Robinson Arithmetic

Andrei Popescu Dmitriy Traytel

March 17, 2025

Abstract

We instantiate our syntax-independent logic infrastructure developed in a separate AFP entry to the FOL theory of Robinson arithmetic (also known as Q). The latter was formalised using Nominal Isabelle by adapting Larry Paulsons formalization of the Hereditarily Finite Set theory.

Contents

1 Terms and Formulas

nat is a pure permutation type

instance *nat* :: *pure* **by** *standard*

atom_decl *name*

declare *fresh_set_empty* [*simp*]

lemma *supp_name* [*simp*]: **fixes** *i*::*name* **shows** *supp i* = {*atom i*}
by (*rule supp_at_base*)

1.1 The datatypes

nominal_datatype *trm* = *zer* | *Var name* | *suc trm* | *pls trm trm* | *tms trm trm*

nominal_datatype *fmla* =
eql trm trm (**infixr** ‹*EQ*› *150*)
| *dsj fmla fmla* (**infixr** ‹*OR*› *130*)
| *neg fmla*
| *exi x*::*name f*::*fmla* **binds** *x* **in** *f*

eql are atomic formulas; dsj, neg, exi are non-atomic

declare *trm.supp* [*simp*] *fmla.supp* [*simp*]

1.2 Substitution

nominal_function *subst* :: *name* ⇒ *trm* ⇒ *trm* ⇒ *trm*
where
subst i x zer = *zer*
| *subst i x* (*Var k*) = (*if i*=*k then x else Var k*)
| *subst i x* (*suc t*) = *suc* (*subst i x t*)
| *subst i x* (*pls t u*) = *pls* (*subst i x t*) (*subst i x u*)
| *subst i x* (*tms t u*) = *tms* (*subst i x t*) (*subst i x u*)
by (*auto simp*: *eqvt_def subst_graph_aux_def*) (*metis trm.strong_exhaust*)

nominal_termination (*eqvt*)
by *lexicographic_order*

lemma *fresh_subst_if* [*simp*]:
j ♯ *subst i x t* ⟷ (*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
by (*induct t rule*: *trm.induct*) (*auto simp*: *fresh_at_base*)

lemma *forget_subst_trm* [*simp*]: *atom a* ♯ *trm* ⟹ *subst a x trm* = *trm*
by (*induct trm rule*: *trm.induct*) (*simp_all add*: *fresh_at_base*)

lemma *subst_trm_id* [*simp*]: *subst a* (*Var a*) *trm* = *trm*
by (*induct trm rule*: *trm.induct*) *simp_all*

lemma *subst_trm_commute* [*simp*]:
atom j ♯ *trm* ⟹ *subst j u* (*subst i t trm*) = *subst i* (*subst j u t*) *trm*
by (*induct trm rule*: *trm.induct*) (*auto simp*: *fresh_Pair*)

lemma *subst_trm_commute2* [*simp*]:
atom j ♯ *t* ⟹ *atom i* ♯ *u* ⟹ *i* ≠ *j* ⟹ *subst j u* (*subst i t trm*) = *subst i t* (*subst j u trm*)
by (*induct trm rule*: *trm.induct*) *auto*

lemma *repeat_subst_trm* [*simp*]: *subst i u* (*subst i t trm*) = *subst i* (*subst i u t*) *trm*

```
  by (induct trm rule: trm.induct) auto

nominal_function  subst_fmla :: fmla ⇒ name ⇒ trm ⇒ fmla (‹_'(_::=_')› [1000, 0, 0] 200)
  where
    eql:   (eql t u)(i::=x)   = eql  (subst i x t) (subst i x u)
  | dsj: (dsj A B)(i::=x) = dsj (A(i::=x)) (B(i::=x))
  | neg:  (neg A)(i::=x)    = neg  (A(i::=x))
  | exi:   atom j ♯ (i, x) ⟹ (exi j A)(i::=x) = exi j (A(i::=x))
subgoal by (simp add: eqvt_def subst_fmla_graph_aux_def)
subgoal by auto
subgoal by (metis fmla.strong_exhaust fresh_star_insert old.prod.exhaust)
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 by auto
subgoal
  by (simp add: eqvt_at_def fresh_star_def fresh_Pair fresh_at_base)
     (metis flip_at_base_simps(3) flip_fresh_fresh) .

nominal_termination (eqvt)
  by lexicographic_order

lemma size_subst_fmla [simp]: size (A(i::=x)) = size A
  by (nominal_induct A avoiding: i x rule: fmla.strong_induct) auto

lemma forget_subst_fmla [simp]: atom a ♯ A ⟹ A(a::=x) = A
  by (nominal_induct A avoiding: a x rule: fmla.strong_induct) (auto simp: fresh_at_base)

lemma subst_fmla_id [simp]: A(a::=Var a) = A
  by (nominal_induct A avoiding: a rule: fmla.strong_induct) (auto simp: fresh_at_base)

lemma fresh_subst_fmla_if [simp]:
  j ♯ (A(i::=x)) ⟷ (atom i ♯ A ∧ j ♯ A) ∨ (j ♯ x ∧ (j ♯ A ∨ j = atom i))
  by (nominal_induct A avoiding: i x rule: fmla.strong_induct) (auto simp: fresh_at_base)

lemma subst_fmla_commute [simp]:
  atom j ♯ A ⟹ (A(i::=t))(j::=u) = A(i ::= subst j u t)
  by (nominal_induct A avoiding: i j t u rule: fmla.strong_induct) (auto simp: fresh_at_base)

lemma repeat_subst_fmla [simp]: (A(i::=t))(i::=u) = A(i ::= subst i u t)
  by (nominal_induct A avoiding: i t u rule: fmla.strong_induct) auto

lemma subst_fmla_exi_with_renaming:
  atom i' ♯ (A, i, j, t) ⟹ (exi i A)(j ::= t) = exi i' (((i ↔ i') · A)(j ::= t))
  by (rule subst [of exi i' ((i ↔ i') · A) exi i A])
     (auto simp: Abs1_eq_iff flip_def swap_commute)
```

the simplifier cannot apply the rule above, because it introduces a new variable at the right hand side.

```
lemma flip_subst_trm: atom y ♯ t ⟹ (x ↔ y) · t = subst x (Var y) t
apply(nominal_induct t avoiding: x y rule: trm.strong_induct)
by auto
```

lemma *flip_subst_fmla*: *atom y ♯ φ ⟹ (x ↔ y) ∙ φ = φ(x::=Var y)*
apply(*nominal_induct φ avoiding*: *x y rule*: *fmla.strong_induct*)
apply (*auto simp*: *flip_subst_trm*)
using *fresh_at_base*(*2*) **by** *blast*

lemma *exi_ren_subst_fresh*: *atom y ♯ φ ⟹ exi x φ = exi y (φ(x::=Var y))*
using *flip_subst_fmla* **by** *auto*

1.3 Semantics

definition *e0* :: (*name*, *nat*) *finfun* — the null environment
where *e0* ≡ *finfun_const 0*

nominal_function *eval_trm* :: (*name*, *nat*) *finfun* ⇒ *trm* ⇒ *nat*
where
eval_trm e zer = 0
| *eval_trm e* (*Var k*) = *finfun_apply e k*
| *eval_trm e* (*suc t*) = *Suc* (*eval_trm e t*)
| *eval_trm e* (*pls t u*) = *eval_trm e t* + *eval_trm e u*
| *eval_trm e* (*tms t u*) = *eval_trm e t* ∗ *eval_trm e u*
by (*auto simp*: *eqvt_def eval_trm_graph_aux_def*) (*metis trm.strong_exhaust*)

nominal_termination (*eqvt*)
by *lexicographic_order*

nominal_function *eval_fmla* :: (*name*, *nat*) *finfun* ⇒ *fmla* ⇒ *bool*
where
eval_fmla e (*t EQ u*) ⟷ *eval_trm e t* = *eval_trm e u*
| *eval_fmla e* (*A OR B*) ⟷ *eval_fmla e A* ∨ *eval_fmla e B*
| *eval_fmla e* (*neg A*) ⟷ (~ *eval_fmla e A*)
| *atom k ♯ e ⟹ eval_fmla e* (*exi k A*) ⟷ (∃ *x*. *eval_fmla* (*finfun_update e k x*) *A*)
supply [[*simproc del*: *defined_all*]]
apply(*simp add*: *eqvt_def eval_fmla_graph_aux_def*)
apply(*auto del*: *iffI*) [*11*]
apply(*rule_tac y*=*b* **and** *c*=(*a*) **in** *fmla.strong_exhaust*)
apply(*auto simp*: *fresh_star_def*)[*4*]
using [[*simproc del*: *alpha_lst*]] **apply** *clarsimp*
apply(*erule_tac c*=(*ea*) **in** *Abs_lst1_fcb2'*)
apply(*rule pure_fresh*)
apply(*simp add*: *fresh_star_def*)
apply (*simp_all add*: *eqvt_at_def*)
apply (*simp_all add*: *perm_supp_eq*)
done

nominal_termination (*eqvt*)
by *lexicographic_order*

lemma *eval_trm_rename*:
assumes *atom k' ♯ t*
shows *eval_trm* (*finfun_update e k x*) *t* =
eval_trm (*finfun_update e k' x*) ((*k'* ↔ *k*) ∙ *t*)
using *assms*
by (*induct t rule*: *trm.induct*) (*auto simp*: *permute_flip_at*)

lemma *eval_fmla_rename*:
assumes *atom k' ♯ A*
shows *eval_fmla* (*finfun_update e k x*) *A* = *eval_fmla* (*finfun_update e k' x*) ((*k'* ↔ *k*) ∙ *A*)
using *assms*

apply (*nominal_induct A avoiding*: *e k k′ x rule*: *fmla.strong_induct*)
apply (*simp_all add*: *eval_trm_rename*[*symmetric*], *metis*)
apply (*simp add*: *fresh_finfun_update fresh_at_base finfun_update_twist*)
done

lemma *better_ex_eval_fmla*[*simp*]:
eval_fmla e (*exi k A*) ⟷ (∃ *x*. *eval_fmla* (*finfun_update e k x*) *A*)
proof −
obtain *k′*::*name* **where** *k′*: *atom k′* ♯ (*k*, *e*, *A*)
by (*rule obtain_fresh*)
then have *eq*: *exi k′* ((*k′* ↔ *k*) ∙ *A*) = *exi k A*
by (*simp add*: *Abs1_eq_iff flip_def*)
have *eval_fmla e* (*exi k′* ((*k′* ↔ *k*) ∙ *A*)) = (∃ *x*. *eval_fmla* (*finfun_update e k′ x*) ((*k′* ↔ *k*) ∙ *A*))
using *k′* **by** *simp*
also have ... = (∃ *x*. *eval_fmla* (*finfun_update e k x*) *A*)
by (*metis eval_fmla_rename k′ fresh_Pair*)
finally show *?thesis*
by (*metis eq*)
qed

lemma *forget_eval_trm* [*simp*]: *atom i* ♯ *t* ⟹
eval_trm (*finfun_update e i x*) *t* = *eval_trm e t*
by (*induct t rule*: *trm.induct*) (*simp_all add*: *fresh_at_base*)

lemma *forget_eval_fmla* [*simp*]:
atom k ♯ *A* ⟹ *eval_fmla* (*finfun_update e k x*) *A* = *eval_fmla e A*
by (*nominal_induct A avoiding*: *k e rule*: *fmla.strong_induct*)
(*simp_all add*: *fresh_at_base finfun_update_twist*)

lemma *eval_subst_trm*: *eval_trm e* (*subst i t u*) =
eval_trm (*finfun_update e i* (*eval_trm e t*)) *u*
by (*induct u rule*: *trm.induct*) (*auto*)

lemma *eval_subst_fmla*: *eval_fmla e* (*fmla*(*i*::= *t*)) =
eval_fmla (*finfun_update e i* (*eval_trm e t*)) *fmla*
by (*nominal_induct fmla avoiding*: *i t e rule*: *fmla.strong_induct*)
(*simp_all add*: *eval_subst_trm finfun_update_twist fresh_at_base*)

