rawpsubst  $\varphi$  ?wxs ?tus [(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  $\varphi$  ?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  $\varphi \in \text{fm}la$  snd ' (set txs)  $\subseteq$  var and fst ' (set txs)  $\subseteq$  trm
  and distinct (map snd txs)

  and  $\bigwedge i j. i < j \implies j < \text{length } txs \implies \text{snd } (txs!j) \notin \text{FvarsT } (\text{fst } (txs!i))$ 
  shows psubst  $\varphi$  txs = rawpsubst  $\varphi$  txs
proof-
define us where us: us = getFrN (map snd txs) (map fst txs) [ $\varphi$ ] (length txs)
note fvt = getFrN_FvarsT[of map snd txs map fst txs [ $\varphi$ ] _ length txs]
and fv = getFrN_Fvars[of map snd txs map fst txs [ $\varphi$ ] _ length txs]
and var = getFrN_var[of map snd txs map fst txs [ $\varphi$ ] _ length txs]
and l = getFrN_length[of map snd txs map fst txs [ $\varphi$ ] length txs]
have us_facts: set us  $\subseteq$  var
  set us  $\cap$  Fvars  $\varphi = \{\}$ 
  set us  $\cap \bigcup (\text{FvarsT } ' (\text{fst } ' (\text{set } txs))) = \{\}$ 
  set us  $\cap$  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 $\varphi \in \text{fmla}$ $x \in \text{var}$ **and** $t \in \text{trm}$
shows $\text{psubst } \varphi [(t,x)] = \text{subst } \varphi t x$

proof–

have $\text{psubst } \varphi [(t,x)] = \text{rawpsubst } \varphi [(t,x)]$ **apply**(*rule psubst_eq_rawpsubst*)
using *assms* **by** *auto*
thus *?thesis* **by** *auto*

qed

lemma *psubst_eq_rawpsubst2*:

assumes $\varphi \in \text{fmla}$ $x1 \in \text{var}$ $x2 \in \text{var}$ $t1 \in \text{trm}$ $t2 \in \text{trm}$
and $x1 \neq x2$ $x2 \notin \text{FvarsT } t1$
shows $\text{psubst } \varphi [(t1,x1),(t2,x2)] = \text{rawpsubst } \varphi [(t1,x1),(t2,x2)]$
apply(*rule psubst_eq_rawpsubst*)
using *assms* **using** *less_SucE* **by** *force+*

lemma *psubst_eq_rawpsubst3*:

assumes $\varphi \in \text{fmla}$ $x1 \in \text{var}$ $x2 \in \text{var}$ $x3 \in \text{var}$ $t1 \in \text{trm}$ $t2 \in \text{trm}$ $t3 \in \text{trm}$
and $x1 \neq x2$ $x1 \neq x3$ $x2 \neq x3$
 $x2 \notin \text{FvarsT } t1$ $x3 \notin \text{FvarsT } t1$ $x3 \notin \text{FvarsT } t2$
shows $\text{psubst } \varphi [(t1,x1),(t2,x2),(t3,x3)] = \text{rawpsubst } \varphi [(t1,x1),(t2,x2),(t3,x3)]$
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 $\varphi \in \text{fmla}$ $x1 \in \text{var}$ $x2 \in \text{var}$ $x3 \in \text{var}$ $x4 \in \text{var}$
 $t1 \in \text{trm}$ $t2 \in \text{trm}$ $t3 \in \text{trm}$ $t4 \in \text{trm}$
and $x1 \neq x2$ $x1 \neq x3$ $x2 \neq x3$ $x1 \neq x4$ $x2 \neq x4$ $x3 \neq x4$
 $x2 \notin \text{FvarsT } t1$ $x3 \notin \text{FvarsT } t1$ $x3 \notin \text{FvarsT } t2$ $x4 \notin \text{FvarsT } t2$ $x4 \notin \text{FvarsT } t3$
shows $\text{psubst } \varphi [(t1,x1),(t2,x2),(t3,x3),(t4,x4)] = \text{rawpsubst } \varphi [(t1,x1),(t2,x2),(t3,x3),(t4,x4)]$
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 $\varphi \in \text{fmla}$ $\text{set } xs \subseteq \text{var}$
shows $\text{rawpsubst } \varphi (\text{map } (\lambda x. (\text{Var } x,x)) xs) = \varphi$
using *assms* **by** (*induct xs*) *auto*

lemma *psubst_same_Var[simp]*:

assumes $\varphi \in \text{fmla}$ $\text{set } xs \subseteq \text{var}$ **and** *distinct xs*
shows $\text{psubst } \varphi (\text{map } (\lambda x. (\text{Var } x,x)) xs) = \varphi$

proof–

have $\text{psubst } \varphi (\text{map } (\lambda x. (\text{Var } x,x)) xs) = \text{rawpsubst } \varphi (\text{map } (\lambda x. (\text{Var } x,x)) xs)$

```

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)  $\subseteq$  var fst ' (set txs)  $\subseteq$  trm  $\varphi \in$  fmla
and Fvars  $\varphi \cap$  snd ' (set txs) = {}
shows rawpsubst  $\varphi$  txs =  $\varphi$ 
using assms by (induct txs) auto

```

```

lemma psubst_notIn[simp]:
assumes x  $\in$  var snd ' (set txs)  $\subseteq$  var fst ' (set txs)  $\subseteq$  trm  $\varphi \in$  fmla
and Fvars  $\varphi \cap$  snd ' (set txs) = {}
shows psubst  $\varphi$  txs =  $\varphi$ 

```

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
using getFrN_Fvars[of map snd txs map fst txs [ $\varphi$ ] _ length txs]
  getFrN_FvarsT[of map snd txs map fst txs [ $\varphi$ ] _ length txs]
  getFrN_var[of map snd txs map fst txs [ $\varphi$ ] _ length txs]
  getFrN_length[of map snd txs map fst txs [ $\varphi$ ] 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  $\varphi$  (zip (map Var us) (map snd txs)) =  $\varphi$ 
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  $\varphi$  (zip (map fst txs) us) =  $\varphi$ 
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_numI[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  $\Rightarrow$  'trm  $\Rightarrow$  'fmla
  and
  cnj :: 'fmla  $\Rightarrow$  'fmla  $\Rightarrow$  'fmla
  and
  imp :: 'fmla  $\Rightarrow$  'fmla  $\Rightarrow$  'fmla
  and
  all :: 'var  $\Rightarrow$  'fmla  $\Rightarrow$  'fmla
  and
  exi :: 'var  $\Rightarrow$  'fmla  $\Rightarrow$  'fmla
assumes
  eql[simp,intro]:  $\bigwedge t1 t2. t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow eql\ t1\ t2 \in fmla$ 
  and
  cnj[simp,intro]:  $\bigwedge \varphi1\ \varphi2. \varphi1 \in fmla \Longrightarrow \varphi2 \in fmla \Longrightarrow cnj\ \varphi1\ \varphi2 \in fmla$ 
  and
  imp[simp,intro]:  $\bigwedge \varphi1\ \varphi2. \varphi1 \in fmla \Longrightarrow \varphi2 \in fmla \Longrightarrow imp\ \varphi1\ \varphi2 \in fmla$ 
  and
  all[simp,intro]:  $\bigwedge x\ \varphi. x \in var \Longrightarrow \varphi \in fmla \Longrightarrow all\ x\ \varphi \in fmla$ 
  and
  exi[simp,intro]:  $\bigwedge x\ \varphi. x \in var \Longrightarrow \varphi \in fmla \Longrightarrow exi\ x\ \varphi \in fmla$ 
  and
  Fvars_eql[simp]:
   $\bigwedge t1\ t2. t1 \in trm \Longrightarrow t2 \in trm \Longrightarrow Fvars\ (eql\ t1\ t2) = FvarsT\ t1 \cup FvarsT\ t2$ 
  and
  Fvars_cnj[simp]:
   $\bigwedge \varphi\ \chi. \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow Fvars\ (cnj\ \varphi\ \chi) = Fvars\ \varphi \cup Fvars\ \chi$ 
  and
  Fvars_imp[simp]:
   $\bigwedge \varphi\ \chi. \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow Fvars\ (imp\ \varphi\ \chi) = Fvars\ \varphi \cup Fvars\ \chi$ 
  and
  Fvars_all[simp]:
   $\bigwedge x\ \varphi. x \in var \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ (all\ x\ \varphi) = Fvars\ \varphi - \{x\}$ 
  and
  Fvars_exi[simp]:
   $\bigwedge x\ \varphi. x \in var \Longrightarrow \varphi \in fmla \Longrightarrow Fvars\ (exi\ x\ \varphi) = Fvars\ \varphi - \{x\}$ 
  and
  — Assumed properties of substitution
  subst_cnj[simp]:
   $\bigwedge x\ \varphi\ \chi\ t. \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow$ 
   $subst\ (cnj\ \varphi\ \chi)\ t\ x = cnj\ (subst\ \varphi\ t\ x)\ (subst\ \chi\ t\ x)$ 
  and
  subst_imp[simp]:
   $\bigwedge x\ \varphi\ \chi\ t. \varphi \in fmla \Longrightarrow \chi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow$ 
   $subst\ (imp\ \varphi\ \chi)\ t\ x = imp\ (subst\ \varphi\ t\ x)\ (subst\ \chi\ t\ x)$ 
  and
  subst_all[simp]:
   $\bigwedge x\ y\ \varphi\ t. \varphi \in fmla \Longrightarrow t \in trm \Longrightarrow x \in var \Longrightarrow y \in var \Longrightarrow$ 
   $x \neq y \Longrightarrow x \notin FvarsT\ t \Longrightarrow subst\ (all\ x\ \varphi)\ t\ y = all\ x\ (subst\ \varphi\ t\ y)$ 
and

```

```

subst_exi[simp]:
 $\bigwedge x y \varphi t. \varphi \in \text{fmla} \implies t \in \text{trm} \implies x \in \text{var} \implies y \in \text{var} \implies$ 
 $x \neq y \implies x \notin \text{FvarsT } t \implies \text{subst } (\text{exi } x \varphi) t y = \text{exi } x (\text{subst } \varphi t y)$ 
and
subst_eql[simp]:
 $\bigwedge t1 t2 t x. t \in \text{trm} \implies t1 \in \text{trm} \implies t2 \in \text{trm} \implies x \in \text{var} \implies$ 
 $\text{subst } (\text{eqL } t1 t2) t x = \text{eqL } (\text{substT } t1 t x) (\text{substT } t2 t x)$ 
begin

```

Formula equivalence, \longleftrightarrow , a derived connective

definition $\text{eqv} :: 'fmla \Rightarrow 'fmla \Rightarrow 'fmla$ **where**
 $\text{eqv } \varphi \chi = \text{cnj } (\text{imp } \varphi \chi) (\text{imp } \chi \varphi)$

lemma

$\text{eqv}[simp]: \bigwedge \varphi \chi. \varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies \text{eqv } \varphi \chi \in \text{fmla}$

and

$\text{Fvars_eqv}[simp]: \bigwedge \varphi \chi. \varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies$

$\text{Fvars } (\text{eqv } \varphi \chi) = \text{Fvars } \varphi \cup \text{Fvars } \chi$

and

$\text{subst_eqv}[simp]:$

$\bigwedge \varphi \chi t x. \varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies t \in \text{trm} \implies x \in \text{var} \implies$

$\text{subst } (\text{eqv } \varphi \chi) t x = \text{eqv } (\text{subst } \varphi t x) (\text{subst } \chi t x)$

unfolding eqv_def **by** auto

lemma $\text{subst_all_idle}[simp]:$

assumes $[simp]: x \in \text{var } \varphi \in \text{fmla } t \in \text{trm}$

shows $\text{subst } (\text{all } x \varphi) t x = \text{all } x \varphi$

by $(\text{intro } \text{subst_notIn}) \text{ auto}$

lemma $\text{subst_exi_idle}[simp]:$

assumes $[simp]: x \in \text{var } \varphi \in \text{fmla } t \in \text{trm}$

shows $\text{subst } (\text{exi } x \varphi) t x = \text{exi } x \varphi$

by $(\text{rule } \text{subst_notIn}) \text{ auto}$

Parallel substitution versus connectives and quantifiers.

lemma $\text{rawpsubst_cnj}:$

assumes $\varphi1 \in \text{fmla } \varphi2 \in \text{fmla}$

and $\text{snd } '(\text{set } \text{txs}) \subseteq \text{var } \text{fst } '(\text{set } \text{txs}) \subseteq \text{trm}$

shows $\text{rawpsubst } (\text{cnj } \varphi1 \varphi2) \text{txs} = \text{cnj } (\text{rawpsubst } \varphi1 \text{txs}) (\text{rawpsubst } \varphi2 \text{txs})$

using assms **by** $(\text{induct } \text{txs } \text{arbitrary: } \varphi1 \varphi2) \text{ auto}$

lemma $\text{psubst_cnj}[simp]:$

assumes $\varphi1 \in \text{fmla } \varphi2 \in \text{fmla}$

and $\text{snd } '(\text{set } \text{txs}) \subseteq \text{var } \text{fst } '(\text{set } \text{txs}) \subseteq \text{trm}$

and $\text{distinct } (\text{map } \text{snd } \text{txs})$

shows $\text{psubst } (\text{cnj } \varphi1 \varphi2) \text{txs} = \text{cnj } (\text{psubst } \varphi1 \text{txs}) (\text{psubst } \varphi2 \text{txs})$

proof-

define us **where** $us: us = \text{getFrN } (\text{map } \text{snd } \text{txs}) (\text{map } \text{fst } \text{txs}) [\text{cnj } \varphi1 \varphi2] (\text{length } \text{txs})$

have $us_facts: \text{set } us \subseteq \text{var}$

$\text{set } us \cap \text{Fvars } \varphi1 = \{\}$

$\text{set } us \cap \text{Fvars } \varphi2 = \{\}$

$\text{set } us \cap \bigcup (\text{FvarsT } '(\text{fst } '(\text{set } \text{txs}))) = \{\}$

$\text{set } us \cap \text{snd } '(\text{set } \text{txs}) = \{\}$

$\text{length } us = \text{length } \text{txs}$

$\text{distinct } us$

using assms **unfolding** us

using $\text{getFrN_Fvars}[of \text{ map } \text{snd } \text{txs } \text{map } \text{fst } \text{txs } [\text{cnj } \varphi1 \varphi2] _ \text{length } \text{txs}]$

$\text{getFrN_FvarsT}[of \text{ map } \text{snd } \text{txs } \text{map } \text{fst } \text{txs } [\text{cnj } \varphi1 \varphi2] _ \text{length } \text{txs}]$

```

  getFrN_var[of map snd txs map fst txs [cnj  $\varphi 1$   $\varphi 2$ ] _ length txs]
  getFrN_length[of map snd txs map fst txs [cnj  $\varphi 1$   $\varphi 2$ ] length txs]
apply –
subgoal by auto
subgoal by fastforce
subgoal by fastforce
subgoal by fastforce
subgoal by (fastforce simp: image_iff)
subgoal by auto
subgoal 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  $\cap \bigcup (FvarsT \text{ ' (fst ' (set txs))}) = \{\}$ 
  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
subgoal by auto
subgoal 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  $\cap \bigcup (FvarsT \text{ ' (fst ' (set txs))}) = \{\}$ 
  set vs2  $\cap$  snd ' (set txs) =  $\{\}$ 
  length vs2 = length txs
  distinct vs2
using assms unfolding vs2
using getFrN_Fvars[of map snd txs map fst txs [ $\varphi 2$ ] _ length txs]
  getFrN_FvarsT[of map snd txs map fst txs [ $\varphi 2$ ] _ length txs]
  getFrN_var[of map snd txs map fst txs [ $\varphi 2$ ] _ length txs]
  getFrN_length[of map snd txs map fst txs [ $\varphi 2$ ] 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  $\varphi 1$   $\varphi 2$ ) ?uxs

```

```

have c: ?c = cnj (rawpsubst  $\varphi 1$  ?uxs) (rawpsubst  $\varphi 2$  ?uxs)
  using assms us_facts
  by (intro rawpsubst_cnj) (auto intro!: rawpsubstT dest!: set_zip_D)
have 0: rawpsubst ?c ?tus =
  cnj (rawpsubst (rawpsubst  $\varphi 1$  ?uxs) ?tus) (rawpsubst (rawpsubst  $\varphi 2$  ?uxs) ?tus)
  unfolding c using assms us_facts
  by (intro rawpsubst_cnj) (auto dest!: set_zip_D intro!: rawpsubst)

have 1: rawpsubst (rawpsubst  $\varphi 1$  ?uxs) ?tus = rawpsubst (rawpsubst  $\varphi 1$  ?vxs1) ?tus1
  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) ?tus2
  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  $\varphi 1 \in \text{fmla}$   $\varphi 2 \in \text{fmla}$ 
  and  $\text{snd} \text{ ' (set txs) } \subseteq \text{var fst ' (set txs) } \subseteq \text{trm}$ 
  shows rawpsubst (imp  $\varphi 1$   $\varphi 2$ ) txs = imp (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_imp[simp]:
  assumes  $\varphi 1 \in \text{fmla}$   $\varphi 2 \in \text{fmla}$ 
  and  $\text{snd} \text{ ' (set txs) } \subseteq \text{var fst ' (set txs) } \subseteq \text{trm}$ 
  and distinct (map snd txs)
  shows psubst (imp  $\varphi 1$   $\varphi 2$ ) txs = imp (psubst  $\varphi 1$  txs) (psubst  $\varphi 2$  txs)
proof-
  define us where us: us = getFrN (map snd txs) (map fst txs) [imp  $\varphi 1$   $\varphi 2$ ] (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  $\cap$   $\text{snd} \text{ ' (set txs) } = \{\}$ 
  length us = length txs
  distinct us
  using assms unfolding us
  using getFrN_Fvars[of map snd txs map fst txs [imp  $\varphi 1$   $\varphi 2$ ] _ length txs]
  getFrN_FvarsT[of map snd txs map fst txs [imp  $\varphi 1$   $\varphi 2$ ] _ length txs]
  getFrN_var[of map snd txs map fst txs [imp  $\varphi 1$   $\varphi 2$ ] _ length txs]
  getFrN_length[of map snd txs map fst txs [imp  $\varphi 1$   $\varphi 2$ ] 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  $\cap$   $\text{snd} \text{ ' (set txs) } = \{\}$ 

```



```

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  $\cap \bigcup (FvarsT \text{ ‘ (fst ‘ (set txs))}) = \{\}$ 
  set vs2  $\cap$  snd ‘ (set txs) =  $\{\}$ 
  length vs2 = length txs
  distinct vs2
using assms unfolding vs2
using getFrN_Fvars[of map snd txs map fst txs [ $\varphi 2$ ] _ length txs]
  getFrN_FvarsT[of map snd txs map fst txs [ $\varphi 2$ ] _ length txs]
  getFrN_var[of map snd txs map fst txs [ $\varphi 2$ ] _ length txs]
  getFrN_length[of map snd txs map fst txs [ $\varphi 2$ ] 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 ?tus1 = zip (map fst txs) vs1
let ?vxs1 = zip (map Var vs1) (map snd txs)
let ?tus2 = zip (map fst txs) vs2
let ?vxs2 = zip (map Var vs2) (map snd txs)

let ?c = rawpsubst (imp  $\varphi 1$   $\varphi 2$ ) ?uxs
have c: ?c = imp (rawpsubst  $\varphi 1$  ?uxs) (rawpsubst  $\varphi 2$  ?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  $\varphi 1$  ?uxs) ?tus) (rawpsubst (rawpsubst  $\varphi 2$  ?uxs) ?tus)
  unfolding c
  using assms us_facts
  by (intro rawpsubst_imp) (auto intro!: rawpsubst dest!: set_zip_D)
have 1: rawpsubst (rawpsubst  $\varphi 1$  ?uxs) ?tus = rawpsubst (rawpsubst  $\varphi 1$  ?vxs1) ?tus1
  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) ?tus2
  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 $\varphi 1 \in \text{fmla}$ $\varphi 2 \in \text{fmla}$
and $\text{snd} \text{ ' (set txs) } \subseteq \text{var fst ' (set txs) } \subseteq \text{trm}$
shows $\text{rawpsubst (eqv } \varphi 1 \varphi 2) \text{ txs} = \text{eqv (rawpsubst } \varphi 1 \text{ txs) (rawpsubst } \varphi 2 \text{ txs)}$
using *assms apply* (*induct txs arbitrary:* $\varphi 1 \varphi 2$ *)*
subgoal by *auto*
subgoal for $\text{tx txs } \varphi 1 \varphi 2$ **by** (*cases tx auto*).

lemma *psubst_eqv[simp]*:

assumes $\varphi 1 \in \text{fmla}$ $\varphi 2 \in \text{fmla}$
and $\text{snd} \text{ ' (set txs) } \subseteq \text{var fst ' (set txs) } \subseteq \text{trm}$
and *distinct (map snd txs)*
shows $\text{psubst (eqv } \varphi 1 \varphi 2) \text{ txs} = \text{eqv (psubst } \varphi 1 \text{ txs) (psubst } \varphi 2 \text{ txs)}$

proof–

define *us* **where** $us = \text{getFrN (map snd txs) (map fst txs) [eqv } \varphi 1 \varphi 2] (\text{length txs})$
have *us_facts*: $\text{set us} \subseteq \text{var}$
 $\text{set us} \cap \text{Fvars } \varphi 1 = \{\}$
 $\text{set us} \cap \text{Fvars } \varphi 2 = \{\}$
 $\text{set us} \cap \bigcup (\text{FvarsT ' (fst ' (set txs))}) = \{\}$
 $\text{set us} \cap \text{snd ' (set txs)} = \{\}$
 $\text{length us} = \text{length txs}$
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** $vs1 = \text{getFrN (map snd txs) (map fst txs) } [\varphi 1] (\text{length txs})$

have *vs1_facts*: $\text{set vs1} \subseteq \text{var}$
 $\text{set vs1} \cap \text{Fvars } \varphi 1 = \{\}$
 $\text{set vs1} \cap \bigcup (\text{FvarsT ' (fst ' (set txs))}) = \{\}$
 $\text{set vs1} \cap \text{snd ' (set txs)} = \{\}$
 $\text{length vs1} = \text{length txs}$
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** $vs2 = \text{getFrN (map snd txs) (map fst txs) } [\varphi 2] (\text{length txs})$

have *vs2_facts*: $\text{set vs2} \subseteq \text{var}$

```

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 ?tus1 = zip (map fst txs) vs1
let ?vxs1 = zip (map Var vs1) (map snd txs)
let ?tus2 = 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) ?tus1
  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) ?tus2
  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  $\subseteq$  var
  set us  $\cap$  (Fvars φ - {y}) = {}
  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 [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  $\subseteq$  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)
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 (all y  $\varphi$ ) (zip (map Var ws) (map snd txs)) =
  all y (rawpsubst  $\varphi$  (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  $\varphi$  (zip (map Var ws) (map snd txs)))) (zip (map fst txs) ws) =
  rawpsubst ((rawpsubst  $\varphi$  (zip (map Var ws) (map snd txs)))) (zip (map fst txs) ws)
apply(rule rawpsubst_compose_freshVar2)
using assms ws_facts vs_facts by (auto intro!: rawpsubst)
have rawpsubst (rawpsubst (all y  $\varphi$ ) (zip (map Var ws) (map snd txs))) (zip (map fst txs) ws) =
  rawpsubst (rawpsubst (all y  $\varphi$ ) (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  $\varphi$  (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  $\varphi$  (zip (map Var ws) (map snd txs))) (zip (map fst txs) ws))
unfolding 1 ..
finally show ?thesis unfolding psubst_def by (simp add: Let_def us[symmetric] vs[symmetric])
qed

```

```

lemma rawpsubst_exi:
assumes  $\varphi \in \text{fmla}$   $y \in \text{var}$ 
  and  $\text{snd } \varphi \subseteq \text{var}$   $\text{fst } \varphi \subseteq \text{trm}$ 
  and  $y \notin \text{snd } \varphi$   $y \notin \bigcup (\text{FvarsT } \varphi \text{ fst } \varphi)$ 
shows rawpsubst (exi y  $\varphi$ ) txs = exi y (rawpsubst  $\varphi$  txs)
using assms apply (induct txs arbitrary:  $\varphi$ )
subgoal by auto
subgoal for tx txs  $\varphi$  by (cases tx) auto .

```

```

lemma psubst_exi[simp]:
assumes  $\varphi \in \text{fmla}$   $y \in \text{var}$ 
  and  $\text{snd } \varphi \subseteq \text{var}$   $\text{fst } \varphi \subseteq \text{trm}$ 
  and  $y \notin \text{snd } \varphi$   $y \notin \bigcup (\text{FvarsT } \varphi \text{ fst } \varphi)$ 
  and distinct (map snd txs)
shows psubst (exi y  $\varphi$ ) txs = exi y (psubst  $\varphi$  txs)
proof-
define us where us: us = getFrN (map snd txs) (map fst txs) [exi y  $\varphi$ ] (length txs)
have us_facts: set us  $\subseteq$  var
  set us  $\cap$  (Fvars  $\varphi$  - {y}) = {}
  set us  $\cap \bigcup (\text{FvarsT } \varphi \text{ fst } \varphi)$  = {}
  set us  $\cap \text{snd } \varphi$  = {}
  length us = length txs
  distinct us
using assms unfolding us
using getFrN_Fvars[of map snd txs map fst txs [exi y  $\varphi$ ] _ length txs]
  getFrN_FvarsT[of map snd txs map fst txs [exi y  $\varphi$ ] _ length txs]
  getFrN_var[of map snd txs map fst txs [exi y  $\varphi$ ] _ length txs]
  getFrN_length[of map snd txs map fst txs [exi y  $\varphi$ ] 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 ⊆ 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)
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 (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 ws) (map snd txs)))) (zip (map fst txs) ws)

```

```

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 us) (map snd txs))) (zip (map fst txs) us)
using assms ws_facts us_facts
by (intro rawpsubst_compose_freshVar2) (auto intro!: rawpsubst)
also have
  ... = exi y (rawpsubst ((rawpsubst φ (zip (map Var us) (map snd txs)))) (zip (map fst txs) us))
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 us) (map snd txs))) (zip (map fst txs) us))
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 φ ∈ fmla x ∈ var y ∈ var n ∈ num
shows x ≠ y ⇒ subst (all x φ) n y = all x (subst φ n y)
using assms by simp

```

lemma *subst_exi_num[simp]*:

```

assumes φ ∈ fmla x ∈ var y ∈ var n ∈ num
shows x ≠ y ⇒ subst (exi x φ) n y = exi x (subst φ 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]*: *φ ∈ fmla ⇒ t ∈ trm ⇒ x ∈ var ⇒ softSubst φ t x ∈ fmla*

unfolding *softSubst_def by auto*

lemma *Fvars_softSubst[simp]*:

```

φ ∈ fmla ⇒ t ∈ trm ⇒ x ∈ var ⇒
Fvars (softSubst φ t x) = (Fvars φ ∪ FvarsT t - {x})
unfolding softSubst_def by auto

```

lemma *Fvars_softSubst_subst_in*:

```

φ ∈ fmla ⇒ t ∈ trm ⇒ x ∈ var ⇒ x ∉ FvarsT t ⇒ x ∈ Fvars φ ⇒

```

$Fvars (softSubst \varphi t x) = Fvars (subst \varphi t x)$
by auto

lemma *Fvars_softSubst_subst_notIn*:

$\varphi \in fmla \implies t \in trm \implies x \in var \implies x \notin FvarsT t \implies x \notin Fvars \varphi \implies$
 $Fvars (softSubst \varphi t x) = Fvars (subst \varphi t x) \cup FvarsT t$
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]:

$\bigwedge t x. t \in trm \implies x \in var \implies subst\ fls\ t\ x = fls$

begin

Negation as a derived connective:

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

neg \varphi = imp \varphi fls

lemma

neg[simp]: \bigwedge \varphi. \varphi \in fmla \implies neg \varphi \in fmla

and

Fvars_neg[simp]: \bigwedge \varphi. \varphi \in fmla \implies Fvars (neg \varphi) = Fvars \varphi

and

subst_neg[simp]:

$\bigwedge \varphi t x. \varphi \in fmla \implies t \in trm \implies x \in var \implies$

$subst (neg \varphi) t x = neg (subst \varphi t x)$

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 \implies x \in var \implies 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  $\Rightarrow$  'fmla where
  lcnj [] = tru
| lcnj ( $\varphi$  #  $\varphi$ s) = cnj  $\varphi$  (lcnj  $\varphi$ s)

```

```

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

```

```

lemma Fvars_lcnj[simp]:
  set  $\varphi$ s  $\subseteq$  fmla  $\Longrightarrow$  finite F  $\Longrightarrow$  Fvars (lcnj  $\varphi$ s) =  $\bigcup$  (set (map Fvars  $\varphi$ s))
  by(induct  $\varphi$ s) auto

```

```

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

```

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

```

definition scnj :: 'fmla set  $\Rightarrow$  'fmla where
  scnj F = lcnj (asList F)

```

```

lemma scnj[simp,intro!]: F  $\subseteq$  fmla  $\Longrightarrow$  finite F  $\Longrightarrow$  scnj F  $\in$  fmla
  unfolding scnj_def by auto

```

```

lemma Fvars_scnj[simp]:
  F  $\subseteq$  fmla  $\Longrightarrow$  finite F  $\Longrightarrow$  Fvars (scnj F) =  $\bigcup$  (Fvars ' F)
  unfolding scnj_def by auto

```

2.3.2 Parallel substitution versus the new connectives

```

lemma rawpsubst_fls:
  snd ' (set txs)  $\subseteq$  var  $\Longrightarrow$  fst ' (set txs)  $\subseteq$  trm  $\Longrightarrow$  rawpsubst fls txs = fls
  by (induct txs) auto

```

```

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

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

```
lemma psubst_lcnj[simp]:  
  set  $\varphi s \subseteq \text{fmla} \implies \text{snd } \text{'(set txs)} \subseteq \text{var} \implies \text{fst } \text{'(set txs)} \subseteq \text{trm} \implies$   
   $\text{distinct (map snd txs)} \implies$   
   $\text{psubst (lcnj } \varphi s) \text{ txs} = \text{lcnj (map } (\lambda \varphi. \text{psubst } \varphi \text{ txs)} \varphi s)$   
  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]:  $\bigwedge \varphi \chi. \varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies \text{dsj } \varphi \chi \in \text{fmla}$   
    and  
    Fvars_dsj[simp]:  $\bigwedge \varphi \chi. \varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies$   
     $\text{Fvars (dsj } \varphi \chi) = \text{Fvars } \varphi \cup \text{Fvars } \chi$   
    and  
    subst_dsj[simp]:  
     $\bigwedge x \varphi \chi t. \varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies t \in \text{trm} \implies x \in \text{var} \implies$   
     $\text{subst (dsj } \varphi \chi) t x = \text{dsj (subst } \varphi t x) (\text{subst } \chi t x)$   
  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 *ldsjsimp.intro!*: $set \varphi s \subseteq fmla \implies ldsj \varphi s \in fmla$
by (*induct* φs) *auto*

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

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

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

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

lemma *sdsjsimp.intro!*: $F \subseteq fmla \implies finite F \implies sdsj F \in fmla$
unfolding *sdsj_def* **by** *auto*

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

2.4.2 Parallel substitution versus the new connectives

lemma *rawpsubst_dsjsimp*:
assumes $\varphi 1 \in fmla \varphi 2 \in fmla$
and $snd ' (set txs) \subseteq var fst ' (set txs) \subseteq trm$
shows $rawpsubst (dsjsimp \varphi 1 \varphi 2) txs = dsjsimp (rawpsubst \varphi 1 txs) (rawpsubst \varphi 2 txs)$
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_dsjsimp*:
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 $psubst (dsjsimp \varphi 1 \varphi 2) txs = dsjsimp (psubst \varphi 1 txs) (psubst \varphi 2 txs)$

proof–

define *us* **where** *us*: $us = getFrN (map snd txs) (map fst txs) [dsjsimp \varphi 1 \varphi 2] (length txs)$
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* $map snd txs map fst txs [dsjsimp \varphi 1 \varphi 2] _ length txs$]
 $getFrN_FvarsT$ [*of* $map snd txs map fst txs [dsjsimp \varphi 1 \varphi 2] _ length txs$]
 $getFrN_var$ [*of* $map snd txs map fst txs [dsjsimp \varphi 1 \varphi 2] _ length txs$]
 $getFrN_length$ [*of* $map snd txs map fst txs [dsjsimp \varphi 1 \varphi 2] length txs$]
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) [ $\varphi 1$ ] (length txs)
have vs1_facts: set vs1  $\subseteq$  var
  set vs1  $\cap$  Fvars  $\varphi 1 = \{\}$ 
  set vs1  $\cap \bigcup (FvarsT \text{ ' (fst ' (set txs))}) = \{\}$ 
  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 fastforce
subgoal by fastforce
subgoal by (fastforce simp: image_iff)
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  $\cap \bigcup (FvarsT \text{ ' (fst ' (set txs))}) = \{\}$ 
  set vs2  $\cap$  snd ' (set txs) =  $\{\}$ 
  length vs2 = length txs
  distinct vs2
using assms unfolding vs2
using getFrN_Fvars[of map snd txs map fst txs [ $\varphi 2$ ] _ length txs]
  getFrN_FvarsT[of map snd txs map fst txs [ $\varphi 2$ ] _ length txs]
  getFrN_var[of map snd txs map fst txs [ $\varphi 2$ ] _ length txs]
  getFrN_length[of map snd txs map fst txs [ $\varphi 2$ ] 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 ?tus1 = zip (map fst txs) vs1
let ?vxs1 = zip (map Var vs1) (map snd txs)
let ?tus2 = zip (map fst txs) vs2
let ?vxs2 = zip (map Var vs2) (map snd txs)

let ?c = rawpsubst (dsj  $\varphi 1$   $\varphi 2$ ) ?uxs
have c: ?c = dsj (rawpsubst  $\varphi 1$  ?uxs) (rawpsubst  $\varphi 2$  ?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  $\varphi 1$  ?uxs) ?tus) (rawpsubst (rawpsubst  $\varphi 2$  ?uxs) ?tus)
  unfolding c using assms us_facts
  by (intro rawpsubst_dsj) (auto intro!: rawpsubst dest!: set_zip_D)
have 1: rawpsubst (rawpsubst  $\varphi 1$  ?uxs) ?tus = rawpsubst (rawpsubst  $\varphi 1$  ?vxs1) ?tus1

```

```

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) ?tus2
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  $\implies$  snd ‘(set txs)  $\subseteq$  var  $\implies$  fst ‘(set txs)  $\subseteq$  trm  $\implies$ 
  distinct (map snd txs)  $\implies$ 
  psubst (ldsj  $\varphi$ s) txs = ldsj (map ( $\lambda\varphi$ . psubst  $\varphi$  txs)  $\varphi$ s)
  by (induct  $\varphi$ s) 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*:

$\varphi \in \text{fm}la \implies x \in \text{var} \implies y \in \text{var} \implies (y = x \vee y \notin \text{Fvars } \varphi) \implies$
 $\text{all } x \varphi = \text{all } y (\text{subst } \varphi (\text{Var } y) x)$
using *all_rename* **by** (*cases* $y = x$) (*auto simp del: Fvars_subst*)

lemma *exi_rename2*:

$\varphi \in \text{fm}la \implies x \in \text{var} \implies y \in \text{var} \implies (y = x \vee y \notin \text{Fvars } \varphi) \implies$
 $\text{exi } x \varphi = \text{exi } y (\text{subst } \varphi (\text{Var } y) x)$
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 \Rightarrow 'fm}la \Rightarrow 'fm}la **where**

$\text{exu } x \varphi \equiv \text{let } y = \text{getFr } [x] [] [\varphi] \text{ in}$
 $\text{cnj } (\text{exi } x \varphi) (\text{exi } y (\text{all } x (\text{imp } \varphi (\text{eql } (\text{Var } x) (\text{Var } y))))))$

lemma *exu[simp,intro]*:

$x \in \text{var} \implies \varphi \in \text{fm}la \implies \text{exu } x \varphi \in \text{fm}la$
unfolding *exu_def* **by** (*simp add: Let_def*)

lemma *Fvars_exu[simp]*:

$x \in \text{var} \implies \varphi \in \text{fm}la \implies \text{Fvars } (\text{exu } x \varphi) = \text{Fvars } \varphi - \{x\}$
unfolding *exu_def* **by** (*auto simp: Let_def getFr_Fvars*)

lemma *exu_def_var*:

assumes [*simp*]: $x \in \text{var } y \in \text{var } y \neq x \ y \notin \text{Fvars } \varphi \ \varphi \in \text{fm}la$
shows

$\text{exu } x \varphi = \text{cnj } (\text{exi } x \varphi) (\text{exi } y (\text{all } x (\text{imp } \varphi (\text{eql } (\text{Var } x) (\text{Var } y))))))$

proof–

have [*simp*]: $x \neq y$ **using** *assms* **by** *blast*

define *z* **where** $z \equiv \text{getFr } [x] [] [\varphi]$

have *z_facts*[*simp*]: $z \in \text{var } z \neq x \ x \neq z \ z \notin \text{Fvars } \varphi$

unfolding *z* **using** *getFr_FvarsT_Fvars*[*of* [*x*] [] [*φ*]] **by** *auto*

define *u* **where** $u \equiv \text{getFr } [x,y,z] [] [\varphi]$

have *u_facts*[*simp*]: $u \in \text{var } u \neq x \ u \neq z \ y \neq u \ u \neq y \ x \neq u \ z \neq u \ u \notin \text{Fvars } \varphi$

unfolding *u* **using** *getFr_FvarsT_Fvars*[*of* [*x,y,z*] [] [*φ*]] **by** *auto*

have $\text{exu } x \varphi = \text{cnj } (\text{exi } x \varphi) (\text{exi } u (\text{all } x (\text{imp } \varphi (\text{eql } (\text{Var } x) (\text{Var } u))))))$

by (*auto simp: exu_def Let_def z[symmetric] exi_rename*[*of* $\text{all } x (\text{imp } \varphi (\text{eql } (\text{Var } x) (\text{Var } z))) z u$])

also have $\dots = \text{cnj } (\text{exi } x \varphi) (\text{exi } y (\text{all } x (\text{imp } \varphi (\text{eql } (\text{Var } x) (\text{Var } y))))))$

by (*auto simp: exi_rename*[*of* $\text{all } x (\text{imp } \varphi (\text{eql } (\text{Var } x) (\text{Var } u))) u y$])

split: if_splits)

finally show *?thesis* .

qed

lemma *subst_exu[simp]*:

assumes [*simp*]: $\varphi \in \text{fm}la \ t \in \text{trm } x \in \text{var } y \in \text{var } x \neq y \ x \notin \text{FvarsT } t$
shows $\text{subst } (\text{exu } x \varphi) t \ y = \text{exu } x (\text{subst } \varphi t \ y)$

proof–

define *u* **where** $u \equiv \text{getFr } [x,y] [t] [\varphi]$

have *u_facts*[*simp*]: $u \in \text{var } u \neq x \ u \neq y \ y \neq u \ x \neq u$

$u \notin \text{FvarsT } t \ u \notin \text{Fvars } \varphi$

unfolding *u* **using** *getFr_FvarsT_Fvars*[*of* [*x,y*] [*t*] [*φ*]] **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]: $x \in \text{var } \varphi \in \text{fmla } t \in \text{trm}$
shows $\text{subst } (\text{exu } x \ \varphi) \ t \ x = \text{exu } x \ \varphi$
by (intro subst_notIn) auto

lemma exu_rename:
assumes [simp]: $\varphi \in \text{fmla } x \in \text{var } y \in \text{var } y \notin \text{Fvars } \varphi$
shows $\text{exu } x \ \varphi = \text{exu } y \ (\text{subst } \varphi \ (\text{Var } y) \ x)$
proof(cases $y = x$)
 case [simp]: *False*
 define z **where** $z = \text{getFr } [x] \ [] \ [\varphi]$
 have $z_facts[simp]: z \in \text{var } z \neq x \ x \neq z \ z \notin \text{Fvars } \varphi$
 unfolding z **using** $\text{getFr_FvarsT_Fvars}$ [of $[x] \ [] \ [\varphi]$] **by** auto
 define u **where** $u \equiv \text{getFr } [x,y,z] \ [] \ [\varphi]$
 have $u_facts[simp]: u \in \text{var } u \neq x \ x \neq u \ u \neq y \ y \neq u \ u \neq z \ z \neq u$
 $u \notin \text{Fvars } \varphi$
 unfolding u **using** $\text{getFr_FvarsT_Fvars}$ [of $[x,y,z] \ [] \ [\varphi]$] **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:
 $\varphi \in \text{fmla} \implies x \in \text{var} \implies y \in \text{var} \implies (y = x \vee y \notin \text{Fvars } \varphi) \implies$
 $\text{exu } x \ \varphi = \text{exu } y \ (\text{subst } \varphi \ (\text{Var } y) \ x)$
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* \Rightarrow *bool*

— Hilbert-style system for intuitionistic logic over $\rightarrow, \wedge, \forall, \exists$. (\perp, \neg and \vee 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:

$\bigwedge \varphi \chi. \varphi \in \text{fmla} \Rightarrow \chi \in \text{fmla} \Rightarrow$

$\text{prv} (\text{imp } \varphi \chi) \Rightarrow \text{prv } \varphi \Rightarrow \text{prv } \chi$

and

— The propositional intuitionistic axioms:

prv_imp_imp_triv:

$\bigwedge \varphi \chi. \varphi \in \text{fmla} \Rightarrow \chi \in \text{fmla} \Rightarrow$

$\text{prv} (\text{imp } \varphi (\text{imp } \chi \varphi))$

and

prv_imp_trans:

$\bigwedge \varphi \chi \psi. \varphi \in \text{fmla} \Rightarrow \chi \in \text{fmla} \Rightarrow \psi \in \text{fmla} \Rightarrow$

$\text{prv} (\text{imp} (\text{imp } \varphi (\text{imp } \chi \psi))$

$(\text{imp} (\text{imp } \varphi \chi) (\text{imp } \varphi \psi)))$

and

prv_imp_cnjL:

$\bigwedge \varphi \chi. \varphi \in \text{fmla} \Rightarrow \chi \in \text{fmla} \Rightarrow$

```

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

```

3.1.1 Properties of the propositional fragment

```

lemma prv_imp_triv:
assumes  $\text{phi}: \varphi \in \text{fmla}$  and  $\text{psi}: \psi \in \text{fmla}$ 
shows  $\text{prv } \psi \implies \text{prv } (\text{imp } \varphi \psi)$ 
    by (meson prv_imp_imp_triv prv_imp_mp imp phi psi)

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

lemma prv_imp_refl2:  $\varphi \in \text{fmla} \implies \psi \in \text{fmla} \implies \varphi = \psi \implies \text{prv } (\text{imp } \varphi \psi)$ 
using prv_imp_refl by auto

```

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

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

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

lemma *prv_prv_imp_trans*:
assumes *phi*: $\varphi \in \text{fmla}$ **and** *chi*: $\chi \in \text{fmla}$ **and** *psi*: $\psi \in \text{fmla}$
assumes *1*: $\text{prv } (\text{imp } \varphi \ \chi)$ **and** *2*: $\text{prv } (\text{imp } \chi \ \psi)$
shows $\text{prv } (\text{imp } \varphi \ \psi)$
proof–
have $\text{prv } (\text{imp } \varphi \ (\text{imp } \chi \ \psi))$ **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*: $\varphi \in \text{fmla}$ **and** *chi*: $\chi \in \text{fmla}$ **and** *psi*: $\psi \in \text{fmla}$
shows $\text{prv } (\text{imp } (\text{imp } \chi \ \psi) \ (\text{imp } (\text{imp } \varphi \ \chi) \ (\text{imp } \varphi \ \psi)))$
by (*meson* *chi* *prv_prv_imp_trans* *prv_imp_imp_triv* *prv_imp_trans* *imp* *phi* *psi*)

lemma *prv_imp_com*:
assumes *phi*: $\varphi \in \text{fmla}$ **and** *chi*: $\chi \in \text{fmla}$ **and** *psi*: $\psi \in \text{fmla}$
assumes $\text{prv } (\text{imp } \varphi \ (\text{imp } \chi \ \psi))$
shows $\text{prv } (\text{imp } \chi \ (\text{imp } \varphi \ \psi))$
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*: $\varphi \in \text{fmla}$ **and** *chi*: $\chi \in \text{fmla}$ **and** *psi*: $\psi \in \text{fmla}$
shows $\text{prv } (\text{imp } (\text{imp } \varphi \ \chi) \ (\text{imp } (\text{imp } \chi \ \psi) \ (\text{imp } \varphi \ \psi)))$
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 $\varphi \in \text{fmla}$ **and** $\chi \in \text{fmla}$ **and** $\psi \in \text{fmla}$
shows $\text{prv } (\text{imp } \varphi \ \psi) \implies \text{prv } (\text{imp } \varphi \ \chi) \implies \text{prv } (\text{imp } \varphi \ (\text{cnj } \psi \ \chi))$
proof –
assume $\text{prv } (\text{imp } \varphi \ \psi)$
moreover
assume $\text{prv } (\text{imp } \varphi \ \chi)$
then **have** $\text{prv } (\text{imp } \varphi \ (\text{imp } \psi \ f))$ **if** $\text{prv } (\text{imp } \chi \ f)$ **for** $f \in \text{fmla}$ **for** f
using *that* **by** (*metis* (*no_types*) *assms* *imp* *prv_imp_imp_triv* *prv_prv_imp_trans*)
moreover **have** $\text{prv } (\text{imp } \varphi \ (\text{imp } \psi \ \psi)) \implies \text{prv } (\text{imp } \varphi \ (\text{imp } \varphi \ \psi))$
using $\langle \text{prv } (\text{imp } \varphi \ \psi) \rangle$ **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 $\varphi \in \text{fmla}$ **and** $\chi \in \text{fmla}$ **and** $\psi \in \text{fmla}$

shows

$prv (imp (imp \varphi (imp \chi \psi))$
 $(imp \chi (imp \varphi \psi)))$
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 $prv (imp \varphi (imp \chi \psi))$

shows $prv (imp (cnj \varphi \chi) \psi)$

proof –

have $prv (imp (cnj \varphi \chi) (cnj \varphi \chi))$

using *prv_imp_refl* **by** (*blast intro: assms(1-3)*)

then have $prv (imp (imp (cnj \varphi \chi) (imp (cnj \varphi \chi) \psi)) (imp (cnj \varphi \chi) \psi))$

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

$prv (imp (imp \varphi (imp \chi \psi))$

$(imp (cnj \varphi \chi) \psi))$

proof–

have $prv (imp \varphi (imp (imp \varphi (imp \chi \psi)) (imp \chi \psi)))$

by (*simp add: assms prv_imp_com prv_imp_refl*)

hence $prv (imp \varphi (imp \chi (imp (imp \varphi (imp \chi \psi)) \psi)))$

by (*metis* (*no_types, lifting*) *assms prv_prv_imp_trans prv_imp_imp_com imp*)

hence $prv (imp (cnj \varphi \chi)$

$(imp (imp \varphi (imp \chi \psi)) \psi))$

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

$prv (imp (imp (cnj \varphi \chi) \psi)$

$(imp \varphi (imp \chi \psi)))$

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 $prv (imp (cnj \varphi \chi) \psi)$

shows $prv (imp \varphi (imp \chi \psi))$

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

Monotonicity of conjunction w.r.t. implication:

lemma *prv_cnj_imp_monoR*:

assumes $\varphi \in fmla$ **and** $\chi \in fmla$ **and** $\psi \in fmla$

shows $prv (imp (imp \varphi \chi) (imp (imp \varphi \psi) (imp \varphi (cnj \chi \psi))))$

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 $\varphi1 \in fmla$ **and** $\varphi2 \in fmla$ **and** $\chi \in fmla$

shows $prv (imp (imp \varphi 1 \chi) (imp (cnj \varphi 1 \varphi 2) \chi))$
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 $prv (imp (imp \varphi 2 \chi) (imp (cnj \varphi 1 \varphi 2) \chi))$
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
 $prv (imp \varphi (imp \chi \psi 1))$
 $(imp (imp \varphi (imp \chi \psi 2)) (imp \varphi (imp \chi (cnj \psi 1 \psi 2))))$
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 $prv (imp \varphi (imp \chi \psi 1)) \implies prv (imp \varphi (imp \chi \psi 2)) \implies prv (imp \varphi (imp \chi (cnj \psi 1 \psi 2)))$
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 $prv (imp (imp \varphi 1 \chi 1) (imp \varphi 1 (imp \varphi 2 (imp \varphi 2 (cnj \chi 1 \chi 2))))$
using $prv_cnj_imp\ prv_imp_mp\ prv_imp_trans2$
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  $\varphi 1 \in \text{fmla}$  and  $\varphi 2 \in \text{fmla}$  and  $\chi 1 \in \text{fmla}$  and  $\chi 2 \in \text{fmla}$ 
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  $\varphi \in \text{fmla}$  and  $\chi \in \text{fmla}$ 
shows prv (imp (eqv  $\varphi$   $\chi$ ) (eqv  $\chi$   $\varphi$ ))
by (simp add: assms prv_imp_cnj prv_imp_cnjL prv_imp_cnjR eqv_def)

```

```

lemma prv_eqv_eqv:
assumes  $\varphi \in \text{fmla}$  and  $\chi \in \text{fmla}$ 
shows prv (eqv (eqv  $\varphi$   $\chi$ ) (eqv  $\chi$   $\varphi$ ))
using assms prv_cnjI prv_eqv_imp eqv_def by auto

```

```

lemma prv_imp_eqvEL:
 $\varphi 1 \in \text{fmla} \implies \varphi 2 \in \text{fmla} \implies \text{prv} (\text{eqv } \varphi 1 \ \varphi 2) \implies \text{prv} (\text{imp } \varphi 1 \ \varphi 2)$ 
unfolding eqv_def by (meson cnj_imp prv_imp_cnjL prv_imp_mp)

```

```

lemma prv_imp_eqvER:
 $\varphi 1 \in \text{fmla} \implies \varphi 2 \in \text{fmla} \implies \text{prv} (\text{eqv } \varphi 1 \ \varphi 2) \implies \text{prv} (\text{imp } \varphi 2 \ \varphi 1)$ 
unfolding eqv_def by (meson cnj_imp prv_imp_cnjR prv_imp_mp)

```

```

lemma prv_eqv_imp_trans:
assumes  $\varphi \in \text{fmla}$  and  $\chi \in \text{fmla}$  and  $\psi \in \text{fmla}$ 
shows prv (imp (eqv  $\varphi$   $\chi$ ) (imp (eqv  $\chi$   $\psi$ ) (eqv  $\varphi$   $\psi$ )))
proof-
have prv (imp (eqv  $\varphi$   $\chi$ ) (imp (imp  $\chi$   $\psi$ ) (imp  $\varphi$   $\psi$ )))
using assms prv_imp_cnjL prv_imp_mp prv_imp_trans2 eqv_def
by (metis prv_imp_cnj3L eqv_imp)
hence prv (imp (eqv  $\varphi$   $\chi$ ) (imp (eqv  $\chi$   $\psi$ ) (imp  $\varphi$   $\psi$ )))
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  $\varphi$   $\chi$ ) (eqv  $\chi$   $\psi$ )) (imp  $\varphi$   $\psi$ ))
using prv_cnj_imp_monoR2
by (simp add: assms(1) assms(2) assms(3))
have prv (imp (eqv  $\varphi$   $\chi$ ) (imp (imp  $\psi$   $\chi$ ) (imp  $\psi$   $\varphi$ )))
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  $\varphi$   $\chi$ ) (imp (eqv  $\chi$   $\psi$ ) (imp  $\psi$   $\varphi$ )))
by (metis assms cnj_eqv_def imp prv_imp_cnj3R prv_prv_imp_trans)
hence 2: prv (imp (cnj (eqv  $\varphi$   $\chi$ ) (eqv  $\chi$   $\psi$ )) (imp  $\psi$   $\varphi$ ))
using prv_cnj_imp_monoR2
by (metis (no_types, lifting) assms eqv_imp)
have prv (imp (cnj (eqv  $\varphi$   $\chi$ ) (eqv  $\chi$   $\psi$ )) (eqv  $\varphi$   $\psi$ ))
using 1 2 using assms prv_imp_cnj by (auto simp: eqv_def[of  $\varphi$   $\psi$ ])
thus ?thesis
by (simp add: assms prv_cnj_imp)
qed

```

lemma *prv_eqv_cnj_trans*:
assumes $\varphi \in \text{fmla}$ **and** $\chi \in \text{fmla}$ **and** $\psi \in \text{fmla}$
shows $\text{prv } (\text{imp } (\text{cnj } (\text{eqv } \varphi \chi) (\text{eqv } \chi \psi)) (\text{eqv } \varphi \psi))$
by (*simp add: assms prv_eqv_imp_trans prv_cnj_imp_monoR2*)

lemma *prv_eqvI*:
assumes $\varphi \in \text{fmla}$ **and** $\chi \in \text{fmla}$
assumes $\text{prv } (\text{imp } \varphi \chi)$ **and** $\text{prv } (\text{imp } \chi \varphi)$
shows $\text{prv } (\text{eqv } \varphi \chi)$
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*: $\varphi \in \text{fmla} \implies \text{prv } (\text{eqv } \varphi \varphi)$
by (*simp add: prv_cnjI prv_imp_refl eqv_def*)

lemma *prv_eqv_sym*:
assumes $\varphi \in \text{fmla}$ **and** $\chi \in \text{fmla}$
shows $\text{prv } (\text{eqv } \varphi \chi) \implies \text{prv } (\text{eqv } \chi \varphi)$
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 $\varphi \in \text{fmla}$ **and** $\chi \in \text{fmla}$ **and** $\psi \in \text{fmla}$
shows $\text{prv } (\text{eqv } \varphi \chi) \implies \text{prv } (\text{eqv } \chi \psi) \implies \text{prv } (\text{eqv } \varphi \psi)$
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 $\varphi_1 \in \text{fmla}$ **and** $\varphi_2 \in \text{fmla}$ **and** $\chi \in \text{fmla}$
shows $\text{prv } (\text{imp } (\text{eqv } \varphi_1 \varphi_2) (\text{eqv } (\text{imp } \varphi_1 \chi) (\text{imp } \varphi_2 \chi)))$
proof –
have f : $\text{prv } (\text{imp } (\text{eqv } \varphi_1 \varphi_2) (\text{eqv } (\text{imp } \varphi_1 \chi) (\text{imp } \varphi_2 \chi)))$
if $\text{prv } (\text{imp } (\text{eqv } \varphi_1 \varphi_2) (\text{imp } (\text{imp } \varphi_2 \chi) (\text{imp } \varphi_1 \chi)))$ $\text{prv } (\text{imp } (\text{eqv } \varphi_1 \varphi_2) (\text{imp } (\text{imp } \varphi_1 \chi) (\text{imp } \varphi_2 \chi)))$
using *assms that prv_imp_cnj by (auto simp: eqv_def)*
moreover have $(\text{prv } (\text{imp } (\text{eqv } \varphi_1 \varphi_2) (\text{imp } \varphi_1 \varphi_2)) \wedge \text{prv } (\text{imp } (\text{eqv } \varphi_1 \varphi_2) (\text{imp } \varphi_2 \varphi_1)))$
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 $\varphi \in \text{fmla}$ **and** $\chi_1 \in \text{fmla}$ **and** $\chi_2 \in \text{fmla}$
shows $\text{prv } (\text{imp } (\text{eqv } \chi_1 \chi_2) (\text{eqv } (\text{imp } \varphi \chi_1) (\text{imp } \varphi \chi_2)))$
by (*simp add: assms prv_cnj_mono prv_imp_trans1 eqv_def*)

lemma *imp_imp_compat_eqv*:
assumes $\varphi_1 \in \text{fmla}$ **and** $\varphi_2 \in \text{fmla}$ **and** $\chi_1 \in \text{fmla}$ **and** $\chi_2 \in \text{fmla}$
shows $\text{prv } (\text{imp } (\text{eqv } \varphi_1 \varphi_2) (\text{imp } (\text{eqv } \chi_1 \chi_2) (\text{eqv } (\text{imp } \varphi_1 \chi_1) (\text{imp } \varphi_2 \chi_2))))$
proof–
have $\text{prv } (\text{imp } (\text{eqv } \varphi_1 \varphi_2) (\text{imp } (\text{eqv } \chi_1 \chi_2) (\text{cnj } (\text{eqv } (\text{imp } \varphi_1 \chi_1) (\text{imp } \varphi_2 \chi_1)) (\text{eqv } (\text{imp } \varphi_2 \chi_1) (\text{imp } \varphi_2 \chi_2))))$
using *prv_imp_cnj5*
 $[\text{OF } ______ \text{imp_imp_compat_eqvL}[\text{of } \varphi_1 \varphi_2 \chi_1] \text{imp_imp_compat_eqvR}[\text{of } \varphi_2 \chi_1 \chi_2]]$ **assms** **by** *auto*
hence $\text{prv } (\text{imp } (\text{cnj } (\text{eqv } \varphi_1 \varphi_2) (\text{eqv } \chi_1 \chi_2)) (\text{cnj } (\text{eqv } (\text{imp } \varphi_1 \chi_1) (\text{imp } \varphi_2 \chi_1)) (\text{eqv } (\text{imp } \varphi_2 \chi_1) (\text{imp } \varphi_2 \chi_2))))$

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 $\varphi 1 \chi$) (imp $\varphi 2 \chi$))*
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 $\varphi \chi 1$) (imp $\varphi \chi 2$))*
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 $\varphi 1 \chi$) (cnj $\varphi 2 \chi$)))*
proof –
have *prv (imp (imp (imp $\varphi 2 \varphi 1$) (imp (cnj $\varphi 2 \chi$) (cnj $\varphi 1 \chi$)))*
(imp (cnj (imp $\varphi 1 \varphi 2$) (imp $\varphi 2 \varphi 1$)) (cnj (imp (cnj $\varphi 1 \chi$) (cnj $\varphi 2 \chi$)))
(imp (cnj $\varphi 2 \chi$) (cnj $\varphi 1 \chi$))))
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 $\varphi \chi 1$) (cnj $\varphi \chi 2$)))*
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 (equiv φ_1 φ_2) (equiv χ_1 χ_2)) (equiv (cnj φ_1 χ_1) (cnj φ_2 χ_2)))*
using *assms prv_equiv_cnj_trans[of cnj φ_1 χ_1 cnj φ_2 χ_1 cnj φ_2 χ_2]*
using *prv_imp_mp prv_imp_trans2*
by (*metis (no_types) cnj_prv_prv_imp_trans equiv*)
thus *?thesis using assms prv_cnj_imp by auto*
qed

lemma *cnj_compat_equivL:*
assumes $\varphi_1 \in \text{fmla}$ **and** $\varphi_2 \in \text{fmla}$ **and** $\chi \in \text{fmla}$
assumes *prv (equiv φ_1 φ_2)*
shows *prv (equiv (cnj φ_1 χ) (cnj φ_2 χ))*
using *assms prv_imp_mp imp_cnj_compat_equivL by (meson equiv cnj)*

lemma *cnj_compat_equivR:*
assumes $\varphi \in \text{fmla}$ **and** $\chi_1 \in \text{fmla}$ **and** $\chi_2 \in \text{fmla}$
assumes *prv (equiv χ_1 χ_2)*
shows *prv (equiv (cnj φ χ_1) (cnj φ χ_2))*
using *assms prv_imp_mp imp_cnj_compat_equivR by (meson equiv cnj)*

lemma *cnj_compat_equiv:*
assumes $\varphi_1 \in \text{fmla}$ **and** $\varphi_2 \in \text{fmla}$ **and** $\chi_1 \in \text{fmla}$ **and** $\chi_2 \in \text{fmla}$
assumes *prv (equiv φ_1 φ_2) and prv (equiv χ_1 χ_2)*
shows *prv (equiv (cnj φ_1 χ_1) (cnj φ_2 χ_2))*
using *assms prv_imp_mp imp_cnj_compat_equiv by (metis equiv imp cnj)*

lemma *prv_equiv_prv:*
assumes $\varphi \in \text{fmla}$ **and** $\chi \in \text{fmla}$
assumes *prv φ and prv (equiv φ χ)*
shows *prv χ*
by (*metis assms prv_imp_cnjL prv_imp_mp equiv_equiv_def imp*)

lemma *prv_equiv_prv_rev:*
assumes $\varphi \in \text{fmla}$ **and** $\chi \in \text{fmla}$
assumes *prv φ and prv (equiv χ φ)*
shows *prv χ*
using *assms prv_equiv_prv prv_equiv_sym by blast*

lemma *prv_imp_equiv_transi:*
assumes $\varphi \in \text{fmla}$ **and** $\chi_1 \in \text{fmla}$ **and** $\chi_2 \in \text{fmla}$
assumes *prv (imp φ χ_1) and prv (equiv χ_1 χ_2)*
shows *prv (imp φ χ_2)*
by (*meson assms imp_imp_compat_equivR prv_equiv_prv*)

lemma *prv_imp_equiv_transi_rev:*
assumes $\varphi \in \text{fmla}$ **and** $\chi_1 \in \text{fmla}$ **and** $\chi_2 \in \text{fmla}$
assumes *prv (imp φ χ_2) and prv (equiv χ_1 χ_2)*
shows *prv (imp φ χ_1)*
by (*meson assms prv_equiv_sym prv_imp_equiv_transi*)

lemma *prv_equiv_imp_transi:*
assumes $\varphi_1 \in \text{fmla}$ **and** $\varphi_2 \in \text{fmla}$ **and** $\chi \in \text{fmla}$
assumes *prv (equiv φ_1 φ_2) and prv (imp φ_2 χ)*
shows *prv (imp φ_1 χ)*
by (*meson assms prv_imp_equiv_transi prv_imp_refl prv_prv_imp_trans*)

lemma *prv_equiv_imp_transi_rev:*
assumes $\varphi_1 \in \text{fmla}$ **and** $\varphi_2 \in \text{fmla}$ **and** $\chi \in \text{fmla}$

assumes $prv (eqv \varphi 1 \varphi 2)$ **and** $prv (imp \varphi 1 \chi)$
shows $prv (imp \varphi 2 \chi)$
by (*meson* *assms* *prv_eqv_imp_transi* *prv_eqv_sym*)

lemma *prv_imp_monoL*: $\varphi \in fmla \implies \chi \in fmla \implies \psi \in fmla \implies$
 $prv (imp \chi \psi) \implies prv (imp (imp \varphi \chi) (imp \varphi \psi))$
by (*meson* *imp* *prv_imp_mp* *prv_imp_trans1*)

lemma *prv_imp_monoR*: $\varphi \in fmla \implies \chi \in fmla \implies \psi \in fmla \implies$
 $prv (imp \psi \chi) \implies prv (imp (imp \chi \varphi) (imp \psi \varphi))$
by (*meson* *imp* *prv_imp_mp* *prv_imp_trans2*)

More properties involving conjunction:

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

lemma *prv_cnj_com*:
assumes $\varphi[simp]: \varphi \in fmla$ **and** $\chi[simp]: \chi \in fmla$
shows $prv (eqv (cnj \varphi \chi) (cnj \chi \varphi))$
by (*simp* *add*: *prv_cnj_com_imp* *prv_eqvI*)

lemma *prv_cnj_assoc_imp1*:
assumes $\varphi[simp]: \varphi \in fmla$ **and** $\chi[simp]: \chi \in fmla$ **and** $\psi[simp]: \psi \in fmla$
shows $prv (imp (cnj \varphi (cnj \chi \psi)) (cnj (cnj \varphi \chi) \psi))$
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 $\varphi[simp]: \varphi \in fmla$ **and** $\chi[simp]: \chi \in fmla$ **and** $\psi[simp]: \psi \in fmla$
shows $prv (imp (cnj (cnj \varphi \chi) \psi) (cnj \varphi (cnj \chi \psi)))$
proof –
have $prv (imp (cnj \varphi \chi) (imp \psi \varphi)) \wedge cnj \chi \psi \in fmla \wedge cnj \varphi \chi \in fmla$
by (*meson* $\chi \varphi \psi$ *cnj_imp* *prv_cnj_imp_monoR2* *prv_imp_imp_triv* *prv_prv_imp_trans*)
then show *?thesis*
using $\chi \varphi \psi$ *cnj_imp* *prv_cnj_imp_monoR2* *prv_imp_cnj4* *prv_imp_cnjI* *prv_imp_triv* **by** *presburger*
qed

lemma *prv_cnj_assoc*:
assumes $\varphi[simp]: \varphi \in fmla$ **and** $\chi[simp]: \chi \in fmla$ **and** $\psi[simp]: \psi \in fmla$
shows $prv (eqv (cnj \varphi (cnj \chi \psi)) (cnj (cnj \varphi \chi) \psi))$
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 $prv (all x \varphi) \implies prv (subst \varphi t x)$
using *assms* *prv_all_inst* *prv_imp_mp* **by** (*meson* *subst* *all*)

lemma *prv_exI*:

assumes $x \in \text{var}$ **and** $\varphi \in \text{fmla}$ **and** $t \in \text{trm}$
shows $\text{prv} (\text{subst } \varphi \ t \ x) \implies \text{prv} (\text{exi } x \ \varphi)$
using $\text{assms } \text{prv_exi_inst } \text{prv_imp_mp}$ **by** ($\text{meson } \text{subst } \text{exi}$)

lemma prv_imp_imp_exi :

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

lemma prv_imp_exi :

assumes $x \in \text{var}$ **and** $\varphi \in \text{fmla}$ **and** $\chi \in \text{fmla}$
shows $x \notin \text{Fvars } \varphi \implies \text{prv} (\text{exi } x \ (\text{imp } \varphi \ \chi)) \implies \text{prv} (\text{imp } \varphi (\text{exi } x \ \chi))$
using $\text{assms } \text{prv_imp_imp_exi } \text{prv_imp_mp}$ **by** ($\text{meson } \text{exi } \text{imp}$)

lemma prv_exi_imp :

assumes $x: x \in \text{var}$ **and** $\varphi \in \text{fmla}$ **and** $\chi \in \text{fmla}$
assumes $x \notin \text{Fvars } \chi$ **and** $d: \text{prv} (\text{all } x \ (\text{imp } \varphi \ \chi))$
shows $\text{prv} (\text{imp} (\text{exi } x \ \varphi) \ \chi)$

proof–

have $\text{prv} (\text{imp } \varphi \ \chi)$ **using** $\text{prv_allE}[\text{of } x \ _ \ \text{Var } x, \ \text{of } \text{imp } \varphi \ \chi]$ assms **by** simp
thus ?thesis **using** $\text{assms } \text{prv_exi_imp_gen}$ **by** blast

qed

lemma prv_all_imp :

assumes $x: x \in \text{var}$ **and** $\varphi \in \text{fmla}$ **and** $\chi \in \text{fmla}$
assumes $x \notin \text{Fvars } \varphi$ **and** $\text{prv} (\text{all } x \ (\text{imp } \varphi \ \chi))$
shows $\text{prv} (\text{imp } \varphi (\text{all } x \ \chi))$

proof–

have $\text{prv} (\text{imp } \varphi \ \chi)$ **using** $\text{prv_allE}[\text{of } x \ _ \ \text{Var } x, \ \text{of } \text{imp } \varphi \ \chi]$ assms **by** simp
thus ?thesis **using** $\text{assms } \text{prv_all_imp_gen}$ **by** blast

qed

lemma prv_exi_inst_same :

assumes $\varphi \in \text{fmla}$ $\chi \in \text{fmla}$ $x \in \text{var}$
shows $\text{prv} (\text{imp } \varphi (\text{exi } x \ \varphi))$

proof–

have $0: \varphi = \text{subst } \varphi \ (\text{Var } x) \ x$ **using** assms **by** simp
show ?thesis
apply($\text{subst } 0$)
using assms **by** ($\text{intro } \text{prv_exi_inst}$) auto

qed

lemma prv_exi_cong :

assumes $\varphi \in \text{fmla}$ $\chi \in \text{fmla}$ $x \in \text{var}$
and $\text{prv} (\text{imp } \varphi \ \chi)$
shows $\text{prv} (\text{imp} (\text{exi } x \ \varphi) (\text{exi } x \ \chi))$

proof–

have $0: \text{prv} (\text{imp } \chi (\text{exi } x \ \chi))$ **using** $\text{assms } \text{prv_exi_inst_same}$ **by** auto
show ?thesis
using assms **apply**($\text{intro } \text{prv_exi_imp_gen}$)
subgoal **by** auto
subgoal **by** auto
subgoal **by** auto
subgoal **by** auto
subgoal **using** $0 \ \text{exi } \text{prv_prv_imp_trans}$ **by** blast .

qed

lemma *prv_exi_congW*:
assumes $\varphi \in \text{fmla}$ $\chi \in \text{fmla}$ $x \in \text{var}$
and *prv* (*imp* φ χ) *prv* (*exi* x φ)
shows *prv* (*exi* x χ)
by (*meson exi assms prv_exi_cong prv_imp_mp*)

lemma *prv_all_cong*:
assumes $\varphi \in \text{fmla}$ $\chi \in \text{fmla}$ $x \in \text{var}$
and *prv* (*imp* φ χ)
shows *prv* (*imp* (*all* x φ) (*all* x χ))
proof–
have *0*: *prv* (*imp* (*all* x φ) 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 $\varphi \in \text{fmla}$ $\chi \in \text{fmla}$ $x \in \text{var}$
and *prv* (*imp* φ χ) *prv* (*all* x φ)
shows *prv* (*all* x χ)
by (*meson all assms prv_all_cong prv_imp_mp*)

Quantifiers versus free variables and substitution:

lemma *exists_no_Fvars*: $\exists \varphi. \varphi \in \text{fmla} \wedge \text{prv } \varphi \wedge \text{Fvars } \varphi = \{\}$
proof–
obtain *n* **where** [*simp*]: $n \in \text{num}$ **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 $x \in \text{var}$ **and** $\varphi \in \text{fmla}$
assumes *prv* φ **shows** *prv* (*all* x φ)
using *assms prv_all_imp_gen prv_imp_mp prv_imp_triv exists_no_Fvars* **by** *blast*

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

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

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

lemma *prv_rawsubst*:
assumes $\varphi \in \text{fmla}$ **and** $\text{snd } \langle \text{set } \text{txs} \rangle \subseteq \text{var}$ **and** $\text{fst } \langle \text{set } \text{txs} \rangle \subseteq \text{trm}$
and *prv* φ
shows *prv* (*rawsubst* φ txs)
using *assms apply (induct txs arbitrary: φ)*
subgoal **by** *auto*

subgoal for $tx\ txs\ \varphi$ **by** $(cases\ tx)\ (auto\ intro:\ prv_subst)$.

lemma prv_psubst :

assumes $\varphi \in fmla$ **and** $snd\ ' (set\ txs) \subseteq var$ **and** $fst\ ' (set\ txs) \subseteq trm$
and $prv\ \varphi$
shows $prv\ (psubst\ \varphi\ txs)$

proof–

define us **where** $us: us \equiv getFrN\ (map\ snd\ txs)\ (map\ fst\ txs)\ [\varphi]\ (length\ txs)$

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 $getFrN_FvarsT[of\ map\ snd\ txs\ map\ fst\ txs\ [\varphi]\ _]\ length\ txs]$

$getFrN_Fvars[of\ map\ snd\ txs\ map\ fst\ txs\ [\varphi]\ _]\ length\ txs]$

$getFrN_var[of\ map\ snd\ txs\ map\ fst\ txs\ [\varphi]\ _]\ length\ txs]$

$getFrN_length[of\ map\ snd\ txs\ map\ fst\ txs\ [\varphi]\ length\ txs]$

$getFrN_length[of\ map\ snd\ txs\ map\ fst\ txs\ [\varphi]\ length\ txs]$

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 $prv_eqv_rawpsubst$:

$\varphi \in fmla \implies \psi \in fmla \implies snd\ ' set\ txs \subseteq var \implies fst\ ' set\ txs \subseteq trm \implies prv\ (eqv\ \varphi\ \psi) \implies$

$prv\ (eqv\ (rawpsubst\ \varphi\ txs)\ (rawpsubst\ \psi\ txs))$

by $(metis\ eqv\ prv_rawpsubst\ rawpsubst_eqv)$

lemma prv_eqv_psubst :

$\varphi \in fmla \implies \psi \in fmla \implies snd\ ' set\ txs \subseteq var \implies fst\ ' set\ txs \subseteq trm \implies prv\ (eqv\ \varphi\ \psi) \implies$

$distinct\ (map\ snd\ txs) \implies$

$prv\ (eqv\ (psubst\ \varphi\ txs)\ (psubst\ \psi\ txs))$

by $(metis\ eqv\ prv_psubst\ psubst_eqv)$

lemma $prv_all_imp_trans$:

assumes $x \in var$ **and** $\varphi \in fmla$ **and** $\chi \in fmla$ **and** $\psi \in fmla$

shows $prv\ (all\ x\ (imp\ \varphi\ \chi)) \implies prv\ (all\ x\ (imp\ \chi\ \psi)) \implies prv\ (all\ x\ (imp\ \varphi\ \psi))$

by $(metis\ Var\ assms\ prv_allE\ prv_all_gen\ prv_prv_imp_trans\ imp\ subst_same_Var)$

lemma $prv_all_imp_cnj$:

assumes $x \in var$ **and** $\varphi \in fmla$ **and** $\chi \in fmla$ **and** $\psi \in fmla$

shows $prv\ (all\ x\ (imp\ \varphi\ (imp\ \psi\ \chi))) \implies prv\ (all\ x\ (imp\ (cnj\ \psi\ \varphi)\ \chi))$

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 $x \in var$ **and** $\varphi \in fmla$ **and** $\chi \in fmla$ **and** $\psi \in fmla$

shows $prv\ (all\ x\ (imp\ (cnj\ \varphi\ \psi)\ \chi)) \implies prv\ (all\ x\ (imp\ \varphi\ (imp\ \psi\ \chi)))$

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 $t: t \in \text{trm}$
shows $\text{prv} (\text{eql } t \ t)$

proof–

obtain x **where** $x: x \in \text{var}$ **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 $xx: x \in \text{var}$ **and** $\varphi: \varphi \in \text{fmla}$ **and** $t1 \in \text{trm}$ **and** $t2 \in \text{trm}$
shows $\text{prv} ((\text{imp} (\text{eql } t1 \ t2)$
 $(\text{imp} (\text{subst } \varphi \ t1 \ x) (\text{subst } \varphi \ t2 \ x))))$

proof–

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

lemma *prv_eql_subst_trm2*:

assumes $x \in \text{var}$ **and** $\varphi \in \text{fmla}$ **and** $t1 \in \text{trm}$ **and** $t2 \in \text{trm}$
assumes $\text{prv} (\text{eql } t1 \ t2)$
shows $\text{prv} (\text{imp} (\text{subst } \varphi \ t1 \ x) (\text{subst } \varphi \ t2 \ x))$
by (*meson assms eql_imp local.subst prv_eql_subst_trm prv_imp_mp*)

lemma *prv_eql_sym*:

assumes [*simp*]: $t1 \in \text{trm}$ $t2 \in \text{trm}$
shows $\text{prv} (\text{imp} (\text{eql } t1 \ t2) (\text{eql } t2 \ t1))$

proof–

obtain x **where** $x[\text{simp}]: x \in \text{var}$ $x \notin \text{FvarsT } t1$
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*: $t1 \in \text{trm} \implies t2 \in \text{trm} \implies \text{prv} (\text{eql } t1 \ t2) \implies \text{prv} (\text{eql } t2 \ t1)$

by (*meson eql prv_eql_sym prv_imp_mp*)

lemma *prv_all_eql*:

assumes $x \in \text{var}$ **and** $y \in \text{var}$ **and** $\varphi \in \text{fmla}$ **and** $t1 \in \text{trm}$ **and** $t2 \in \text{trm}$
shows $\text{prv} (\text{all } x ((\text{imp} (\text{eql } t1 \ t2)$
 $(\text{imp} (\text{subst } \varphi \ t2 \ y) (\text{subst } \varphi \ t1 \ y))))$
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 $t1 \in \text{trm}$ **and** $t2 \in \text{trm}$ **and** $\varphi \in \text{fmla}$ **and** $y \in \text{var}$
shows
 $\text{prv} ((\text{imp} (\text{eql } t1 \ t2)$
 $(\text{imp} (\text{subst } \varphi \ t2 \ y) (\text{subst } \varphi \ t1 \ y))))$
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 $x \in \text{var}$ **and** $\varphi \in \text{fmla}$ **and** $t1 \in \text{trm}$ **and** $t2 \in \text{trm}$
assumes $\text{prv} (\text{eql } t1 \ t2)$
shows $\text{prv} (\text{imp} (\text{subst } \varphi \ t2 \ x) (\text{subst } \varphi \ t1 \ x))$
by (meson assms eql imp local.subst prv_eql_subst_trm_rev prv_imp_mp)

lemma *prv_eql_subst_trm_eqv*:

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

lemma *prv_eql_subst_trm_id*:

assumes $y \in \text{var}$ $\varphi \in \text{fmla}$ **and** $n \in \text{num}$
shows $\text{prv} (\text{imp} (\text{eql} (\text{Var } y) \ n) (\text{imp } \varphi (\text{subst } \varphi \ n \ y)))$
using *assms* prv_eql_subst_trm
by (metis *Var in_num subst_same_Var*)

lemma *prv_eql_subst_trm_id_back*:

assumes $y \in \text{var}$ $\varphi \in \text{fmla}$ **and** $n \in \text{num}$
shows $\text{prv} (\text{imp} (\text{eql} (\text{Var } y) \ n) (\text{imp} (\text{subst } \varphi \ n \ y) \ \varphi))$
by (metis *Var assms in_num prv_eql_subst_trm_rev subst_same_Var*)

lemma *prv_eql_subst_trm_id_rev*:

assumes $y \in \text{var}$ $\varphi \in \text{fmla}$ **and** $n \in \text{num}$
shows $\text{prv} (\text{imp} (\text{eql } n \ (\text{Var } y)) (\text{imp } \varphi (\text{subst } \varphi \ n \ y)))$
using *assms* $\text{prv_eql_subst_trm_rev}$ **by** (metis *Var in_num subst_same_Var*)

lemma *prv_eql_subst_trm_id_back_rev*:

assumes $y \in \text{var}$ $\varphi \in \text{fmla}$ **and** $n \in \text{num}$
shows $\text{prv} (\text{imp} (\text{eql } n \ (\text{Var } y)) (\text{imp} (\text{subst } \varphi \ n \ y) \ \varphi))$
by (metis (full_types) *Var assms in_num prv_eql_subst_trm subst_same_Var*)

lemma *prv_eql_imp_trans_rev*:

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

proof–

obtain $y1$ **where** $y1$ [simp]: $y1 \in \text{var}$ **and** $y1 \notin \text{FvarsT } t1 \cup \text{FvarsT } t2 \cup \text{FvarsT } t3$
using *finite_FvarsT finite_subset infinite_var subsetI t1 t2 t3*
by (metis (full_types) *infinite_Un*)

hence [simp]: $y1 \notin \text{FvarsT } t1$ $y1 \notin \text{FvarsT } t2$ $y1 \notin \text{FvarsT } t3$ **by** auto

obtain $y2$ **where** $y2$ [simp]: $y2 \in \text{var}$ **and** $y2 \notin \text{FvarsT } t1 \cup \text{FvarsT } t2 \cup \text{FvarsT } t3 \cup \{y1\}$

using *finite_FvarsT finite_subset infinite_var subsetI t1 t2 t3*

by (metis (full_types) *finite_insert infinite_Un insert_is_Un*)

hence [simp]: $y2 \notin \text{FvarsT } t1$ $y2 \notin \text{FvarsT } t2$ $y2 \notin \text{FvarsT } t3$ $y1 \neq y2$ **by** auto

have 0: $\text{prv} (\text{imp} (\text{eql} (\text{Var } y1) \ (\text{Var } y2))$
 $(\text{imp} (\text{eql} (\text{Var } y1) \ t3) (\text{eql} (\text{Var } y2) \ t3)))$

using prv_eql_subst [OF $y1 \ y2$, of $\text{eql} (\text{Var } y1) \ t3$] **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]: $t1 \in \text{trm}$ **and** $t2$ [simp]: $t2 \in \text{trm}$ **and** $t3$ [simp]: $t3 \in \text{trm}$
shows $\text{prv} (\text{imp} (\text{eql } t1 \ t2) (\text{imp} (\text{eql } t2 \ t3) (\text{eql } t1 \ t3)))$

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

lemma *prv_eql_trans*:

assumes $t1[simp]: t1 \in trm$ **and** $t2[simp]: t2 \in trm$ **and** $t3[simp]: t3 \in trm$

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 $[simp,intro!]: x \in var t \in trm \varphi \in fmla x \notin FvarsT t$

shows $prv (imp (subst \varphi t x) (softSubst \varphi t x))$

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 $x \in var t \in trm \varphi \in fmla$

and $x \notin FvarsT t$

and $prv (subst \varphi t x)$

shows $prv (softSubst \varphi t x)$

using *assms prv_subst_imp_softSubst*

by (*metis Var cnj eql_exi subst prv_imp_mp softSubst_def*)

lemma *prv_softSubst_imp_subst*:

assumes $[simp,intro!]: x \in var t \in trm \varphi \in fmla x \notin FvarsT t$

shows $prv (imp (softSubst \varphi t x) (subst \varphi t x))$

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 $x \in var t \in trm \varphi \in fmla$

and $x \notin FvarsT t$

and $prv (softSubst \varphi t x)$

shows $prv (subst \varphi t x)$

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

lemma *prv_softSubst_eqv_subst*:

assumes $[simp,intro!]: x \in var t \in trm \varphi \in fmla x \notin FvarsT t$

shows $prv (eqv (softSubst \varphi t x) (subst \varphi t x))$

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_fl[simp,intro]:  $\bigwedge \varphi. prv (imp fls \varphi)$ 
begin