1.4 Derived logical connectives

abbreviation *imp* :: *fmla* ⇒ *fmla* ⇒ *fmla* (**infixr** ‹*IMP*› *125*)
where *imp A B* ≡ *dsj* (*neg A*) *B*

abbreviation *all* :: *name* ⇒ *fmla* ⇒ *fmla*
where *all i A* ≡ *neg* (*exi i* (*neg A*))

1.4.1 Conjunction

definition *cnj* :: *fmla* ⇒ *fmla* ⇒ *fmla* (**infixr** ‹*AND*› *135*)
where *cnj A B* ≡ *neg* (*dsj* (*neg A*) (*neg B*))

lemma *cnj_eqvt* [*eqvt*]: *p* ∙ (*A AND B*) = (*p* ∙ *A*) *AND* (*p* ∙ *B*)
by (*simp add*: *cnj_def*)

lemma *fresh_cnj* [*simp*]: *a* ♯ *A AND B* ⟷ (*a* ♯ *A* ∧ *a* ♯ *B*)
by (*auto simp*: *cnj_def*)

lemma *supp_cnj* [*simp*]: *supp* (*A AND B*) = *supp A* ∪ *supp B*
by (*auto simp*: *cnj_def*)

lemma *size_cnj* [*simp*]: *size* (*A AND B*) = *size A* + *size B* + *4*
by (*simp add*: *cnj_def*)

lemma *cnj_injective_iff* [*iff*]: (*A AND B*) = (*A'* *AND B'*) ⟷ (*A* = *A'* ∧ *B* = *B'*)
by (*auto simp*: *cnj_def*)

lemma *subst_fmla_cnj* [*simp*]: (*A AND B*)(*i*::=*x*) = (*A*(*i*::=*x*)) *AND* (*B*(*i*::=*x*))
by (*auto simp*: *cnj_def*)

lemma *eval_fmla_cnj* [*simp*]: *eval_fmla e* (*cnj A B*) ⟷ (*eval_fmla e A* ∧ *eval_fmla e B*)
by (*auto simp*: *cnj_def*)

1.4.2 If and only if

definition *Iff* :: *fmla* ⇒ *fmla* ⇒ *fmla* (**infixr** ‹*IFF*› *125*)
where *Iff A B* = *cnj* (*imp A B*) (*imp B A*)

lemma *Iff_eqvt* [*eqvt*]: *p* ∙ (*A IFF B*) = (*p* ∙ *A*) *IFF* (*p* ∙ *B*)
by (*simp add*: *Iff_def*)

lemma *fresh_Iff* [*simp*]: *a* ♯ *A IFF B* ⟷ (*a* ♯ *A* ∧ *a* ♯ *B*)
by (*auto simp*: *cnj_def Iff_def*)

lemma *size_Iff* [*simp*]: *size* (*A IFF B*) = *2*∗(*size A* + *size B*) + *8*
by (*simp add*: *Iff_def*)

lemma *Iff_injective_iff* [*iff*]: (*A IFF B*) = (*A'* *IFF B'*) ⟷ (*A* = *A'* ∧ *B* = *B'*)
by (*auto simp*: *Iff_def*)

lemma *subst_fmla_Iff* [*simp*]: (*A IFF B*)(*i*::=*x*) = (*A*(*i*::=*x*)) *IFF* (*B*(*i*::=*x*))
by (*auto simp*: *Iff_def*)

lemma *eval_fmla_Iff* [*simp*]: *eval_fmla e* (*Iff A B*) ⟷ (*eval_fmla e A* ⟷ *eval_fmla e B*)
by (*auto simp*: *Iff_def*)

1.4.3 False

definition *fls* **where** *fls* ≡ *neg* (*zer EQ zer*)

lemma *fls_eqvt* [*eqvt*]: (*p* ∙ *fls*) = *fls*
by (*simp add*: *fls_def*)

lemma *fls_fresh* [*simp*]: *a* ♯ *fls*
by (*simp add*: *fls_def*)

2 Axioms and Theorems

2.1 Logical axioms

inductive_set *boolean_axioms* :: *fmla set*
where
Ident: *A IMP A* ∈ *boolean_axioms*
| *dsjI1*: *A IMP* (*A OR B*) ∈ *boolean_axioms*
| *dsjCont*: (*A OR A*) *IMP A* ∈ *boolean_axioms*
| *dsjAssoc*: (*A OR* (*B OR C*)) *IMP* ((*A OR B*) *OR C*) ∈ *boolean_axioms*
| *dsjcnj*: (*C OR A*) *IMP* (((*neg C*) *OR B*) *IMP* (*A OR B*)) ∈ *boolean_axioms*

lemma *boolean_axioms_hold*: $A \in boolean\_axioms \Longrightarrow eval\_fmla\ e\ A$
by (*induct rule*: *boolean_axioms.induct*, *auto*)

inductive_set *special_axioms* :: *fmla set* **where**
I: $A(i::=x)\ IMP\ (exi\ i\ A) \in special\_axioms$

lemma *special_axioms_hold*: $A \in special\_axioms \Longrightarrow eval\_fmla\ e\ A$
by (*induct rule*: *special_axioms.induct*, *auto*) (*metis eval_subst_fmla*)

lemma *twist_forget_eval_fmla* [*simp*]:
$atom\ j \sharp (i, A)$
$\Longrightarrow eval\_fmla\ (finfun\_update\ (finfun\_update\ (finfun\_update\ e\ i\ x)\ j\ y)\ i\ z)\ A =$
$eval\_fmla\ (finfun\_update\ e\ i\ z)\ A$
by (*metis finfun_update_twice finfun_update_twist forget_eval_fmla fresh_Pair*)

2.2 Concrete variables

declare *Abs_name_inject*[*simp*]

abbreviation
$X0 \equiv Abs\_name\ (Atom\ (Sort\ ''Theory\_Syntax\_Q.name''\ [])\ 0)$

abbreviation
$X1 \equiv Abs\_name\ (Atom\ (Sort\ ''Robinson\_Arithmetic.name''\ [])\ (Suc\ 0))$
— We prefer *Suc 0* because simplification will transform 1 to that form anyway.

abbreviation
$X2 \equiv Abs\_name\ (Atom\ (Sort\ ''Robinson\_Arithmetic.name''\ [])\ 2)$

abbreviation
$X3 \equiv Abs\_name\ (Atom\ (Sort\ ''Robinson\_Arithmetic.name''\ [])\ 3)$

abbreviation
$X4 \equiv Abs\_name\ (Atom\ (Sort\ ''Robinson\_Arithmetic.name''\ [])\ 4)$

2.3 Equality axioms

definition *refl_ax* :: *fmla* **where**
$refl\_ax = Var\ X1\ EQ\ Var\ X1$

lemma *refl_ax_holds*: *eval_fmla e refl_ax*
by (*auto simp*: *refl_ax_def*)

definition *eq_cong_ax* :: *fmla* **where**
$eq\_cong\_ax = ((Var\ X1\ EQ\ Var\ X2)\ AND\ (Var\ X3\ EQ\ Var\ X4))\ IMP$
$((Var\ X1\ EQ\ Var\ X3)\ IMP\ (Var\ X2\ EQ\ Var\ X4))$

lemma *eq_cong_ax_holds*: *eval_fmla e eq_cong_ax*
by (*auto simp*: *cnj_def eq_cong_ax_def*)

definition *syc_cong_ax* :: *fmla* **where**
$syc\_cong\_ax = ((Var\ X1\ EQ\ Var\ X2))\ IMP$
$((suc\ (Var\ X1))\ EQ\ (suc\ (Var\ X2)))$

lemma *syc_cong_ax_holds*: *eval_fmla e syc_cong_ax*
by (*auto simp*: *cnj_def syc_cong_ax_def*)

definition *pls_cong_ax* :: *fmla* **where**
$pls\_cong\_ax = ((Var\ X1\ EQ\ Var\ X2)\ AND\ (Var\ X3\ EQ\ Var\ X4))\ IMP$

((*pls* (*Var X1*) (*Var X3*)) *EQ* (*pls* (*Var X2*) (*Var X4*)))

lemma *pls_cong_ax_holds*: *eval_fmla e pls_cong_ax*
by (*auto simp*: *cnj_def pls_cong_ax_def*)

definition *tms_cong_ax* :: *fmla* **where**
tms_cong_ax = ((*Var X1 EQ Var X2*) *AND* (*Var X3 EQ Var X4*)) *IMP*
((*tms* (*Var X1*) (*Var X3*)) *EQ* (*tms* (*Var X2*) (*Var X4*)))

lemma *tms_cong_ax_holds*: *eval_fmla e tms_cong_ax*
by (*auto simp*: *cnj_def tms_cong_ax_def*)

definition *equality_axioms* :: *fmla set* **where**
equality_axioms = {*refl_ax*, *eq_cong_ax*, *syc_cong_ax*, *pls_cong_ax*, *tms_cong_ax*}

lemma *equality_axioms_hold*: *A* ∈ *equality_axioms* ⟹ *eval_fmla e A*
by (*auto simp*: *equality_axioms_def refl_ax_holds eq_cong_ax_holds*
syc_cong_ax_holds pls_cong_ax_holds tms_cong_ax_holds)

2.4 The Q (Robinson-arithmetic-specific) axioms

definition *Q_axioms* ≡
{*A* | *A X1 X2*.
X1 ≠ *X2* ∧
(*A* = *neg* (*zer EQ suc* (*Var X1*)) ∨
A = *suc* (*Var X1*) *EQ suc* (*Var X2*) *IMP Var X1 EQ Var X2* ∨
A = *Var X2 EQ zer OR exi X1* (*Var X2 EQ suc* (*Var X1*)) ∨
A = *pls* (*Var X1*) *zer EQ Var X1* ∨
A = *pls* (*Var X1*) (*suc* (*Var X2*)) *EQ suc* (*pls* (*Var X1*) (*Var X2*)) ∨
A = *tms* (*Var X1*) *zer EQ zer* ∨
A = *tms* (*Var X1*) (*suc* (*Var X2*)) *EQ pls* (*tms* (*Var X1*) (*Var X2*)) (*Var X1*))}

2.5 The proof system

inductive *nprv* :: *fmla set* ⇒ *fmla* ⇒ *bool* (**infixl** ‹⊢› *55*)
where
Hyp: *A* ∈ *H* ⟹ *H* ⊢ *A*
| *Q*: *A* ∈ *Q_axioms* ⟹ *H* ⊢ *A*
| *Bool*: *A* ∈ *boolean_axioms* ⟹ *H* ⊢ *A*
| *eql*: *A* ∈ *equality_axioms* ⟹ *H* ⊢ *A*
| *Spec*: *A* ∈ *special_axioms* ⟹ *H* ⊢ *A*
| *MP*: *H* ⊢ *A IMP B* ⟹ *H'* ⊢ *A* ⟹ *H* ∪ *H'* ⊢ *B*
| *exiists*: *H* ⊢ *A IMP B* ⟹ *atom i* ♯ *B* ⟹ ∀ *C* ∈ *H*. *atom i* ♯ *C* ⟹ *H* ⊢ (*exi i A*) *IMP B*