```

3.2.1 Basic properties of False (fls)

```

lemma prv_expl:
  assumes  $\varphi \in fmla$ 
  assumes prv fls
  shows prv  $\varphi$ 
  using assms prv_imp_mp by blast

lemma prv_cnjR_fl:  $\varphi \in fmla \implies prv (eqv (cnj fls \varphi) fls)$ 
  by (simp add: prv_eqvI prv_imp_cnjL)

lemma prv_cnjL_fl:  $\varphi \in fmla \implies prv (eqv (cnj \varphi fls) fls)$ 
  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_fl:
  assumes  $\varphi \in fmla$ 
  shows prv (imp  $\varphi (imp (neg \varphi) fls)$ )
  using assms prv_imp_com prv_imp_refl neg_def by auto

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

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

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

lemma prv_imp_imp_neg_rev:

```

assumes $\varphi \in \text{fmla}$ **and** $\chi \in \text{fmla}$
shows $\text{prv } (\text{imp } (\text{imp } \varphi \chi)$
 $\quad (\text{imp } (\text{neg } \chi) (\text{neg } \varphi)))$
unfolding neg_def **using** $\text{prv_imp_trans2}[OF \text{ assms } \text{fls}]$.

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

lemma $\text{prv_eqv_neg_prv_fls}$:
 $\varphi \in \text{fmla} \implies$
 $\text{prv } (\text{eqv } \varphi (\text{neg } \varphi)) \implies \text{prv } \text{fls}$
by ($\text{metis cnj fls neg neg_def prv_cnj_imp_monoR2 prv_eqv_imp_transi prv_imp_cnj prv_imp_eqvER}$
 $\text{prv_imp_mp prv_imp_neg_rev}$)

lemma $\text{prv_eqv_eqv_neg_prv_fls}$:
 $\varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies$
 $\text{prv } (\text{eqv } \varphi \chi) \implies \text{prv } (\text{eqv } \varphi (\text{neg } \chi)) \implies \text{prv } \text{fls}$
by ($\text{meson neg prv_eqv_neg_prv_fls prv_eqv_sym prv_eqv_trans}$)

lemma $\text{prv_eqv_eqv_neg_prv_fls2}$:
 $\varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies$
 $\text{prv } (\text{eqv } \varphi \chi) \implies \text{prv } (\text{eqv } \chi (\text{neg } \varphi)) \implies \text{prv } \text{fls}$
by ($\text{simp add: prv_eqv_eqv_neg_prv_fls prv_eqv_sym}$)

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

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

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

lemma $\text{prv_imp_neg_imp_cnjL}$:
assumes $\varphi \in \text{fmla}$ $\varphi1 \in \text{fmla}$ $\varphi2 \in \text{fmla}$
and $\text{prv } (\text{imp } \varphi (\text{neg } \varphi1))$
shows $\text{prv } (\text{imp } \varphi (\text{neg } (\text{cnj } \varphi1 \varphi2)))$
unfolding neg_def **by** ($\text{metis assms cnj fls neg neg_def prv_imp_cnj3L prv_prv_imp_trans}$)

lemma $\text{prv_imp_neg_imp_cnjR}$:
assumes $\varphi \in \text{fmla}$ $\varphi1 \in \text{fmla}$ $\varphi2 \in \text{fmla}$
and $\text{prv } (\text{imp } \varphi (\text{neg } \varphi2))$
shows $\text{prv } (\text{imp } \varphi (\text{neg } (\text{cnj } \varphi1 \varphi2)))$
unfolding neg_def **by** ($\text{metis assms cnj fls neg neg_def prv_imp_cnj3R prv_prv_imp_trans}$)

Negation versus quantifiers:

```

lemma prv_all_neg_imp_neg_exi:
  assumes  $x: x \in \text{var}$  and  $\varphi: \varphi \in \text{fmla}$ 
  shows  $\text{prv} (\text{imp} (\text{all } x (\text{neg } \varphi)) (\text{neg} (\text{exi } x \varphi)))$ 
proof-
  have 0:  $\text{prv} (\text{imp} (\text{all } x (\text{neg } \varphi)) (\text{imp } \varphi \text{ fls}))$ 
    using prv_all_inst[OF x, of neg  $\varphi$  Var x, simplified] assms by (auto simp: neg_def)
  have 1:  $\text{prv} (\text{imp } \varphi (\text{imp} (\text{all } x (\text{neg } \varphi)) \text{ fls}))$ 
    using 0 by (simp add: assms prv_imp_com)
  have 2:  $\text{prv} (\text{imp} (\text{exi } x \varphi) (\text{imp} (\text{all } x (\text{neg } \varphi)) \text{ fls}))$ 
    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  $x: x \in \text{var}$  and  $\varphi: \varphi \in \text{fmla}$ 
  shows  $\text{prv} (\text{imp} (\text{neg} (\text{exi } x \varphi)) (\text{all } x (\text{neg } \varphi)))$ 
  using assms
  by (intro prv_all_imp_gen[of x neg (exi x  $\varphi$ )])
  (auto simp: prv_exi_inst_same prv_imp_neg_rev)

```

```

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

```

```

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

```

```

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

```

```

lemma prv_exi_imp_exiI:
  assumes [simp]:  $x \in \text{var}$   $y \in \text{var}$   $\varphi \in \text{fmla}$  and  $yx: y = x \vee y \notin \text{Fvars } \varphi$ 
  shows  $\text{prv} (\text{imp} (\text{exi } x \varphi) (\text{exi } y (\text{subst } \varphi (\text{Var } y) x)))$ 
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  $\varphi \in \text{fmla}$   $\chi \in \text{fmla}$   $t \in \text{trm}$   $x \in \text{var}$ 
  and  $\text{prv} (\text{imp } \varphi (\text{neg} (\text{subst } \chi t x)))$ 
  shows  $\text{prv} (\text{imp } \varphi (\text{neg} (\text{all } x \chi)))$ 
  by (meson all assms subst neg prv_all_inst prv_imp_neg_rev prv_prv_imp_trans)

```

lemma *prv_imp_neg_allWI*:
assumes $\chi \in \text{fm}la$ $t \in \text{trm}$ $x \in \text{var}$
and $\text{prv } (\text{neg } (\text{subst } \chi \ t \ x))$
shows $\text{prv } (\text{neg } (\text{all } x \ \chi))$
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*: $\varphi \in \text{fm}la \implies \text{prv } (\text{imp } \varphi \ \text{tru})$
by (*simp add: neg_def prv_imp_triv tru_def*)

lemma *prv_tru_imp*: $\varphi \in \text{fm}la \implies \text{prv } (\text{equiv } (\text{imp } \text{tru } \varphi) \ \varphi)$
by (*metis imp neg_def prv_equivI prv_fls prv_imp_com prv_imp_imp_triv prv_imp_imp prv_imp_refl tru tru_def*)

lemma *prv_fls_neg_tru*: $\varphi \in \text{fm}la \implies \text{prv } (\text{equiv } \text{fls } (\text{neg } \text{tru}))$
using *neg_def prv_equivI prv_neg_neg tru_def* **by** *auto*

lemma *prv_tru_neg_fls*: $\varphi \in \text{fm}la \implies \text{prv } (\text{equiv } \text{tru } (\text{neg } \text{fls}))$
by (*simp add: prv_equiv_refl tru_def*)

lemma *prv_cnjR_tru*: $\varphi \in \text{fm}la \implies \text{prv } (\text{equiv } (\text{cnj } \text{tru } \varphi) \ \varphi)$
by (*simp add: prv_equivI prv_imp_cnj prv_imp_cnjR prv_imp_refl prv_imp_tru*)

lemma *prv_cnjL_tru*: $\varphi \in \text{fm}la \implies \text{prv } (\text{equiv } (\text{cnj } \varphi \ \text{tru}) \ \varphi)$
by (*simp add: prv_equivI 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 $\text{set } \varphi s \subseteq \text{fm}la$
and $\varphi \in \text{set } \varphi s$
shows $\text{prv } (\text{imp } (\text{lcnj } \varphi s) \ \varphi)$
proof–
have $\varphi \in \text{fm}la$ **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 $\chi \in \text{fm}la$ **and** $\text{set } \varphi s \subseteq \text{fm}la$
and $\varphi \in \text{set } \varphi s$ **and** $\text{prv } (\text{imp } \varphi \ \chi)$
shows $\text{prv } (\text{imp } (\text{lcnj } \varphi s) \ \chi)$
using *assms lcnj prv_lcnj_imp_in prv_prv_imp_trans* **by** *blast*

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

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

```

case (Cons  $\varphi_1 \varphi_{1s}$ )
have [intro!]: set ( $\varphi_{1s} @ \varphi_{2s}$ )  $\subseteq$  fmla set ( $\varphi_{1s} @ \varphi \# \varphi_{2s}$ )  $\subseteq$  fmla using Cons by auto
have 0: prv (imp (cnj  $\varphi$  (cnj  $\varphi_1$  (lcnj ( $\varphi_{1s} @ \varphi_{2s}$ ))))
      (cnj  $\varphi_1$  (cnj  $\varphi$  (lcnj ( $\varphi_{1s} @ \varphi_{2s}$ ))))
      using Cons by (intro prv_cnj_com_imp3) fastforce+
have 1: prv (imp (cnj  $\varphi_1$  (cnj  $\varphi$  (lcnj ( $\varphi_{1s} @ \varphi_{2s}$ ))))
      (cnj  $\varphi_1$  (lcnj ( $\varphi_{1s} @ \varphi \# \varphi_{2s}$ ))))
      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  $\varphi_s \subseteq$  fmla
shows prv (imp (lcnj (remdups  $\varphi_s$ )) (lcnj  $\varphi_s$ ))
using assms apply(induct  $\varphi_s$ )
by (auto simp: prv_imp_cnj prv_lcnj_imp_in prv_cnj_mono prv_imp_refl)

lemma prv_lcnj_mono:
assumes  $\varphi_{1s}$ : set  $\varphi_{1s} \subseteq$  fmla and set  $\varphi_{2s} \subseteq$  set  $\varphi_{1s}$ 
shows prv (imp (lcnj  $\varphi_{1s}$ ) (lcnj  $\varphi_{2s}$ ))
proof-
define  $\varphi_{2s}'$  where  $\varphi_{2s}'$ :  $\varphi_{2s}' =$  remdups  $\varphi_{2s}$ 
have A: set  $\varphi_{2s}' \subseteq$  set  $\varphi_{1s}$  distinct  $\varphi_{2s}'$  unfolding  $\varphi_{2s}'$  using assms by auto
have B: prv (imp (lcnj  $\varphi_{1s}$ ) (lcnj  $\varphi_{2s}'$ ))
      using  $\varphi_{1s}$  A proof(induction  $\varphi_{1s}$  arbitrary:  $\varphi_{2s}'$ )
      case (Cons  $\varphi_1 \varphi_{1s} \varphi_{2ss}$ )
      show ?case proof(cases  $\varphi_1 \in$  set  $\varphi_{2ss}$ )
        case True
          then obtain  $\varphi_{2ss1} \varphi_{2ss2}$  where  $\varphi_{2ss}$ :  $\varphi_{2ss} = \varphi_{2ss1} @ \varphi_1 \# \varphi_{2ss2}$ 
          by (meson split_list)
          define  $\varphi_{2s}$  where  $\varphi_{2s}$ :  $\varphi_{2s} \equiv \varphi_{2ss1} @ \varphi_{2ss2}$ 
          have nin:  $\varphi_1 \notin$  set  $\varphi_{2s}$  using  $\varphi_{2ss}$  (distinct  $\varphi_{2ss}$ ) unfolding  $\varphi_{2s}$  by auto
          have [intro!]: set  $\varphi_{2s} \subseteq$  fmla unfolding  $\varphi_{2s}$  using  $\varphi_{2ss}$  Cons by auto
          have 0: prv (imp (cnj  $\varphi_1$  (lcnj  $\varphi_{2s}$ )) (lcnj  $\varphi_{2ss}$ ))
            unfolding  $\varphi_{2s} \varphi_{2ss}$  using Cons  $\varphi_{2ss}$ 
            by (intro prv_lcnj_imp_inner) auto
          have 1: prv (imp (lcnj  $\varphi_{1s}$ ) (lcnj  $\varphi_{2s}$ ))
            using nin Cons.prem1 True  $\varphi_{2s} \varphi_{2ss}$  by (intro Cons.IH) auto
          have 2: prv (imp (cnj  $\varphi_1$  (lcnj  $\varphi_{1s}$ )) (cnj  $\varphi_1$  (lcnj  $\varphi_{2s}$ )))
            using Cons  $\varphi_{2ss} \varphi_{2s} 1$  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])
        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  $\varphi_{2s}'$ ) (lcnj  $\varphi_{2s}$ ))
      unfolding  $\varphi_{2s}'$  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  $\varphi_{1s} \subseteq$  fmla and set  $\varphi_{2s} =$  set  $\varphi_{1s}$ 
shows prv (eqv (lcnj  $\varphi_{1s}$ ) (lcnj  $\varphi_{2s}$ ))
using assms prv_lcnj_mono by (intro prv_eqvI) auto

```

lemma *prv_lcnj_mono_imp*:
assumes $set \ \varphi 1s \subseteq fmla$ $set \ \varphi 2s \subseteq fmla$ **and** $\forall \ \varphi 2 \in set \ \varphi 2s. \exists \ \varphi 1 \in set \ \varphi 1s. prv \ (imp \ \varphi 1 \ \varphi 2)$
shows $prv \ (imp \ (lcnj \ \varphi 1s) \ (lcnj \ \varphi 2s))$
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 $F \subseteq fmla$ *finite* F $t \in trm$ $x \in var$
shows $prv \ (eqv \ (subst \ (scnj \ F) \ t \ x) \ (scnj \ ((\lambda \varphi. \ subst \ \varphi \ t \ x) \ 'F)))$
using *assms* **unfolding** *scnj_def* **by** (*fastforce intro!*: *prv_lcnj_eqv*)

lemma *prv_imp_subst_scnj*:
assumes $F \subseteq fmla$ *finite* F $t \in trm$ $x \in var$
shows $prv \ (imp \ (subst \ (scnj \ F) \ t \ x) \ (scnj \ ((\lambda \varphi. \ subst \ \varphi \ t \ x) \ 'F)))$
using *prv_subst_scnj*[*OF assms*] *assms* **by** (*intro prv_imp_eqvEL*) *auto*

lemma *prv_subst_scnj_imp*:
assumes $F \subseteq fmla$ *finite* F $t \in trm$ $x \in var$
shows $prv \ (imp \ (scnj \ ((\lambda \varphi. \ subst \ \varphi \ t \ x) \ 'F)) \ (subst \ (scnj \ F) \ t \ x))$
using *prv_subst_scnj*[*OF assms*] *assms* **by** (*intro prv_imp_eqvER*) *auto*

lemma *prv_scnj_imp_in*:
assumes $F \subseteq fmla$ *finite* F
and $\varphi \in F$
shows $prv \ (imp \ (scnj \ F) \ \varphi)$
unfolding *scnj_def* **using** *assms* **by** (*intro prv_lcnj_imp_in*) *auto*

lemma *prv_scnj_imp*:
assumes $\chi \in fmla$ **and** $F \subseteq fmla$ *finite* F
and $\varphi \in F$ **and** $prv \ (imp \ \varphi \ \chi)$
shows $prv \ (imp \ (scnj \ F) \ \chi)$
unfolding *scnj_def* **using** *assms* **by** (*intro prv_lcnj_imp*) *auto*

lemma *prv_imp_scnj*:
assumes $\chi \in fmla$ **and** $F \subseteq fmla$ *finite* F
and $\bigwedge \varphi. \ \varphi \in F \implies prv \ (imp \ \chi \ \varphi)$
shows $prv \ (imp \ \chi \ (scnj \ F))$
unfolding *scnj_def* **using** *assms* **by** (*intro prv_imp_lcnj*) *auto*

lemma *prv_scnj_mono*:
assumes $F1 \subseteq fmla$ **and** $F2 \subseteq F1$ **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 $F1 \subseteq fmla$ $F2 \subseteq fmla$ *finite* $F1$ *finite* $F2$
and $\forall \ \varphi 2 \in F2. \exists \ \varphi 1 \in F1. prv \ (imp \ \varphi 1 \ \varphi 2)$
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 $F \subseteq fmla$ **and** *finite* F **and** $\varphi \in fmla$

shows $prv (imp (scnj (insert \varphi F)) (cnj \varphi (scnj F)))$
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 $prv (scnj (insert \varphi F))$
shows $prv (cnj \varphi (scnj F))$
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 $prv (imp (cnj \varphi (scnj F)) (scnj (insert \varphi F)))$
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 $prv (cnj \varphi (scnj F))$
shows $prv (scnj (insert \varphi F))$
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 $prv (eqv (scnj (insert \varphi F)) (cnj \varphi (scnj F)))$
by (*simp add: assms prv_eqvI prv_imp_cnj_scnj prv_imp_scnj_insert*)

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

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

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

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

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

lemma *prv_rawpsubst_scnj*:
assumes $F \subseteq fmla$ *finite F*
and $snd \text{ ' (set txs) } \subseteq var \text{ fst ' (set txs) } \subseteq trm$
shows $prv (eqv (rawpsubst (scnj F) txs) (scnj ((\lambda \varphi. rawpsubst \varphi txs) \text{ ' F})))$

```

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]:  $\bigwedge x f. f \in F \implies x \in \text{set } (vs f) \implies x \in \text{var}$ 
  using vs_facts
  by (meson subsetD)

let ?tus = zip (map fst txs) us
let ?uxs = zip (map Var us) (map snd txs)
let ?tvs =  $\lambda \varphi. \text{zip } (map \text{fst } txs) (vs \varphi)$ 
let ?vxs =  $\lambda \varphi. \text{zip } (map \text{Var } (vs \varphi)) (map \text{snd } txs)$ 

let ?c = rawpsubst (scnj F) ?uxs
have c: prv (eqv ?c
  (scnj (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?uxs}$ ) ' 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 (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?uxs}$ ) ' F)) ?tus))
  using assms us_facts
  by (intro prv_eqv_rawpsubst) (auto intro!: rawpsubst dest!: set_zip_D)
moreover
have prv (eqv (rawpsubst (scnj (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?uxs}$ ) ' F)) ?tus)
  (scnj (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?tus}$ ) ' (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?uxs}$ ) ' 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 (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?tus}$ ) ' (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?uxs}$ ) ' F))))
  using assms us_facts apply - by (rule prv_eqv_trans) (auto intro!: rawpsubst dest!: set_zip_D)
moreover
have prv (eqv (scnj (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?tus}$ ) ' (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?uxs}$ ) ' F)))
  (scnj (( $\lambda \varphi. \text{rawpsubst } (\text{rawpsubst } \varphi (\text{?vxs } \varphi)) (\text{?tvs } \varphi))$ ) ' 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  $\varphi$  apply (rule bestI[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  $\varphi$  apply(rule beXI[of _  $\varphi$ ]) 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 (( $\lambda\varphi$ . rawpsubst (rawpsubst  $\varphi$  (?vxs  $\varphi$ )) (?tus  $\varphi$ )) ' 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  $F \subseteq \text{fmla}$  finite F snd ' set txs  $\subseteq$  var fst ' set txs  $\subseteq$  trm
  and distinct (map snd txs)
  shows prv (imp (psubst (scnj F) txs) (scnj (( $\lambda\varphi$ . psubst  $\varphi$  txs) ' F)))
  using prv_psubst_scnj[OF assms] assms apply(intro prv_imp_eqvEL) by auto

```

```

lemma prv_psubst_scnj_imp:
  assumes  $F \subseteq \text{fmla}$  finite F snd ' set txs  $\subseteq$  var fst ' set txs  $\subseteq$  trm
  and distinct (map snd txs)
  shows prv (imp (scnj (( $\lambda\varphi$ . psubst  $\varphi$  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  $\equiv$   $\neg$  prv fls

```

```

lemma consistent_def2: consistent  $\longleftrightarrow$  ( $\exists \varphi \in \text{fmla}$ .  $\neg$  prv  $\varphi$ )
  unfolding consistent_def using prv_expl by blast

```

```

lemma consistent_def3: consistent  $\longleftrightarrow$  ( $\forall \varphi \in \text{fmla}$ .  $\neg$  (prv  $\varphi \wedge$  prv (neg  $\varphi$ )))
  unfolding consistent_def using prv_neg_fls neg_def by auto

```

```

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

```

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

```

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

```

```

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

```

```

lemma  $\omega$ consistent_impliesStd1:

```

```

 $\omega$ consistent  $\implies$ 
 $\omega$ consistentStd1
unfolding  $\omega$ consistent_def  $\omega$ consistentStd1_def using prv_neg_neg by blast

```

```

lemma  $\omega$ consistent_impliesStd2:
 $\omega$ consistent  $\implies$ 
 $\omega$ consistentStd2
by (auto dest!:  $\omega$ consistent_impliesStd1 bspec[of _ _ neg _])
  simp:  $\omega$ consistentStd1_def  $\omega$ consistentStd2_def prv_neg_neg

```

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

```

lemma  $\omega$ consistent_iffStd1:
assumes  $\bigwedge \varphi. \varphi \in \text{fmla} \implies \text{prv} (\text{imp} (\text{neg} (\text{neg} \varphi)) \varphi)$ 
shows  $\omega$ consistent  $\longleftrightarrow$   $\omega$ consistentStd1
apply standard
subgoal using  $\omega$ consistent_impliesStd1 by auto
subgoal unfolding  $\omega$ consistentStd1_def  $\omega$ consistent_def
  by (meson assms exi neg prv_imp_mp) .