2.6 Derived rules of inference

lemma *contraction*: *insert A* (*insert A H*) ⊢ *B* ⟹ *insert A H* ⊢ *B*
by (*metis insert_absorb2*)

lemma *thin_Un*: *H* ⊢ *A* ⟹ *H* ∪ *H'* ⊢ *A*
by (*metis Bool MP boolean_axioms.Ident sup_commute*)

lemma *thin*: *H* ⊢ *A* ⟹ *H* ⊆ *H'* ⟹ *H'* ⊢ *A*
by (*metis Un_absorb1 thin_Un*)

lemma *thin0*: {} ⊢ *A* ⟹ *H* ⊢ *A*
by (*metis sup_bot_left thin_Un*)

lemma *thin1*: *H* ⊢ *B* ⟹ *insert A H* ⊢ *B*

by (*metis subset_insertI thin*)

lemma *thin2*: *insert A1 H* $\vdash$ *B* $\Longrightarrow$ *insert A1* (*insert A2 H*) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *thin3*: *insert A1* (*insert A2 H*) $\vdash$ *B* $\Longrightarrow$ *insert A1* (*insert A2* (*insert A3 H*)) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *thin4*:
insert A1 (*insert A2* (*insert A3 H*)) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4 H*))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate2*: *insert A2* (*insert A1 H*) $\vdash$ *B* $\Longrightarrow$ *insert A1* (*insert A2 H*) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate3*: *insert A3* (*insert A1* (*insert A2 H*)) $\vdash$ *B* $\Longrightarrow$ *insert A1* (*insert A2* (*insert A3 H*)) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate4*:
insert A4 (*insert A1* (*insert A2* (*insert A3 H*))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4 H*))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate5*:
insert A5 (*insert A1* (*insert A2* (*insert A3* (*insert A4 H*)))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5 H*)))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate6*:
insert A6 (*insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5 H*))))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6 H*))))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate7*:
insert A7 (*insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6 H*)))))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert A7 H*)))))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate8*:
insert A8 (*insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert A7 H*)))))))
$\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert A7* (*insert A8 H*)))))))
$\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate9*:
insert A9 (*insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert A7* (*insert A8 H*)))))))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert A7* (*insert A8* (*insert A9 H*)))))))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate10*:
insert A10 (*insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert A7* (*insert A8* (*insert A9 H*))))))))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert A7* (*insert A8* (*insert A9* (*insert A10 H*))))))))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate11*:
insert A11 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 H)))))))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 H)))))))))) ⊢ B
by (*blast intro*: *thin*)

lemma *rotate12*:
insert A12 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 H)))))))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 H)))))))))) ⊢ B
by (*blast intro*: *thin*)

lemma *rotate13*:
insert A13 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 H)))))))))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 (insert A13 H)))))))))))) ⊢ B
by (*blast intro*: *thin*)

lemma *rotate14*:
insert A14 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 (insert A13 H)))))))))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 (insert A13 (insert A14 H)))))))))))) ⊢ B
by (*blast intro*: *thin*)

lemma *rotate15*:
insert A15 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 (insert A13 (insert A14 H)))))))))))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 (insert A13 (insert A14 (insert A15 H)))))))))))))) ⊢ B
by (*blast intro*: *thin*)

lemma *MP_same*: *H ⊢ A IMP B ⟹ H ⊢ A ⟹ H ⊢ B*
by (*metis MP Un_absorb*)

lemma *MP_thin*: *HA ⊢ A IMP B ⟹ HB ⊢ A ⟹ HA ∪ HB ⊆ H ⟹ H ⊢ B*
by (*metis MP_same le_sup_iff thin*)

lemma *MP_null*: *{} ⊢ A IMP B ⟹ H ⊢ A ⟹ H ⊢ B*
by (*metis MP_same thin0*)

lemma *dsj_commute*: *H ⊢ B OR A ⟹ H ⊢ A OR B*
using *dsjcnj* [*of B A B*] *Ident* [*of B*]
by (*metis Bool MP_same*)

lemma *S*: **assumes** *H ⊢ A IMP (B IMP C) H′ ⊢ A IMP B* **shows** *H ∪ H′ ⊢ A IMP C*
proof −
have *H′ ∪ H ⊢ (neg A) OR (C OR (neg A))*
by (*metis Bool MP MP_same boolean_axioms.dsjcnj dsj_commute dsjAssoc assms*)
thus *?thesis*
by (*metis Bool dsj_commute Un_commute MP_same dsjAssoc dsjCont dsjI1*)
qed

lemma *Assume*: *insert A H ⊢ A*

by (*metis Hyp insertI1*)

lemmas *AssumeH* = *Assume Assume* [*THEN rotate2*] *Assume* [*THEN rotate3*] *Assume* [*THEN rotate4*] *Assume* [*THEN rotate5*]
Assume [*THEN rotate6*] *Assume* [*THEN rotate7*] *Assume* [*THEN rotate8*] *Assume* [*THEN rotate9*] *Assume* [*THEN rotate10*]
Assume [*THEN rotate11*] *Assume* [*THEN rotate12*]
declare *AssumeH* [*intro*!]

lemma *imp_triv_I*: $H \vdash B \Longrightarrow H \vdash A\ IMP\ B$
by (*metis Bool dsj_commute MP_same boolean_axioms.dsjI1*)

lemma *dsjAssoc1*: $H \vdash A\ OR\ (B\ OR\ C) \Longrightarrow H \vdash (A\ OR\ B)\ OR\ C$
by (*metis Bool MP_same boolean_axioms.dsjAssoc*)

lemma *dsjAssoc2*: $H \vdash (A\ OR\ B)\ OR\ C \Longrightarrow H \vdash A\ OR\ (B\ OR\ C)$
by (*metis dsjAssoc1 dsj_commute*)

lemma *dsj_commute_imp*: $H \vdash (B\ OR\ A)\ IMP\ (A\ OR\ B)$
using *dsjcnj* [*of B A B*] *Ident* [*of B*]
by (*metis Bool dsjAssoc2 dsj_commute MP_same*)

lemma *dsj_Semicong_1*: $H \vdash A\ OR\ C \Longrightarrow H \vdash A\ IMP\ B \Longrightarrow H \vdash B\ OR\ C$
using *dsjcnj* [*of A C B*]
by (*metis Bool dsj_commute MP_same*)

lemma *imp_imp_commute*: $H \vdash B\ IMP\ (A\ IMP\ C) \Longrightarrow H \vdash A\ IMP\ (B\ IMP\ C)$
by (*metis dsjAssoc1 dsjAssoc2 dsj_Semicong_1 dsj_commute_imp*)

2.7 The deduction theorem

lemma *deduction_Diff*: **assumes** $H \vdash B$ **shows** $H - \{C\} \vdash C\ IMP\ B$
using *assms*
proof (*induct*)
case (*Hyp A H*) **thus** *?case*
by (*metis Bool imp_triv_I boolean_axioms.Ident nprv.Hyp member_remove remove_def*)
next
case (*Q H*) **thus** *?case*
by (*metis imp_triv_I nprv.Q*)
next
case (*Bool A H*) **thus** *?case*
by (*metis imp_triv_I nprv.Bool*)
next
case (*eql A H*) **thus** *?case*
by (*metis imp_triv_I nprv.eql*)
next
case (*Spec A H*) **thus** *?case*
by (*metis imp_triv_I nprv.Spec*)
next
case (*MP H A B H'*)
hence $(H-\{C\}) \cup (H'-\{C\}) \vdash imp\ C\ B$
by (*simp add*: *S*)
thus *?case*
by (*metis Un_Diff*)
next
case (*exiists H A B i*) **show** *?case*
proof (*cases* $C \in H$)
case *True*

hence *atom i ♯ C* **using** *exiists* **by** *auto*
moreover have $H - \{C\} \vdash A\ IMP\ C\ IMP\ B$ **using** *exiists*
by (*metis imp_imp_commute*)
ultimately have $H - \{C\} \vdash (exi\ i\ A)\ IMP\ C\ IMP\ B$ **using** *exiists*
using *nprv.eql*
by (*simp add*: *nprv.exiists*)
thus *?thesis*
by (*metis imp_imp_commute*)
next
case *False*
hence $H - \{C\} = H$ **by** *auto*
thus *?thesis* **using** *exiists*
by (*metis imp_triv_I nprv.exiists*)
qed
qed

theorem *imp_I* [*intro*!]: $insert\ A\ H \vdash B \Longrightarrow H \vdash A\ IMP\ B$
by (*metis Diff_insert_absorb imp_triv_I deduction_Diff insert_absorb*)

lemma *anti_deduction*: $H \vdash A\ IMP\ B \Longrightarrow insert\ A\ H \vdash B$
by (*metis Assume MP_same thin1*)

2.8 Cut rules

lemma *cut*: $H \vdash A \Longrightarrow insert\ A\ H' \vdash B \Longrightarrow H \cup H' \vdash B$
by (*metis MP Un_commute imp_I*)

lemma *cut_same*: $H \vdash A \Longrightarrow insert\ A\ H \vdash B \Longrightarrow H \vdash B$
by (*metis Un_absorb cut*)

lemma *cut_thin*: $HA \vdash A \Longrightarrow insert\ A\ HB \vdash B \Longrightarrow HA \cup HB \subseteq H \Longrightarrow H \vdash B$
by (*metis thin cut*)

lemma *cut0*: $\{\} \vdash A \Longrightarrow insert\ A\ H \vdash B \Longrightarrow H \vdash B$
by (*metis cut_same thin0*)

lemma *cut1*: $\{A\} \vdash B \Longrightarrow H \vdash A \Longrightarrow H \vdash B$
by (*metis cut sup_bot_right*)

lemma *rcut1*: $\{A\} \vdash B \Longrightarrow insert\ B\ H \vdash C \Longrightarrow insert\ A\ H \vdash C$
by (*metis Assume cut1 cut_same rotate2 thin1*)

lemma *cut2*: $[\![\{A,B\} \vdash C;\ H \vdash A;\ H \vdash B]\!] \Longrightarrow H \vdash C$
by (*metis Un_empty_right Un_insert_right cut cut_same*)

lemma *rcut2*: $\{A,B\} \vdash C \Longrightarrow insert\ C\ H \vdash D \Longrightarrow H \vdash B \Longrightarrow insert\ A\ H \vdash D$
by (*metis Assume cut2 cut_same insert_commute thin1*)

lemma *cut3*: $[\![\{A,B,C\} \vdash D;\ H \vdash A;\ H \vdash B;\ H \vdash C]\!] \Longrightarrow H \vdash D$
by (*metis MP_same cut2 imp_I*)

lemma *cut4*: $[\![\{A,B,C,D\} \vdash E;\ H \vdash A;\ H \vdash B;\ H \vdash C;\ H \vdash D]\!] \Longrightarrow H \vdash E$
by (*metis MP_same cut3* [*of B C D*] *imp_I*)