```

```

lemma  $\omega$ consistent_iffStd2:
assumes  $\bigwedge \varphi. \varphi \in \text{fmla} \implies \text{prv} (\text{imp} (\text{neg} (\text{neg} \varphi)) \varphi)$ 
shows  $\omega$ consistent  $\longleftrightarrow$   $\omega$ consistentStd2
unfolding  $\omega$ consistent_iffStd1[OF assms, simplified]
   $\omega$ consistentStd1_def  $\omega$ consistentStd2_def apply safe
subgoal for  $\varphi$   $x$ 
  by (auto simp: prv_neg_neg dest: bspec[of _ _ neg _])
subgoal for  $\varphi$   $x$ 
  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  $\omega$ consistentStd1_implies_consistent:
assumes  $\omega$ consistentStd1
shows consistent
unfolding consistent_def
proof safe
assume pf: prv fls
then obtain  $x$  where  $x: x \in \text{var } x \notin \text{Fvars } fls$ 
  using finite_Fvars getFresh by auto
let ?fls = cnj (fls) (eq1 (Var  $x$ ) (Var  $x$ ))
have 0:  $\forall n \in \text{num}. \text{prv} (\text{neg} (\text{subst } ?fls \ n \ x))$  and 1: prv (exi  $x$  fls)
  using  $x$  fls by (auto simp: pf prv_expl)
have 2:  $\neg \text{prv} (\text{exi } x \ ?fls)$  using 0 fls  $x$  assms
  unfolding  $\omega$ consistentStd1_def by simp
show False using 1 2 consistent_def consistent_def2 pf  $x$ (1) by blast
qed

```

```

lemma  $\omega$ consistentStd2_implies_consistent:
assumes  $\omega$ consistentStd2
shows consistent
unfolding consistent_def
proof safe
assume pf: prv fls
then obtain  $x$  where  $x: x \in \text{var } x \notin \text{Fvars } fls$ 
  using finite_Fvars getFresh by auto
let ?fls = cnj (fls) (eq1 (Var  $x$ ) (Var  $x$ ))
have 0:  $\forall n \in \text{num}. \text{prv} (\text{subst } ?fls \ n \ x)$  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:

$\bigwedge \varphi \chi. \varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies$

prv (imp φ (dsj φ χ))

and

prv_dsj_impR:

$\bigwedge \varphi \chi. \varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies$

prv (imp χ (dsj φ χ))

and

prv_imp_dsjE:

$\bigwedge \varphi \chi \psi. \varphi \in \text{fmla} \implies \chi \in \text{fmla} \implies \psi \in \text{fmla} \implies$

prv (imp (imp φ ψ) (imp (imp χ ψ) (imp (dsj φ χ) ψ)))

begin

lemma *prv_imp_dsjEE*:

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

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

3.3.1 Disjunction vs. disjunction

lemma *prv_dsj_com_imp*:

assumes $\varphi[\text{simp}]: \varphi \in \text{fmla}$ **and** $\chi[\text{simp}]: \chi \in \text{fmla}$
shows $\text{prv} (\text{imp} (\text{dsj } \varphi \chi) (\text{dsj } \chi \varphi))$
by (*metis* χ φ *dsj_imp* *prv_dsj_impL* *prv_dsj_impR* *prv_imp_dsjE* *prv_imp_mp*)

lemma *prv_dsj_com*:

assumes $\varphi[\text{simp}]: \varphi \in \text{fmla}$ **and** $\chi[\text{simp}]: \chi \in \text{fmla}$
shows $\text{prv} (\text{equiv} (\text{dsj } \varphi \chi) (\text{dsj } \chi \varphi))$
by (*simp* *add*: *prv_dsj_com_imp* *prv_equivI*)

lemma *prv_dsj_assoc_imp1*:

assumes $\varphi[\text{simp}]: \varphi \in \text{fmla}$ **and** $\chi[\text{simp}]: \chi \in \text{fmla}$ **and** $\psi[\text{simp}]: \psi \in \text{fmla}$
shows $\text{prv} (\text{imp} (\text{dsj } \varphi (\text{dsj } \chi \psi)) (\text{dsj} (\text{dsj } \varphi \chi) \psi))$

proof –

have $f1: \bigwedge f \text{ fa fb. } f \notin \text{fmla} \vee \neg \text{prv} (\text{imp } \text{fa } \text{fb}) \vee \text{fb} \notin \text{fmla} \vee \text{fa} \notin \text{fmla} \vee \text{prv} (\text{imp } \text{fa} (\text{dsj } \text{fb } f))$
by (*meson* *dsj* *prv_dsj_impL* *prv_prv_imp_trans*)

have $\text{prv} (\text{imp } \varphi (\text{dsj } \varphi \chi))$

by (*simp* *add*: *prv_dsj_impL*)

then show *?thesis*

using $f1$ χ φ ψ *dsj* *prv_dsj_impR* *prv_imp_dsjEE* **by** *presburger*

qed

lemma *prv_dsj_assoc_imp2*:

assumes $\varphi[\text{simp}]: \varphi \in \text{fmla}$ **and** $\chi[\text{simp}]: \chi \in \text{fmla}$ **and** $\psi[\text{simp}]: \psi \in \text{fmla}$
shows $\text{prv} (\text{imp} (\text{dsj} (\text{dsj } \varphi \chi) \psi) (\text{dsj } \varphi (\text{dsj } \chi \psi)))$

proof –

have $f1: \forall f \text{ fa fb. } (((\text{prv} (\text{imp } f (\text{dsj } \text{fa } \text{fb})) \vee \neg \text{prv} (\text{imp } f (\text{dsj } \text{fb } \text{fa}))) \vee f \notin \text{fmla}) \vee \text{fa} \notin \text{fmla}) \vee \text{fb} \notin \text{fmla}$

by (*meson* *dsj* *prv_dsj_com_imp* *prv_prv_imp_trans*)

have $\forall f \text{ fa fb. } (((\text{prv} (\text{imp } f (\text{dsj } \text{fa } \text{fb})) \vee \neg \text{prv} (\text{imp } f \text{fa})) \vee \text{fa} \notin \text{fmla}) \vee f \notin \text{fmla}) \vee \text{fb} \notin \text{fmla}$

by (*meson* *dsj* *prv_dsj_impL* *prv_prv_imp_trans*)

then show *?thesis*

using $f1$ **by** (*metis* χ φ ψ *dsj* *prv_dsj_impR* *prv_imp_dsjEE*)

qed

lemma *prv_dsj_assoc*:

assumes $\varphi[\text{simp}]: \varphi \in \text{fmla}$ **and** $\chi[\text{simp}]: \chi \in \text{fmla}$ **and** $\psi[\text{simp}]: \psi \in \text{fmla}$
shows $\text{prv} (\text{equiv} (\text{dsj } \varphi (\text{dsj } \chi \psi)) (\text{dsj} (\text{dsj } \varphi \chi) \psi))$
by (*simp* *add*: *prv_dsj_assoc_imp1* *prv_dsj_assoc_imp2* *prv_equivI*)

lemma *prv_dsj_com_imp3*:

assumes $\varphi 1 \in \text{fmla}$ $\varphi 2 \in \text{fmla}$ $\varphi 3 \in \text{fmla}$
shows $\text{prv} (\text{imp} (\text{dsj } \varphi 1 (\text{dsj } \varphi 2 \varphi 3)) (\text{dsj } \varphi 2 (\text{dsj } \varphi 1 \varphi 3)))$

proof –

have $\forall f \text{ fa fb. } (((\text{prv} (\text{imp } f (\text{dsj } \text{fb } \text{fa})) \vee \neg \text{prv} (\text{imp } f \text{fa})) \vee \text{fa} \notin \text{fmla}) \vee f \notin \text{fmla}) \vee \text{fb} \notin \text{fmla}$
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*:

$\varphi 1 \in fmla \implies \varphi 2 \in fmla \implies \chi 1 \in fmla \implies \chi 2 \in fmla \implies$
 $prv (imp \varphi 1 \chi 1) \implies prv (imp \varphi 2 \chi 2) \implies prv (imp (dsj \varphi 1 \varphi 2) (dsj \chi 1 \chi 2))$
 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 $\varphi[simp]: \varphi \in fmla$ **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 $\varphi[simp]: \varphi \in fmla$ **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 $\varphi[simp]: \varphi \in fmla$ **and** $\chi 1[simp]: \chi 1 \in fmla$ **and** $\chi 2[simp]: \chi 2 \in fmla$
shows $prv (equiv (cnj \varphi (dsj \chi 1 \chi 2)) (dsj (cnj \varphi \chi 1) (cnj \varphi \chi 2)))$
 by (simp add: prv_cnj_dsj_distrib1 prv_cnj_dsj_distrib2 prv_equivI)

lemma *prv_dsj_cnj_distrib1*:

assumes $\varphi[simp]: \varphi \in fmla$ **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 $\varphi[simp]: \varphi \in fmla$ **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 $\chi 1 \chi 2 \varphi$ 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 $\varphi[simp]: \varphi \in fmla$ **and** $\chi 1[simp]: \chi 1 \in fmla$ **and** $\chi 2[simp]: \chi 2 \in fmla$
shows $prv (equiv (dsj \varphi (cnj \chi 1 \chi 2)) (cnj (dsj \varphi \chi 1) (dsj \varphi \chi 2)))$
 by (simp add: prv_dsj_cnj_distrib1 prv_dsj_cnj_distrib2 prv_equivI)

3.3.3 Disjunction vs. True and False

lemma *prv_dsjR_fls*: $\varphi \in fmla \implies prv (equiv (dsj fls \varphi) \varphi)$

by (simp add: prv_dsj_impR prv_equivI prv_imp_dsjEE prv_imp_refl)

lemma *prv_dsjL_fls*: $\varphi \in fmla \implies prv (equiv (dsj \varphi fls) \varphi)$

by (simp add: prv_dsj_impL prv_equivI prv_imp_dsjEE prv_imp_refl)

lemma *prv_dsjR_tru*: $\varphi \in fmla \implies prv (equiv (dsj tru \varphi) tru)$

by (simp add: prv_dsj_impR prv_equivI prv_imp_tru)

lemma *prv_dsjL_tru*: $\varphi \in fmla \implies prv (equiv (dsj \varphi tru) tru)$

by (simp add: prv_dsj_impL prv_equivI 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* $\varphi s \subseteq fmla$

and $\varphi \in set\ \varphi s$

shows *prv* (*imp* φ (*ldsj* φs))

proof–

have $\varphi \in fmla$ **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 $\chi \in fmla$ **and** *set* $\varphi s \subseteq fmla$

and $\varphi \in set\ \varphi s$ **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 $\chi \in fmla$ **and** *set* $\varphi s \subseteq fmla$

and $\bigwedge \varphi. \varphi \in set\ \varphi s \implies prv\ (imp\ \varphi\ \chi)$

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 $\varphi \in fmla$ *set* $\varphi 1s \subseteq fmla$ *set* $\varphi 2s \subseteq fmla$

shows *prv* (*imp* (*ldsj* ($\varphi 1s$ @ $\varphi \# \varphi 2s$)) (*dsj* φ (*ldsj* ($\varphi 1s$ @ $\varphi 2s$))))

using *assms* **proof**(*induction* $\varphi 1s$)

case (*Cons* $\varphi 1$ $\varphi 1s$)

have [*intro!*]: *set* ($\varphi 1s$ @ $\varphi 2s$) $\subseteq fmla$ *set* ($\varphi 1s$ @ $\varphi \# \varphi 2s$) $\subseteq fmla$ **using** *Cons* **by** *auto*

have 0: *prv* (*imp* (*dsj* $\varphi 1$ (*dsj* φ (*ldsj* ($\varphi 1s$ @ $\varphi 2s$))))

(*dsj* φ (*dsj* $\varphi 1$ (*ldsj* ($\varphi 1s$ @ $\varphi 2s$))))

using *Cons* **by** (*intro* *prv_dsj_com_imp3*) *fastforce+*

have 1: *prv* (*imp* (*dsj* $\varphi 1$ (*ldsj* ($\varphi 1s$ @ $\varphi \# \varphi 2s$))))

(*dsj* $\varphi 1$ (*dsj* φ (*ldsj* ($\varphi 1s$ @ $\varphi 2s$))))

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* $\varphi s \subseteq fmla$

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 $\varphi 2s$: *set* $\varphi 2s \subseteq fmla$ **and** *set* $\varphi 1s \subseteq set\ \varphi 2s$

shows *prv* (*imp* (*ldsj* $\varphi 1s$) (*ldsj* $\varphi 2s$))

proof–

define $\varphi 1s'$ **where** $\varphi 1s'$: $\varphi 1s' = remdups\ \varphi 1s$

have *A*: *set* $\varphi 1s' \subseteq set\ \varphi 2s$ *distinct* $\varphi 1s'$ **unfolding** $\varphi 1s'$ **using** *assms* **by** *auto*

```

have B: prv (imp (ldsjs  $\varphi 1s'$ ) (ldsjs  $\varphi 2s$ ))
using  $\varphi 2s$  A proof(induction  $\varphi 2s$  arbitrary:  $\varphi 1s'$ )
  case (Cons  $\varphi 2$   $\varphi 2s$   $\varphi 1ss$ )
  show ?case proof(cases  $\varphi 2 \in set \varphi 1ss$ )
    case True
    then obtain  $\varphi 1ss1$   $\varphi 1ss2$  where  $\varphi 1ss$ :  $\varphi 1ss = \varphi 1ss1 @ \varphi 2 \# \varphi 1ss2$ 
    by (meson split_list)
    define  $\varphi 1s$  where  $\varphi 1s$ :  $\varphi 1s \equiv \varphi 1ss1 @ \varphi 1ss2$ 
    have nin:  $\varphi 2 \notin set \varphi 1s$  using  $\varphi 1ss$  (distinct  $\varphi 1ss$ ) unfolding  $\varphi 1s$  by auto
    have [intro!,simp]:  $set \varphi 1s \subseteq fmla$  unfolding  $\varphi 1s$  using  $\varphi 1ss$  Cons by auto
    have 0: prv (imp (ldsjs  $\varphi 1ss$ ) (dsj  $\varphi 2$  (ldsjs  $\varphi 1s$ )))
      unfolding  $\varphi 1s$   $\varphi 1ss$ 
      apply(rule prv_ldsj_imp_inner) using Cons  $\varphi 1ss$  by auto
    have 1: prv (imp (ldsjs  $\varphi 1s$ ) (ldsjs  $\varphi 2s$ )) apply(rule Cons.IH) using nin Cons.prem True
    using  $\varphi 1s$   $\varphi 1ss$  by auto
    have 2: prv (imp (dsj  $\varphi 2$  (ldsjs  $\varphi 1s$ )) (dsj  $\varphi 2$  (ldsjs  $\varphi 2s$ )))
    using Cons  $\varphi 1ss$   $\varphi 1s$  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 ldsjs_order.trans prv_imp_ldsj_in prv_ldsj_imp_subset_code(1))
qed
qed(simp add: prv_imp_refl)
have C: prv (imp (ldsjs  $\varphi 1s$ ) (ldsjs  $\varphi 1s'$ ))
  unfolding  $\varphi 1s'$  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  $set \varphi 1s \subseteq fmla$  and  $set \varphi 2s = set \varphi 1s$ 
shows prv (eqv (ldsjs  $\varphi 1s$ ) (ldsjs  $\varphi 2s$ ))
  using assms prv_ldsj_mono by (intro prv_eqvI) auto

```

```

lemma prv_ldsj_mono_imp:
assumes  $set \varphi 1s \subseteq fmla$   $set \varphi 2s \subseteq fmla$  and  $\forall \varphi 1 \in set \varphi 1s. \exists \varphi 2 \in set \varphi 2s. prv (imp \varphi 1 \varphi 2)$ 
shows prv (imp (ldsjs  $\varphi 1s$ ) (ldsjs  $\varphi 2s$ ))
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:
 $F \subseteq fmla \implies finite F \implies t \in trm \implies x \in var \implies$ 
prv (eqv (subst (sdsjs F) t x) (sdsjs (( $\lambda \varphi. subst \varphi t x$ ) ' F)))
unfolding sdsjs_def by (fastforce intro!: prv_ldsj_eqv)

```

```

lemma prv_imp_sdsj_in:
assumes  $\varphi \in fmla$  and  $F \subseteq fmla$  finite F
and  $\varphi \in F$ 
shows prv (imp  $\varphi$  (sdsjs F))
unfolding sdsjs_def using assms by (intro prv_imp_ldsj_in) auto

```

```

lemma prv_imp_sdsj:
assumes  $\chi \in fmla$  and  $F \subseteq fmla$  finite F

```


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

lemma prv_sdsj_imp :
assumes $\chi \in fmla$ **and** $F \subseteq fmla$ *finite* F
and $\bigwedge \varphi. \varphi \in F \implies prv (imp \varphi \chi)$
shows $prv (imp (sdsj F) \chi)$
unfolding $sdsj_def$ **using** $assms$ **by** $(intro\ prv_ldsjs_imp)$ *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** $(intro\ prv_ldsjs_mono)$
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 $\forall \varphi1 \in F1. \exists \varphi2 \in F2. prv (imp \varphi1 \varphi2)$
shows $prv (imp (sdsj F1) (sdsj F2))$
unfolding $sdsj_def$ **using** $assms$ **by** $(intro\ prv_ldsjs_mono_imp)$ *auto*

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

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

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

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

lemma $prv_dsj_imp_sdsj2$:
 $\varphi \in fmla \implies \psi \in fmla \implies prv (imp (dsj \varphi \psi) (sdsj \{\varphi, \psi\}))$
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 $snd \text{ ' (set } txs) \subseteq var \text{ fst ' (set } txs) \subseteq trm$
shows $prv (equiv (rawpsubst (sdsj F) txs) (sdsj ((\lambda \varphi. rawpsubst \varphi txs) \text{ ' } F)))$
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** $Cons.prem$ s **by** *auto*
have $0: (\lambda \varphi. rawpsubst (subst \varphi t\ x) txs) \text{ ' } F =$
 $(\lambda \varphi. rawpsubst \varphi txs) \text{ ' } ((\lambda \varphi. subst \varphi t\ x) \text{ ' } F)$
using $Cons.prem$ s **by** *auto*
have $prv (equiv (subst (sdsj F) t\ x)$

```

      (sdsj ((λφ. subst φ t x) ' F)))
using Cons.prem1 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.prem2 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.prem3 by (intro Cons.IH) auto
ultimately show ?case
using Cons.prem4 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 vs: ∧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]:  $\bigwedge x f. f \in F \implies x \in \text{set } (vs f) \implies x \in \text{var}$ 
using vs_facts by (meson subsetD)

let ?tus = zip (map fst txs) us
let ?uxs = zip (map Var us) (map snd txs)
let ?tvs =  $\lambda \varphi. \text{zip } (\text{map fst txs}) (vs \varphi)$ 
let ?vxs =  $\lambda \varphi. \text{zip } (\text{map Var } (vs \varphi)) (\text{map snd txs})$ 

let ?c = rawpsubst (sdsj F) ?uxs
have c: prv (eqv ?c
  (sdsj (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?uxs}$ ) ' 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 (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?uxs}$ ) ' F)) ?tus))
using assms us_facts by (intro prv_eqv_rawpsubst) (auto intro!: rawpsubst dest!: set_zip_D)
moreover
have prv (eqv (rawpsubst (sdsj (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?uxs}$ ) ' F)) ?tus)
  (sdsj (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?tus}$ ) ' (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?uxs}$ ) ' 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 (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?tus}$ ) ' (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?uxs}$ ) ' F))))
using assms us_facts apply- by (rule prv_eqv_trans) (auto intro!: rawpsubst dest!: set_zip_D)
moreover
have prv (eqv (sdsj (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?tus}$ ) ' (( $\lambda \varphi. \text{rawpsubst } \varphi \text{ ?uxs}$ ) ' F)))
  (sdsj (( $\lambda \varphi. \text{rawpsubst } (\text{rawpsubst } \varphi \text{ (?vxs } \varphi)) \text{ (?tvs } \varphi)$ ) ' 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  $\varphi$  apply (rule bexI[of _  $\varphi$ ]) 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  $\varphi$  apply (rule bexI[of _  $\varphi$ ]) 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 (( $\lambda \varphi. \text{rawpsubst } (\text{rawpsubst } \varphi \text{ (?vxs } \varphi)) \text{ (?tvs } \varphi)$ ) ' 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 L_q 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 = ( $\lambda$  t1 t2. psubst Lq [(t1,zz), (t2,yy)] in
 $\forall \varphi \in \text{fmla}$ .  $\forall q \in \text{num}$ .  $\text{Fvars } \varphi = \{\text{zz}\} \wedge (\forall p \in \text{num}$ . prv (subst  $\varphi$  p zz))
 $\rightarrow \text{prv}$  (all zz (imp (LLq (Var zz) q)  $\varphi$ ))
and
Lq_num2:
let LLq = ( $\lambda$  t1 t2. psubst Lq [(t1,zz), (t2,yy)] in
 $\forall p \in \text{num}$ .  $\exists P \subseteq \text{num}$ . finite P  $\wedge$  prv (dsj (sdsj {eql (Var yy) r | r. r  $\in$  P}) (LLq p (Var yy)))
begin

lemma LLq_num:
assumes  $\varphi \in \text{fmla}$   $q \in \text{num}$   $\text{Fvars } \varphi = \{\text{zz}\}$   $\forall p \in \text{num}$ . prv (subst  $\varphi$  p zz)
shows prv (all zz (imp (LLq (Var zz) q)  $\varphi$ ))
using assms Lq_num unfolding LLq_def by auto

lemma LLq_num2:
assumes  $p \in \text{num}$ 
shows  $\exists P \subseteq \text{num}$ . finite P  $\wedge$  prv (dsj (sdsj {eql (Var yy) r | r. r  $\in$  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  $\Rightarrow$  'fmla  $\Rightarrow$  bool where
nprv F  $\varphi \equiv$  prv (imp (scnj F)  $\varphi$ )
```

```
lemma nprv_hyp[simp,intro]:
 $\varphi \in F \Longrightarrow F \subseteq$  fmla  $\Longrightarrow$  finite F  $\Longrightarrow$  nprv F  $\varphi$ 
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  $\varphi \Longrightarrow \varphi \in$  fmla  $\Longrightarrow$  nprv {}  $\varphi$ 
unfolding nprv_def by (simp add: prv_imp_triv)
```

```
lemma prv_nprv_emp:  $\varphi \in$  fmla  $\Longrightarrow$  prv  $\varphi \longleftrightarrow$  nprv {}  $\varphi$ 
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  $\varphi$ 
and  $F \subseteq$  fmla finite F  $G \subseteq F$   $\varphi \in$  fmla
shows nprv F  $\varphi$ 
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  $\varphi$  and nprv (insert  $\varphi$  F)  $\psi$ 
and  $F \subseteq$  fmla finite F  $\varphi \in$  fmla  $\psi \in$  fmla
shows nprv F  $\psi$ 
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:
```

$nprv\ F\ \varphi1 \implies nprv\ (insert\ \varphi1\ F)\ \varphi2 \implies nprv\ (insert\ \varphi2\ (insert\ \varphi1\ F))\ \psi \implies$
 $F \subseteq fmla \implies finite\ F \implies \varphi1 \in fmla \implies \varphi2 \in fmla \implies \psi \in fmla \implies$
 $nprv\ F\ \psi$
by (*meson finite.insertI insert_subsetI nprv_cut*)

lemma *nprv_cut2*:

$nprv\ F\ \varphi1 \implies nprv\ F\ \varphi2 \implies$
 $F \subseteq fmla \implies finite\ F \implies \varphi1 \in fmla \implies \varphi2 \in fmla \implies \psi \in fmla \implies$
 $nprv\ (insert\ \varphi2\ (insert\ \varphi1\ F))\ \psi \implies nprv\ F\ \psi$
by (*meson finite.insertI insert_subsetI nprv_mono nprv_strong_cut2 subset_insertI*)

Useful for fine control of the eigenformula:

lemma *nprv_insertShiftI*:

$nprv\ (insert\ \varphi1\ (insert\ \varphi2\ F))\ \psi \implies nprv\ (insert\ \varphi2\ (insert\ \varphi1\ F))\ \psi$
by (*simp add: insert_commute*)

lemma *nprv_insertShift2I*:

$nprv\ (insert\ \varphi3\ (insert\ \varphi1\ (insert\ \varphi2\ F)))\ \psi \implies nprv\ (insert\ \varphi1\ (insert\ \varphi2\ (insert\ \varphi3\ F)))\ \psi$
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*: $prv\ \varphi \implies \varphi \in fmla \implies F \subseteq fmla \implies finite\ F \implies nprv\ F\ \varphi$
using *prv_nprv0I*
by (*simp add: nprv_def prv_imp_triv*)

thm *prv_nprv0I*

lemma *prv_nprv1I*:

assumes $\varphi \in fmla\ \psi \in fmla$ **and** $prv\ (imp\ \varphi\ \psi)$
shows $nprv\ \{\varphi\}\ \psi$
using *assms unfolding nprv_def* **by** (*simp add: prv_scnj_imp*)

lemma *prv_nprv2I*:

assumes $prv\ (imp\ \varphi1\ (imp\ \varphi2\ \psi))\ \varphi1 \in fmla\ \varphi2 \in fmla\ \psi \in fmla$
shows $nprv\ \{\varphi1, \varphi2\}\ \psi$
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*: $nprv\ \{\}\ \varphi \implies \varphi \in fmla \implies prv\ \varphi$
using *prv_nprv_emp* **by** *auto*

4.4 More Structural Properties

lemma *nprv_clear*: $nprv\ \{\}\ \varphi \implies F \subseteq fmla \implies finite\ F \implies \varphi \in fmla \implies nprv\ F\ \varphi$
by (*rule nprv_mono*) *auto*

lemma *nprv_cut_set*:

assumes $F: finite\ F\ F \subseteq fmla$ **and** $G: finite\ G\ G \subseteq fmla\ \chi \in fmla$
and $n1: \bigwedge \psi. \psi \in G \implies nprv\ F\ \psi$ **and** $n2: nprv\ (G \cup F)\ \chi$
shows $nprv\ F\ \chi$
using $G\ F\ n1\ n2$ **proof**(*induction arbitrary: F* χ)
case (*insert* $\psi\ G\ F\ \chi$)
hence $0: nprv\ F\ \psi$ **by** *auto*

```

have 1: nprv (insert  $\psi$  F)  $\chi$ 
using insert.premis 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.premis by (intro nprv_cut[OF 0 1]) auto
qed(insert nprv_clear, auto)

```

```

lemma nprv_clear2_1:
nprv { $\varphi_2$ }  $\psi \implies \varphi_1 \in \text{fmla} \implies \varphi_2 \in \text{fmla} \implies \psi \in \text{fmla} \implies$ 
nprv { $\varphi_1, \varphi_2$ }  $\psi$ 
by (rule nprv_mono) auto

```

```

lemma nprv_clear2_2:
nprv { $\varphi_1$ }  $\psi \implies \varphi_1 \in \text{fmla} \implies \varphi_2 \in \text{fmla} \implies \psi \in \text{fmla} \implies$ 
nprv { $\varphi_1, \varphi_2$ }  $\psi$ 
by (rule nprv_mono) auto

```

```

lemma nprv_clear3_1:
nprv { $\varphi_2, \varphi_3$ }  $\psi \implies \varphi_1 \in \text{fmla} \implies \varphi_2 \in \text{fmla} \implies \varphi_3 \in \text{fmla} \implies \psi \in \text{fmla} \implies$ 
nprv { $\varphi_1, \varphi_2, \varphi_3$ }  $\psi$ 
by (rule nprv_mono) auto

```

```

lemma nprv_clear3_2:
nprv { $\varphi_1, \varphi_3$ }  $\psi \implies \varphi_1 \in \text{fmla} \implies \varphi_2 \in \text{fmla} \implies \varphi_3 \in \text{fmla} \implies \psi \in \text{fmla} \implies$ 
nprv { $\varphi_1, \varphi_2, \varphi_3$ }  $\psi$ 
by (rule nprv_mono) auto

```

```

lemma nprv_clear3_3:
nprv { $\varphi_1, \varphi_2$ }  $\psi \implies \varphi_1 \in \text{fmla} \implies \varphi_2 \in \text{fmla} \implies \varphi_3 \in \text{fmla} \implies \psi \in \text{fmla} \implies$ 
nprv { $\varphi_1, \varphi_2, \varphi_3$ }  $\psi$ 
by (rule nprv_mono) auto

```

```

lemma nprv_clear4_1:
nprv { $\varphi_2, \varphi_3, \varphi_4$ }  $\psi \implies \varphi_1 \in \text{fmla} \implies \varphi_2 \in \text{fmla} \implies \varphi_3 \in \text{fmla} \implies \varphi_4 \in \text{fmla} \implies \psi \in \text{fmla} \implies$ 
nprv { $\varphi_1, \varphi_2, \varphi_3, \varphi_4$ }  $\psi$ 
by (rule nprv_mono) auto

```

```

lemma nprv_clear4_2:
nprv { $\varphi_1, \varphi_3, \varphi_4$ }  $\psi \implies \varphi_1 \in \text{fmla} \implies \varphi_2 \in \text{fmla} \implies \varphi_3 \in \text{fmla} \implies \varphi_4 \in \text{fmla} \implies \psi \in \text{fmla} \implies$ 
nprv { $\varphi_1, \varphi_2, \varphi_3, \varphi_4$ }  $\psi$ 
by (rule nprv_mono) auto

```

```

lemma nprv_clear4_3:
nprv { $\varphi_1, \varphi_2, \varphi_4$ }  $\psi \implies \varphi_1 \in \text{fmla} \implies \varphi_2 \in \text{fmla} \implies \varphi_3 \in \text{fmla} \implies \varphi_4 \in \text{fmla} \implies \psi \in \text{fmla} \implies$ 
nprv { $\varphi_1, \varphi_2, \varphi_3, \varphi_4$ }  $\psi$ 
by (rule nprv_mono) auto

```

```

lemma nprv_clear4_4:
nprv { $\varphi_1, \varphi_2, \varphi_3$ }  $\psi \implies \varphi_1 \in \text{fmla} \implies \varphi_2 \in \text{fmla} \implies \varphi_3 \in \text{fmla} \implies \varphi_4 \in \text{fmla} \implies \psi \in \text{fmla} \implies$ 
nprv { $\varphi_1, \varphi_2, \varphi_3, \varphi_4$ }  $\psi$ 
by (rule nprv_mono) auto

```

```

lemma nprv_clear5_1:
nprv { $\varphi_2, \varphi_3, \varphi_4, \varphi_5$ }  $\psi \implies \varphi_1 \in \text{fmla} \implies \varphi_2 \in \text{fmla} \implies \varphi_3 \in \text{fmla} \implies \varphi_4 \in \text{fmla} \implies \varphi_5 \in \text{fmla} \implies$ 

```


$\psi \in fmla \implies$
 $nprv \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5\} \psi$
by (rule nprv_mono) auto

lemma nprv_clear5_2:
 $nprv \{\varphi_1, \varphi_3, \varphi_4, \varphi_5\} \psi \implies \varphi_1 \in fmla \implies \varphi_2 \in fmla \implies \varphi_3 \in fmla \implies \varphi_4 \in fmla \implies \varphi_5 \in fmla \implies$
 $\psi \in fmla \implies$
 $nprv \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5\} \psi$
by (rule nprv_mono) auto

lemma nprv_clear5_3:
 $nprv \{\varphi_1, \varphi_2, \varphi_4, \varphi_5\} \psi \implies \varphi_1 \in fmla \implies \varphi_2 \in fmla \implies \varphi_3 \in fmla \implies \varphi_4 \in fmla \implies \varphi_5 \in fmla \implies$
 $\psi \in fmla \implies$
 $nprv \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5\} \psi$
by (rule nprv_mono) auto

lemma nprv_clear5_4:
 $nprv \{\varphi_1, \varphi_2, \varphi_3, \varphi_5\} \psi \implies \varphi_1 \in fmla \implies \varphi_2 \in fmla \implies \varphi_3 \in fmla \implies \varphi_4 \in fmla \implies \varphi_5 \in fmla \implies$
 $\psi \in fmla \implies$
 $nprv \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5\} \psi$
by (rule nprv_mono) auto

lemma nprv_clear5_5:
 $nprv \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\} \psi \implies \varphi_1 \in fmla \implies \varphi_2 \in fmla \implies \varphi_3 \in fmla \implies \varphi_4 \in fmla \implies \varphi_5 \in fmla \implies$
 $\psi \in fmla \implies$
 $nprv \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5\} \psi$
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!: bezI[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*:

$nprv\ F\ (imp\ \varphi\ \psi) \implies nprv\ F\ \varphi \implies$
 $F \subseteq fmla \implies finite\ F \implies \varphi \in fmla \implies \psi \in fmla \implies$
 $nprv\ F\ \psi$

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*:

$nprv\ F\ (imp\ \varphi\ \psi) \implies nprv\ F\ \varphi \implies nprv\ (insert\ \psi\ F)\ \chi \implies$
 $F \subseteq fmla \implies finite\ F \implies \varphi \in fmla \implies \psi \in fmla \implies \chi \in fmla \implies$
 $nprv\ F\ \chi$

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*:

$nprv\ F\ \varphi \implies nprv\ F\ \psi \implies$
 $F \subseteq fmla \implies finite\ F \implies \varphi \in fmla \implies \psi \in fmla \implies$
 $nprv\ F\ (cnj\ \varphi\ \psi)$

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

lemma *nprv_cnjE*:

$nprv\ F\ (cnj\ \varphi1\ \varphi2) \implies nprv\ (insert\ \varphi1\ (insert\ \varphi2\ F))\ \psi \implies$
 $F \subseteq fmla \implies finite\ F \implies \varphi1 \in fmla \implies \varphi2 \in fmla \implies \psi \in fmla \implies$
 $nprv\ F\ \psi$

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*:

$nprv\ F\ \varphi \implies$
 $F \subseteq fmla \implies finite\ F \implies \varphi \in fmla \implies \psi \in fmla \implies$
 $nprv\ F\ (dsj\ \varphi\ \psi)$

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

lemma *nprv_dsjIR*:

$nprv\ F\ \psi \implies$
 $F \subseteq fmla \implies finite\ F \implies \varphi \in fmla \implies \psi \in fmla \implies$
 $nprv\ F\ (dsj\ \varphi\ \psi)$

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 $F \subseteq fmla\ finite\ F\ \varphi \in fmla\ \psi \in fmla\ \chi \in fmla$
shows *nprv F χ*

proof–

have $nprv\ F\ (imp\ (dsj\ \varphi\ \psi)\ \chi)$
by $(meson\ assms\ dsj\ imp\ nprv_def\ nprv_impI\ prv_imp_com\ prv_imp_dsjEE\ scnj)$
hence $nprv\ (insert\ (dsj\ \varphi\ \psi)\ F)\ \chi$ **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 $nprv_flsE: nprv\ F\ fls \implies F \subseteq fmla \implies finite\ F \implies \varphi \in fmla \implies nprv\ F\ \varphi$
unfolding $nprv_def$ **using** $prv_prv_imp_trans\ scnj$ **by** $blast$

lemmas $nprv_flsE0 = nprv_flsE[OF\ nprv_hyp,\ simped]$

lemma $nprv_truI: F \subseteq fmla \implies finite\ F \implies nprv\ F\ tru$
unfolding $nprv_def$ **by** $(simp\ add:\ prv_imp_tru)$

lemma $nprv_negI:$
 $nprv\ (insert\ \varphi\ F)\ fls \implies$
 $F \subseteq fmla \implies finite\ F \implies \varphi \in fmla \implies$
 $nprv\ F\ (neg\ \varphi)$
unfolding neg_def **by** $(auto\ intro:\ nprv_impI)$

lemma $nprv_neg_fls:$
 $nprv\ F\ (neg\ \varphi) \implies nprv\ F\ \varphi \implies$
 $F \subseteq fmla \implies finite\ F \implies \varphi \in fmla \implies \psi \in fmla \implies$
 $nprv\ F\ fls$
unfolding neg_def **using** $nprv_mp$ **by** $blast$

lemma $nprv_negE:$
 $nprv\ F\ (neg\ \varphi) \implies nprv\ F\ \varphi \implies$
 $F \subseteq fmla \implies finite\ F \implies \varphi \in fmla \implies \psi \in fmla \implies$
 $nprv\ F\ \psi$
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 $nprv_scnjI:$
 $(\bigwedge\ \psi.\ \psi \in G \implies nprv\ F\ \psi) \implies$
 $F \subseteq fmla \implies finite\ F \implies G \subseteq fmla \implies finite\ G \implies$
 $nprv\ F\ (scnj\ G)$
unfolding $nprv_def$ **by** $(simp\ add:\ prv_imp_scnj)$

lemma $nprv_scnjE:$
 $nprv\ F\ (scnj\ G) \implies nprv\ (G \cup F)\ \psi \implies$
 $F \subseteq fmla \implies finite\ F \implies G \subseteq fmla \implies finite\ G \implies \psi \in fmla \implies$
 $nprv\ F\ \psi$
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*:
 $(\bigwedge \psi. \psi \in \text{set } \psi s \implies \text{nprv } F \psi) \implies$
 $F \subseteq \text{fmla} \implies \text{finite } F \implies \text{set } \psi s \subseteq \text{fmla} \implies$
 $\text{nprv } F (\text{lcj } \psi s)$
unfolding *nprv_def* **by** (*simp add: prv_imp_lcnj*)

lemma *nprv_lcnjE*:
 $\text{nprv } F (\text{lcj } \varphi s) \implies \text{nprv } (\text{set } \varphi s \cup F) \psi \implies$
 $F \subseteq \text{fmla} \implies \text{finite } F \implies \text{set } \varphi s \subseteq \text{fmla} \implies \psi \in \text{fmla} \implies$
 $\text{nprv } F \psi$
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*:
 $\text{nprv } F \varphi \implies$
 $F \subseteq \text{fmla} \implies \text{finite } F \implies G \subseteq \text{fmla} \implies \text{finite } G \implies \varphi \in G \implies$
 $\text{nprv } F (\text{sdsj } G)$
unfolding *nprv_def* **by** (*simp add: prv_imp_sdsj*)

lemma *nprv_sdsjE*:
assumes *nprv F (sdsj G)*
and $\bigwedge \psi. \psi \in G \implies \text{nprv } (\text{insert } \psi F) \chi$
and $F \subseteq \text{fmla} \text{ finite } F \ G \subseteq \text{fmla} \text{ finite } G \ \chi \in \text{fmla}$
shows *nprv F \chi*
proof–
have *0: prv (imp (sdsj G) (imp (scnj F) \chi))*
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) \chi)*
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 $nprv_ldsji$:
 $nprv\ F\ \varphi \implies$
 $F \subseteq fmla \implies finite\ F \implies set\ \varphi s \subseteq fmla \implies \varphi \in set\ \varphi s \implies$
 $nprv\ F\ (lds\ j\ \varphi s)$
unfolding $nprv_def$ **by** ($simp\ add:\ prv_imp_lds\ j$)

lemma $nprv_ldsje$:
assumes $nprv\ F\ (lds\ j\ \psi s)$
and $\bigwedge\ \psi.\ \psi \in set\ \psi s \implies nprv\ (insert\ \psi\ F)\ \chi$
and $F \subseteq fmla\ finite\ F\ set\ \psi s \subseteq fmla\ \chi \in fmla$
shows $nprv\ F\ \chi$
proof–
have $0:\ prv\ (imp\ (lds\ j\ \psi s)\ (imp\ (scnj\ F)\ \chi))$
using $assms$ **apply** ($intro\ prv_lds\ j_imp$)
subgoal **by** $auto$
subgoal **by** $auto$
subgoal **by** ($meson\ nprv_def\ nprv_impI\ prv_imp_com\ scnj\ set_rev_mp$) .
hence $nprv\ F\ (imp\ (lds\ j\ \psi s)\ \chi)$
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 $nprv_allI$:
 $nprv\ F\ \varphi \implies$
 $F \subseteq fmla \implies finite\ F \implies \varphi \in fmla \implies x \in var \implies x \notin \bigcup\ (Fvars\ 'F) \implies$
 $nprv\ F\ (all\ x\ \varphi)$
unfolding $nprv_def$ **by** ($rule\ prv_all_imp_gen$) $auto$

lemma $nprv_allE$:
assumes $nprv\ F\ (all\ x\ \varphi)\ nprv\ (insert\ (subst\ \varphi\ t\ x)\ F)\ \psi$
 $F \subseteq fmla\ finite\ F\ \varphi \in fmla\ t \in trm\ x \in var\ \psi \in fmla$
shows $nprv\ F\ \psi$
proof–
have $nprv\ F\ (subst\ \varphi\ t\ x)$
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 $nprv_exiI$:
 $nprv\ F\ (subst\ \varphi\ t\ x) \implies$
 $F \subseteq fmla \implies finite\ F \implies \varphi \in fmla \implies t \in trm \implies x \in var \implies$
 $nprv\ F\ (exi\ x\ \varphi)$
unfolding $nprv_def$ **by** ($meson\ exi\ local.subst\ prv_exi_inst\ prv_prv_imp_trans\ scnj$)

lemma $nprv_exiE$:
assumes $n:\ nprv\ F\ (exi\ x\ \varphi)$

```

and  $nn$ :  $nprv$  ( $insert\ \varphi\ F$ )  $\psi$ 
and  $0[simp]$ :  $F \subseteq fmla$   $finite\ F$   $\varphi \in fmla$   $x \in var$   $\psi \in fmla$ 
and  $x$ :  $x \notin \bigcup (Fvars\ 'F)$   $x \notin Fvars$   $\psi$ 
shows  $nprv\ F\ \psi$ 
proof-
  have  $nprv\ F$  ( $imp$  ( $exi\ x\ \varphi$ )  $\psi$ ) 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  $blast\ \dots$ 
  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  $prv\ \varphi$   $nprv$  ( $insert\ \varphi\ F$ )  $\psi$ 
and  $\varphi \in fmla$   $\psi \in fmla$  and  $F \subseteq fmla$  and  $finite\ F$ 
shows  $nprv\ F\ \psi$ 
using  $assms$   $nprv\_cut$   $prv\_nprvI$  by  $blast$ 

```

```

lemmas  $nprv\_addLemmaE1 = nprv\_addLemmaE[OF\ \_\ nprv\_hyp,\ simped]$ 

```

```

lemma  $nprv\_addImpLemmaI$ :
assumes  $prv$  ( $imp\ \varphi1\ \varphi2$ )
and  $F \subseteq fmla$   $finite\ F$   $\varphi1 \in fmla$   $\varphi2 \in fmla$ 
and  $nprv\ F\ \varphi1$ 
shows  $nprv\ F\ \varphi2$ 
by ( $meson\ assms\ nprv\_def\ prv\_prv\_imp\_trans\ scnj$ )

```

```

lemma  $nprv\_addImpLemmaE$ :
assumes  $prv$  ( $imp\ \varphi1\ \varphi2$ ) and  $nprv\ F\ \varphi1$  and  $nprv$  ( $(insert\ \varphi2)\ F$ )  $\psi$ 
and  $F \subseteq fmla$   $finite\ F$   $\varphi1 \in fmla$   $\varphi2 \in fmla$   $\psi \in fmla$ 
shows  $nprv\ F\ \psi$ 
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  $prv$  ( $imp\ \varphi1$ ) ( $imp\ \varphi2\ \varphi3$ )
and  $F \subseteq fmla$   $finite\ F$   $\varphi1 \in fmla$   $\varphi2 \in fmla$   $\varphi3 \in fmla$ 

```

and $nprv\ F\ \varphi_1\ nprv\ F\ \varphi_2$
shows $nprv\ F\ \varphi_3$
by (*meson* *assms* *imp* *nprv_addImpLemmaI* *nprv_mp*)

lemma *nprv_addImp2LemmaE*:
assumes $prv\ (imp\ \varphi_1\ (imp\ \varphi_2\ \varphi_3))$ **and** $nprv\ F\ \varphi_1$ **and** $nprv\ F\ \varphi_2$ **and** $nprv\ ((insert\ \varphi_3)\ F)\ \psi$
and $F \subseteq fmla\ finite\ F\ \varphi_1 \in fmla\ \varphi_2 \in fmla\ \varphi_3 \in fmla\ \psi \in fmla$
shows $nprv\ F\ \psi$
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 $prv\ (imp\ \varphi_1\ (imp\ \varphi_2\ (imp\ \varphi_3\ \varphi_4)))$
and $F \subseteq fmla\ finite\ F\ \varphi_1 \in fmla\ \varphi_2 \in fmla\ \varphi_3 \in fmla\ \varphi_4 \in fmla$
and $nprv\ F\ \varphi_1\ nprv\ F\ \varphi_2\ nprv\ F\ \varphi_3$
shows $nprv\ F\ \varphi_4$
by (*meson* *assms* *imp* *nprv_addImpLemmaI* *nprv_mp*)

lemma *nprv_addImp3LemmaE*:
assumes $prv\ (imp\ \varphi_1\ (imp\ \varphi_2\ (imp\ \varphi_3\ \varphi_4)))$ **and** $nprv\ F\ \varphi_1$ **and** $nprv\ F\ \varphi_2$ **and** $nprv\ F\ \varphi_3$
and $nprv\ ((insert\ \varphi_4)\ F)\ \psi$
and $F \subseteq fmla\ finite\ F\ \varphi_1 \in fmla\ \varphi_2 \in fmla\ \varphi_3 \in fmla\ \varphi_4 \in fmla\ \psi \in fmla$
shows $nprv\ F\ \psi$
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 $prv\ (dsj\ \varphi_1\ \varphi_2)$ **and** $nprv\ (insert\ \varphi_1\ F)\ \psi$ **and** $nprv\ ((insert\ \varphi_2)\ F)\ \psi$
and $F \subseteq fmla\ finite\ F\ \varphi_1 \in fmla\ \varphi_2 \in fmla\ \psi \in fmla$
shows $nprv\ F\ \psi$

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_eq_reflI: $F \subseteq \text{fm}la \implies \text{finite } F \implies t \in \text{trm} \implies \text{nprv } F \text{ (eq } t \ t)$

by (simp add: prv_eq_reflT prv_nprvI)

lemma nprv_eq_eqLI: $t1 = t2 \implies F \subseteq \text{fm}la \implies \text{finite } F \implies t1 \in \text{trm} \implies \text{nprv } F \text{ (eq } t1 \ t2)$

by (simp add: prv_eq_reflT prv_nprvI)

Symmetry:

lemmas nprv_eq_symI = nprv_addImpLemmaI[OF prv_eq_sym, simped, rotated 4]

lemmas nprv_eq_symE = nprv_addImpLemmaE[OF prv_eq_sym, simped, rotated 2]

lemmas nprv_eq_symE0 = nprv_eq_symE[OF nprv_hyp _, simped]

lemmas nprv_eq_symE1 = nprv_eq_symE[OF _ nprv_hyp, simped]

lemmas nprv_eq_symE01 = nprv_eq_symE[OF nprv_hyp nprv_hyp, simped]

Transitivity:

lemmas nprv_eq_transI = nprv_addImp2LemmaI[OF prv_eq_imp_trans, simped, rotated 5]

lemmas nprv_eq_transE = nprv_addImp2LemmaE[OF prv_eq_imp_trans, simped, rotated 3]

lemmas nprv_eq_transE0 = nprv_eq_transE[OF nprv_hyp _ _, simped]

lemmas nprv_eq_transE1 = nprv_eq_transE[OF _ nprv_hyp _, simped]

lemmas nprv_eq_transE2 = nprv_eq_transE[OF _ _ nprv_hyp, simped]

lemmas nprv_eq_transE01 = nprv_eq_transE[OF nprv_hyp nprv_hyp _, simped]

lemmas nprv_eq_transE02 = nprv_eq_transE[OF nprv_hyp _ nprv_hyp, simped]

lemmas nprv_eq_transE12 = nprv_eq_transE[OF _ nprv_hyp nprv_hyp, simped]

lemmas nprv_eq_transE012 = nprv_eq_transE[OF nprv_hyp nprv_hyp nprv_hyp, simped]

Substitutivity:

lemmas nprv_eq_substI =

nprv_addImp2LemmaI[OF prv_eq_subst_trm_rev, simped, rotated 6]

lemmas nprv_eq_substE = nprv_addImp2LemmaE[OF prv_eq_subst_trm_rev, simped, rotated 4]

lemmas nprv_eq_substE0 = nprv_eq_substE[OF nprv_hyp _ _, simped]

lemmas nprv_eq_substE1 = nprv_eq_substE[OF _ nprv_hyp _, simped]

lemmas nprv_eq_substE2 = nprv_eq_substE[OF _ _ nprv_hyp, simped]

lemmas nprv_eq_substE01 = nprv_eq_substE[OF nprv_hyp nprv_hyp _, simped]

lemmas nprv_eq_substE02 = nprv_eq_substE[OF nprv_hyp _ nprv_hyp, simped]

lemmas nprv_eq_substE12 = nprv_eq_substE[OF _ nprv_hyp nprv_hyp, simped]

lemmas nprv_eq_substE012 = nprv_eq_substE[OF nprv_hyp nprv_hyp nprv_hyp, simped]

4.9 Other Rules

lemma nprv_cnjH:

nprv (insert $\varphi1$ (insert $\varphi2$ F)) $\psi \implies$

$F \subseteq \text{fm}la \implies \text{finite } F \implies \varphi1 \in \text{fm}la \implies \varphi2 \in \text{fm}la \implies \psi \in \text{fm}la \implies$

nprv (insert (cnj $\varphi1$ $\varphi2$) F) ψ

apply(rule nprv_cut2[of _ $\varphi1$ $\varphi2$])

```

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

```

```

lemma prv_exi_commute:
assumes [simp]:  $x \in \text{var } y \in \text{var } \varphi \in \text{fmla}$ 
shows  $\text{prv } (\text{imp } (\text{exi } x (\text{exi } y \varphi)) (\text{exi } y (\text{exi } x \varphi)))$ 
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  $n1: \text{nprv } F (\text{subst } \varphi \ t \ x)$  and  $n2: \text{nprv } (\text{insert } \varphi \ F) (\text{eql } (\text{Var } x) \ t)$ 
and  $i[\text{simp}]: F \subseteq \text{fmla finite } F \ \varphi \in \text{fmla } t \in \text{trm } x \in \text{var } \ x \notin \text{Fvars } T \ t$ 
and  $u: x \notin (\bigcup \varphi \in F. \text{Fvars } \varphi)$ 
shows  $\text{nprv } F (\text{exu } x \ \varphi)$ 
proof-
  define  $z$  where  $z \equiv \text{getFr } [x] [t] [\varphi]$ 
  have  $z\_facts[\text{simp}]: z \in \text{var } z \neq x \ x \neq z \ z \notin \text{Fvars } T \ t \ z \notin \text{Fvars } \varphi$ 
  using  $\text{getFr\_Fvars } T \ \text{Fvars}$ [of  $[x] [t] [\varphi]$ ] unfolding  $z\_def[\text{symmetric}]$  by auto
  have  $0: \text{exu } x \ \varphi = \text{cnj } (\text{exi } x \ \varphi) (\text{exi } z (\text{all } x (\text{imp } \varphi (\text{eql } (\text{Var } x) (\text{Var } z))))))$ 
  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  $n1: \text{nprv } F (\text{subst } \varphi \ t \ x)$  and  $n2: \text{nprv } (\text{insert } (\text{subst } \varphi (\text{Var } y) \ x) \ F) (\text{eql } (\text{Var } y) \ t)$ 

```

and $i[simp]: F \subseteq fmla$ finite $F \varphi \in fmla$ $t \in trm$ $x \in var$
 $y \in var$ $y \notin FvarsT$ t **and** $u: y \notin (\bigcup \varphi \in F. Fvars \varphi)$ **and** $yx: y = x \vee y \notin Fvars \varphi$
shows $nprv F (exu x \varphi)$
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: $nprv F (exi x \varphi)$ **and** n2: $nprv (insert (subst \varphi (Var y) x) (insert \varphi F)) (eql (Var y) (Var x))$
and $i[simp]: F \subseteq fmla$ finite $F \varphi \in fmla$ $x \in var$ $y \in var$ $y \neq x$ $y \notin Fvars \varphi$
and $u: x \notin (\bigcup \varphi \in F. Fvars \varphi)$ $y \notin (\bigcup \varphi \in F. Fvars \varphi)$
shows $nprv F (exu x \varphi)$
proof–
have e: $nprv (insert \varphi F) (exu x \varphi)$
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 \varphi) (exi x \varphi))
proof–
define z **where** z: $z \equiv getFr [x] [] [\varphi]$
have z_facts[simp]: $z \in var$ $z \neq x$ $x \neq z$ $z \notin Fvars \varphi$
using getFr_FvarsT_Fvars[of [x] [] [\varphi]] **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 \varphi)
shows prv (exi x \varphi)
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: $nprv F (exu x \varphi)$ **and** n2: $nprv (insert \varphi F) \psi$
and $i[simp]: F \subseteq fmla$ finite $F \varphi \in fmla$ $x \in var$ $\psi \in fmla$ $x \notin Fvars \psi$
and $u: x \notin (\bigcup \varphi \in F. Fvars \varphi)$
shows $nprv F \psi$
using assms **apply**– **apply**(rule nprv_exiE[of _ x \varphi])
subgoal by (rule nprv_addImpLemmaI[OF prv_exu_imp_exi[of \varphi x]]) **auto**
by auto

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

```

and  $u: x \notin (\bigcup \varphi \in F. Fvars \varphi)$ 
shows  $nprv F \psi$ 
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  $prv (imp (exu x \varphi) (imp (subst \varphi t1 x) (imp (subst \varphi t2 x) (eql t1 t2))))$ 
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  $getFr\_FvarsT\_Fvars[of [x] [t1,t2] [\varphi]]$  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 $nprv F \varphi$, 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* \equiv *Variable 0*

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

abbreviation *inp* \equiv *Variable (Suc 0)*

lemma *out_inp_distinct[simp]*:
out \neq *inp* *inp* \neq *out*
out \neq *xx* *out* \neq *yy* *yy* \neq *out* *out* \neq *zz* *zz* \neq *out* *out* \neq *xx'* *xx'* \neq *out*
out \neq *yy'* *yy'* \neq *out* *out* \neq *zz'* *zz'* \neq *out*
inp \neq *xx* *inp* \neq *yy* *yy* \neq *inp* *inp* \neq *zz* *zz* \neq *inp* *inp* \neq *xx'* *xx'* \neq *inp*
inp \neq *yy'* *yy'* \neq *inp* *inp* \neq *zz'* *zz'* \neq *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* \Rightarrow '*fmla set* **where**
ptrm n \equiv $\{\sigma \in \text{fmla} . \text{Fvars } \sigma = \text{Variable } \{0..n\} \wedge \text{prv } (\text{exu } \text{out } \sigma)\}$

lemma *ptrm[intro,simp]*: $\sigma \in \text{ptrm } n \implies \sigma \in \text{fmla}$
unfolding *ptrm_def* **by** *auto*

lemma *ptrm_1_Fvars[simp]*: $\sigma \in \text{ptrm } (\text{Suc } 0) \implies \text{Fvars } \sigma = \{\text{out}, \text{inp}\}$
unfolding *ptrm_def* **by** *auto*

lemma *ptrm_prv_exu*: $\sigma \in \text{ptrm } n \implies \text{prv } (\text{exu out } \sigma)$
unfolding *ptrm_def* **by** *auto*

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

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

lemma *nprv_ptrmE_uni*:
 $\sigma \in \text{ptrm } n \implies \text{nprv } F (\text{subst } \sigma t1 \text{ out}) \implies \text{nprv } F (\text{subst } \sigma t2 \text{ out}) \implies$
 $\text{nprv } (\text{insert } (\text{eql } t1 t2) F) \psi \implies$
 $F \subseteq \text{fmla} \implies \text{finite } F \implies \psi \in \text{fmla} \implies t1 \in \text{trm} \implies t2 \in \text{trm}$
 $\implies \text{nprv } F \psi$
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*:
 $\sigma \in \text{ptrm } n \implies \text{nprv } F \sigma \implies \text{nprv } F (\text{subst } \sigma t \text{ out}) \implies$
 $\text{nprv } (\text{insert } (\text{eql } (\text{Var out}) t) F) \psi \implies$
 $F \subseteq \text{fmla} \implies \text{finite } F \implies \psi \in \text{fmla} \implies t \in \text{trm}$
 $\implies \text{nprv } F \psi$
by (*rule nprv_ptrmE_uni[of σ _ _ Var out t]*) *auto*

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

5.2 The \forall - \exists Equivalence

There are two natural ways to state that (unique) "output" of a pseudo-term σ satisfies a property φ : (1) using \exists : there exists an "out" such that σ and φ hold for it; (2) using \forall : 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 $\sigma: \sigma \in \text{ptrm } n$ **and** [*simp*]: $\varphi \in \text{fmla}$
shows $\text{nprv } \{\sigma, \text{exi out } (\text{cnj } \sigma \varphi)\} \varphi$
proof–
have [*simp*]: $\sigma \in \text{fmla}$ **using** σ **by** *simp*
define z **where** $z \equiv \text{getFr } [\text{out}] [] [\sigma, \varphi]$
have $z_facts[\text{simp}]$: $z \in \text{var } z \neq \text{out } z \notin \text{Fvars } \sigma \ z \notin \text{Fvars } \varphi$
using *getFr_FvarsT_Fvars*[*of* [*out*] [] [σ, φ]] **unfolding** $z_def[\text{symmetric}]$ **by** *auto*
have 0 : $\text{exi out } (\text{cnj } \sigma \varphi) = \text{exi } z (\text{subst } (\text{cnj } \sigma \varphi) (\text{Var } z) \text{ out})$
by(*rule exi_rename, auto*)
show *?thesis*
unfolding 0

```

apply(nrule r: nprv_exiE0[of z subst (cnj  $\sigma$   $\varphi$ ) (Var z) out])
apply(nrule2 r: nprv_ptrmE0_uni0[OF  $\sigma$ , 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  $\_$   $\varphi$  out  $\varphi$ ]) . .
qed

```

```

lemma ptrm_nprv_exi_all:
  assumes  $\sigma$ :  $\sigma \in ptrm\ n$  and [simp]:  $\varphi \in fmla$ 
  shows nprv {exi out (cnj  $\sigma$   $\varphi$ )} (all out (imp  $\sigma$   $\varphi$ ))
proof-
  have [simp]:  $\sigma \in fmla$  using  $\sigma$  by simp
  show ?thesis
  apply(nrule r: nprv_allI)
  apply(nrule r: nprv_impI)
  apply(nrule r: ptrm_nprv_exi[OF  $\sigma$ ]) .
qed

```

```

lemma ptrm_prv_exi_imp_all:
  assumes  $\sigma$ :  $\sigma \in ptrm\ n$  and [simp]:  $\varphi \in fmla$ 
  shows prv (imp (exi out (cnj  $\sigma$   $\varphi$ )) (all out (imp  $\sigma$   $\varphi$ )))
proof-
  have [simp]:  $\sigma \in fmla$  using  $\sigma$  by simp
  show ?thesis
  apply(nrule r: nprv_prvI)
  apply(nrule r: nprv_impI)
  apply(nrule r: ptrm_nprv_exi_all[OF  $\sigma$ ]) .
qed

```

```

lemma ptrm_nprv_all_imp_exi:
  assumes  $\sigma$ :  $\sigma \in ptrm\ n$  and [simp]:  $\varphi \in fmla$ 
  shows nprv {all out (imp  $\sigma$   $\varphi$ )} (exi out (cnj  $\sigma$   $\varphi$ ))
proof-
  have [simp]:  $\sigma \in fmla$  using  $\sigma$  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] [] [ $\sigma, \varphi$ ]] unfolding z_def[symmetric] by auto
  show ?thesis
  apply(nrule r: nprv_ptrmE_exi[OF  $\sigma$ ])
  apply(nrule r: nprv_exiI[of  $\_$  cnj  $\sigma$   $\varphi$  Var out out])
  apply(nrule r: nprv_allE0[of out imp  $\sigma$   $\varphi$   $\_$  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  $\sigma$ :  $\sigma \in ptrm\ n$  and [simp]:  $\varphi \in fmla$ 
  shows prv (imp (all out (imp  $\sigma$   $\varphi$ )) (exi out (cnj  $\sigma$   $\varphi$ )))
proof-
  have [simp]:  $\sigma \in fmla$  using  $\sigma$  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
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  $\varphi$  t  $\equiv$  subst  $\varphi$  t inp

```

```

lemma instInp_fmla[simp,intro]:
assumes  $\varphi \in$  fmla and t  $\in$  trm
shows instInp  $\varphi$  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  $\varphi$  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  $\tau: \tau \in$  ptrm (Suc 0) and t  $\in$  trm
shows  $Fvars$  (instInp  $\tau$  t) = insert out ( $FvarsT$  t)
using assms instInp_def by auto

```

```

lemma instInp:
assumes  $\tau: \tau \in$  ptrm (Suc 0) and [simp]: t  $\in$  trm
and [simp]:  $FvarsT$  t = Variable '{(Suc 0)..n}
shows instInp  $\tau$  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  $\tau$  t) = insert out ( $FvarsT$  t)
using  $\tau$  by (subst  $Fvars$  instInp_ptrm_1) auto
have 0: exu out (instInp  $\tau$  t) = subst (exu out  $\tau$ ) t inp

```

```

unfolding instInp_def by (subst subst_exu) auto
have 1: prv (exu out  $\tau$ ) using  $\tau$  unfolding ptrm_def by auto
have prv (exu out (instInp  $\tau$  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  $\tau: \tau \in \text{ptrm } (Suc\ 0)$  and  $t \in \text{trm}$  and  $FvarsT\ t = \{\}$ 
shows instInp  $\tau$  t  $\in$  ptrm 0
using assms by (intro instInp) auto

```

```

lemma instInp_1:
assumes  $\tau: \tau \in \text{ptrm } (Suc\ 0)$  and  $t \in \text{trm}$  and  $FvarsT\ t = \{inp\}$ 
shows instInp  $\tau$  t  $\in$  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 $instInpP\ \varphi\ n\ \tau$ where n is the number of input variables of τ . In the name $instInpP$, Inp is as before a reminder that we instantiate φ on its variable "inp" and the suffix "P" stands for "Pseudo".

```

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

```

```

lemma instInpP_fmla[simp, intro]:
assumes  $\varphi \in \text{fmla}$  and  $\tau \in \text{fmla}$ 
shows instInpP  $\varphi$  n  $\tau \in \text{fmla}$ 
using assms unfolding instInpP_def by (auto simp: Let_def)

```

```

lemma Fvars_instInpP[simp]:
assumes  $\varphi \in \text{fmla}$  and  $\tau: \tau \in \text{ptrm } n$  Variable (Suc (Suc n))  $\notin$  Fvars  $\varphi$ 
shows Fvars (instInpP  $\varphi$  n  $\tau$ ) = Fvars  $\varphi$  - {inp}  $\cup$  Variable ' {(Suc 0)..n}
using assms unfolding instInpP_def Let_def ptrm_def by auto

```

```

lemma Fvars_instInpP2[simp]:
assumes  $\varphi \in \text{fmla}$  and  $\tau: \tau \in \text{ptrm } n$  and  $Fvars\ \varphi \subseteq \{inp\}$ 
shows Fvars (instInpP  $\varphi$  n  $\tau$ ) = Fvars  $\varphi$  - {inp}  $\cup$  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  $\sigma: \sigma \in \text{ptrm } (Suc\ 0)$  and  $\tau: \tau \in \text{ptrm } n$ 
shows instInpP  $\sigma$  n  $\tau \in \text{ptrm } n$ 
proof-
  note Let_def[simp]
  have [simp]:  $\sigma \in \text{fmla}$   $\tau \in \text{fmla}$   $Fvars\ \sigma = \{out, inp\}$ 
     $Fvars\ \tau = \text{Variable ' } \{0..n\}$ 
    using assms unfolding ptrm_def by auto
  define zz where zz  $\equiv$  Variable (Suc (Suc n))
  have zz_facts[simp]:  $zz \in \text{var} \wedge i. i \leq n \implies \text{Variable } i \neq zz \wedge zz \neq \text{Variable } i$ 
     $out \neq zz \wedge out \neq inp \wedge zz \neq inp$ 

```

unfolding *zz_def* **by** *auto*

define *x* **where** $x \equiv \text{getFr } [out, inp, zz] [] [\sigma, \tau]$
have *x_facts*[*simp*]: $x \in \text{var } x \neq out \ x \neq inp$
 $x \neq zz \ z \neq x \ x \notin Fvars \ \sigma \ x \notin Fvars \ \tau$
using *getFr_FvarsT_Fvars*[of [out,inp,zz] [] [\sigma,\tau]] **unfolding** *x_def*[*symmetric*] **by** *auto*
have [*simp*]: $x \neq \text{Variable } (Suc \ (Suc \ n))$
using *x_facts*(4) *zz_def* **by** *auto*
define *z* **where** $z \equiv \text{getFr } [out, inp, zz, x] [] [\sigma, \tau]$
have *z_facts*[*simp*]: $z \in \text{var } z \neq out \ z \neq inp \ z \neq x \ z \neq zz \ z \notin Fvars \ \sigma \ z \notin Fvars \ \tau$
using *getFr_FvarsT_Fvars*[of [out,inp,zz,x] [] [\sigma,\tau]] **unfolding** *z_def*[*symmetric*] **by** *auto*

have [*simp*]: $\bigwedge i. z = \text{Variable } i \implies \neg i \leq n$
and [*simp*]: $\bigwedge i. x = \text{Variable } i \implies \neg i \leq n$
using $\langle Fvars \ \tau = \text{Variable } \{0..n\} \rangle$ *atLeastAtMost_iff* *z_facts*(7) *x_facts*(7)
by *blast+*

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

have *exi_σ*: $prv \ (exi \ out \ \sigma)$ **and** *exi_τ*: $prv \ (exi \ zz \ (\text{subst } \tau \ (\text{Var } zz) \ out))$
using $\sigma \ \tau$ *ptrm_prv_exi* *tt* **by** *fastforce+*
have *exi_σ*: $prv \ (exi \ out \ (\text{subst } \sigma \ (\text{Var } zz) \ inp))$
using *prv_subst*[OF _ _ _ *exi_σ*, of *inp* *Var* *zz*] **by** *auto*

have *exu_σ*: $prv \ (exu \ out \ \sigma)$
using σ *ptrm_prv_exu* **by** *blast*
have *exu_σ*: $prv \ (exu \ out \ (\text{subst } \sigma \ (\text{Var } zz) \ inp))$
using *prv_subst*[OF _ _ _ *exu_σ*, of *inp* *Var* *zz*] **by** *auto*

have *zz_z*: $exi \ zz \ (\text{cnj} \ (\text{subst } \tau \ (\text{Var } zz) \ out) \ (\text{subst } \sigma \ (\text{Var } zz) \ inp)) =$
 $exi \ z \ (\text{cnj} \ (\text{subst } \tau \ (\text{Var } z) \ out) \ (\text{subst } \sigma \ (\text{Var } z) \ inp))$
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_τ*])
apply(*nrule* *r*: *nprv_exiE*[of _ *zz* *subst* τ (*Var* *zz*) *out*])
apply(*nrule* *r*: *nprv_clear2_2*)
apply(*nrule* *r*: *nprv_exiI*[of _ _ *Var* *zz*])
apply(*nrule* *r*: *nprv_addLemmaE*[OF *exi_σ*])
apply(*nrule* *r*: *nprv_exiE*[of _ *out* *subst* σ (*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  $\tau$ , 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  $\sigma$  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  $\sigma$ ])
    apply(nrule r: nprv_exuE_uni[of _ out subst  $\sigma$  (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  $\varphi \in \text{fmla}$  Fvars  $\varphi \subseteq \{\text{inp}\}$  and  $\tau \in \text{ptrm}$  (Suc 0)
and  $t \in \text{trm}$  and FvarsT  $t = \{\}$ 
shows instInp (instInpP  $\varphi$  (Suc 0)  $\tau$ )  $t = \text{instInpP}$   $\varphi$  0 (instInp  $\tau$   $t$ )
proof-
  define x1 and x2 where
    x12:  $x1 \equiv \text{Variable}$  (Suc (Suc 0))
    x2  $\equiv \text{Variable}$  (Suc (Suc (Suc 0)))
  have x_facts[simp]:  $x1 \in \text{var}$   $x2 \in \text{var}$   $x1 \neq \text{inp}$   $x2 \neq \text{inp}$ 
     $x1 \neq \text{out}$   $\text{out} \neq x1$   $x2 \neq \text{out}$   $\text{out} \neq x2$   $x1 \neq x2$   $x2 \neq x1$ 
  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  $\varphi$  _ x1]
) .
qed

```

Pseudo-term instantiation also composes smoothly with itself:

```

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

```

define $\chi\sigma$ **and** $\sigma\tau$ **where** $\chi\sigma_def: \chi\sigma \equiv instInpP \chi (Suc\ 0) \sigma$ **and** $\sigma\tau_def: \sigma\tau \equiv instInpP \sigma\ 0 \tau$

have $[simp]: \sigma \in fmla\ Fvars\ \sigma = \{out, inp\}$ $\tau \in fmla\ Fvars\ \tau = \{out\}$

using $\sigma\ \tau$ **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 $z_facts[simp]: z \in var$

$out \neq z \wedge z \neq out\ inp \neq z \wedge z \neq inp\ z \notin Fvars\ \chi\ z \notin Fvars\ \sigma\ z \notin Fvars\ \tau$

unfolding z_def **by** $auto$

define zz **where** $zz \equiv Variable\ (Suc\ (Suc\ (Suc\ 0)))$

have $zz_facts[simp]: zz \in var$

$out \neq zz \wedge zz \neq out\ inp \neq zz \wedge zz \neq inp\ z \neq zz \wedge zz \neq z$

$zz \notin Fvars\ \chi\ zz \notin Fvars\ \sigma\ zz \notin Fvars\ \tau$

unfolding $zz_def\ z_def$ **by** $auto$

define z' **where** $z' \equiv getFr\ [out, inp, z, zz] []\ [\chi, \sigma, \tau]$

have $z'_facts[simp]: z' \in var\ z' \neq out\ z' \neq inp\ z' \neq z\ z \neq z'\ z' \neq zz\ zz \neq z'$