3 Miscellaneous Logical Rules

lemma *dsj_I1*: $H \vdash A \Longrightarrow H \vdash A\ OR\ B$
by (*metis Bool MP_same boolean_axioms.dsjI1*)

lemma *dsj_I2*: $H \vdash B \Longrightarrow H \vdash A\ OR\ B$
by (*metis dsj_commute dsj_I1*)

lemma *Peirce*: $H \vdash (neg\ A)\ IMP\ A \Longrightarrow H \vdash A$
using *dsjcnj* [*of neg A A A*] *dsjCont* [*of A*]
by (*metis Bool MP_same boolean_axioms.Ident*)

lemma *Contra*: $insert\ (neg\ A)\ H \vdash A \Longrightarrow H \vdash A$
by (*metis Peirce imp_I*)

lemma *imp_neg_I*: $H \vdash A\ IMP\ B \Longrightarrow H \vdash A\ IMP\ (neg\ B) \Longrightarrow H \vdash neg\ A$
by (*metis dsjcnj* [*of B neg A neg A*] *dsjCont Bool dsj_commute MP_same*)

lemma *negneg_I*: $H \vdash A \Longrightarrow H \vdash neg\ (neg\ A)$
using *dsjcnj* [*of neg* (*neg A*) *neg A neg* (*neg A*)]
by (*metis Bool Ident MP_same*)

lemma *negneg_D*: $H \vdash neg\ (neg\ A) \Longrightarrow H \vdash A$
by (*metis dsj_I1 Peirce*)

lemma *neg_D*: $H \vdash neg\ A \Longrightarrow H \vdash A \Longrightarrow H \vdash B$
by (*metis imp_neg_I imp_triv_I negneg_D*)

lemma *dsj_neg_1*: $H \vdash A\ OR\ B \Longrightarrow H \vdash neg\ B \Longrightarrow H \vdash A$
by (*metis dsj_I1 dsj_Semicong_1 dsj_commute Peirce*)

lemma *dsj_neg_2*: $H \vdash A\ OR\ B \Longrightarrow H \vdash neg\ A \Longrightarrow H \vdash B$
by (*metis dsj_neg_1 dsj_commute*)

lemma *neg_dsj_I*: $H \vdash neg\ A \Longrightarrow H \vdash neg\ B \Longrightarrow H \vdash neg\ (A\ OR\ B)$
by (*metis Bool dsj_neg_1 MP_same boolean_axioms.Ident dsjAssoc*)

lemma *cnj_I* [*intro*!]: $H \vdash A \Longrightarrow H \vdash B \Longrightarrow H \vdash A\ AND\ B$
by (*metis cnj_def negneg_I neg_dsj_I*)

lemma *cnj_E1*: $H \vdash A\ AND\ B \Longrightarrow H \vdash A$
by (*metis cnj_def Bool dsj_neg_1 negneg_D boolean_axioms.dsjI1*)

lemma *cnj_E2*: $H \vdash A\ AND\ B \Longrightarrow H \vdash B$
by (*metis cnj_def Bool dsj_I2 dsj_neg_2 MP_same dsjAssoc Ident*)

lemma *cnj_commute*: $H \vdash B\ AND\ A \Longrightarrow H \vdash A\ AND\ B$
by (*metis cnj_E1 cnj_E2 cnj_I*)

lemma *cnj_E*: **assumes** $insert\ A\ (insert\ B\ H) \vdash C$ **shows** $insert\ (A\ AND\ B)\ H \vdash C$
apply (*rule cut_same* [**where** *A=A*], *metis cnj_E1 Hyp insertI1*)
by (*metis* (*full_types*) *AssumeH*(*2*) *cnj_E2 assms cut_same* [**where** *A=B*] *insert_commute thin2*)

lemmas *cnj_EH* = *cnj_E cnj_E* [*THEN rotate2*] *cnj_E* [*THEN rotate3*] *cnj_E* [*THEN rotate4*] *cnj_E* [*THEN rotate5*]
cnj_E [*THEN rotate6*] *cnj_E* [*THEN rotate7*] *cnj_E* [*THEN rotate8*] *cnj_E* [*THEN rotate9*] *cnj_E* [*THEN rotate10*]
declare *cnj_EH* [*intro*!]

lemma *neg_I0*: **assumes** $(\bigwedge B.\ atom\ i \sharp B \Longrightarrow insert\ A\ H \vdash B)$ **shows** $H \vdash neg\ A$
by (*meson fls_fresh imp_I imp_neg_I assms fmla.fresh*(*3*))

lemma *neg_mono*: *insert A H ⊢ B ⟹ insert (neg B) H ⊢ neg A*
by (*rule neg_I0*) (*metis Hyp neg_D insert_commute insertI1 thin1*)

lemma *cnj_mono*: *insert A H ⊢ B ⟹ insert C H ⊢ D ⟹ insert (A AND C) H ⊢ B AND D*
by (*metis cnj_E1 cnj_E2 cnj_I Hyp Un_absorb2 cut insertI1 subset_insertI*)

lemma *dsj_mono*:
assumes *insert A H ⊢ B insert C H ⊢ D* **shows** *insert (A OR C) H ⊢ B OR D*
proof −
{ fix *A B C H*
have *insert (A OR C) H ⊢ (A IMP B) IMP C OR B*
by (*metis Bool Hyp MP_same boolean_axioms.dsjcnj insertI1*)
hence *insert A H ⊢ B ⟹ insert (A OR C) H ⊢ C OR B*
by (*metis MP_same Un_absorb Un_insert_right imp_I thin_Un*)
}
thus *?thesis*
by (*metis cut_same assms thin2*)
qed

lemma *dsj_E*:
assumes *A*: *insert A H ⊢ C* **and** *B*: *insert B H ⊢ C* **shows** *insert (A OR B) H ⊢ C*
by (*metis A B dsj_mono negneg_I Peirce*)

lemmas *dsj_EH = dsj_E dsj_E* [*THEN rotate2*] *dsj_E* [*THEN rotate3*] *dsj_E* [*THEN rotate4*] *dsj_E* [*THEN rotate5*]
dsj_E [*THEN rotate6*] *dsj_E* [*THEN rotate7*] *dsj_E* [*THEN rotate8*] *dsj_E* [*THEN rotate9*] *dsj_E* [*THEN rotate10*]
declare *dsj_EH* [*intro*!]

lemma *Contra′*: *insert A H ⊢ neg A ⟹ H ⊢ neg A*
by (*metis Contra neg_mono*)

lemma *negneg_E* [*intro*!]: *insert A H ⊢ B ⟹ insert (neg (neg A)) H ⊢ B*
by (*metis negneg_D neg_mono*)

declare *negneg_E* [*THEN rotate2*, *intro*!]
declare *negneg_E* [*THEN rotate3*, *intro*!]
declare *negneg_E* [*THEN rotate4*, *intro*!]
declare *negneg_E* [*THEN rotate5*, *intro*!]
declare *negneg_E* [*THEN rotate6*, *intro*!]
declare *negneg_E* [*THEN rotate7*, *intro*!]
declare *negneg_E* [*THEN rotate8*, *intro*!]

lemma *imp_E*:
assumes *A*: *H ⊢ A* **and** *B*: *insert B H ⊢ C* **shows** *insert (A IMP B) H ⊢ C*
proof −
have *insert (A IMP B) H ⊢ B*
by (*metis Hyp A thin1 MP_same insertI1*)
thus *?thesis*
by (*metis cut* [**where** *B=C*] *Un_insert_right sup_commute sup_idem B*)
qed

lemma *imp_cut*:
assumes *insert C H ⊢ A IMP B* {*A*} *⊢ C*
shows *H ⊢ A IMP B*
by (*metis Contra dsj_I1 neg_mono assms rcut1*)

lemma *Iff_I* [*intro*!]: *insert A H ⊢ B ⟹ insert B H ⊢ A ⟹ H ⊢ A IFF B*

by (*metis Iff_def cnj_I imp_I*)

lemma *Iff_MP_same*: $H \vdash A\ IFF\ B \Longrightarrow H \vdash A \Longrightarrow H \vdash B$
by (*metis Iff_def cnj_E1 MP_same*)

lemma *Iff_MP2_same*: $H \vdash A\ IFF\ B \Longrightarrow H \vdash B \Longrightarrow H \vdash A$
by (*metis Iff_def cnj_E2 MP_same*)

lemma *Iff_refl* [*intro*!]: $H \vdash A\ IFF\ A$
by (*metis Hyp Iff_I insertI1*)

lemma *Iff_sym*: $H \vdash A\ IFF\ B \Longrightarrow H \vdash B\ IFF\ A$
by (*metis Iff_def cnj_commute*)

lemma *Iff_trans*: $H \vdash A\ IFF\ B \Longrightarrow H \vdash B\ IFF\ C \Longrightarrow H \vdash A\ IFF\ C$
unfolding *Iff_def*
by (*metis cnj_E1 cnj_E2 cnj_I dsj_Semicong_1 dsj_commute*)

lemma *Iff_E*:
$insert\ A\ (insert\ B\ H) \vdash C \Longrightarrow insert\ (neg\ A)\ (insert\ (neg\ B)\ H) \vdash C \Longrightarrow insert\ (A\ IFF\ B)\ H \vdash C$
by (*simp add*: *Assume Iff_def anti_deduction cnj_E dsj_EH*(*2*) *dsj_I1 insert_commute*)

lemma *Iff_E1*:
assumes *A*: $H \vdash A$ **and** *B*: $insert\ B\ H \vdash C$ **shows** $insert\ (A\ IFF\ B)\ H \vdash C$
by (*metis Iff_def A B cnj_E imp_E insert_commute thin1*)

lemma *Iff_E2*:
assumes *A*: $H \vdash A$ **and** *B*: $insert\ B\ H \vdash C$ **shows** $insert\ (B\ IFF\ A)\ H \vdash C$
by (*metis Iff_def A B Bool cnj_E2 cnj_mono imp_E boolean_axioms.Ident*)

lemma *Iff_MP_left*: $H \vdash A\ IFF\ B \Longrightarrow insert\ A\ H \vdash C \Longrightarrow insert\ B\ H \vdash C$
by (*metis Hyp Iff_E2 cut_same insertI1 insert_commute thin1*)

lemma *Iff_MP_left'*: $H \vdash A\ IFF\ B \Longrightarrow insert\ B\ H \vdash C \Longrightarrow insert\ A\ H \vdash C$
by (*metis Iff_MP_left Iff_sym*)

lemma *Swap*: $insert\ (neg\ B)\ H \vdash A \Longrightarrow insert\ (neg\ A)\ H \vdash B$
by (*metis negneg_D neg_mono*)

lemma *Cases*: $insert\ A\ H \vdash B \Longrightarrow insert\ (neg\ A)\ H \vdash B \Longrightarrow H \vdash B$
by (*metis Contra neg_D neg_mono*)

lemma *neg_cnj_E*: $H \vdash B \Longrightarrow insert\ (neg\ A)\ H \vdash C \Longrightarrow insert\ (neg\ (A\ AND\ B))\ H \vdash C$
by (*metis cnj_I Swap thin1*)

lemma *dsj_CI*: $insert\ (neg\ B)\ H \vdash A \Longrightarrow H \vdash A\ OR\ B$
by (*metis Contra dsj_I1 dsj_I2 Swap*)

lemma *dsj_3I*: $insert\ (neg\ A)\ (insert\ (neg\ C)\ H) \vdash B \Longrightarrow H \vdash A\ OR\ B\ OR\ C$
by (*metis dsj_CI dsj_commute insert_commute*)

lemma *Contrapos1*: $H \vdash A\ IMP\ B \Longrightarrow H \vdash neg\ B\ IMP\ neg\ A$
by (*metis Bool MP_same boolean_axioms.dsjcnj boolean_axioms.Ident*)

lemma *Contrapos2*: $H \vdash (neg\ B)\ IMP\ (neg\ A) \Longrightarrow H \vdash A\ IMP\ B$
by (*metis Bool MP_same boolean_axioms.dsjcnj boolean_axioms.Ident*)

lemma *ContraAssumeN* [*intro*]: $B \in H \Longrightarrow insert\ (neg\ B)\ H \vdash A$

by (*metis Hyp Swap thin1*)

lemma *ContraAssume*: *neg B* ∈ *H* ⟹ *insert B H* ⊢ *A*
by (*metis dsj_I1 Hyp anti_deduction*)

lemma *ContraProve*: *H* ⊢ *B* ⟹ *insert* (*neg B*) *H* ⊢ *A*
by (*metis Swap thin1*)

lemma *dsj_IE1*: *insert B H* ⊢ *C* ⟹ *insert* (*A OR B*) *H* ⊢ *A OR C*
by (*metis Assume dsj_mono*)

lemmas *dsj_IE1H* = *dsj_IE1 dsj_IE1* [*THEN rotate2*] *dsj_IE1* [*THEN rotate3*] *dsj_IE1* [*THEN rotate4*] *dsj_IE1* [*THEN rotate5*]
dsj_IE1 [*THEN rotate6*] *dsj_IE1* [*THEN rotate7*] *dsj_IE1* [*THEN rotate8*]
declare *dsj_IE1H* [*intro*!]