$z' \notin Fvars\ \chi\ z' \notin Fvars\ \sigma\ z' \notin Fvars\ \tau$

using $getFr_FvarsT_Fvars[of\ [out, inp, z, zz] []\ [\chi, \sigma, \tau]]$ **unfolding** $z'_def[symmetric]$ **by** $auto$

have $\chi\sigma': instInpP\ \chi\sigma\ 0\ \tau = exi\ z'\ (cnj\ (subst\ \tau\ (Var\ z')\ out)\ (subst\ \chi\sigma\ (Var\ z')\ inp))$

unfolding $instInpP_def\ Let_def\ z_def[symmetric]$

by $(auto\ simp: exi_rename[of_ z\ z'])$

have $\chi\sigma z': subst\ \chi\sigma\ (Var\ z')\ inp =$

$exi\ zz\ (cnj\ (subst\ (subst\ \sigma\ (Var\ zz)\ out)\ (Var\ z')\ inp)\ (subst\ \chi\ (Var\ zz)\ inp))$

unfolding $\chi\sigma_def\ instInpP_def\ Let_def\ zz_def[symmetric]$

by $(auto\ simp: subst_compose_same)$

have $\sigma\tau zz: subst\ \sigma\tau\ (Var\ zz)\ out =$

$exi\ z\ (cnj\ (subst\ \tau\ (Var\ z)\ out)\ (subst\ (subst\ \sigma\ (Var\ zz)\ out)\ (Var\ z)\ inp))$

unfolding $\sigma\tau_def\ instInpP_def\ Let_def\ z_def[symmetric]$

by $(variousSubsts2\ auto\ s1: subst_compose_same\ s2: subst_compose_diff)$

have $nprv\ \{instInpP\ \chi\sigma\ 0\ \tau\}\ (instInpP\ \chi\ 0\ \sigma\tau)$

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 $\chi\sigma\tau: instInpP\ \chi\ 0\ \sigma\tau = exi\ z'\ (cnj\ (subst\ \sigma\tau\ (Var\ z')\ out)\ (subst\ \chi\ (Var\ z')\ inp))$

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  $\tau$  (Var  $z$ ) out) (subst (subst  $\sigma$  (Var  $z$ ) inp) (Var  $z'$ ) out))
unfolding  $\sigma\tau\_def$  instInpP\_def Let\_def  $z\_def$ [symmetric]
by (auto simp: subst\_compose\_same)

have  $\chi\sigma z$ : subst  $\chi\sigma$  (Var  $z$ ) inp =
  exi  $zz$  (cnj (subst (subst  $\sigma$  (Var  $z$ ) inp) (Var  $zz$ ) out) (subst  $\chi$  (Var  $zz$ ) inp))
unfolding  $\chi\sigma\_def$  instInpP\_def Let\_def  $zz\_def$ [symmetric]
by (variousSubsts2 auto s1: subst\_compose\_same s2: subst\_compose\_diff)

have nprv {instInpP  $\chi$  0  $\sigma\tau$ } (instInpP  $\chi\sigma$  0  $\tau$ )
unfolding  $\chi\sigma\tau$ 
apply(nrule  $r$ : nprv\_exiE0)
apply(nrule  $r$ : nprv\_clear2\_2)
apply(nrule  $r$ : nprv\_cnjE0)
apply(nrule  $r$ : nprv\_clear3\_3)
unfolding  $\sigma\tau z'$ 
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  $\chi\sigma z$ 
apply(nrule  $r$ : nprv\_exiI[of  $\_ \_$  Var  $z'$ ])
apply(nrule  $r$ : nprv\_cnjI) .
thus ?B unfolding  $\chi\sigma\_def$   $\sigma\tau\_def$  .
qed

```

```

lemma prv_instInpP_compose:
assumes [simp]:  $\chi \in fmla$  Fvars  $\chi = \{inp\}$ 
and  $\sigma$ [simp]:  $\sigma \in ptrm$  (Suc 0) and  $\tau$ [simp]:  $\tau \in ptrm$  0
shows prv (imp (instInpP (instInpP  $\chi$  (Suc 0)  $\sigma$ ) 0  $\tau$ )
  (instInpP  $\chi$  0 (instInpP  $\sigma$  0  $\tau$ ))) (is ?A)
and
  prv (imp (instInpP  $\chi$  0 (instInpP  $\sigma$  0  $\tau$ ))
    (instInpP (instInpP  $\chi$  (Suc 0)  $\sigma$ ) 0  $\tau$ )) (is ?B)
and
  prv (equiv (instInpP (instInpP  $\chi$  (Suc 0)  $\sigma$ ) 0  $\tau$ )
    (instInpP  $\chi$  0 (instInpP  $\sigma$  0  $\tau$ ))) (is ?C)
proof-
  have [simp]:  $\sigma \in fmla$  Fvars  $\sigma = \{out,inp\}$   $\tau \in fmla$  Fvars  $\tau = \{out\}$ 
  using  $\sigma$   $\tau$  unfolding ptrm\_def by auto
  show ?A ?B by (intro nprv\_prvI nprv\_impI nprv_instInpP_compose, auto)+
  thus ?C by (intro prv\_equivI) 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* \Rightarrow '*trm* \Rightarrow *bool* **where**
prveqlPT τ $t \equiv prv$ (*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** $\tau: \tau \in ptrm \ 0$
and $[simp]: t \in trm \ FvarsT \ t = \{\}$
and $\tau t: prveqlPT \ \tau \ t$
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* = $nprv_addLemmaE[OF \ \tau t[unfolding \ prveqlPT_def]]$

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* $r: nprv_ptrmE_uni[OF \ \tau, \ of \ _ \ t \ Var \ zz]$)
apply(*nrule* $r: nprv_clear4_2$)
apply(*nrule* $r: nprv_clear3_3$)
apply(*nrule* $r: nprv_eqL_substE012[of \ t \ Var \ zz \ _ \ \varphi \ inp]$) .
qed

lemma *prveqlPT_prv_instInp_instInpP*:
assumes $\varphi \in fmla$ **and** $f: Fvars \ \varphi \subseteq \{inp\}$ **and** $\tau: \tau \in ptrm \ 0$
and $t \in trm \ FvarsT \ t = \{\}$
and $\tau t: prveqlPT \ \tau \ t$
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** $\tau: \tau \in ptrm \ 0$
and $[simp]: t \in trm \ FvarsT \ t = \{\}$
and $\tau t: prveqlPT \ \tau \ t$
shows $nprv \ \{instInpP \ \varphi \ t\} \ (instInpP \ \varphi \ 0 \ \tau)$
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* = $nprv_addLemmaE[OF \ \tau t[unfolding \ prveqlPT_def]]$

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** $\tau: \tau \in ptrm \ 0$
and $t \in trm \ FvarsT \ t = \{\}$
and $\tau t: prveqlPT \ \tau \ t$
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  $\varphi \in \text{fm}la$  and  $f: \text{Fvars } \varphi \subseteq \{inp\}$  and  $\tau: \tau \in \text{ptrm } 0$ 
and  $t \in \text{trm } \text{Fvars}T \ t = \{\}$ 
and  $\tau t: \text{prveqlPT } \tau \ t$ 
shows  $\text{prv } (\text{eqv } (\text{instInpP } \varphi \ 0 \ \tau) (\text{instInp } \varphi \ t))$ 
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  $\Rightarrow$  bool
assumes
  not_isTrue_fl:  $\neg$  isTrue fl
and
isTrue_imp:
 $\bigwedge \varphi \psi. \varphi \in \text{fmla} \Rightarrow \psi \in \text{fmla} \Rightarrow \text{Fvars } \varphi = \{\} \Rightarrow \text{Fvars } \psi = \{\} \Rightarrow$ 
 $\text{isTrue } \varphi \Rightarrow \text{isTrue } (\text{imp } \varphi \psi) \Rightarrow \text{isTrue } \psi$ 
and
isTrue_all:
 $\bigwedge x \varphi. x \in \text{var} \Rightarrow \varphi \in \text{fmla} \Rightarrow \text{Fvars } \varphi = \{x\} \Rightarrow$ 
 $(\forall n \in \text{num}. \text{isTrue } (\text{subst } \varphi n x)) \Rightarrow \text{isTrue } (\text{all } x \varphi)$ 
and
isTrue_exi:
 $\bigwedge x \varphi. x \in \text{var} \Rightarrow \varphi \in \text{fmla} \Rightarrow \text{Fvars } \varphi = \{x\} \Rightarrow$ 
 $\text{isTrue } (\text{exi } x \varphi) \Rightarrow (\exists n \in \text{num}. \text{isTrue } (\text{subst } \varphi n x))$ 
and
isTrue_neg:
 $\bigwedge \varphi. \varphi \in \text{fmla} \Rightarrow \text{Fvars } \varphi = \{\} \Rightarrow$ 
 $\text{isTrue } \varphi \vee \text{isTrue } (\text{neg } \varphi)$ 
begin

lemma isTrue_neg_excl:
```

```

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

```

```

lemma isTrue_neg_neg:
assumes  $\varphi \in \text{fmla}$   $\text{Fvars } \varphi = \{\}$ 
and  $\text{isTrue } (\text{neg } (\text{neg } \varphi))$ 
shows  $\text{isTrue } \varphi$ 
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:  $\bigwedge \varphi. \varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{prv } \varphi \implies \text{isTrue } \varphi$ 
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:
 $\varphi \in \text{fmla} \implies \text{Fvars } \varphi = \{\} \implies \text{prv } \varphi \implies \text{prv } (\text{neg } \varphi) \implies \text{False}$ 
using isTrue_neg_excl[of \varphi] sound_isTrue by auto

```

```

lemma prv_imp_implies_isTrue:
assumes [simp]:  $\varphi \in \text{fmla}$   $\chi \in \text{fmla}$   $\text{Fvars } \varphi = \{\}$   $\text{Fvars } \chi = \{\}$ 
and  $p$ :  $\text{prv } (\text{imp } \varphi \chi)$  and  $i$ :  $\text{isTrue } \varphi$ 
shows  $\text{isTrue } \chi$ 
proof—
  have  $\text{isTrue } (\text{imp } \varphi \chi)$  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 *ω consistent: ω consistent*
unfolding *ω consistent_def* **proof** (*safe del: notI*)
fix φ x **assume** *0[simp,intro]: $\varphi \in \text{fmLa}$ $x \in \text{var}$* **and** *1: Fvars $\varphi = \{x\}$*
and *00: $\forall n \in \text{num. prv}(\text{neg}(\text{subst } \varphi n x))$*
hence *$\forall n \in \text{num. isTrue}(\text{neg}(\text{subst } \varphi n x))$*
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 *\neg isTrue (neg (neg (exi x φ)))*
using *1 isTrue_neg_excl* **by** *force*
thus *\neg prv (neg (neg (exi x φ)))*
using *1 sound_isTrue* **by** *auto*
qed

lemma *ω consistentStd1: ω consistentStd1*
using *ω consistent ω consistent_impliesStd1* **by** *blast*

lemma *ω consistentStd2: ω consistentStd2*
using *ω consistent ω consistent_impliesStd2* **by** *blast*

end — context *Minimal_Truth_Soundness*

Chapter 7

Arithmetic Constructs

Less generic 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  $\Rightarrow$  'trm  
and  
pls :: 'trm  $\Rightarrow$  'trm  $\Rightarrow$  'trm  
and  
tms :: 'trm  $\Rightarrow$  'trm  $\Rightarrow$  'trm  
assumes  
Fvars_zero[simp,intro!]: FvarsT zer = {}  
and  
substT_zer[simp]:  $\bigwedge t x. t \in trm \implies x \in var \implies$   
  substT zer t x = zer  
and  
suc[simp]:  $\bigwedge t. t \in trm \implies suc t \in trm$   
and  
FvarsT_suc[simp]:  $\bigwedge t. t \in trm \implies$   
  FvarsT (suc t) = FvarsT t  
and  
substT_suc[simp]:  $\bigwedge t1 t x. t1 \in trm \implies t \in trm \implies x \in var \implies$ 
```

```

  substT (suc t1) t x = suc (substT t1 t x)
and
pls[simp]:  $\bigwedge t1 t2. t1 \in trm \implies t2 \in trm \implies pls\ t1\ t2 \in trm$ 
and
Fvars_pls[simp]:  $\bigwedge t1 t2. t1 \in trm \implies t2 \in trm \implies$ 
  FvarsT (pls t1 t2) = FvarsT t1  $\cup$  FvarsT t2
and
substT_pls[simp]:  $\bigwedge t1 t2 t x. t1 \in trm \implies t2 \in trm \implies t \in trm \implies x \in var \implies$ 
  substT (pls t1 t2) t x = pls (substT t1 t x) (substT t2 t x)
and
tms[simp]:  $\bigwedge t1 t2. t1 \in trm \implies t2 \in trm \implies tms\ t1\ t2 \in trm$ 
and
Fvars_tms[simp]:  $\bigwedge t1 t2. t1 \in trm \implies t2 \in trm \implies$ 
  FvarsT (tms t1 t2) = FvarsT t1  $\cup$  FvarsT t2
and
substT_tms[simp]:  $\bigwedge t1 t2 t x. t1 \in trm \implies t2 \in trm \implies t \in trm \implies x \in var \implies$ 
  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  $\Rightarrow$  '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  $\in$  num
  using num_Num by auto

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

lemma substT_Num[simp]:  $x \in var \implies t \in trm \implies substT\ (Num\ n)\ t\ x = Num\ n$ 
  by auto

```

lemma *zer*[*simp,intro*]: $zer \in num$
and *suc_num*[*simp*]: $\bigwedge n. n \in num \implies suc\ n \in num$
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*]: $n \in num \implies n \in atrm$
atrm_Var[*simp,intro*]: $x \in var \implies Var\ x \in atrm$
atrm_suc[*simp,intro*]: $t \in atrm \implies suc\ t \in atrm$
atrm_pls[*simp,intro*]: $t \in atrm \implies t' \in atrm \implies pls\ t\ t' \in atrm$
atrm_tms[*simp,intro*]: $t \in atrm \implies t' \in atrm \implies tms\ t\ t' \in atrm$

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 $s \in \text{atrm } x \in \text{var}$
shows $\text{substT } s \text{ (Var } x) x = s$
using *assms by induct auto*

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

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

lemma *substT_compose_same*:
 $\bigwedge s \ t1 \ t2 \ x. s \in \text{atrm} \implies t1 \in \text{trm} \implies t2 \in \text{trm} \implies x \in \text{var} \implies$
 $\text{substT } (\text{substT } s \ t1 \ x) \ t2 \ x = \text{substT } s \ (\text{substT } t1 \ t2 \ x) \ x$
using *substT_compose_eq_or by blast*

lemma *substT_substT*[*simp*]:
assumes $s[\text{simp}]: s \in \text{atrm}$ **and** $t[\text{simp}]: t \in \text{trm}$ **and** $x[\text{simp}]: x \in \text{var}$ **and** $y[\text{simp}]: y \in \text{var}$
assumes $yy: x \neq y \ y \notin \text{FvarsT } s$
shows $\text{substT } (\text{substT } s \ \text{(Var } y) \ x) \ t \ y = \text{substT } s \ t \ x$
using *substT_compose_eq_or*[*OF s _ t x y, of Var y*] **using** *subst_notIn yy by simp*

lemma *substT_comp*:
 $\bigwedge x \ y \ s \ t. s \in \text{atrm} \implies t \in \text{trm} \implies x \in \text{var} \implies y \in \text{var} \implies$
 $x \neq y \implies y \notin \text{FvarsT } t \implies$
 $\text{substT } (\text{substT } s \ \text{(Var } x) \ y) \ t \ x = \text{substT } (\text{substT } s \ t \ x) \ t \ y$
by (*simp add: substT_compose_diff*)

Now the corresponding development of parallel substitution for arithmetic terms:

lemma *rawpsubstT_atrm*[*simp,intro*]:
assumes $r \in \text{atrm}$ **and** $\text{snd } ' \text{ (set } \text{txs}) \subseteq \text{var}$ **and** $\text{fst } ' \text{ (set } \text{txs}) \subseteq \text{atrm}$
shows $\text{rawpsubstT } r \ \text{txs} \in \text{atrm}$
using *assms by (induct txs arbitrary: r) auto*

lemma *psubstT_atrm*[*simp,intro*]:
assumes $r \in \text{atrm}$ **and** $\text{snd } ' \text{ (set } \text{txs}) \subseteq \text{var}$ **and** $\text{fst } ' \text{ (set } \text{txs}) \subseteq \text{atrm}$
shows $\text{psubstT } r \ \text{txs} \in \text{atrm}$

proof–

have $\text{txs_trm}: \text{fst } ' \text{ (set } \text{txs}) \subseteq \text{trm}$ **using** *assms atrm_trm by auto*
define us **where** $us \equiv \text{getFrN } (\text{map } \text{snd } \text{txs}) \ (r \ \# \ \text{map } \text{fst } \text{txs}) \ [] \ (\text{length } \text{txs})$
have $us_facts: \text{set } us \subseteq \text{var}$
 $\text{set } us \cap \text{FvarsT } r = \{\}$
 $\text{set } us \cap \bigcup (\text{FvarsT } ' \text{ (fst } ' \text{ (set } \text{txs}))) = \{\}$
 $\text{set } us \cap \text{snd } ' \text{ (set } \text{txs}) = \{\}$
 $\text{length } us = \text{length } \text{txs}$
 $\text{distinct } us$
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  $r \in \text{atrm}$  and  $\text{snd}'(\text{set txs}) \subseteq \text{var}$  and  $\text{fst}'(\text{set txs}) \subseteq \text{atrm}$ 
shows  $FvarsT(\text{rawpsubstT } r \text{ txs}) \subseteq$ 
     $(FvarsT r - \text{snd}'(\text{set txs})) \cup (\bigcup \{FvarsT t \mid t x. (t,x) \in \text{set txs}\})$ 
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 \in trm and x: x \in var using Cons.prem.s unfolding tx by auto
define \chi where \chi \equiv substT r t x
have 0: FvarsT \chi = FvarsT r - \{x\} \cup (if x \in FvarsT r then FvarsT t else \{\})
using Cons.prem.s unfolding \chi_def by (auto simp: tx t FvarsT_substT)
have \chi: \chi \in trm \chi \in atrm unfolding \chi_def using Cons.prem.s t x by (auto simp add: tx)
have FvarsT (rawpsubstT \chi txs) \subseteq
     $(FvarsT \chi - \text{snd}'(\text{set txs})) \cup$ 
     $(\bigcup \{FvarsT t \mid t x. (t,x) \in \text{set txs}\})$ 
using Cons.prem.s \chi by (intro Cons.IH) auto
also have ... \subseteq FvarsT r - insert x (snd' set txs) \cup \bigcup \{FvarsT ta \mid ta. \exists xa. ta = t \wedge xa = x \vee (ta,
 $xa) \in \text{set txs}\}$ 
(is _ \subseteq ?R) by (auto simp: 0 tx Cons.prem.s)
finally have 1: FvarsT (rawpsubstT \chi txs) \subseteq ?R .
have 2: FvarsT \chi = FvarsT r - \{x\} \cup (if x \in FvarsT r then FvarsT t else \{\})
using Cons.prem.s t x unfolding \chi_def using FvarsT_substT by auto
show ?case using 1 by (simp add: tx \chi_def[symmetric] 2)
qed auto

lemma in_FvarsT_rawpsubstT_imp:
assumes  $y \in FvarsT(\text{rawpsubstT } r \text{ txs})$ 
and  $r \in \text{atrm}$  and  $\text{snd}'(\text{set txs}) \subseteq \text{var}$  and  $\text{fst}'(\text{set txs}) \subseteq \text{atrm}$ 
shows  $(y \in FvarsT r - \text{snd}'(\text{set txs})) \vee$ 
     $(y \in \bigcup \{FvarsT t \mid t x. (t,x) \in \text{set txs}\})$ 
using Fvars_rawpsubst_su[OF assms(2-4)]
using assms(1) by blast

lemma FvarsT_rawpsubstT:
assumes  $r \in \text{atrm}$  and  $\text{snd}'(\text{set txs}) \subseteq \text{var}$  and  $\text{fst}'(\text{set txs}) \subseteq \text{atrm}$ 
and distinct (map snd txs) and  $\forall x \in \text{snd}'(\text{set txs}). \forall t \in \text{fst}'(\text{set txs}). x \notin FvarsT t$ 
shows  $FvarsT(\text{rawpsubstT } r \text{ txs}) =$ 
     $(FvarsT r - \text{snd}'(\text{set txs})) \cup$ 
     $(\bigcup \{\text{if } x \in FvarsT r \text{ then } FvarsT t \text{ else } \{\} \mid t x. (t,x) \in \text{set txs}\})$ 
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 \in trm and x: x \in var using Cons.prem.s unfolding a by auto
have tx: x \notin FvarsT t \wedge \text{snd}' set txs \cap FvarsT t = \{\} using Cons.prem.s unfolding a by auto
hence 0: FvarsT r - \{x\} \cup FvarsT t - \text{snd}' set txs = FvarsT r - insert x (snd' set txs) \cup FvarsT t
by auto
have x_txs: \bigwedge ta xa. (ta, xa) \in \text{set txs} \implies x \neq xa using (distinct (map snd (a \# txs)))

```


unfolding a by (*auto simp: rev_image_eqI*)

define χ **where** $\chi_def: \chi \equiv substT r t x$

have $\chi: \chi \in trm \chi \in atrm$ **unfolding** χ_def **using** *Cons.prem*s $t x$ **by** (*auto simp: a*)

have 1: $FvarsT (rawpsubstT \chi txs) =$

$(FvarsT \chi - snd \text{' (set txs)}) \cup$
 $(\bigcup \{if x \in FvarsT \chi then FvarsT t else \{\} \mid t x . (t,x) \in set txs\})$

using *Cons.prem*s χ **by** (*intro Cons.IH*) *auto*

have 2: $FvarsT \chi = FvarsT r - \{x\} \cup (if x \in FvarsT r then FvarsT t else \{\})$

using *Cons.prem*s $t x$ **unfolding** χ_def **using** $FvarsT_substT$ **by** *auto*

define f **where** $f \equiv \lambda ta xa. if xa \in FvarsT r then FvarsT ta else \{\}$

have 3: $\bigcup \{f ta xa \mid ta xa. (ta, xa) \in set ((t, x) \# txs)\} =$

$f t x \cup (\bigcup \{f ta xa \mid ta xa. (ta, xa) \in set txs\})$ **by** *auto*

have 4: $snd \text{' set } ((t, x) \# txs) = \{x\} \cup snd \text{' set txs$ **by** *auto*

have 5: $f t x \cap snd \text{' set txs} = \{\}$ **unfolding** f_def **using** xt **by** *auto*

have 6: $\bigcup \{if xa \in FvarsT r - \{x\} \cup f t x then FvarsT ta else \{\} \mid ta xa. (ta, xa) \in set txs\}$

$= (\bigcup \{f ta xa \mid ta xa. (ta, xa) \in set txs\})$

unfolding f_def **using** $xt x_txs$ **by** (*fastforce split: if_splits*)

have $FvarsT r - \{x\} \cup f t x - snd \text{' set txs} \cup$

$\bigcup \{if xa \in FvarsT r - \{x\} \cup f t x then FvarsT ta else \{\}$
 $\mid ta xa. (ta, xa) \in set txs\} =$

$FvarsT r - snd \text{' set } ((t, x) \# txs) \cup$

$\bigcup \{f ta xa \mid ta xa. (ta, xa) \in set ((t, x) \# txs)\}$

unfolding 3 4 6 **unfolding** $Un_Diff2[OF 5]$ Un_assoc **unfolding** $Diff_Diff_Un ..$

thus *?case* **unfolding** a $rawpsubstT.simps 1 2$ $\chi_def[symmetric]$ f_def **by** *simp*
qed *auto*

lemma $in_FvarsT_rawpsubstTD$:

assumes $y \in FvarsT (rawpsubstT r txs)$

and $r \in atrm$ **and** $snd \text{' (set txs)} \subseteq var$ **and** $fst \text{' (set txs)} \subseteq atrm$

and *distinct* ($map snd txs$) **and** $\forall x \in snd \text{' (set txs)}. \forall t \in fst \text{' (set txs)}. x \notin FvarsT t$

shows ($y \in FvarsT r - snd \text{' (set txs)}$) \vee

$(y \in \bigcup \{if x \in FvarsT r then FvarsT t else \{\} \mid t x . (t,x) \in set txs\})$

using $FvarsT_rawpsubstT$ *assms* **by** *auto*

lemma $FvarsT_psubstT$:

assumes $r \in atrm$ **and** $snd \text{' (set txs)} \subseteq var$ **and** $fst \text{' (set txs)} \subseteq atrm$

and *distinct* ($map snd txs$)

shows $FvarsT (psubstT r txs) =$

$(FvarsT r - snd \text{' (set txs)}) \cup$

$(\bigcup \{if x \in FvarsT r then FvarsT t else \{\} \mid t x . (t,x) \in set txs\})$

proof-

have $txs_trm: fst \text{' (set txs)} \subseteq trm$ **using** *assms* **by** *auto*

define us **where** $us: us \equiv getFrN (map snd txs) (r \# map fst txs) [] (length txs)$

have $us_facts: set us \subseteq var$

$set us \cap FvarsT r = \{\}$

$set us \cap \bigcup (FvarsT \text{' (fst \text{' (set txs))}) = \{\}$

$set us \cap snd \text{' (set txs)} = \{\}$

$length us = length txs$

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]:  $\bigwedge aa b. b \in \text{set } (\text{map snd txs}) \implies aa \in \text{set } (\text{map Var us}) \implies b \notin \text{FvarsT aa}$ 
using us_facts by (fastforce simp: image_def Int_def)
have [simp]:
 $\bigwedge b ac bc. b \in \text{set us} \implies b \in \text{FvarsT ac} \implies (ac, bc) \notin \text{set txs}$ 
using us_facts(3) by (fastforce simp: image_def Int_def)

define  $\chi$  where  $\chi\_def: \chi \equiv \text{rawpsubstT } r \ (\text{zip } (\text{map Var us}) \ (\text{map snd txs}))$ 
have  $\chi: \chi \in \text{atrm}$  unfolding  $\chi\_def$ 
using assms using us_facts by (intro rawpsubstT_atrm) (force dest!: set_zip_D)+

hence  $\chi \in \text{trm}$  by auto
note  $\chi = \chi$  this
have set_us:  $\text{set us} = \text{snd } '(\text{set } (\text{zip } (\text{map fst txs}) \ \text{us}))$ 
  using us_facts by (intro snd_set_zip[symmetric]) auto
have set_txs:  $\text{snd } ' \text{set txs} = \text{snd } '(\text{set } (\text{zip } (\text{map Var us}) \ (\text{map snd txs})))$ 
  using us_facts by (intro snd_set_zip_map_snd[symmetric]) auto
have  $\bigwedge t x. (t, x) \in \text{set } (\text{zip } (\text{map Var us}) \ (\text{map snd txs})) \implies \exists u. t = \text{Var } u$ 
  using us_facts set_zip_leftD by fastforce
hence 00:  $\bigwedge t x. (t, x) \in \text{set } (\text{zip } (\text{map Var us}) \ (\text{map snd txs}))$ 
 $\longleftrightarrow (\exists u \in \text{var}. t = \text{Var } u \wedge (\text{Var } u, x) \in \text{set } (\text{zip } (\text{map Var us}) \ (\text{map snd txs})))$ 
  using us_facts set_zip_leftD by fastforce
have FvarsT  $\chi =$ 
  FvarsT r -  $\text{snd } ' \text{set txs} \cup$ 
 $\bigcup \{ \text{if } x \in \text{FvarsT } r \text{ then FvarsT } t \text{ else } \{ \} \mid t x. (t, x) \in \text{set } (\text{zip } (\text{map Var us}) \ (\text{map snd txs})) \}$ 
  unfolding  $\chi\_def$  set_txs using assms us_facts set_txs
  by (intro FvarsT_rawpsubstT) (force dest!: set_zip_D)+
also have ... =
  FvarsT r -  $\text{snd } ' \text{set txs} \cup$ 
 $\bigcup \{ \text{if } x \in \text{FvarsT } r \text{ then } \{u\} \text{ else } \{ \} \mid u x. u \in \text{var} \wedge (\text{Var } u, x) \in \text{set } (\text{zip } (\text{map Var us}) \ (\text{map snd txs})) \}$ 
(is ... = ?R)
apply(subst 00)
by (metis (no_types, hide_lams) FvarsT_Var)
finally have 0: FvarsT  $\chi = ?R$  .
have 1: FvarsT (rawpsubstT  $\chi$  (zip (map fst txs) us)) =
  (FvarsT  $\chi$  - set us)  $\cup$ 
 $\bigcup \{ \text{if } u \in \text{FvarsT } \chi \text{ then FvarsT } t \text{ else } \{ \} \mid t u. (t, u) \in \text{set } (\text{zip } (\text{map fst txs}) \ \text{us}) \}$ 
unfolding us_facts set_us using assms  $\chi$  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  $\chi$  - set us = FvarsT r -  $\text{snd } ' \text{set txs}$ 
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:
( $\bigcup \{ \text{if } u \in \text{FvarsT } \chi \text{ then } \text{FvarsT } t \text{ else } \{ \} \mid t \ u . (t, u) \in \text{set } (\text{zip } (\text{map } \text{fst } \text{txs}) \text{ us}) \} \} =$ 
 $\bigcup \{ \text{if } x \in \text{FvarsT } r \text{ then } \text{FvarsT } t \text{ else } \{ \} \mid t \ x . (t, x) \in \text{set } \text{txs} \}$ )
proof safe
  fix xx tt y
  assume xx:  $xx \in (\text{if } y \in \text{FvarsT } \chi \text{ then } \text{FvarsT } tt \text{ else } \{ \})$ 
  and ty:  $(tt, y) \in \text{set } (\text{zip } (\text{map } \text{fst } \text{txs}) \text{ us})$ 
  have ttin:  $tt \in \text{fst } ' \text{set } \text{txs} \text{ using } ty \text{ using } \text{set\_zip\_leftD} \text{ by } \text{fastforce}$ 
  have yin:  $y \in \text{set } \text{us} \text{ using } ty \text{ by } (\text{meson } \text{set\_zip\_D})$ 
  have yvar:  $y \in \text{var} \text{ using } us\_facts \text{ yin} \text{ by } \text{auto}$ 
  have ynotin:  $y \notin \text{snd } ' \text{set } \text{txs} \text{ y} \notin \text{FvarsT } r \text{ using } yin \text{ us\_facts} \text{ by } \text{auto}$ 
  show  $xx \in \bigcup \{ \text{if } x \in \text{FvarsT } r \text{ then } \text{FvarsT } t \text{ else } \{ \} \mid t \ x . (t, x) \in \text{set } \text{txs} \}$ 
  proof(cases  $y \in \text{FvarsT } \chi$ )
    case True note  $y = \text{True}$ 
      hence xx:  $xx \in \text{FvarsT } tt \text{ using } xx \text{ by } \text{simp}$ 
      obtain x where  $xx: x \in \text{FvarsT } r$ 
      and yx:  $(\text{Var } y, x) \in \text{set } (\text{zip } (\text{map } \text{Var } \text{us}) (\text{map } \text{snd } \text{txs}))$ 
        using y ynotin unfolding 0 by (auto split: if_splits)
      have yx:  $(y, x) \in \text{set } (\text{zip } \text{us } (\text{map } \text{snd } \text{txs}))$ 
      using yvar us_facts by (intro inj_on_set_zip_map[OF inj_on_Var yx]) auto
      have  $(tt, x) \in \text{set } \text{txs} \text{ apply}(\text{rule } \text{set\_zip\_map\_fst\_snd}[OF \text{yx } ty])$ 
      using  $\langle \text{distinct } (\text{map } \text{snd } \text{txs}) \rangle \text{ us\_facts} \text{ by } \text{auto}$ 
      thus ?thesis using xx  $xr$  by auto
    qed(insert xx, auto)
  next
    fix y tt xx
    assume y:  $y \in (\text{if } xx \in \text{FvarsT } r \text{ then } \text{FvarsT } tt \text{ else } \{ \})$ 
    and tx:  $(tt, xx) \in \text{set } \text{txs}$ 
    hence xxsnd:  $xx \in \text{snd } ' \text{set } \text{txs} \text{ by } \text{force}$ 
    obtain u where uin:  $u \in \text{set } \text{us}$  and uxx:  $(u, xx) \in \text{set } (\text{zip } \text{us } (\text{map } \text{snd } \text{txs}))$ 
      by (metis xxsnd in_set_impl_in_set_zip2 length_map set_map_set_zip_leftD us_facts(5))
    have uvar:  $u \in \text{var} \text{ using } us\_facts \text{ by } \text{auto}$ 
    show  $y \in \bigcup \{ \text{if } u \in \text{FvarsT } \chi \text{ then } \text{FvarsT } t \text{ else } \{ \} \mid t \ u . (t, u) \in \text{set } (\text{zip } (\text{map } \text{fst } \text{txs}) \text{ us}) \}$ 
    proof(cases  $xx \in \text{FvarsT } r$ )
      case True note  $xx = \text{True}$ 
        hence y:  $y \in \text{FvarsT } tt \text{ using } y \text{ by } \text{auto}$ 
        have  $(\text{Var } u, xx) \in \text{set } (\text{zip } (\text{map } \text{Var } \text{us}) (\text{map } \text{snd } \text{txs}))$ 
        apply(rule set_zip_length_map[OF uxx]) using us_facts by auto
        hence ux:  $u \in \text{FvarsT } \chi \text{ using } uin \text{ xx } uvar \text{ unfolding } 0 \text{ by } \text{auto}$ 
        have ttu:  $(tt, u) \in \text{set } (\text{zip } (\text{map } \text{fst } \text{txs}) \text{ us})$ 
        apply(rule set_zip_map_fst_snd2[OF uxx tx]) using assms us_facts by auto
        show ?thesis using ux ttu y by auto
      qed(insert y, auto)
    qed
    show ?thesis
    by (simp add: psubstT_def Let_def us[symmetric]  $\chi\_def$ [symmetric] 1 2 3)
  qed

```

```

lemma in_FvarsT_psubstTD:
assumes  $y \in \text{FvarsT } (\text{psubstT } r \text{ txs})$ 
and  $r \in \text{atrm}$  and  $\text{snd } ' (\text{set } \text{txs}) \subseteq \text{var}$  and  $\text{fst } ' (\text{set } \text{txs}) \subseteq \text{atrm}$ 
and  $\text{distinct } (\text{map } \text{snd } \text{txs})$ 
shows  $y \in (\text{FvarsT } r - \text{snd } ' (\text{set } \text{txs})) \cup$ 
 $(\bigcup \{ \text{if } x \in \text{FvarsT } r \text{ then } \text{FvarsT } t \text{ else } \{ \} \mid t \ x . (t, x) \in \text{set } \text{txs} \})$ 
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 $x \in var$ **and** $t \in trm$ **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** $\dots = rawpsubstT\ (rawpsubstT\ r\ [(s,y), (t,x)])\ txs$
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** $\dots = rawpsubstT\ (substT\ r\ t\ x)\ (txs\ @\ [(s,y)])$ **using** *Cons* **by** *auto*
also **have** $\dots = rawpsubstT\ r\ (((t,x) \# txs) \ @\ [(s,y)])$ **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 $x \in var$ **and** $t \in trm$ **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.prem*s **by** (*intro* *Cons.IH*) *auto*
also **have** $\dots = rawpsubstT\ (rawpsubstT\ r\ [(t,x)])\ ((s,y) \# txs)$ **by** *simp*

```

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