3.1 Quantifier reasoning

lemma *exi_I*: *H* ⊢ *A*(*i*::=*x*) ⟹ *H* ⊢ *exi i A*
by (*metis MP_same Spec special_axioms.intros*)

lemma *exi_E*:
assumes *insert A H* ⊢ *B atom i* ♯ *B* ∀ *C* ∈ *H*. *atom i* ♯ *C*
shows *insert* (*exi i A*) *H* ⊢ *B*
by (*metis exiists imp_I anti_deduction assms*)

lemma *exi_E_with_renaming*:
assumes *insert* ((*i* ↔ *i*′) · *A*) *H* ⊢ *B atom i*′ ♯ (*A*,*i*,*B*) ∀ *C* ∈ *H*. *atom i*′ ♯ *C*
shows *insert* (*exi i A*) *H* ⊢ *B*
proof −
have *exi i A* = *exi i*′ ((*i* ↔ *i*′) · *A*)
using *assms* **using** *flip_subst_fmla* **by** *auto*
thus *?thesis*
by (*metis exi_E assms fresh_Pair*)
qed

lemmas *exi_EH* = *exi_E exi_E* [*THEN rotate2*] *exi_E* [*THEN rotate3*] *exi_E* [*THEN rotate4*] *exi_E* [*THEN rotate5*]
exi_E [*THEN rotate6*] *exi_E* [*THEN rotate7*] *exi_E* [*THEN rotate8*] *exi_E* [*THEN rotate9*] *exi_E* [*THEN rotate10*]
declare *exi_EH* [*intro*!]

lemma *exi_mono*: *insert A H* ⊢ *B* ⟹ ∀ *C* ∈ *H*. *atom i* ♯ *C* ⟹ *insert* (*exi i A*) *H* ⊢ (*exi i B*)
by (*auto simp add*: *intro*: *exi_I* [**where** *x*=*Var i*])

lemma *all_I* [*intro*!]: *H* ⊢ *A* ⟹ ∀ *C* ∈ *H*. *atom i* ♯ *C* ⟹ *H* ⊢ *all i A*
by (*auto intro*: *ContraProve neg_I0*)

lemma *all_D*: *H* ⊢ *all i A* ⟹ *H* ⊢ *A*(*i*::=*x*)
by (*metis Assume exi_I negneg_D neg_mono neg cut_same*)

lemma *all_E*: *insert* (*A*(*i*::=*x*)) *H* ⊢ *B* ⟹ *insert* (*all i A*) *H* ⊢ *B*
by (*metis exi_I negneg_D neg_mono neg*)

lemma *all_E*′: *H* ⊢ *all i A* ⟹ *insert* (*A*(*i*::=*x*)) *H* ⊢ *B* ⟹ *H* ⊢ *B*
by (*metis all_D cut_same*)

3.2 Congruence rules

lemma *neg_cong*: $H \vdash A\ IFF\ A' \Longrightarrow H \vdash neg\ A\ IFF\ neg\ A'$
by (*metis Iff_def cnj_E1 cnj_E2 cnj_I Contrapos1*)

lemma *dsj_cong*: $H \vdash A\ IFF\ A' \Longrightarrow H \vdash B\ IFF\ B' \Longrightarrow H \vdash A\ OR\ B\ IFF\ A'\ OR\ B'$
by (*metis cnj_E1 cnj_E2 dsj_mono Iff_I Iff_def anti_deduction*)

lemma *cnj_cong*: $H \vdash A\ IFF\ A' \Longrightarrow H \vdash B\ IFF\ B' \Longrightarrow H \vdash A\ AND\ B\ IFF\ A'\ AND\ B'$
by (*metis cnj_def dsj_cong neg_cong*)

lemma *imp_cong*: $H \vdash A\ IFF\ A' \Longrightarrow H \vdash B\ IFF\ B' \Longrightarrow H \vdash (A\ IMP\ B)\ IFF\ (A'\ IMP\ B')$
by (*metis dsj_cong neg_cong*)

lemma *Iff_cong*: $H \vdash A\ IFF\ A' \Longrightarrow H \vdash B\ IFF\ B' \Longrightarrow H \vdash (A\ IFF\ B)\ IFF\ (A'\ IFF\ B')$
by (*metis Iff_def cnj_cong imp_cong*)

lemma *exi_cong*: $H \vdash A\ IFF\ A' \Longrightarrow \forall C \in H.\ atom\ i \sharp C \Longrightarrow H \vdash (exi\ i\ A)\ IFF\ (exi\ i\ A')$
apply (*rule Iff_I*)
apply (*metis exi_mono Hyp Iff_MP_same Un_absorb Un_insert_right insertI1 thin_Un*)
apply (*metis exi_mono Hyp Iff_MP2_same Un_absorb Un_insert_right insertI1 thin_Un*)
done

lemma *all_cong*: $H \vdash A\ IFF\ A' \Longrightarrow \forall C \in H.\ atom\ i \sharp C \Longrightarrow H \vdash (all\ i\ A)\ IFF\ (all\ i\ A')$
by (*metis exi_cong neg_cong*)

lemma *Subst*: $H \vdash A \Longrightarrow \forall B \in H.\ atom\ i \sharp B \Longrightarrow H \vdash A\ (i::=x)$
by (*metis all_D all_I*)

4 Eqluality Reasoning

4.1 The congruence property for (EQ), and other basic properties of equality

lemma *eql_cong1*: $\{\} \vdash (t\ EQ\ t'\ AND\ u\ EQ\ u')\ IMP\ (t\ EQ\ u\ IMP\ t'\ EQ\ u')$
proof −
obtain *v2*::*name* **and** *v3*::*name* **and** *v4*::*name*
where *v2*: *atom v2* $\sharp$ (*t*,*X1*,*X3*,*X4*)
and *v3*: *atom v3* $\sharp$ (*t*,*t'*,*X1*,*v2*,*X4*)
and *v4*: *atom v4* $\sharp$ (*t*,*t'*,*u*,*X1*,*v2*,*v3*)
by (*metis obtain_fresh*)
have $\{\} \vdash$ (*Var X1 EQ Var X2 AND Var X3 EQ Var X4*) *IMP* (*Var X1 EQ Var X3 IMP Var X2 EQ Var X4*)
by (*rule eql*) (*simp add*: *eq_cong_ax_def equality_axioms_def*)
hence $\{\} \vdash$ (*Var X1 EQ Var X2 AND Var X3 EQ Var X4*) *IMP* (*Var X1 EQ Var X3 IMP Var X2 EQ Var X4*)
by (*drule_tac i=X1* **and** *x=Var X1* **in** *Subst*) *simp_all*
hence $\{\} \vdash$ (*Var X1 EQ Var v2 AND Var X3 EQ Var X4*) *IMP* (*Var X1 EQ Var X3 IMP Var v2 EQ Var X4*)
by (*drule_tac i=X2* **and** *x=Var v2* **in** *Subst*) *simp_all*
hence $\{\} \vdash$ (*Var X1 EQ Var v2 AND Var v3 EQ Var X4*) *IMP* (*Var X1 EQ Var v3 IMP Var v2 EQ Var X4*)
using *v2*
by (*drule_tac i=X3* **and** *x=Var v3* **in** *Subst*) *simp_all*
hence $\{\} \vdash$ (*Var X1 EQ Var v2 AND Var v3 EQ Var v4*) *IMP* (*Var X1 EQ Var v3 IMP Var v2 EQ Var v4*)
using *v2 v3*
by (*drule_tac i=X4* **and** *x=Var v4* **in** *Subst*) *simp_all*

```
  hence {} ⊢ (t EQ Var v2 AND Var v3 EQ Var v4) IMP (t EQ Var v3 IMP Var v2 EQ Var v4)
    using v2 v3 v4
    by (drule_tac i=X1 and x=t in Subst) simp_all
  hence {} ⊢ (t EQ t' AND Var v3 EQ Var v4) IMP (t EQ Var v3 IMP t' EQ Var v4)
    using v2 v3 v4
    by (drule_tac i=v2 and x=t' in Subst) simp_all
  hence {} ⊢ (t EQ t' AND u EQ Var v4) IMP (t EQ u IMP t' EQ Var v4)
    using v3 v4
    by (drule_tac i=v3 and x=u in Subst) simp_all
  thus ?thesis
    using v4
    by (drule_tac i=v4 and x=u' in Subst) simp_all
qed

lemma Refl [iff]: H ⊢ t EQ t
proof −
  have {} ⊢ Var X1 EQ Var X1
    by (rule eql) (simp add: equality_axioms_def refl_ax_def)
  hence {} ⊢ t EQ t
    by (drule_tac i=X1 and x=t in Subst) simp_all
  thus ?thesis
    by (metis empty_subsetI thin)
qed
```

Apparently necessary in order to prove the congruence property.

```
lemma Sym: assumes H ⊢ t EQ u shows H ⊢ u EQ t
proof −
  have {} ⊢  (t EQ u AND t EQ t) IMP (t EQ t IMP u EQ t)
    by (rule eql_cong1)
   moreover have {t EQ u} ⊢ t EQ u AND t EQ t
    by (metis Assume cnj_I Refl)
  ultimately have {t EQ u} ⊢ u EQ t
    by (metis MP_same MP Refl sup_bot_left)
  thus H ⊢ u EQ t by (metis assms cut1)
qed

lemma Sym_L: insert (t EQ u) H ⊢ A ⟹ insert (u EQ t) H ⊢ A
  by (metis Assume Sym Un_empty_left Un_insert_left cut)

lemma Trans: assumes H ⊢ x EQ y H ⊢ y EQ z shows H ⊢ x EQ z
proof −
  have ⋀H. H ⊢ (x EQ x AND y EQ z) IMP (x EQ y IMP x EQ z)
    by (metis eql_cong1 bot_least thin)
  moreover have {x EQ y, y EQ z} ⊢ x EQ x AND y EQ z
    by (metis Assume cnj_I Refl thin1)
  ultimately have {x EQ y, y EQ z} ⊢ x EQ z
    by (metis Hyp MP_same insertI1)
  thus ?thesis
    by (metis assms cut2)
qed

lemma eql_cong:
  assumes H ⊢ t EQ t' H ⊢ u EQ u' shows H ⊢ t EQ u IFF t' EQ u'
proof −
  { fix t t' u u'
    assume  H ⊢ t EQ t' H ⊢ u EQ u'
    moreover have {t EQ t', u EQ u'} ⊢ t EQ u IMP t' EQ u' using eql_cong1
      by (metis Assume cnj_I MP_null insert_commute)
```

ultimately have $H \vdash t\ EQ\ u\ IMP\ t'\ EQ\ u'$
by (*metis cut2*)
}
thus *?thesis*
by (*metis Iff_def cnj_I assms Sym*)
qed

lemma *eql_Trans_E*: $H \vdash x\ EQ\ u \Longrightarrow insert\ (t\ EQ\ u)\ H \vdash A \Longrightarrow insert\ (x\ EQ\ t)\ H \vdash A$
by (*metis Assume Sym_L Trans cut_same thin1 thin2*)