```

lemma rawpsubstT_compose_fresh Var:

assumes $r \in \text{atrm}$ $\text{snd}'(\text{set } txs) \subseteq \text{var}$ **and** $\text{fst}'(\text{set } txs) \subseteq \text{atrm}$

and $\text{distinct}(\text{map } \text{snd } txs)$

and $\bigwedge i j. i < j \implies j < \text{length } txs \implies \text{snd}(txs!j) \notin \text{FvarsT}(\text{fst}(txs!i))$

and $us_facts: \text{set } us \subseteq \text{var}$

$\text{set } us \cap \text{FvarsT } r = \{\}$

$\text{set } us \cap \bigcup (\text{FvarsT}'(\text{fst}'(\text{set } txs))) = \{\}$

$\text{set } us \cap \text{snd}'(\text{set } txs) = \{\}$

$\text{length } us = \text{length } txs$

$\text{distinct } us$

shows $\text{rawpsubstT}(\text{rawpsubstT } r(\text{zip}(\text{map } \text{Var } us)(\text{map } \text{snd } txs))) (\text{zip}(\text{map } \text{fst } txs) us) = \text{rawpsubstT } r txs$

using $assms$ **proof**($\text{induction } txs$ $\text{arbitrary: } us$ r)

case ($\text{Cons } tx \text{ } txs \text{ } uus \text{ } r$)

obtain $t \ x$ **where** $tx[\text{simp}]: tx = (t,x)$ **by force**

obtain $u \ us$ **where** $uus[\text{simp}]: uus = u \# us$ **using** Cons **by** ($\text{cases } uus$) **auto**

have $us_facts: \text{set } us \subseteq \text{var}$

$\text{set } us \cap \text{FvarsT } r = \{\}$

$\text{set } us \cap \bigcup (\text{FvarsT}'(\text{fst}'(\text{set } txs))) = \{\}$

$\text{set } us \cap \text{snd}'(\text{set } txs) = \{\}$

$\text{length } us = \text{length } txs$

$\text{distinct } us$ **and** $u_facts: u \in \text{var } u \notin \text{FvarsT } r$

$u \notin \bigcup (\text{FvarsT}'(\text{fst}'(\text{set } txs)))$

$u \notin \text{snd}'(\text{set } txs) \ u \notin \text{set } us$

using Cons **by auto**

have $[\text{simp}]: \bigwedge bb \ xaa \ ab. bb \in \text{FvarsT}(\text{Var } xaa) \implies$

$(ab, bb) \in \text{set } txs \implies xaa \notin \text{set } us$

using $us_facts(1,4)$ **by force**

let $?uxs = \text{zip}(\text{map } \text{Var } us)(\text{map } \text{snd } txs)$

have $1: \text{rawpsubstT}(\text{substT } r(\text{Var } u) \ x) \ ?uxs = \text{rawpsubstT } r(\ ?uxs @ [(\text{Var } u, x)])$

using Cons.prem u_facts **apply**($\text{intro rawpsubstT_subst_fresh_switch}$)

subgoal by auto

subgoal by ($\text{auto dest!}: \text{set_zip_D}$)

subgoal by ($\text{fastforce dest!}: \text{set_zip_D}$)

subgoal by ($\text{auto dest!}: \text{set_zip_D}$)

subgoal by ($\text{fastforce dest!}: \text{set_zip_D}$)

by ($\text{auto dest!}: \text{set_zip_D}$)

let $?uuxs = \text{zip}(\text{map } \text{Var } uus)(\text{map } \text{snd } (tx \# txs))$

let $?tus = \text{zip}(\text{map } \text{fst } txs) \ us$ **let** $?txs = \text{zip}(\text{map } \text{fst } (tx \# txs)) \ uus$

have $2: u \in \text{FvarsT}(\text{rawpsubstT } r(\text{zip}(\text{map } \text{Var } us)(\text{map } \text{snd } txs))) \implies \text{False}$

apply($\text{drule in_FvarsT_rawpsubstTD}$) **apply**–

subgoal using Cons.prem **by auto**

subgoal using Cons.prem **by** ($\text{auto dest!}: \text{set_zip_D}$)

subgoal using Cons.prem **by** ($\text{force dest!}: \text{set_zip_D}$)

```

subgoal using Cons.prem1 by (auto dest!: set_zip_D)
subgoal by (auto dest!: set_zip_D)
subgoal using us_facts(1,4,5) Cons.prem1(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.prem1(4) Cons.prem1(5)[of 0 Suc i] tx
    by (auto simp: nth_Cons' split: if_splits dest: sym)
qed

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

have rawsubstT r (?uxs @ [(Var u, x)]) = rawsubstT (rawsubstT r ?uxs) [(Var u, x)]
  using Cons.prem1
  by (intro rawsubstT_compose[symmetric]) (auto 0 3 dest!: set_zip_D)
also have rawsubstT (rawsubstT r ?uxs) [(Var u, x)] = substT (rawsubstT r ?uxs) (Var u) x by
simp
finally have substT (rawsubstT r (?uxs @ [(Var u, x)])) t u =
  substT (substT (rawsubstT r ?uxs) (Var u) x) t u by simp
also have ... = substT (rawsubstT r ?uxs) t x
  using Cons 2 by (intro substT_substT) (auto 0 3 intro!: rawsubstT_atrm[of r] dest!: set_zip_D)
also have ... = rawsubstT r ((t,x) # ?uxs)
  using Cons.prem1 3
  by (intro substT_rawsubstT_fresh_switch) (auto 0 3 dest!: set_zip_D FvarsT_VarD)
also have ... = rawsubstT r ([[(t,x)] @ ?uxs]) by simp
also have ... = rawsubstT (rawsubstT r [(t,x)]) ?uxs
  using Cons.prem1 by (intro rawsubstT_compose[symmetric]) (auto 0 3 dest!: set_zip_D)
finally have rawsubstT (substT (rawsubstT r (?uxs @ [(Var u, x)])) t u) ?tus =
  rawsubstT (rawsubstT (rawsubstT r [(t,x)]) ?uxs) ?tus by auto
hence rawsubstT (rawsubstT r ?uxs) ?txs = rawsubstT (rawsubstT (rawsubstT r [(t,x)]) ?uxs)
?tus
  using 00 by auto
also have ... = rawsubstT (rawsubstT r [(t,x)]) txs
using Cons.prem1 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!: rawsubstT dest!: set_zip_D in_FvarsT_substTD
split: if_splits)
  by auto
finally show ?case by simp
qed auto

lemma rawsubstT_compose_fresh Var2_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 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 ?ruv = substT r (Var u) x let ?rvx = substT r (Var v) x

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

have 1: rawpsubstT ?ruv (zip (map Var vs) xs) =
    substT (rawpsubstT r (zip (map Var vs) xs)) (Var u) x
using Cons.prem
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.prem
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.prem
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.prem
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 ?ruv (zip (map Var us) xs)) t u) (zip ts us) =
    substT (rawpsubstT (rawpsubstT ?rvx (zip (map Var vs) xs)) (zip ts us)) t u

```

```

using Cons.premis
by (intro substT_rawsubstT_fresh_switch[simplified,symmetric])
  (auto 0 3 intro!: rawsubstT_atrm[of ?ruv] substT_atrm dest!: set_zip_D)
also have ... = substT (rawsubstT (rawsubstT ?ruv (zip (map Var vs) xs)) (zip ts vs)) t u
unfolding 0 ..
also have ... = rawsubstT (substT (rawsubstT ?ruv (zip (map Var vs) xs)) t u) (zip ts vs)
using Cons.premis
by (intro substT_rawsubstT_fresh_switch[simplified])
  (auto 0 3 intro!: rawsubstT_atrm[of ?ruv] dest!: set_zip_D)
also have ... = rawsubstT (substT (substT (rawsubstT r (zip (map Var vs) xs)) (Var u) x) t u) (zip
ts vs)
unfolding 1 ..
also have ... = rawsubstT (substT (substT (rawsubstT r (zip (map Var vs) xs)) (Var v) x) t v) (zip
ts vs)
unfolding 2 ..
also have ... = rawsubstT (substT (rawsubstT ?ruv (zip (map Var vs) xs)) t v) (zip ts vs)
unfolding 11 ..
finally have rawsubstT (substT (rawsubstT ?ruv (zip (map Var vs) xs)) t u) (zip ts vs) =
rawsubstT (substT (rawsubstT ?ruv (zip (map Var vs) xs)) t v) (zip ts vs) .
thus ?case by simp
qed auto

```

lemma rawsubstT_compose_freshVar2:

assumes r[simp]: $r \in \text{atrm}$

and ts: $\text{set } ts \subseteq \text{atrm}$

and xs: $\text{set } xs \subseteq \text{var}$ *distinct* xs

and us_facts: $\text{set } us \subseteq \text{var}$ *distinct* us

$\text{set } us \cap \text{FvarsT } r = \{\}$

$\text{set } us \cap \bigcup (\text{FvarsT } '(\text{set } ts)) = \{\}$

$\text{set } us \cap \text{set } xs = \{\}$

and vs_facts: $\text{set } vs \subseteq \text{var}$ *distinct* vs

$\text{set } vs \cap \text{FvarsT } r = \{\}$

$\text{set } vs \cap \bigcup (\text{FvarsT } '(\text{set } ts)) = \{\}$

$\text{set } vs \cap \text{set } xs = \{\}$

and l: $\text{length } us = \text{length } xs$ $\text{length } vs = \text{length } xs$ $\text{length } ts = \text{length } xs$

shows rawsubstT (rawsubstT r (zip (map Var us) xs)) (zip ts us) =

rawsubstT (rawsubstT r (zip (map Var vs) xs)) (zip ts vs) (**is** ?L = ?R)

proof–

have ts_trm: $\text{set } ts \subseteq \text{trm}$ **using** ts **by** auto

define ws **where** ws = getFrN (xs @ us @ vs) (r # ts) [] (length xs)

have ws_facts: $\text{set } ws \subseteq \text{var}$ *distinct* ws

$\text{set } ws \cap \text{FvarsT } r = \{\}$

$\text{set } ws \cap \bigcup (\text{FvarsT } '(\text{set } ts)) = \{\}$

$\text{set } ws \cap \text{set } xs = \{\}$ $\text{set } ws \cap \text{set } us = \{\}$ $\text{set } ws \cap \text{set } vs = \{\}$

$\text{length } ws = \text{length } xs$ **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*: $a \in \text{fst} \text{ ` } AB \iff (\exists b. (a,b) \in AB)$ **by** *force*

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

and $\bigwedge i j. i < j \implies j < \text{length } txs \implies \text{snd } (txs!j) \notin \text{FvarsT } (\text{fst } (txs!i))$
shows $\text{psubstT } r \text{ txs} = \text{rawpsubstT } r \text{ txs}$

proof–

have $txs_trm: r \in \text{trm}$ $\text{fst} \text{ ` } (\text{set } txs) \subseteq \text{trm}$ **using** *assms* **by** *auto*

note $\text{frt} = \text{getFrN_FvarsT}[of \text{ map } \text{snd } txs \text{ r } \# \text{ map } \text{fst } txs \text{ [] } _ \text{ length } txs]$
and $\text{fr} = \text{getFrN_Fvars}[of \text{ map } \text{snd } txs \text{ r } \# \text{ map } \text{fst } txs \text{ [] } _ \text{ length } txs]$
and $\text{var} = \text{getFrN_var}[of \text{ map } \text{snd } txs \text{ r } \# \text{ map } \text{fst } txs \text{ [] } _ \text{ length } txs]$
and $l = \text{getFrN_length}[of \text{ map } \text{snd } txs \text{ r } \# \text{ map } \text{fst } txs \text{ [] } \text{ length } txs]$
define *us* **where** $us: us \equiv \text{getFrN } (\text{map } \text{snd } txs) \text{ (r } \# \text{ map } \text{fst } txs) \text{ [] } (\text{length } txs)$
have $us_facts: \text{set } us \subseteq \text{var}$
 $\text{set } us \cap \text{FvarsT } r = \{\}$
 $\text{set } us \cap \bigcup (\text{FvarsT } \text{ ` } (\text{fst } \text{ ` } (\text{set } txs))) = \{\}$
 $\text{set } us \cap \text{snd} \text{ ` } (\text{set } txs) = \{\}$
 $\text{length } us = \text{length } txs$
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 $r \in \text{atrm}$ $x \in \text{var}$ **and** $t \in \text{atrm}$
shows $\text{psubstT } r [(t,x)] = \text{substT } r \text{ t } x$

proof–

have $\text{psubstT } r [(t,x)] = \text{rawpsubstT } r [(t,x)]$

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

thus *?thesis* **by** *auto*

qed

lemma *psubstT_eq_rawpsubst2*:
assumes $r \in \text{atrm}$ $x1 \in \text{var}$ $x2 \in \text{var}$ $t1 \in \text{atrm}$ $t2 \in \text{atrm}$

and $x1 \neq x2$ $x2 \notin FvarsT\ t1$
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 $r \in atrm$ $x1 \in var$ $x2 \in var$ $x3 \in var$ $t1 \in atrm$ $t2 \in atrm$ $t3 \in atrm$
and $x1 \neq x2$ $x1 \neq x3$ $x2 \neq x3$
 $x2 \notin FvarsT\ t1$ $x3 \notin FvarsT\ t1$ $x3 \notin FvarsT\ t2$
shows $psubstT\ r\ [(t1,x1),(t2,x2),(t3,x3)] = rawpsubstT\ r\ [(t1,x1),(t2,x2),(t3,x3)]$
using *assms* *less_SucE* *less_Suc_eq_0_disj*
by (*intro psubstT_eq_rawpsubstT*) *auto*

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

lemma *rawpsubstT_same_Var[simp]*:
assumes $r \in atrm$ $set\ xs \subseteq var$
shows $rawpsubstT\ r\ (map\ (\lambda x. (Var\ x,x))\ xs) = r$
using *assms* **by** (*induct xs*) *auto*

lemma *psubstT_same_Var[simp]*:
assumes $r \in atrm$ $set\ xs \subseteq var$ **and** *distinct xs*
shows $psubstT\ r\ (map\ (\lambda x. (Var\ x,x))\ xs) = r$
proof-
have $psubstT\ r\ (map\ (\lambda x. (Var\ x,x))\ xs) = rawpsubstT\ r\ (map\ (\lambda x. (Var\ x,x))\ xs)$
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_eq1*:
assumes $t1 \in trm$ $t2 \in trm$
and $snd\ ' (set\ txs) \subseteq var$ $fst\ ' (set\ txs) \subseteq trm$
shows $rawpsubst\ (eql\ t1\ t2)\ txs = eql\ (rawpsubstT\ t1\ txs)\ (rawpsubstT\ t2\ txs)$
using *assms* **apply** (*induct txs arbitrary: t1 t2*)
subgoal **by** *auto*
subgoal **for** $tx\ txs\ t1\ t2$ **by** (*cases tx*) *auto* .

lemma *psubst_eq1[simp]*:
assumes $t1 \in atrm$ $t2 \in atrm$
and $snd\ ' (set\ txs) \subseteq var$ $fst\ ' (set\ txs) \subseteq atrm$
and *distinct (map snd txs)*
shows $psubst\ (eql\ t1\ t2)\ txs = eql\ (psubstT\ t1\ txs)\ (psubstT\ t2\ txs)$
proof-

```

have t12: fst ‘ (set txs)  $\subseteq$  trm using assms by auto
define us where us: us  $\equiv$  getFrN (map snd txs) (map fst txs) [eql t1 t2] (length txs)
have us_facts: set us  $\subseteq$  var
set us  $\cap$  FvarsT t1 = {}
set us  $\cap$  FvarsT t2 = {}
set us  $\cap \bigcup$  (FvarsT ‘ (fst ‘ (set txs))) = {}
set us  $\cap$  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  $\equiv$  getFrN (map snd txs) (t1 # map fst txs) [] (length txs)
have vs1_facts: set vs1  $\subseteq$  var
set vs1  $\cap$  FvarsT t1 = {}
set vs1  $\cap \bigcup$  (FvarsT ‘ (fst ‘ (set txs))) = {}
set vs1  $\cap$  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  $\equiv$  getFrN (map snd txs) (t2 # map fst txs) [] (length txs)
have vs2_facts: set vs2  $\subseteq$  var
set vs2  $\cap$  FvarsT t2 = {}
set vs2  $\cap \bigcup$  (FvarsT ‘ (fst ‘ (set txs))) = {}
set vs2  $\cap$  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  $\varphi \in \text{fm}la$   $x \in \text{var } \text{snd}$  ' set txs  $\subseteq$  var fst ' set txs  $\subseteq$  atrm
 $x \notin \text{snd}$  ' set txs  $x \notin (\bigcup t \in \text{fst}$  ' set txs. FvarsT t) distinct (map snd txs)
shows psubst (exu x  $\varphi$ ) txs = exu x (psubst  $\varphi$  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) [ $\varphi$ ]
have u_facts: u  $\in$  var u  $\neq$  x
u  $\notin$  snd ' set txs u  $\notin$  ( $\bigcup t \in \text{fst}$  ' set txs. FvarsT t) u  $\notin$  Fvars  $\varphi$ 
unfolding u using f getFr_FvarsT_Fvars[of x # map snd txs map fst txs [ $\varphi$ ]] by (auto simp: assms)
hence [simp]: psubst (subst  $\varphi$  (Var u) x) txs = subst (psubst  $\varphi$  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  $\varphi$  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  $\bigwedge i j. i < j \implies j < \text{length } txs \implies \text{snd } (txs!j) \notin \text{FvarsT } (\text{fst } (txs!i))$ 
shows  $\text{rawpsubstT } (\text{Var } y) txs = s$ 
using assms proof(induction txs)
  case (Cons tx txs)
    obtain  $t x$  where  $tx[\text{simp}]: tx = (t,x)$  by (cases tx) auto

    have  $00: \text{FvarsT } t \cap \text{snd } \text{' set } txs = \{\}$ 
    using Cons.prem5[of 0 Suc _] by (auto simp: set_conv_nth)

    have  $\text{rawpsubstT } (\text{substT } (\text{Var } y) t x) txs = s$ 
    proof(cases y = x)
      case [simp]:True hence [simp]:  $s = t$  using  $\langle \text{distinct } (\text{map } \text{snd } (tx \# txs)) \rangle$ 
       $\langle (s, y) \in \text{set } (tx \# txs) \rangle$  using image_iff by fastforce
      show ?thesis using Cons.prem5 00 by auto
    next
      case False
      hence [simp]:  $\text{substT } (\text{Var } y) t x = \text{Var } y$ 
      using Cons.prem5 by (intro substT_notIn) auto
      have  $\text{rawpsubstT } (\text{Var } y) txs = s$ 
      using Cons.prem5 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  $y \in \text{var } \text{snd } \text{' (set } txs) \subseteq \text{var } \text{fst } \text{' (set } txs) \subseteq \text{trm}$ 
and  $\text{distinct } (\text{map } \text{snd } txs)$  and  $(s,y) \in \text{set } txs$ 
shows  $\text{psubstT } (\text{Var } y) txs = s$ 
proof-
  define us where  $us \equiv \text{getFrN } (\text{map } \text{snd } txs) (\text{Var } y \# \text{map } \text{fst } txs) [] (\text{length } txs)$ 
  have us_facts:  $\text{set } us \subseteq \text{var}$ 
   $\text{set } us \cap \bigcup (\text{FvarsT } \text{' (fst } \text{' (set } txs))) = \{\}$ 
   $y \notin \text{set } us$ 
   $\text{set } us \cap \text{snd } \text{' (set } txs) = \{\}$ 
   $\text{length } us = \text{length } txs$ 
  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  $i[\text{simp}]: i < \text{length } txs \text{ txs!}i = (s,y)$  using  $\langle (s,y) \in \text{set } txs \rangle$ 
  by (metis in_set_conv_nth)
  hence  $00[\text{simp}]: \bigwedge j. j < \text{length } txs \implies txs ! j = txs ! i \implies j = i$ 
  using  $\langle \text{distinct } (\text{map } \text{snd } txs) \rangle$  distinct_Ex1 nth_mem by fastforce
  have  $000[\text{simp}]: \bigwedge j \text{ ia. } j < \text{length } txs \implies \text{ia} < \text{length } txs \implies \text{snd } (txs ! j) \neq us ! \text{ia}$ 
  using assms us_facts

```

```

  by (metis IntI empty_iff length_map list.set_map nth_map nth_mem)
  have [simp]:  $\bigwedge ii\ jj. ii < jj \implies jj < \text{length } txs \implies us ! ii \in var$ 
    using nth_mem us_facts(1) us_facts(5) by auto
  have [simp]:  $\bigwedge i\ j. i < j \implies j < \text{length } txs \implies us ! j \notin FvarsT (fst (txs ! i))$ 
    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 $y \in var\ x \in var\ t \in atrm$

$snd \text{ ' } set\ txs \subseteq var\ fst \text{ ' } set\ txs \subseteq atrm\ x \notin snd \text{ ' } set\ txs$

$distinct (map\ snd\ txs)\ y \neq x$

shows $psubstT (Var\ y) ((t, x) \# txs) = psubstT (Var\ y) txs$

proof–

have $txs_trm: t \in trm\ fst \text{ ' } set\ txs \subseteq trm$ **using** assms **by** auto

note $assms1 = assms(1,2)\ assms(4)\ assms(6-8)\ txs_trm$

note $fst = getFrN_FvarsT[of\ x \# map\ snd\ txs\ Var\ y \# t \# map\ fst\ txs\ []_ Suc\ (length\ txs)]$

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 :

$uus \equiv getFrN (x \# map\ snd\ txs) (Var\ y \# t \# map\ fst\ txs) [] (Suc (length\ txs))$

have $uus_facts: set\ uus \subseteq var$

$set\ uus \cap \bigcup (FvarsT \text{ ' } (fst \text{ ' } (set\ txs))) = \{\}$

$set\ uus \cap snd \text{ ' } (set\ txs) = \{\}$

$set\ uus \cap FvarsT\ t = \{\}$

$x \notin set\ uus$

$y \notin set\ uus$

$length\ uus = Suc (length\ txs)$

$distinct\ uus$

using $assms1$ **unfolding** uus

apply –

subgoal **by** auto

subgoal **using** fst **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** fst **by** auto

subgoal **using** var **by** (fastforce simp: in_fst_image Int_def)

subgoal **using** fst **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 $us_facts: set\ us \subseteq var$

$set\ us \cap \bigcup (FvarsT \text{ ' } (fst \text{ ' } (set\ txs))) = \{\}$

$set\ us \cap snd \text{ ' } (set\ txs) = \{\}$

$set\ us \cap FvarsT\ t = \{\}$

$x \notin set\ us$

```

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 (rawsubstT (Var y) (zip (map Var us) (map snd txs))) t u =
rawsubstT (Var y) (zip (map Var us) (map snd txs))
using assms u_facts us_facts
by (intro substT_notIn)
(auto 0 3 intro!: rawsubstT dest!: set_zip_D in_FvarsT_rawsubstT_imp FvarsT_VarD)

have rawsubstT (rawsubstT (Var y) (zip (map Var us) (map snd txs))) (zip (map fst txs) us) =
rawsubstT (rawsubstT (Var y) (zip (map Var vs) (map snd txs))) (zip (map fst txs) vs)
using assms vs_facts us_facts by (intro rawsubstT_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) ⊆ atrm
shows psubstT zer txs = zer

```

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

lemma *rawpsubstT_suc*:

assumes $r \in \text{trm}$ **and** $\text{snd} \text{ ` (set txs) } \subseteq \text{var}$ **and** $\text{fst} \text{ ` (set txs) } \subseteq \text{trm}$

shows $\text{rawpsubstT} (\text{suc } r) \text{ txs} = \text{suc} (\text{rawpsubstT } r \text{ txs})$

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

subgoal **by** *simp*

subgoal **for** $\text{tx txs } r$ **by** (*cases tx*) *auto* .

lemma *psubstT_suc[simp]*:

assumes $r \in \text{atrm}$ **and** $\text{snd} \text{ ` (set txs) } \subseteq \text{var}$ **and** $\text{fst} \text{ ` (set txs) } \subseteq \text{atrm}$

and *distinct* (*map snd txs*)

shows $\text{psubstT} (\text{suc } r) \text{ txs} = \text{suc} (\text{psubstT } r \text{ txs})$

proof–

have *000*: $r \in \text{trm}$ $\text{fst} \text{ ` (set txs) } \subseteq \text{trm}$ **using** *assms* **by** *auto*

define *us* **where** *us*: $\text{us} \equiv \text{getFrN} (\text{map snd txs}) (\text{suc } r \# \text{map fst txs}) [] (\text{length txs})$

have *us_facts*: $\text{set us} \subseteq \text{var}$

$\text{set us} \cap \text{FvarsT } r = \{\}$

$\text{set us} \cap \bigcup (\text{FvarsT} \text{ ` (fst ` (set txs))}) = \{\}$

$\text{set us} \cap \text{snd} \text{ ` (set txs) } = \{\}$

$\text{length us} = \text{length txs}$

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*: $\text{vs} \equiv \text{getFrN} (\text{map snd txs}) (r \# \text{map fst txs}) [] (\text{length txs})$

have *vs_facts*: $\text{set vs} \subseteq \text{var}$

$\text{set vs} \cap \text{FvarsT } r = \{\}$

$\text{set vs} \cap \bigcup (\text{FvarsT} \text{ ` (fst ` (set txs))}) = \{\}$

$\text{set vs} \cap \text{snd} \text{ ` (set txs) } = \{\}$

$\text{length vs} = \text{length txs}$

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*: $\text{rawpsubstT} (\text{suc } r) (\text{zip} (\text{map Var vs}) (\text{map snd txs})) =$

$\text{suc} (\text{rawpsubstT } r (\text{zip} (\text{map Var vs}) (\text{map snd txs})))$

using *assms* *vs_facts* **by** (*intro rawpsubstT_suc*) (*auto dest!: set_zip_D*)

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


```

      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  $r1 \in trm$   $r2 \in trm$  and  $snd \text{ ' (set txs) } \subseteq var$  and  $fst \text{ ' (set txs) } \subseteq trm$ 
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  $r1 \in atrm$   $r2 \in atrm$  and  $snd \text{ ' (set txs) } \subseteq var$  and  $fst \text{ ' (set txs) } \subseteq atrm$ 
and distinct (map snd txs)
shows psubstT (pls r1 r2) txs = pls (psubstT r1 txs) (psubstT r2 txs)
proof-

```

```

  have 000:  $fst \text{ ' (set txs) } \subseteq trm$  using assms by auto
  define us where us:  $us \equiv getFrN$  (map snd txs) (pls r1 r2 # map fst txs) [] (length txs)
  have us_facts:  $set\ us \subseteq var$ 
   $set\ us \cap FvarsT\ r1 = \{\}$ 
   $set\ us \cap FvarsT\ r2 = \{\}$ 
   $set\ us \cap \bigcup (FvarsT \text{ ' (fst \text{ ' (set txs))})} = \{\}$ 
   $set\ us \cap snd \text{ ' (set txs) } = \{\}$ 
   $length\ us = length\ txs$ 
  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:  $vs1 \equiv getFrN$  (map snd txs) (r1 # map fst txs) [] (length txs)
  have vs1_facts:  $set\ vs1 \subseteq var$ 
   $set\ vs1 \cap FvarsT\ r1 = \{\}$ 
   $set\ vs1 \cap \bigcup (FvarsT \text{ ' (fst \text{ ' (set txs))})} = \{\}$ 
   $set\ vs1 \cap snd \text{ ' (set txs) } = \{\}$ 
   $length\ vs1 = length\ txs$ 
  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  $\equiv$  getFrN (map snd txs) (r2 # map fst txs) [] (length txs)
have vs2_facts: set vs2  $\subseteq$  var
set vs2  $\cap$  FvarsT r2 = {}
set vs2  $\cap \bigcup$  (FvarsT ' (fst ' (set txs))) = {}
set vs2  $\cap$  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  $\in$  trm r2  $\in$  trm and snd ' (set txs)  $\subseteq$  var and fst ' (set txs)  $\subseteq$  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 $r1 \in atrm$ $r2 \in atrm$ **and** $snd \text{ ` (set txs) } \subseteq var$ **and** $fst \text{ ` (set txs) } \subseteq atrm$
and *distinct (map snd txs)*
shows $psubstT (tms r1 r2) txs = tms (psubstT r1 txs) (psubstT r2 txs)$
proof–
have *000: $fst \text{ ` (set txs) } \subseteq trm$ using *assms* by *auto**
define *us* **where** $us \equiv getFrN (map snd txs) (tms r1 r2 \# map fst txs) [] (length txs)$
have *us_facts: $set us \subseteq var$*
 $set us \cap FvarsT r1 = \{\}$
 $set us \cap FvarsT r2 = \{\}$
 $set us \cap \bigcup (FvarsT \text{ ` (fst \text{ ` (set txs))})} = \{\}$
 $set us \cap snd \text{ ` (set txs) } = \{\}$
 $length us = length txs$
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** $vs1 \equiv getFrN (map snd txs) (r1 \# map fst txs) [] (length txs)$
have *vs1_facts: $set vs1 \subseteq var$*
 $set vs1 \cap FvarsT r1 = \{\}$
 $set vs1 \cap \bigcup (FvarsT \text{ ` (fst \text{ ` (set txs))})} = \{\}$
 $set vs1 \cap snd \text{ ` (set txs) } = \{\}$
 $length vs1 = length txs$
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** $vs2 \equiv getFrN (map snd txs) (r2 \# map fst txs) [] (length txs)$
have *vs2_facts: $set vs2 \subseteq var$*
 $set vs2 \cap FvarsT r2 = \{\}$
 $set vs2 \cap \bigcup (FvarsT \text{ ` (fst \text{ ` (set txs))})} = \{\}$
 $set vs2 \cap snd \text{ ` (set txs) } = \{\}$
 $length vs2 = length txs$
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: rawsubstT (tms r1 r2) (zip (map Var us) (map snd txs)) =
      tms (rawsubstT r1 (zip (map Var us) (map snd txs)))
      (rawsubstT r2 (zip (map Var us) (map snd txs)))
using assms us_facts by (intro rawsubstT_tms) (auto dest!: set_zip_D)

have 1: rawsubstT (rawsubstT r1 (zip (map Var us) (map snd txs))) (zip (map fst txs) us) =
      rawsubstT (rawsubstT r1 (zip (map Var vs1) (map snd txs))) (zip (map fst txs) vs1)
using assms us_facts vs1_facts by (intro rawsubstT_compose_freshVar2) auto

have 2: rawsubstT (rawsubstT r2 (zip (map Var us) (map snd txs))) (zip (map fst txs) us) =
      rawsubstT (rawsubstT r2 (zip (map Var vs2) (map snd txs))) (zip (map fst txs) vs2)
using assms us_facts vs2_facts by (intro rawsubstT_compose_freshVar2) auto

have 3: rawsubstT (rawsubstT (tms r1 r2) (zip (map Var us) (map snd txs))) (zip (map fst txs) us)
=
      tms (rawsubstT (rawsubstT r1 (zip (map Var us) (map snd txs))) (zip (map fst txs) us))
      (rawsubstT (rawsubstT r2 (zip (map Var us) (map snd txs))) (zip (map fst txs) us))
unfolding 0 using assms us_facts apply(intro rawsubstT_tms)
subgoal by (auto dest!: set_zip_D intro!: rawsubstT)
subgoal by (force dest!: set_zip_D intro!: rawsubstT simp: Int_def)
subgoal by (auto dest!: set_zip_D intro!: rawsubstT)
subgoal by (fastforce dest!: set_zip_D intro!: rawsubstT 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**
 $Lq \equiv \text{exi } zz \text{ (eql (Var } yy) \text{ (pls (Var } zz) \text{ (Var } xx))}$

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

lemma $Lq_def2: z \in \text{var} \implies z \neq yy \implies z \neq xx \implies Lq = \text{exi } z \text{ (eql (Var } yy) \text{ (pls (Var } z) \text{ (Var } xx))}$
unfolding Lq_def **using** exi_rename [of $\text{eql (Var } yy) \text{ (pls (Var } zz) \text{ (Var } xx) \text{ } zz \text{]}$] **by auto**

lemma $Lq[\text{simp}, \text{intro!}]: Lq \in \text{fmla}$
unfolding Lq_def **by auto**

lemma $Fvars_Lq[\text{simp}]: Fvars Lq = \{xx, yy\}$
unfolding Lq_def **by auto**

As usual, we also define a predicate version:

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

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

lemma *LLq2[simp,intro!]*:
 $n \in num \implies LLq\ n\ (Var\ yy') \in fmla$
by *auto*

lemma *Fvars_LLq[simp]*: $t1 \in trm \implies t2 \in trm \implies$
 $Fvars\ (LLq\ t1\ t2) = FvarsT\ t1 \cup FvarsT\ t2$
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*]: $t1 \in atrm$ $t2 \in atrm$ $z \in var$ $z \notin FvarsT\ t1$ $z \notin FvarsT\ t2$
shows $LLq\ t1\ t2 = exi\ z\ (eql\ t2\ (pls\ (Var\ z)\ t1))$
proof–
define z' **where** $z' \equiv getFr\ [xx,yy,z]\ [t1,t2]\ []$
have $z_facts[simp]$: $z' \in var$ $z' \neq yy$ $z' \neq xx$ $z' \neq z$ $z' \notin FvarsT\ t1$ $z' \notin FvarsT\ t2$
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 $\dots = exi\ z\ (eql\ t2\ (pls\ (Var\ z)\ t1))$
using *exi_rename[of eql t2 (pls (Var z') t1) z' z, simplified]* .
finally show *?thesis* .
qed

lemma *LLq_pls_zz*:
assumes $t1 \in atrm$ $t2 \in atrm$ $zz \notin FvarsT\ t1$ $zz \notin FvarsT\ t2$
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*]: $t1 \in atrm$ $t2 \in atrm$ $s \in atrm$ $x \in var$
shows $subst\ (LLq\ t1\ t2)\ s\ x = LLq\ (substT\ t1\ s\ x)\ (substT\ t2\ s\ x)$
proof–
define z **where** $z \equiv getFr\ [xx,yy,x]\ [t1,t2,s]\ []$
have $z_facts[simp]$: $z \in var$ $z \neq xx$ $z \neq yy$ $z \neq x$ $z \notin FvarsT\ t1$ $z \notin FvarsT\ t2$ $z \notin FvarsT\ s$
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:  $t1 \in atrm$   $t2 \in atrm$   $fst \text{ ' set } txs \subseteq atrm$ 
and 2:  $snd \text{ ' set } txs \subseteq var$ 
and 3:  $distinct (map\ snd\ txs)$ 
shows  $psubst (LLq\ t1\ t2)\ txs = LLq (psubstT\ t1\ txs) (psubstT\ t2\ txs)$ 
proof-
  have 0:  $t1 \in trm$   $t2 \in trm$   $fst \text{ ' set } txs \subseteq trm$  using 1 by auto
  define  $z$  where  $z \equiv getFr ([xx,yy] @ map\ snd\ txs) ([t1,t2] @ map\ fst\ txs) []$ 
  have  $us\_facts: z \in var$   $z \neq xx$   $z \neq yy$ 
   $z \notin FvarsT\ t1$   $z \notin FvarsT\ t2$ 
   $z \notin \bigcup (FvarsT \text{ ' } (fst \text{ ' } (set\ txs)))$ 
   $z \notin snd \text{ ' } (set\ txs)$ 
  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
 $Ls \equiv cnj\ Lq (neg (eql (Var\ xx) (Var\ yy)))$ 

```