4.2 The congruence properties for *suc*, *pls* and *tms*

lemma *suc_cong1*: $\{\} \vdash (t\ EQ\ t')\ IMP\ (suc\ t\ EQ\ suc\ t')$
proof −
obtain *v2*::*name* **and** *v3*::*name* **and** *v4*::*name*
where *v2*: $atom\ v2 \sharp (t,X1,X3,X4)$
and *v3*: $atom\ v3 \sharp (t,t',X1,v2,X4)$
and *v4*: $atom\ v4 \sharp (t,t',X1,v2,v3)$
by (*metis obtain_fresh*)
have $\{\} \vdash (Var\ X1\ EQ\ Var\ X2)\ IMP\ (suc\ (Var\ X1)\ EQ\ suc\ (Var\ X2))$
by (*metis syc_cong_ax_def equality_axioms_def insert_iff eql*)
hence $\{\} \vdash (Var\ X1\ EQ\ Var\ v2)\ IMP\ (suc\ (Var\ X1)\ EQ\ suc\ (Var\ v2))$
by (*drule_tac i=X2* **and** *x=Var v2* **in** *Subst*) *simp_all*
hence $\{\} \vdash (t\ EQ\ Var\ v2)\ IMP\ (suc\ t\ EQ\ suc\ (Var\ v2))$
using *v2 v3 v4*
by (*drule_tac i=X1* **and** *x=t* **in** *Subst*) *simp_all*
hence $\{\} \vdash (t\ EQ\ t')\ IMP\ (suc\ t\ EQ\ suc\ t')$
using *v2 v3 v4*
by (*drule_tac i=v2* **and** *x=t'* **in** *Subst*) *simp_all*
thus *?thesis*
using *v4*
by (*drule_tac i=v4* **in** *Subst*) *simp_all*
qed

lemma *suc_cong*: $[\![H \vdash t\ EQ\ t']\!] \Longrightarrow H \vdash suc\ t\ EQ\ suc\ t'$
by (*metis anti_deduction suc_cong1 cut1*)

lemma *pls_cong1*: $\{\} \vdash (t\ EQ\ t'\ AND\ u\ EQ\ u')\ IMP\ (pls\ t\ u\ EQ\ pls\ t'\ u')$
proof −
obtain *v2*::*name* **and** *v3*::*name* **and** *v4*::*name*
where *v2*: $atom\ v2 \sharp (t,X1,X3,X4)$
and *v3*: $atom\ v3 \sharp (t,t',X1,v2,X4)$
and *v4*: $atom\ v4 \sharp (t,t',u,X1,v2,v3)$
by (*metis obtain_fresh*)
have $\{\} \vdash (Var\ X1\ EQ\ Var\ X2\ AND\ Var\ X3\ EQ\ Var\ X4)\ IMP\ (pls\ (Var\ X1)\ (Var\ X3)\ EQ\ pls\ (Var\ X2)\ (Var\ X4))$
by (*metis pls_cong_ax_def equality_axioms_def insert_iff eql*)
hence $\{\} \vdash (Var\ X1\ EQ\ Var\ v2\ AND\ Var\ X3\ EQ\ Var\ X4)\ IMP\ (pls\ (Var\ X1)\ (Var\ X3)\ EQ\ pls\ (Var\ v2)\ (Var\ X4))$
by (*drule_tac i=X2* **and** *x=Var v2* **in** *Subst*) *simp_all*
hence $\{\} \vdash (Var\ X1\ EQ\ Var\ v2\ AND\ Var\ v3\ EQ\ Var\ X4)\ IMP\ (pls\ (Var\ X1)\ (Var\ v3)\ EQ\ pls\ (Var\ v2)\ (Var\ X4))$
using *v2*
by (*drule_tac i=X3* **and** *x=Var v3* **in** *Subst*) *simp_all*
hence $\{\} \vdash (Var\ X1\ EQ\ Var\ v2\ AND\ Var\ v3\ EQ\ Var\ v4)\ IMP\ (pls\ (Var\ X1)\ (Var\ v3)\ EQ\ pls\ (Var\ v2)\ (Var\ v4))$
using *v2 v3*
by (*drule_tac i=X4* **and** *x=Var v4* **in** *Subst*) *simp_all*

hence {} ⊢ (*t EQ Var v2 AND Var v3 EQ Var v4*) *IMP* (*pls t* (*Var v3*) *EQ pls* (*Var v2*) (*Var v4*))
using *v2 v3 v4*
by (*drule_tac i=X1* **and** *x=t* **in** *Subst*) *simp_all*
hence {} ⊢ (*t EQ t′ AND Var v3 EQ Var v4*) *IMP* (*pls t* (*Var v3*) *EQ pls t′* (*Var v4*))
using *v2 v3 v4*
by (*drule_tac i=v2* **and** *x=t′* **in** *Subst*) *simp_all*
hence {} ⊢ (*t EQ t′ AND u EQ Var v4*) *IMP* (*pls t u EQ pls t′* (*Var v4*))
using *v3 v4*
by (*drule_tac i=v3* **and** *x=u* **in** *Subst*) *simp_all*
thus *?thesis*
using *v4*
by (*drule_tac i=v4* **and** *x=u′* **in** *Subst*) *simp_all*
qed

lemma *pls_cong*: ⟦*H* ⊢ *t EQ t′*; *H* ⊢ *u EQ u′*⟧ ⟹ *H* ⊢ *pls t u EQ pls t′ u′*
by (*metis cnj_I anti_deduction pls_cong1 cut1*)

lemma *tms_cong1*: {} ⊢ (*t EQ t′ AND u EQ u′*) *IMP* (*tms t u EQ tms t′ u′*)
proof −
obtain *v2*::*name* **and** *v3*::*name* **and** *v4*::*name*
where *v2*: *atom v2* ♯ (*t*,*X1*,*X3*,*X4*)
and *v3*: *atom v3* ♯ (*t*,*t′*,*X1*,*v2*,*X4*)
and *v4*: *atom v4* ♯ (*t*,*t′*,*u*,*X1*,*v2*,*v3*)
by (*metis obtain_fresh*)
have {} ⊢ (*Var X1 EQ Var X2 AND Var X3 EQ Var X4*) *IMP* (*tms* (*Var X1*) (*Var X3*) *EQ tms* (*Var X2*) (*Var X4*))
by (*metis tms_cong_ax_def equality_axioms_def insert_iff eql*)
hence {} ⊢ (*Var X1 EQ Var v2 AND Var X3 EQ Var X4*) *IMP* (*tms* (*Var X1*) (*Var X3*) *EQ tms* (*Var v2*) (*Var X4*))
by (*drule_tac i=X2* **and** *x=Var v2* **in** *Subst*) *simp_all*
hence {} ⊢ (*Var X1 EQ Var v2 AND Var v3 EQ Var X4*) *IMP* (*tms* (*Var X1*) (*Var v3*) *EQ tms* (*Var v2*) (*Var X4*))
using *v2*
by (*drule_tac i=X3* **and** *x=Var v3* **in** *Subst*) *simp_all*
hence {} ⊢ (*Var X1 EQ Var v2 AND Var v3 EQ Var v4*) *IMP* (*tms* (*Var X1*) (*Var v3*) *EQ tms* (*Var v2*) (*Var v4*))
using *v2 v3*
by (*drule_tac i=X4* **and** *x=Var v4* **in** *Subst*) *simp_all*
hence {} ⊢ (*t EQ Var v2 AND Var v3 EQ Var v4*) *IMP* (*tms t* (*Var v3*) *EQ tms* (*Var v2*) (*Var v4*))
using *v2 v3 v4*
by (*drule_tac i=X1* **and** *x=t* **in** *Subst*) *simp_all*
hence {} ⊢ (*t EQ t′ AND Var v3 EQ Var v4*) *IMP* (*tms t* (*Var v3*) *EQ tms t′* (*Var v4*))
using *v2 v3 v4*
by (*drule_tac i=v2* **and** *x=t′* **in** *Subst*) *simp_all*
hence {} ⊢ (*t EQ t′ AND u EQ Var v4*) *IMP* (*tms t u EQ tms t′* (*Var v4*))
using *v3 v4*
by (*drule_tac i=v3* **and** *x=u* **in** *Subst*) *simp_all*
thus *?thesis*
using *v4*
by (*drule_tac i=v4* **and** *x=u′* **in** *Subst*) *simp_all*
qed

lemma *tms_cong*: ⟦*H* ⊢ *t EQ t′*; *H* ⊢ *u EQ u′*⟧ ⟹ *H* ⊢ *tms t u EQ tms t′ u′*
by (*metis cnj_I anti_deduction tms_cong1 cut1*)

4.3 Substitution for eqlualities

lemma *eql_subst_trm_Iff*: {*t EQ u*} ⊢ *subst i t trm EQ subst i u trm*

by (*induct trm rule*: *trm.induct*) (*auto simp*: *suc_cong pls_cong tms_cong*)

lemma *eql_subst_fmla_Iff*: *insert* (*t EQ u*) *H* ⊢ *A*(*i*::=*t*) *IFF A*(*i*::=*u*)
proof −
have { *t EQ u* } ⊢ *A*(*i*::=*t*) *IFF A*(*i*::=*u*)
by (*nominal_induct A avoiding*: *i t u rule*: *fmla.strong_induct*)
(*auto simp*: *dsj_cong neg_cong exi_cong eql_cong eql_subst_trm_Iff*)
thus *?thesis*
by (*metis Assume cut1*)
qed

lemma *Var_eql_subst_Iff*: *insert* (*Var i EQ t*) *H* ⊢ *A*(*i*::=*t*) *IFF A*
by (*metis eql_subst_fmla_Iff Iff_sym subst_fmla_id*)

lemma *Var_eql_imp_subst_Iff*: *H* ⊢ *Var i EQ t* ⟹ *H* ⊢ *A*(*i*::=*t*) *IFF A*
by (*metis Var_eql_subst_Iff cut_same*)

4.4 Congruence rules for predicates

lemma *P1_cong*:
fixes *tms* :: *trm list*
assumes ⋀*i t x*. *atom i* ♯ *tms* ⟹ (*P t*)(*i*::=*x*) = *P* (*subst i x t*) **and** *H* ⊢ *x EQ x′*
shows *H* ⊢ *P x IFF P x′*
proof −
obtain *i*::*name* **where** *i*: *atom i* ♯ *tms*
by (*metis obtain_fresh*)
have *insert* (*x EQ x′*) *H* ⊢ (*P* (*Var i*))(*i*::=*x*) *IFF* (*P*(*Var i*))(*i*::=*x′*)
by (*rule eql_subst_fmla_Iff*)
thus *?thesis* **using** *assms i*
by (*metis cut_same subst.simps*(*2*))
qed

lemma *P2_cong*:
fixes *tms* :: *trm list*
assumes *sub*: ⋀*i t u x*. *atom i* ♯ *tms* ⟹ (*P t u*)(*i*::=*x*) = *P* (*subst i x t*) (*subst i x u*)
and *eq*: *H* ⊢ *x EQ x′ H* ⊢ *y EQ y′*
shows *H* ⊢ *P x y IFF P x′ y′*
proof −
have *yy′*: { *y EQ y′* } ⊢ *P x′ y IFF P x′ y′*
by (*rule P1_cong* [**where** *tms*=[*y*,*x′*]@*tms*]) (*auto simp*: *fresh_Cons sub*)
have { *x EQ x′* } ⊢ *P x y IFF P x′ y*
by (*rule P1_cong* [**where** *tms*=[*y*,*x′*]@*tms*]) (*auto simp*: *fresh_Cons sub*)
hence {*x EQ x′*, *y EQ y′*} ⊢ *P x y IFF P x′ y′*
by (*metis Assume Iff_trans cut1 rotate2 yy′*)
thus *?thesis*
by (*metis cut2 eq*)
qed

lemma *P3_cong*:
fixes *tms* :: *trm list*
assumes *sub*: ⋀*i t u v x*. *atom i* ♯ *tms* ⟹
(*P t u v*)(*i*::=*x*) = *P* (*subst i x t*) (*subst i x u*) (*subst i x v*)
and *eq*: *H* ⊢ *x EQ x′ H* ⊢ *y EQ y′ H* ⊢ *z EQ z′*
shows *H* ⊢ *P x y z IFF P x′ y′ z′*
proof −
obtain *i*::*name* **where** *i*: *atom i* ♯ (*z*,*z′*,*y*,*y′*,*x*,*x′*)
by (*metis obtain_fresh*)
have *tl*: { *y EQ y′*, *z EQ z′* } ⊢ *P x′ y z IFF P x′ y′ z′*

by (*rule P2_cong* [**where** *tms*=[*z*,*z'*,*y*,*y'*,*x*,*x'*]@*tms*]) (*auto simp*: *fresh_Cons sub*)
have *hd*: { *x EQ x'* } ⊢ *P x y z IFF P x' y z*
by (*rule P1_cong* [**where** *tms*=[*z*,*y*,*x'*]@*tms*]) (*auto simp*: *fresh_Cons sub*)
have {*x EQ x'*, *y EQ y'*, *z EQ z'*} ⊢ *P x y z IFF P x' y' z'*
by (*metis Assume thin1 hd* [*THEN cut1*] *tl Iff_trans*)
thus *?thesis*
by (*rule cut3*) (*rule eq*)+
qed

lemma *P4_cong*:
fixes *tms* :: *trm list*
assumes *sub*: ⋀*i t1 t2 t3 t4 x*. *atom i* ♯ *tms* ⟹
(*P t1 t2 t3 t4*)(*i*::=*x*) = *P* (*subst i x t1*) (*subst i x t2*) (*subst i x t3*) (*subst i x t4*)
and *eq*: *H* ⊢ *x1 EQ x1'* *H* ⊢ *x2 EQ x2'* *H* ⊢ *x3 EQ x3'* *H* ⊢ *x4 EQ x4'*
shows *H* ⊢ *P x1 x2 x3 x4 IFF P x1' x2' x3' x4'*
proof −
obtain *i*::*name* **where** *i*: *atom i* ♯ (*x4*,*x4'*,*x3*,*x3'*,*x2*,*x2'*,*x1*,*x1'*)
by (*metis obtain_fresh*)
have *tl*: { *x2 EQ x2'*, *x3 EQ x3'*, *x4 EQ x4'* } ⊢ *P x1' x2 x3 x4 IFF P x1' x2' x3' x4'*
by (*rule P3_cong* [**where** *tms*=[*x4*,*x4'*,*x3*,*x3'*,*x2*,*x2'*,*x1*,*x1'*]@*tms*]) (*auto simp*: *fresh_Cons sub*)
have *hd*: { *x1 EQ x1'* } ⊢ *P x1 x2 x3 x4 IFF P x1' x2 x3 x4*
by (*auto simp*: *fresh_Cons sub intro*!: *P1_cong* [**where** *tms*=[*x4*,*x3*,*x2*,*x1'*]@*tms*])
have {*x1 EQ x1'*, *x2 EQ x2'*, *x3 EQ x3'*, *x4 EQ x4'*} ⊢ *P x1 x2 x3 x4 IFF P x1' x2' x3' x4'*
by (*metis Assume thin1 hd* [*THEN cut1*] *tl Iff_trans*)
thus *?thesis*
by (*rule cut4*) (*rule eq*)+
qed

4.5 The formula *fls*

lemma *neg_I* [*intro*!]: *insert A H* ⊢ *fls* ⟹ *H* ⊢ *neg A*
unfolding *fls_def*
by (*meson neg_D neg_I0 Refl*)

lemma *neg_E* [*intro*!]: *H* ⊢ *A* ⟹ *insert* (*neg A*) *H* ⊢ *fls*
by (*rule ContraProve*)

declare *neg_E* [*THEN rotate2*, *intro*!]
declare *neg_E* [*THEN rotate3*, *intro*!]
declare *neg_E* [*THEN rotate4*, *intro*!]
declare *neg_E* [*THEN rotate5*, *intro*!]
declare *neg_E* [*THEN rotate6*, *intro*!]
declare *neg_E* [*THEN rotate7*, *intro*!]
declare *neg_E* [*THEN rotate8*, *intro*!]

lemma *neg_imp_I* [*intro*!]: *H* ⊢ *A* ⟹ *insert B H* ⊢ *fls* ⟹ *H* ⊢ *neg* (*A IMP B*)
by (*metis negneg_I neg_dsj_I neg_I*)

lemma *neg_imp_E* [*intro*!]: *insert* (*neg B*) (*insert A H*) ⊢ *C* ⟹ *insert* (*neg* (*A IMP B*)) *H* ⊢ *C*
apply (*rule cut_same* [**where** *A*=*A*])
apply (*metis Assume dsj_I1 negneg_D neg_mono*)
apply (*metis Swap imp_I rotate2 thin1*)
done

declare *neg_imp_E* [*THEN rotate2*, *intro*!]
declare *neg_imp_E* [*THEN rotate3*, *intro*!]
declare *neg_imp_E* [*THEN rotate4*, *intro*!]
declare *neg_imp_E* [*THEN rotate5*, *intro*!]

declare *neg_imp_E* [*THEN rotate6*, *intro*!]
declare *neg_imp_E* [*THEN rotate7*, *intro*!]
declare *neg_imp_E* [*THEN rotate8*, *intro*!]

lemma *fls_E* [*intro*!]: *insert fls H* ⊢ *A*
by (*simp add*: *ContraProve fls_def*)

declare *fls_E* [*THEN rotate2*, *intro*!]
declare *fls_E* [*THEN rotate3*, *intro*!]
declare *fls_E* [*THEN rotate4*, *intro*!]
declare *fls_E* [*THEN rotate5*, *intro*!]
declare *fls_E* [*THEN rotate6*, *intro*!]
declare *fls_E* [*THEN rotate7*, *intro*!]
declare *fls_E* [*THEN rotate8*, *intro*!]

lemma *truth_provable*: *H* ⊢ (*neg fls*)
by (*metis fls_E neg_I*)

lemma *exFalso*: *H* ⊢ *fls* ⟹ *H* ⊢ *A*
by (*metis neg_D truth_provable*)

Soundness of the provability relation

theorem *nprv_sound*: **assumes** *H* ⊢ *A* **shows** (∀ *B*∈*H*. *eval_fmla e B*) ⟹ *eval_fmla e A*
using *assms*
proof (*induct arbitrary*: *e*)
case (*Hyp A H*) **thus** *?case*
by *auto*
next
case (*Q H*) **thus** *?case*
unfolding *Q_axioms_def*
using *not0_implies_Suc* **by** *fastforce*
next
case (*Bool A H*) **thus** *?case*
by (*metis boolean_axioms_hold*)
next
case (*eql A H*) **thus** *?case*
by (*metis equality_axioms_hold*)
next
case (*Spec A H*) **thus** *?case*
by (*metis special_axioms_hold*)
next
case (*MP H A B H'*) **thus** *?case*
by *auto*
next
case (*exiists H A B i e*) **thus** *?case*
by *auto* (*metis forget_eval_fmla*)
qed

5 Instantiation of Syntax-Independent Logic Infrastructure

5.1 Preliminaries

inductive_set *num* :: *trm set* **where**
zer[*intro*!,*simp*]: *zer* ∈ *num*
|*suc*[*simp*]: *t* ∈ *num* ⟹ *suc t* ∈ *num*

definition *ground_aux* :: *trm* ⇒ *atom set* ⇒ *bool*

where *ground_aux t S* ≡ (*supp t* ⊆ *S*)

abbreviation *ground* :: *trm* ⇒ *bool*
where *ground t* ≡ *ground_aux t* {}

definition *ground_fmla_aux* :: *fmla* ⇒ *atom set* ⇒ *bool*
where *ground_fmla_aux A S* ≡ (*supp A* ⊆ *S*)

abbreviation *ground_fmla* :: *fmla* ⇒ *bool*
where *ground_fmla A* ≡ *ground_fmla_aux A* {}

lemma *ground_aux_simps*[*simp*]:
ground_aux zer S = *True*
ground_aux (*Var k*) *S* = (*if atom k* ∈ *S then True else False*)
ground_aux (*suc t*) *S* = (*ground_aux t S*)
ground_aux (*pls t u*) *S* = (*ground_aux t S* ∧ *ground_aux u S*)
ground_aux (*tms t u*) *S* = (*ground_aux t S* ∧ *ground_aux u S*)
unfolding *ground_aux_def*
by (*simp_all add*: *supp_at_base*)

lemma *ground_fmla_aux_simps*[*simp*]:
ground_fmla_aux fls S = *True*
ground_fmla_aux (*t EQ u*) *S* = (*ground_aux t S* ∧ *ground_aux u S*)
ground_fmla_aux (*A OR B*) *S* = (*ground_fmla_aux A S* ∧ *ground_fmla_aux B S*)
ground_fmla_aux (*A AND B*) *S* = (*ground_fmla_aux A S* ∧ *ground_fmla_aux B S*)
ground_fmla_aux (*A IFF B*) *S* = (*ground_fmla_aux A S* ∧ *ground_fmla_aux B S*)
ground_fmla_aux (*neg A*) *S* = (*ground_fmla_aux A S*)
ground_fmla_aux (*exi x A*) *S* = (*ground_fmla_aux A* (*S* ∪ {*atom x*}))
by (*auto simp*: *ground_fmla_aux_def ground_aux_def supp_conv_fresh*)

lemma *ground_fresh*[*simp*]:
ground t ⟹ *atom i* ♯ *t*
ground_fmla A ⟹ *atom i* ♯ *A*
unfolding *ground_aux_def ground_fmla_aux_def fresh_def*
by *simp_all*

definition *Fvars t* = {*a* :: *name*. ¬ *atom a* ♯ *t*}

lemma *Fvars_trm_simps*[*simp*]:
Fvars zer = {}
Fvars (*Var a*) = {*a*}
Fvars (*suc x*) = *Fvars x*
Fvars (*pls x y*) = *Fvars x* ∪ *Fvars y*
Fvars (*tms x y*) = *Fvars x* ∪ *Fvars y*
by (*auto simp*: *Fvars_def fresh_at_base*(*2*))

lemma *finite_Fvars_trm*[*simp*]:
fixes *t* :: *trm*
shows *finite* (*Fvars t*)
by (*induct t rule*: *trm.induct*) *auto*

lemma *Fvars_fmla_simps*[*simp*]:
Fvars (*x EQ y*) = *Fvars x* ∪ *Fvars y*
Fvars (*A OR B*) = *Fvars A* ∪ *Fvars B*
Fvars (*A AND B*) = *Fvars A* ∪ *Fvars B*
Fvars (*A IMP B*) = *Fvars A* ∪ *Fvars B*
Fvars fls = {}

```
  Fvars (neg A) = Fvars A
  Fvars (exi a A) = Fvars A − {a}
  Fvars (all a A) = Fvars A − {a}
  by (auto simp: Fvars_def fresh_at_base(2))

lemma finite_Fvars_fmla[simp]:
  fixes A :: fmla
  shows finite (Fvars A)
  by (induct A rule: fmla.induct) auto

lemma subst_trm_subst_trm[simp]:
  x ≠ y ⟹ atom x ♯ u ⟹ subst y u (subst x t v) = subst x (subst y u t) (subst y u v)
  by (induct v rule: trm.induct) auto

lemma subst_fmla_subst_fmla[simp]:
  x ≠ y ⟹ atom x ♯ u ⟹ (A(x::=t))(y::=u) = (A(y::=u))(x::=subst y u t)
  by (nominal_induct A avoiding: x t y u rule: fmla.strong_induct) auto

lemma Fvars_empty_ground[simp]: Fvars t = {} ⟹ ground t
  by (induct t rule: trm.induct) auto

lemma Fvars_ground_aux: Fvars t ⊆ B ⟹ ground_aux t (atom ‘ B)
  by (induct t rule: trm.induct) auto

lemma ground_Fvars: ground t ⟷ Fvars t = {}
  apply (rule iffI)
  subgoal by (auto simp only: Fvars_def ground_fresh) []
  by auto

lemma Fvars_ground_fmla_aux: Fvars A ⊆ B ⟹ ground_fmla_aux A (atom ‘ B)
  apply (induct A arbitrary: B rule: fmla.induct)
  subgoal by (auto simp: Diff_subset_conv Fvars_ground_aux)
  subgoal by (auto simp: Diff_subset_conv Fvars_ground_aux)
  subgoal by (auto simp: Diff_subset_conv Fvars_ground_aux)
  subgoal by (metis Diff_subset_conv Fvars_fmla_simps(7) Un_insert_left
          Un_insert_right ground_fmla_aux_simps(7)
          image_insert sup_bot.left_neutral sup_bot.right_neutral) .

lemma ground_fmla_Fvars: ground_fmla A ⟷ Fvars A = {}
  apply (rule iffI)
  subgoal by (auto simp only: Fvars_def ground_fresh)
  by (auto intro: Fvars_ground_fmla_aux[of A {}, simplified])

lemma obtain_const_trm:
obtains t where eval_trm e t = x t ∈ num
apply (induct x)
using eval_trm.simps(1) eval_trm.simps(3) num.suc by blast+

lemma ex_eval_fmla_iff_exists_num:
  eval_fmla e (exi k A) ⟷ (∃ t. eval_fmla e (A(k::=t)) ∧ t ∈ num)
by (auto simp: eval_subst_fmla) (metis obtain_const_trm)

lemma exi_ren: y ∉ Fvars φ ⟹ exi x φ = exi y (φ(x::=Var y))
using exi_ren_subst_fresh Fvars_def by blast

lemma all_ren: y ∉ Fvars φ ⟹ all x φ = all y (φ(x::=Var y))
by (simp add: exi_ren)
```

lemma *Fvars_num*[*simp*]: $t \in num \Longrightarrow Fvars\ t = \{\}$
by (*induct t rule*: *trm.induct*) (*auto elim*: *num.cases*)