```

lemma  $Ls[simp,intro!]: Ls \in fmla$ 
unfolding  $Ls\_def$  by auto

```

```

lemma  $Fvars\_Ls[simp]: Fvars\ Ls = \{xx,yy\}$ 
unfolding  $Ls\_def$  by auto

```

```

definition  $LLs$  where  $LLs \equiv \lambda\ t1\ t2. psubst\ Ls\ [(t1,xx), (t2,yy)]$ 

```

```

lemma  $LLs[simp,intro!]:$ 
assumes  $t1 \in trm$   $t2 \in trm$ 
shows  $LLs\ t1\ t2 \in fmla$ 
  using  $assms$  unfolding  $LLs\_def$  by auto

```

```

lemma  $LLs2[simp,intro!]:$ 
 $n \in num \implies LLs\ n\ (Var\ yy') \in fmla$ 
  by auto

```

```

lemma  $Fvars\_LLs[simp]: t1 \in trm \implies t2 \in trm \implies$ 

```

```

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 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**

$ball\ x\ t\ \varphi \equiv all\ x\ (imp\ (LLq\ (Var\ x)\ t)\ \varphi)$

lemma $ball[simp, intro]$: $x \in var \implies t \in trm \implies \varphi \in fmla \implies ball\ x\ t\ \varphi \in fmla$
unfolding $ball_def$ **by** $auto$

lemma $Fvars_ball[simp]$:
 $x \in var \implies \varphi \in fmla \implies t \in trm \implies Fvars\ (ball\ x\ t\ \varphi) = (Fvars\ \varphi \cup FvarsT\ t) - \{x\}$
unfolding $ball_def$ **by** $auto$

lemma $subst_ball$:
 $\varphi \in fmla \implies t \in atrm \implies t1 \in atrm \implies x \in var \implies y \in var \implies x \neq y \implies x \notin FvarsT\ t1 \implies$
 $subst\ (ball\ x\ t\ \varphi)\ t1\ y = ball\ x\ (substT\ t\ t1\ y)\ (subst\ \varphi\ t1\ y)$
unfolding $ball_def$ **by** $simp$

lemma $psubst_ball$:
 $\varphi \in fmla \implies y \in var \implies snd\ 'set\ txs \subseteq var \implies t \in atrm \implies$
 $fst\ 'set\ txs \subseteq trm \implies fst\ 'set\ txs \subseteq atrm \implies y \notin snd\ 'set\ txs \implies y \notin (\bigcup t \in fst\ 'set\ txs.\ FvarsT\ t)$
 \implies
 $distinct\ (map\ snd\ txs) \implies$
 $psubst\ (ball\ y\ t\ \varphi)\ txs = ball\ y\ (psubstT\ t\ txs)\ (psubst\ \varphi\ txs)$
unfolding $ball_def$ **by** $simp$

Bounded exists

definition $bexi :: 'var \Rightarrow 'trm \Rightarrow 'fmla \Rightarrow 'fmla\ where$
 $bexi\ x\ t\ \varphi \equiv exi\ x\ (cnj\ (LLq\ (Var\ x)\ t)\ \varphi)$

lemma $bexi[simp, intro]$: $x \in var \implies t \in trm \implies \varphi \in fmla \implies bexi\ x\ t\ \varphi \in fmla$
unfolding $bexi_def$ **by** $auto$

lemma $Fvars_bexi[simp]$:
 $x \in var \implies \varphi \in fmla \implies t \in trm \implies Fvars\ (bexi\ x\ t\ \varphi) = (Fvars\ \varphi \cup FvarsT\ t) - \{x\}$
unfolding $bexi_def$ **by** $auto$

lemma $subst_bexi$:
 $\varphi \in fmla \implies t \in atrm \implies t1 \in atrm \implies x \in var \implies y \in var \implies x \neq y \implies x \notin FvarsT\ t1 \implies$
 $subst\ (bexi\ x\ t\ \varphi)\ t1\ y = bexi\ x\ (substT\ t\ t1\ y)\ (subst\ \varphi\ t1\ y)$
unfolding $bexi_def$ **by** $simp$

lemma $psubst_bexi$:
 $\varphi \in fmla \implies y \in var \implies snd\ 'set\ txs \subseteq var \implies t \in atrm \implies$
 $fst\ 'set\ txs \subseteq trm \implies fst\ 'set\ txs \subseteq atrm \implies y \notin snd\ 'set\ txs \implies y \notin (\bigcup t \in fst\ 'set\ txs.\ FvarsT\ t)$
 \implies
 $distinct\ (map\ snd\ txs) \implies$
 $psubst\ (bexi\ y\ t\ \varphi)\ txs = bexi\ y\ (psubstT\ t\ txs)\ (psubst\ \varphi\ txs)$
unfolding $bexi_def$ **by** $simp$

end — context $Syntax_Arith$

Chapter 8

Deduction in a System Embedding the Intuitionistic Robinson Arithmetic

NB: Robinson arithmetic, also known 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) $\varphi \implies$
 $F \subseteq \text{fmla} \implies \text{finite } F \implies \varphi \in \text{fmla} \implies t \in \text{trm} \implies x \in \text{var} \implies$
 $x \notin (\bigcup \varphi \in F. \text{Fvars } \varphi) \implies x \notin \text{FvarsT } t \implies$
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) \implies *nprv* F (LLq t1 t) \implies
 $F \subseteq \text{fmla} \implies \text{finite } F \implies \varphi \in \text{fmla} \implies t \in \text{atrm} \implies t1 \in \text{atrm} \implies x \in \text{var} \implies x \notin \text{FvarsT } t \implies$
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) \implies *nprv* F (LLq t1 t) \implies *nprv* (insert (subst φ t1 x) F) $\psi \implies$
 $F \subseteq \text{fmla} \implies \text{finite } F \implies \varphi \in \text{fmla} \implies t \in \text{atrm} \implies t1 \in \text{atrm} \implies x \in \text{var} \implies \psi \in \text{fmla} \implies$
 $x \notin \text{FvarsT } t \implies$
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) \implies *nprv* F (LLq t1 t) \implies
 $F \subseteq \text{fmla} \implies \text{finite } F \implies \varphi \in \text{fmla} \implies t \in \text{atrm} \implies t1 \in \text{atrm} \implies x \in \text{var} \implies$
 $x \notin \text{FvarsT } t \implies$
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) \implies *nprv* (insert (LLq (Var x) t) (insert φ F)) $\psi \implies$
 $F \subseteq \text{fmla} \implies \text{finite } F \implies \varphi \in \text{fmla} \implies x \in \text{var} \implies \psi \in \text{fmla} \implies t \in \text{atrm} \implies$
 $x \notin (\bigcup \varphi \in F. \text{Fvars } \varphi) \implies x \notin \text{Fvars } \psi \implies x \notin \text{FvarsT } t \implies$
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 conj 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 conj 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 nI[simp]: nprv F (subst  $\varphi$  (Var y) x)
and i[simp]:  $F \subseteq \text{fmla finite } F \ \varphi \in \text{fmla} \ x \in \text{var} \ y \in \text{var}$ 
and u:  $y \notin (\bigcup \varphi \in F. \text{Fvars } \varphi)$  and yx[simp]:  $y = x \vee y \notin \text{Fvars } \varphi$ 
shows nprv F (all x  $\varphi$ )
proof–
  have [simp]:  $\bigwedge \varphi. \varphi \in F \implies y \notin \text{Fvars } \varphi$  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 $n: nprv F (exi x \varphi)$
and $nn: nprv (insert (subst \varphi (Var y) x) F) \psi$
and $0: F \subseteq fmla$ $finite F$ $\varphi \in fmla$ $x \in var$ $y \in var$ $\psi \in fmla$
and $yx: y = x \vee y \notin Fvars$ $\varphi y \notin \bigcup (Fvars ' F) y \notin Fvars$ ψ
shows $nprv F \psi$
proof-
have $e: exi x \varphi = exi y (subst \varphi (Var y) x)$
using 0 yx n nn **by** $(subst\ exi_rename2[of_ _ _ y])$ $(auto\ simp: 0)$
show $?thesis$
using $assms$ **unfolding** e
by $(auto\ intro: nprv_exiE[of_ _ _ y\ subst\ \varphi\ (Var\ y)\ x])$
qed

lemma prv_neg_zer_suc:
assumes $[simp]: t \in atrm$ **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 $t \in atrm$ **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:
 $nprv F (eql t zer) \implies nprv F (eql t (suc t1)) \implies$
 $finite F \implies F \subseteq fmla \implies t \in atrm \implies t1 \in atrm \implies \varphi \in fmla \implies$
 $nprv F \varphi$
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:
 $t \in atrm \implies t' \in atrm \implies$
 $prv (imp (eql (suc t) (suc t'))$
 $(eql t t'))$

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]*: $t \in \text{atrm}$ and *x[simp]*: $x \in \text{var } x \notin \text{FvarsT } t$
 shows *prv* (*dsj* (*eql* *t* *zer*)
 (*exi* *x* (*eql* *t* (*suc* (*Var* *x*))))))

proof-

define *x'* where *x'*: $x' \equiv \text{getFr } [x,yy] [t] []$
 have *x'_facts*[*simp*]: $x' \in \text{var } x' \neq x \ x' \neq yy \ x' \notin \text{FvarsT } t$ **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*) $\varphi \implies$ *nprv* (*insert* (*eql* *t* (*suc* (*Var* *x*))) *F*) $\varphi \implies$

finite *F* $\implies F \subseteq \text{fmla} \implies \varphi \in \text{fmla} \implies x \in \text{var} \implies t \in \text{atrm} \implies$

$x \notin \text{Fvars } \varphi \implies x \notin \text{FvarsT } t \implies x \notin \bigcup (\text{Fvars } 'F) \implies$

nprv *F* φ

by (*nprover*3 *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*]: $t \in \text{atrm}$ shows *prv* (*eql* (*pls* *t* *zer*) *t*)

using *prv_psubst*[*OF* _ _ _ *prv_pls_zer_var*, of [(*t,xx*)]]

by *simp*

lemma *prv_pls_suc*:

$t \in atrm \implies t' \in atrm \implies$
 $prv (eql (pls t (suc t'))$
 $(suc (pls t t')))$
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 :
 $t \in atrm \implies t' \in atrm \implies$
 $prv (eql (tms t (suc t'))$
 $(pls (tms t t') t))$
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 $prv (imp (eql t1 t2)$
 $(eql (suc t1) (suc t2)))$

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 $prv (imp (eql t1 t1')$
 $(imp (eql t2 t2') (eql (pls t1 t2) (pls t1' t2'))))$

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(nrul r: nprv_prvI)
apply(nrul r: nprv_impI)
apply(nrul r: nprv_impI)
apply(nrul r: nprv_eql_substE02[of t1 t1' _ eq (pls (Var z) t2) (pls t1' t2') z])
apply(nrul r: nprv_eql_substE02[of t2 t2' _ eq (pls t1' (Var z)) (pls t1' t2') z])
apply(nrul r: nprv_eq_eqI) .
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 (eq t1 t1') and prv (eq t2 t2')
shows prv (eq (pls t1 t2) (pls t1' t2'))
by (metis assms atrm_imp_trm cnj eq 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 (eq t1 t1') ⇒ prv (eq (pls t1 t2) (pls t1' t2))
by (rule prv_pls_cong[OF _ _ _ _ _ prv_eq_reflT]) auto

```

```

lemma prv_pls_congR:
t1 ∈ atrm ⇒ t2 ∈ atrm ⇒ t2' ∈ atrm ⇒
prv (eq t2 t2') ⇒ prv (eq (pls t1 t2) (pls t1 t2'))
by (rule prv_pls_cong[OF _ _ _ _ _ prv_eq_reflT]) auto

```

```

lemma nprv_pls_cong:
assumes [simp]: t1 ∈ atrm t1' ∈ atrm t2 ∈ atrm t2' ∈ atrm
shows nprv {eq t1 t1', eq t2 t2'} (eq (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 (eq t1 t1')
(eq (pls t1 t2) (pls 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 z ≠ 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(nrul r: nprv_prvI)
apply(nrul r: nprv_impI)
apply(nrul 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_eqI) .
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 $t1 \in \text{atrm } t1' \in \text{atrm } t2 \in \text{atrm } t2' \in \text{atrm}$

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]: $t1 \in \text{atrm } t1' \in \text{atrm } t2 \in \text{atrm } t2' \in \text{atrm}$

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:

$t1 \in \text{atrm} \implies t1' \in \text{atrm} \implies t2 \in \text{atrm} \implies$

$\text{prv} (\text{eql } t1 \ t1') \implies \text{prv} (\text{eql } (\text{tms } t1 \ t2) \ (\text{tms } t1' \ t2'))$

by (rule prv_tms_cong[OF _____ prv_eql_reflT]) auto

lemma prv_tms_congR:

$t1 \in \text{atrm} \implies t2 \in \text{atrm} \implies t2' \in \text{atrm} \implies$

$\text{prv} (\text{eql } t2 \ t2') \implies \text{prv} (\text{eql } (\text{tms } t1 \ t2) \ (\text{tms } t1 \ t2'))$

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:

$t \in \text{atrm} \implies \text{prv} (\text{eql} (\text{pls } t \ (\text{suc } \text{zer})) \ (\text{suc } t))$

by (metis (no_types, hide_lams) 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]: $t1 \in \text{atrm } t2 \in \text{atrm}$

shows prv (imp (LLq (suc t1) (suc t2)) (LLq t1 t2))

proof— **define** z **where** $z \equiv \text{getFr } [xx,yy,zz] \ [t1,t2] \ []$

have z_facts [simp]: $z \in \text{var } z \neq xx \ z \neq yy \ z \neq zz \ zz \neq z \ z \notin \text{FvarsT } t1 \ z \notin \text{FvarsT } t2$

using $\text{getFr_FvarsT_Fvars}$ [of $[xx,yy,zz] \ [t1,t2] \ []$] **unfolding** z_def [symmetric] **by** auto

note LLq_pls [of $__ \ z, \text{simp}$]

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 t1 x ∉ 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** $n: n = \text{Suc } n'$ **by** (*cases n*) *auto*
thus ?*case* **unfolding** n **by** (*simp add: prv_neg_zer_suc*)
next
case (*Suc m n*) **note** $s = \text{Suc}$
show ?*case*
proof(*cases n*)
case 0
thus ?*thesis* **by** (*simp add: prv_neg_eql_suc_Num_zer*)
next
case (*Suc n'*) **note** $n = \text{Suc}$
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 $0 = \text{prv_pls_suc}$ [*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 ≠ 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:  $\bigwedge k'. k = \text{Suc } k' \implies$ 
     $\text{prv } (\text{neg } (\text{eql } (\text{pls } (\text{Num } m) (\text{Num } n)) (\text{Num } k')) \wedge m + n \neq k')$ 
  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 z ≠ 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_nprvI) .
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  $mm: m * n \neq k'$  using False Suc.prem by auto
note  $IH = \text{Suc.IH}[OF\ mm]$ 
show ?thesis unfolding  $k$ 
apply(nrule  $r: \text{nprv\_negI}$ )
apply(nrule  $r: \text{nprv\_addLemmaE}[OF\ \text{prv\_prv\_eql\_sym}[OF\ \_\_\ \text{prv\_eql\_pls\_plus}[of\ k'\ m]]])$ )
apply(nrule  $r: \text{nprv\_eql\_transE01}[of\ \_\ \text{Num}\ (k' + m)]$ )
apply(nrule  $r: \text{nprv\_clear3\_2}$ )
apply(nrule  $r: \text{nprv\_clear2\_2}$ )
apply(nrule  $r: \text{nprv\_pls\_Num\_injectE0}$ )
apply(nrule  $r: \text{nprv\_clear2\_2}$ )
apply(nrule  $r: \text{nprv\_addLemmaE}[OF\ IH]$ )
apply(nrule  $r: \text{nprv\_negE0}$ ) .
qed
show ?case
apply(nrule  $r: \text{nprv\_prvI}$ )
apply(nrule  $r: \text{nprv\_addLemmaE}[OF\ \text{prv\_tms\_suc},\ of\ \text{Num}\ m\ \text{Num}\ n]$ )
apply(nrule  $r: \text{nprv\_eql\_substE}[of\ \_\ \text{tms}\ (\text{Num}\ m)\ (\text{Suc}\ (\text{Num}\ n))\ \text{pls}\ (\text{tms}\ (\text{Num}\ m)\ (\text{Num}\ n))\ (\text{Num}\ m)$ )
   $\text{neg}\ (\text{eql}\ (\text{Var}\ xx)\ (\text{Num}\ k))\ xx]$ )
apply(nrule  $r: \text{nprv\_clear}$ ) .
qed

```

```

lemma consistent_prv_eql_tms_times_rev:
assumes consistent_prv ( $\text{eql}\ (\text{tms}\ (\text{Num}\ m)\ (\text{Num}\ n))\ (\text{Num}\ k)$ )
shows  $k = m * n$ 
by (metis  $\text{Num}\ \text{assms}\ \text{consistent\_def}\ \text{eql}\ \text{not\_times\_prv\_neg\_eql\_tms}\ \text{num}\ \text{tms}\ \text{prv\_neg\_fls}\ \text{subsetCE}$ )

```

Representability of the order

```

lemma leq_prv_LLq_Num:
assumes  $m \leq n$ 
shows  $\text{prv}\ (\text{LLq}\ (\text{Num}\ m)\ (\text{Num}\ n))$ 
proof-
  obtain  $i$  where  $n = i + m$  using assms add.commute le_Suc_ex by blast
  note  $\text{prv\_eql\_pls\_plus}[simp]$ 
  have  $\text{prv}\ (\text{exi}\ zz\ (\text{eql}\ (\text{Num}\ (i + m))\ (\text{pls}\ (\text{Var}\ zz)\ (\text{Num}\ m))))$ 
  by(nprver2  $r1: \text{prv\_exiI}[of\ \_\ \_\ \text{Num}\ i]$   $r2: \text{prv\_prv\_eql\_sym}$ )
  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]:  $t \in \text{atrm}$ 
shows  $\text{prv}\ (\text{LLq}\ \text{zer}\ t)$ 
proof-
  define  $z$  where  $z \equiv \text{getFr}\ [xx,yy]\ [t]\ []$ 
  have  $z\_facts[simp]: z \in \text{var}\ z \neq xx\ z \neq yy\ z \notin \text{FvarsT}\ t$ 
  using  $\text{getFr\_FvarsT\_Fvars}[of\ [xx,yy]\ [t]\ []]$  unfolding  $z\_def[symmetric]$  by auto
  have  $\text{prv}\ (\text{exi}\ z\ (\text{eql}\ t\ (\text{pls}\ (\text{Var}\ z)\ \text{zer})))$ 
  apply(nrule  $r: \text{nprv\_prvI}$ )
  apply(nrule  $r: \text{nprv\_exiI}[of\ \_\ \_\ t]$ )
  apply(nrule  $r: \text{nprv\_eql\_symI}$ )
  apply(nrule  $r: \text{prv\_nprvI}$ )
  apply(nrule  $r: \text{prv\_pls\_zer}$ ) .
  thus ?thesis by (simp add: LLq_pls[of  $\_\ \_\ z$ ])

```

qed

lemmas $Q1 = \text{prv_LLq_zer}$

lemma $\text{prv_LLq_zer_imp_eql}$:

assumes $[\text{simp}]$: $t \in \text{atrm}$

shows $\text{prv} (\text{imp} (\text{LLq } t \text{ zer}) (\text{eql } t \text{ zer}))$

proof-

define y where $y \equiv \text{getFr } [] [t] []$

have $y_facts[\text{simp}]$: $y \in \text{var } y \notin \text{FvarsT } t$

using $\text{getFr_FvarsT_Fvars}[\text{of } [] [t] []]$ **unfolding** $y_def[\text{symmetric}]$ **by auto**

define z where $z \equiv \text{getFr } [y] [t] []$

have $z_facts[\text{simp}]$: $z \in \text{var } z \neq y \ z \notin \text{FvarsT } t$

using $\text{getFr_FvarsT_Fvars}[\text{of } [y] [t] []]$ **unfolding** $z_def[\text{symmetric}]$ **by auto**

define x where $x \equiv \text{getFr } [y,z] [t] []$

have $x_facts[\text{simp}]$: $x \in \text{var } x \neq y \ x \neq z \ x \notin \text{FvarsT } t$

using $\text{getFr_FvarsT_Fvars}[\text{of } [y,z] [t] []]$ **unfolding** x_def **by auto**

note $\text{LLq_pls}[\text{of } _ _ z, \text{simp}]$

show $?thesis$

apply($\text{nrule } r$: nprv_prvI)

apply($\text{nrule } r$: nprv_impI)

apply($\text{nrule } r$: $\text{nprv_zer_suc_casesE}[\text{of } t _ _ y]$)

apply($\text{nrule } r$: $\text{nprv_exiE0}[\text{of } z \text{ eql } \text{zer} (\text{pls } (\text{Var } z) t)]$)

apply($\text{nrule } r$: nprv_clear3_3)

apply($\text{nrule } r$: $\text{nprv_eql_symE0}[\text{of } t]$)

apply($\text{nrule } r$: $\text{nprv_eql_substE01}[\text{of } \text{suc } (\text{Var } y) t _ \text{eql } \text{zer} (\text{pls } (\text{Var } z) (\text{Var } x)) x]$)

apply($\text{nrule } r$: $\text{nprv_addLemmaE}[\text{OF } \text{prv_pls_suc}[\text{of } \text{Var } z \text{ Var } y, \text{simplified}]]$)

apply($\text{nrule } r$: $\text{nprv_eql_transE01}[\text{of } \text{zer } \text{pls } (\text{Var } z) (\text{suc } (\text{Var } y)) _ \text{suc } (\text{pls } (\text{Var } z) (\text{Var } y))]$)

apply($\text{nrule } r$: $\text{nprv_zer_suc_contrE0}[\text{of } \text{pls } (\text{Var } z) (\text{Var } y)]$) .

qed

lemmas $\text{nprv_LLq_zer_eqlI} = \text{nprv_addImpLemmaI}[\text{OF } \text{prv_LLq_zer_imp_eql}, \text{simped}, \text{rotated } 3]$

lemmas $\text{nprv_LLq_zer_eqlE} = \text{nprv_addImpLemmaE}[\text{OF } \text{prv_LLq_zer_imp_eql}, \text{simped}, \text{rotated } 1]$

lemmas $\text{nprv_LLq_zer_eqlE0} = \text{nprv_LLq_zer_eqlE}[\text{OF } \text{nprv_hyp } _ , \text{simped}]$

lemmas $\text{nprv_LLq_zer_eqlE1} = \text{nprv_LLq_zer_eqlE}[\text{OF } _ \text{nprv_hyp}, \text{simped}]$

lemmas $\text{nprv_LLq_zer_eqlE01} = \text{nprv_LLq_zer_eqlE}[\text{OF } \text{nprv_hyp } \text{nprv_hyp}, \text{simped}]$

lemma $\text{prv_dsj_eql_imp_LLq}$: — O2

assumes $[\text{simp}]$: $t \in \text{atrm}$

shows $\text{prv} (\text{imp} (\text{dsj} (\text{map } (\lambda i. \text{eql } t (\text{Num } i)) (\text{toN } n))) (\text{LLq } t (\text{Num } n)))$

proof-

define z where $z \equiv \text{getFr } [xx,yy] [t] []$

have $z_facts[\text{simp}]$: $z \in \text{var } z \neq xx \ z \neq yy \ z \notin \text{FvarsT } t$

using $\text{getFr_FvarsT_Fvars}[\text{of } [xx,yy] [t] []]$ **unfolding** $z_def[\text{symmetric}]$ **by auto**

note $\text{imp}[\text{rule } \text{del}, \text{intro!}]$ **note** $\text{dsj}[\text{intro!}]$

show $?thesis$

apply($\text{nrule } r$: nprv_prvI)

apply($\text{nrule } r$: nprv_impI)

apply($\text{nrule } r$: nprv_dsjE0)

subgoal for i

apply($\text{nrule } r$: $\text{nprv_eql_substE}[\text{of } _ t \text{ Num } i \text{ LLq } (\text{Var } z) (\text{Num } n) z]$)

subgoal by ($\text{nrule } r$: nprv_hyp)

subgoal by ($\text{nprv_clear3 } r1$: $\text{nprv_addLemmaE}[\text{OF } \text{leq_prv_LLq_Num}]$)

subgoal by ($\text{nrule } r$: 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)) (lds (map (λi. 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 ∈ (λx. eql t (Num x)) ‘ {0..Suc n} by (force simp: image_def)
  have [simp]: ∧i. i ≤ n ⇒
    eql (suc (Var z)) (suc (Num i)) ∈ (λx. 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))) (lds (map (λi. 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 $prv_LLq_sdsj_eqlE0 = prv_LLq_sdsj_eqlE[OF\ nprv_hyp_ ,\ simpd]$
lemmas $prv_LLq_sdsj_eqlE1 = prv_LLq_sdsj_eqlE[OF_ \ nprv_hyp,\ simpd]$
lemmas $prv_LLq_sdsj_eqlE01 = prv_LLq_sdsj_eqlE[OF\ nprv_hyp\ nprv_hyp,\ simpd]$

lemmas $O3I = prv_LLq_sdsj_eqlI$
lemmas $O3E = prv_LLq_sdsj_eqlE$
lemmas $O3E0 = prv_LLq_sdsj_eqlE0$
lemmas $O3E1 = prv_LLq_sdsj_eqlE1$
lemmas $O3E01 = prv_LLq_sdsj_eqlE01$

lemma $not_leq_prv_neg_LLq_Num$:

assumes $\neg m \leq n$

shows $prv\ (neg\ (LLq\ (Num\ m)\ (Num\ n)))$

proof–

have $[simp]: \bigwedge i. i \leq n \implies prv\ (imp\ (eql\ (Num\ m)\ (Num\ i))\ fls)$

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 $(nrule\ r:\ prv_nprvI1)$.

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 $[simp]: \bigwedge i. i \leq n \implies prv\ (subst\ \varphi\ (Num\ i)\ x)$

shows $prv\ (ball\ x\ (Num\ n)\ \varphi)$

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 $(nrule\ r:\ nprv_eql_substE[of_ \ Var\ x\ Num\ _ \ \varphi\ x])$

apply $(nrule\ r:\ prv_nprvI)$.

lemmas $O4 = prv_ball_NumI$

lemma prv_bexi_NumI : — O5

assumes $[simp]: x \in var\ \varphi \in fmla$

and $[simp]: i \leq n\ prv\ (subst\ \varphi\ (Num\ i)\ x)$

shows $prv\ (bexi\ x\ (Num\ n)\ \varphi)$

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]: $t \in \text{atrm}$

shows prv (imp (LLq t (Num n)) (LLq t (suc (Num n))))

proof—

have [simp]: $\bigwedge i. i \leq n \implies \text{prv} (\text{LLq} (\text{Num } i) (\text{suc} (\text{Num } n)))$

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]: $t \in \text{atrm}$

shows prv (LLq (suc (Num n)) (pls (suc t) (Num n)))

proof—

define z where $z \equiv \text{getFr} [xx,yy,zz] [t] []$

have z_facts[simp]: $z \in \text{var } z \neq xx \ z \neq yy \ z \neq zz \ z \neq z \ z \notin \text{FvarsT } t$

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 (eq (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 eq (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_eqlE0)
  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: nprv_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(nrule 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)  $\psi \implies$ 
nprv (insert (LLq (Num n) t) F)  $\psi \implies$ 
t ∈ atrm  $\implies$  F ⊆ fmla  $\implies$  finite F  $\implies$   $\psi \in$  fmla  $\implies$ 
nprv F  $\psi$ 
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 $O8E = \text{prv_LLq_Num_casesE}$
lemmas $O8E0 = \text{prv_LLq_Num_casesE0}$
lemmas $O8E1 = \text{prv_LLq_Num_casesE1}$
lemmas $O8E01 = \text{prv_LLq_Num_casesE01}$

lemma $\text{prv_imp_LLq_neg_Num_suc}$:
assumes $[\text{simp}]$: $t \in \text{atrm}$
shows prv (imp (LLq t (suc (Num n))))
 $(\text{imp} ((\text{neg} (\text{eql } t (\text{suc} (\text{Num } n))))$
 $(\text{LLq } t (\text{Num } n))))$
apply($\text{nrule } r$: nprv_prvI)
apply($\text{nrule } r$: nprv_impI)
apply($\text{nrule } r$: nprv_impI)
apply($\text{nrule } r$: $O3E0[\text{of } t \text{ Suc } n]$)
apply($\text{nrule } r$: nprv_clear3_3)
apply($\text{nrule } r$: nprv_ldsje0)
subgoal for i
apply($\text{nrule } r$: nprv_clear3_2)
apply($\text{nrule } r$: $\text{nprv_impI_rev2}[\text{of } \{ \text{eql } t (\text{Num } i) \} \text{ neg } (\text{eql } t (\text{suc} (\text{Num } n)))]$)
apply($\text{nrule } r$: $\text{nprv_eql_substE}[\text{of } _ t \text{ Num } i$
 $\text{imp } (\text{neg } (\text{eql } (\text{Var } xx) (\text{suc} (\text{Num } n)))) (\text{LLq } (\text{Var } xx) (\text{Num } n)) \text{ xx}]$)
apply($\text{nrule } r$: nprv_clear)
apply($\text{nrule } r$: nprv_impI)
apply($\text{cases } i = \text{Suc } n$)
subgoal by ($\text{nprover2 } r1$: nprv_negE0 $r2$: nprv_eql_reflI)
subgoal by ($\text{nprover2 } r1$: prv_nprvI $r2$: leq_prv_LLq_Num) . .

lemmas $\text{nprv_LLq_neg_Num_sucI} = \text{nprv_addImp2LemmaI}[\text{OF } \text{prv_imp_LLq_neg_Num_suc}, \text{simped}, \text{rotated } 3]$
lemmas $\text{nprv_LLq_neg_Num_sucE} = \text{nprv_addImp2LemmaE}[\text{OF } \text{prv_imp_LLq_neg_Num_suc}, \text{simped}, \text{rotated } 1]$

lemmas $\text{nprv_LLq_neg_Num_sucE0} = \text{nprv_LLq_neg_Num_sucE}[\text{OF } \text{nprv_hyp } _ _, \text{simped}]$
lemmas $\text{nprv_LLq_neg_Num_sucE1} = \text{nprv_LLq_neg_Num_sucE}[\text{OF } _ \text{nprv_hyp } _ _, \text{simped}]$
lemmas $\text{nprv_LLq_neg_Num_sucE2} = \text{nprv_LLq_neg_Num_sucE}[\text{OF } _ _ \text{nprv_hyp}, \text{simped}]$
lemmas $\text{nprv_LLq_neg_Num_sucE01} = \text{nprv_LLq_neg_Num_sucE}[\text{OF } \text{nprv_hyp } \text{nprv_hyp } _ _, \text{simped}]$
lemmas $\text{nprv_LLq_neg_Num_sucE02} = \text{nprv_LLq_neg_Num_sucE}[\text{OF } \text{nprv_hyp } _ \text{nprv_hyp}, \text{simped}]$
lemmas $\text{nprv_LLq_neg_Num_sucE12} = \text{nprv_LLq_neg_Num_sucE}[\text{OF } _ \text{nprv_hyp } \text{nprv_hyp}, \text{simped}]$
lemmas $\text{nprv_LLq_neg_Num_sucE012} = \text{nprv_LLq_neg_Num_sucE}[\text{OF } \text{nprv_hyp } \text{nprv_hyp } \text{nprv_hyp}, \text{simped}]$

lemma $\text{prv_ball_Num_imp_ball_suc}$: — O9
assumes $[\text{simp}]$: $x \in \text{var } \varphi \in \text{fmla}$
shows prv (imp ($\text{ball } x$ ($\text{Num } n$) φ)
 $(\text{ball } x (\text{suc} (\text{Num } n)) (\text{imp} (\text{neg} (\text{eql} (\text{Var } x) (\text{suc} (\text{Num } n)))) \varphi))$
apply($\text{nrule } r$: nprv_prvI)
apply($\text{nrule } r$: nprv_impI)
apply($\text{nrule } r$: nprv_ballI)
apply($\text{nrule } r$: nprv_impI)
apply($\text{nrule } r$: $\text{nprv_LLq_neg_Num_sucE01}$)
apply($\text{nrule } r$: nprv_clear4_2)
apply($\text{nrule } r$: nprv_clear3_2)
apply($\text{nrule } r$: $\text{nprv_ballE0}[\text{of } x \text{ Num } n \varphi _ \text{Var } x]$) .

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*]: $\varphi \in \text{fmla}$ *Fvars* $\varphi = \{zz\}$ **and** $q: q \in \text{num}$ **and** $p: \forall p \in \text{num}. \text{prv}(\text{subst } \varphi \text{ } p \text{ } zz)$
shows *prv* (*all zz (imp (LLq (Var zz) q) φ)*)

proof–

obtain *n* **where** $q: q = \text{Num } n$ **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 $p \in \text{num}$

shows $\exists P \subseteq \text{num}. \text{finite } P \wedge \text{prv}(\text{dsj}(\text{sdsj}\{ \text{eql}(\text{Var } yy) \text{ } r \mid r. r \in P\}) (\text{LLq } p (\text{Var } yy)))$

proof–

obtain *n* **where** q [*simp*]: $p = \text{Num } n$ **using** *assms num_Num* **by** *auto*

have [*simp*]: $\{ \text{eql}(\text{Var } yy) \text{ } r \mid r. \exists i. r = \text{Num } i \wedge i \leq n \} \subseteq \text{fmla}$ **by** *auto*

show *?thesis*

apply(*nrule* $r: \text{exI}[of_ \{ \text{Num } i \mid i. i \leq n \}]$)

apply(*nrule* $r: \text{nprv_prvI}$)

apply(*nrule* $r: \text{O8E}[of \text{Var } yy \text{ } n]$)

subgoal

apply(*nrule* $r: \text{nprv_dsjIL}$)

apply(*nrule* $r: \text{O3E0}$)

apply(*nrule* $r: \text{nprv_ldsje0}$)

apply(*nrule* $r: \text{nprv_sdsjI}[of_ \text{eql}(\text{Var } yy) (\text{Num } _)]$)

apply(*nrule* $r: \text{nprv_hyp}$) .

subgoal **by** (*nrule* $r: \text{nprv_dsjIR}$) .

qed

end — context *Deduct_Q*

sublocale *Deduct_Q* < *lab: Deduct_with_PseudoOrder* **where** $Lq = \text{LLq}(\text{Var } zz) (\text{Var } yy)$

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.