5.2 Instantiation of the generic syntax and deduction relation

interpretation *Generic_Syntax* **where**
  var = *UNIV* :: *name set*
 and *trm* = *UNIV* :: *trm set*
 and *fmla* = *UNIV* :: *fmla set*
 and *Var* = *Var*
 and *FvarsT* = *Fvars*
 and *substT* = $\lambda t\ u\ x.$ *subst x u t*
 and *Fvars* = *Fvars*
 and *subst* = $\lambda A\ u\ x.$ *subst_fmla A x u*
 apply *unfold_locales*
 subgoal by *simp*
 subgoal by *simp*
 subgoal by *simp*
 subgoal by *simp*
 subgoal by *simp*
 subgoal by *simp*
 subgoal by *simp*
 subgoal by *simp*
 subgoal for *t* **by** (*induct t rule*: *trm.induct*) *auto*
 subgoal by *simp*
 subgoal by *simp*
 subgoal by *simp*
 subgoal unfolding *Fvars_def fresh_subst_fmla_if* **by** *auto*
 subgoal unfolding *Fvars_def* **by** *auto*
 subgoal unfolding *Fvars_def* **by** *simp*
 subgoal by *simp*
 subgoal unfolding *Fvars_def* **by** *simp* **.**

interpretation *Syntax_with_Numerals* **where**
  var = *UNIV* :: *name set*
 and *trm* = *UNIV* :: *trm set*
 and *fmla* = *UNIV* :: *fmla set*
 and *num* = *num*
 and *Var* = *Var*
 and *FvarsT* = *Fvars*
 and *substT* = $\lambda t\ u\ x.$ *subst x u t*
 and *Fvars* = *Fvars*
 and *subst* = $\lambda A\ u\ x.$ *subst_fmla A x u*
 apply *unfold_locales*
 subgoal by (*auto intro*!: *exI*[*of* _ *zer*])
 subgoal by *simp*
 subgoal by (*simp add*: *ground_Fvars*) **.**

interpretation *Deduct_with_False* **where**
  var = *UNIV* :: *name set*
 and *trm* = *UNIV* :: *trm set*
 and *fmla* = *UNIV* :: *fmla set*
 and *num* = *num*
 and *Var* = *Var*
 and *FvarsT* = *Fvars*
 and *substT* = $\lambda t\ u\ x.$ *subst x u t*
 and *Fvars* = *Fvars*
 and *subst* = $\lambda A\ u\ x.$ *subst_fmla A x u*

and *eql* = *eql* **and** *cnj* = *cnj* **and** *imp* = *imp* **and** *all* = *all*
and *exi* = *exi* **and** *fls* = *fls*
and *prv* = (⊢) {}
apply *unfold_locales*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal unfolding *Fvars_def* **by** *simp*
subgoal unfolding *Fvars_def* **by** *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal using *MP_null* **by** *blast*
subgoal by *blast*
subgoal for *A B C*
apply (*rule imp_I*)+
apply (*rule MP_same*[*of* _ *B*])
apply (*rule MP_same*[*of* _ *C*])
apply (*auto intro*: *neg_D*) **.**
subgoal by *blast*
subgoal by *blast*
subgoal by *blast*
subgoal unfolding *Fvars_def* **by** (*auto intro*: *MP_null*)
subgoal unfolding *Fvars_def* **by** (*auto intro*: *MP_null*)
subgoal by (*auto intro*: *all_D*)
subgoal by (*auto intro*: *exi_I*)
subgoal by *simp*
subgoal by (*metis cnj_E2 Iff_def imp_I Var_eql_subst_Iff*)
subgoal by *blast* **.**

interpretation *Deduct_with_False_Disj* **where**
var = *UNIV* :: *name set*
and *trm* = *UNIV* :: *trm set*
and *fmla* = *UNIV* :: *fmla set*
and *num* = *num*
and *Var* = *Var*
and *FvarsT* = *Fvars*
and *substT* = λ*t u x*. *subst x u t*
and *Fvars* = *Fvars*
and *subst* = λ*A u x*. *subst_fmla A x u*
and *eql* = *eql* **and** *cnj* = *cnj* **and** *dsj* = *dsj* **and** *imp* = *imp*
and *all* = *all* **and** *exi* = *exi* **and** *fls* = *fls*
and *prv* = (⊢) {}
apply *unfold_locales*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by (*auto intro*: *dsj_I1*)

subgoal by (*auto intro*: *dsj_I2*)
subgoal by (*auto intro*: *ContraAssume*) **.**

5.3 Instantiation of the arithmetic-enriched generic syntax and deduction relation

interpretation *Syntax_Arith_aux* **where**
var = *UNIV* :: *name set*
and *trm* = *UNIV* :: *trm set*
and *fmla* = *UNIV* :: *fmla set*
and *num* = *num*
and *Var* = *Var*
and *FvarsT* = *Fvars*
and *substT* = λ*t u x*. *subst x u t*
and *Fvars* = *Fvars*
and *subst* = λ*A u x*. *subst_fmla A x u*
and *eql* = *eql* **and** *cnj* = *cnj* **and** *imp* = *imp* **and** *all* = *all*
and *exi* = *exi* **and** *dsj* = *dsj* **and** *fls* = *fls*
and *zer* = *zer* **and** *suc* = *suc* **and** *pls* = *pls* **and** *tms* = *tms*
by *unfold_locales* (*auto simp*: *exi_ren all_ren*)

lemma *num_range_Num*: *num* = *range Num*
proof−
{fix *t* **assume** *t* ∈ *num*
then have ∃ *n*. *t* = *Num n*
apply(*induct t rule*: *trm.induct*)
subgoal by (*auto intro*: *exI*[*of* _ *0*])
subgoal by (*auto elim*: *num.cases*)
subgoal by (*metis Num.simps*(*2*) *num.cases trm.distinct*(*3*) *trm.eq_iff*(*3*))
by (*auto elim*: *num.cases*)
}
moreover
{fix *n* **have** *Num n* ∈ *num*
by (*induct n*) *auto*
}
ultimately show *?thesis* **by** *auto*
qed

lemma [*simp*]: {} ⊢ *neg* (*zer EQ suc* (*Var xx*))
proof−
have *0*: {} ⊢ *Robinson_Arithmetic.neg* (*zer EQ suc* (*Var xx*))
by (*intro nprv.Q*) (*auto intro*!: *exI*[*of* _ *zz*] *simp*: *Q_axioms_def*)
show *?thesis* **unfolding** *neg_def*
by (*simp add*: *0 dsj_I1*)
qed

lemma [*simp*]: {} ⊢ *Var yy EQ zer OR exi xx* (*Var yy EQ suc* (*Var xx*))
by (*intro nprv.Q*) (*auto intro*!: *exI*[*of* _ *zz*] *simp*: *Q_axioms_def*)

lemma [*simp*]: {} ⊢ *pls* (*Var xx*) *zer EQ Var xx*
by (*intro nprv.Q*) (*auto intro*!: *exI*[*of* _ *zz*] *simp*: *Q_axioms_def*)

lemma [*simp*]: {} ⊢ *tms* (*Var xx*) *zer EQ zer*
by (*intro nprv.Q*) (*auto intro*!: *exI*[*of* _ *zz*] *simp*: *Q_axioms_def*)

interpretation *S*: *Syntax_Arith* **where**
var = *UNIV* :: *name set*
and *trm* = *UNIV* :: *trm set*

and *fmla* = *UNIV* :: *fmla set*
and *num* = *num*
and *Var* = *Var*
and *FvarsT* = *Fvars*
and *substT* = λ*t u x*. *subst x u t*
and *Fvars* = *Fvars*
and *subst* = λ*A u x*. *subst_fmla A x u*
and *eql* = *eql* **and** *cnj* = *cnj* **and** *imp* = *imp* **and** *all* = *all*
and *exi* = *exi* **and** *dsj* = *dsj* **and** *fls* = *fls* **and** *zer* = *zer*
and *suc* = *suc* **and** *pls* = *pls* **and** *tms* = *tms*
using *num_range_Num* **by** *unfold_locales auto*

interpretation *Deduct_Q* **where**
var = *UNIV* :: *name set*
and *trm* = *UNIV* :: *trm set*
and *fmla* = *UNIV* :: *fmla set*
and *num* = *num*
and *Var* = *Var*
and *FvarsT* = *Fvars*
and *substT* = λ*t u x*. *subst x u t*
and *Fvars* = *Fvars*
and *subst* = λ*A u x*. *subst_fmla A x u*
and *eql* = *eql* **and** *cnj* = *cnj* **and** *imp* = *imp* **and** *all* = *all*
and *exi* = *exi* **and** *dsj* = *dsj* **and** *fls* = *fls* **and** *zer* = *zer*
and *suc* = *suc* **and** *pls* = *pls* **and** *tms* = *tms*
and *prv* = (⊢) {}
by *unfold_locales* (*auto simp add*: *Q Q_axioms_def*)

5.4 Instantiation of the abstract notion of standard model and truth

interpretation *Minimal_Truth_Soundness* **where**
var = *UNIV* :: *name set*
and *trm* = *UNIV* :: *trm set*
and *fmla* = *UNIV* :: *fmla set*
and *num* = *num*
and *Var* = *Var*
and *FvarsT* = *Fvars*
and *substT* = λ*t u x*. *subst x u t*
and *Fvars* = *Fvars*
and *subst* = λ*A u x*. *subst_fmla A x u*
and *eql* = *eql* **and** *cnj* = *cnj* **and** *dsj* = *dsj* **and** *imp* = *imp*
and *all* = *all* **and** *exi* = *exi* **and** *fls* = *fls*
and *prv* = (⊢) {}
and *isTrue* = *eval_fmla e0*
apply *unfold_locales*
subgoal by (*auto simp*: *fls_def*)
subgoal by *simp*
subgoal by (*auto simp only*: *ex_eval_fmla_iff_exists_num eval_fmla.simps subst_fmla.simps*)
subgoal by (*auto simp only*: *ex_eval_fmla_iff_exists_num*)
subgoal by (*simp add*: *neg_def*)
subgoal by (*auto dest*: *nprv_sound*) **.**