Nominal 2

Christian Urban, Stefan Berghofer, and Cezary Kaliszyk

May 26, 2024

Abstract

Dealing with binders, renaming of bound variables, capture-avoiding substitution, etc., is very often a major problem in formal proofs, especially in proofs by structural and rule induction. Nominal Isabelle is designed to make such proofs easy to formalise: it provides an infrastructure for declaring nominal datatypes (that is alpha-equivalence classes) and for defining functions over them by structural recursion. It also provides induction principles that have Barendregts variable convention already built in.

This entry can be used as a more advanced replacement for HOL/Nominal in the Isabelle distribution.

Contents

theory *Nominal2-Base*
imports *HOL−Library.Infinite-Set*
HOL−Library.Multiset
HOL−Library.FSet
FinFun.FinFun
keywords
atom-decl equivariance :: *thy-decl*
begin

declare [[*typedef-overloaded*]]

1 Atoms and Sorts

A simple implementation for *atom-sorts* is strings.

To deal with Church-like binding we use trees of strings as sorts.

datatype *atom-sort* = *Sort string atom-sort list*

datatype *atom* = *Atom atom-sort nat*

Basic projection function.

primrec
sort-of :: *atom* ⇒ *atom-sort*
where
sort-of (*Atom s n*) = *s*

primrec
nat-of :: *atom* ⇒ *nat*
where
nat-of (*Atom s n*) = *n*

There are infinitely many atoms of each sort.

lemma *INFM-sort-of-eq*:
shows *INFM a. sort-of a* = *s*
⟨*proof*⟩

lemma *infinite-sort-of-eq*:
shows *infinite* {*a. sort-of a* = *s*}
⟨*proof*⟩

lemma *atom-infinite* [*simp*]:
shows *infinite* (*UNIV* :: *atom set*)

⟨*proof*⟩

lemma *obtain-atom*:
fixes X :: *atom set*
assumes X: *finite* X
obtains a **where** $a \notin X$ *sort-of* $a = s$
⟨*proof*⟩

lemma *atom-components-eq-iff*:
fixes a b :: *atom*
shows $a = b \longleftrightarrow$ *sort-of* a = *sort-of* $b \wedge$ *nat-of* a = *nat-of* b
⟨*proof*⟩

2 Sort-Respecting Permutations

definition
perm $\equiv \{f.$ *bij* $f \wedge$ *finite* $\{a.\ f\ a \neq a\} \wedge (\forall a.$ *sort-of* $(f\ a)$ = *sort-of* $a)\}$

typedef *perm* = *perm*
⟨*proof*⟩

lemma *permI*:
assumes *bij* f **and** *MOST* $x.\ f\ x = x$ **and** $\bigwedge a.$ *sort-of* $(f\ a)$ = *sort-of* a
shows $f \in$ *perm*
⟨*proof*⟩

lemma *perm-is-bij*: $f \in$ *perm* $\Longrightarrow$ *bij* f
⟨*proof*⟩

lemma *perm-is-finite*: $f \in$ *perm* $\Longrightarrow$ *finite* $\{a.\ f\ a \neq a\}$
⟨*proof*⟩

lemma *perm-is-sort-respecting*: $f \in$ *perm* $\Longrightarrow$ *sort-of* $(f\ a)$ = *sort-of* a
⟨*proof*⟩

lemma *perm-MOST*: $f \in$ *perm* $\Longrightarrow$ *MOST* $x.\ f\ x = x$
⟨*proof*⟩

lemma *perm-id*: *id* $\in$ *perm*
⟨*proof*⟩

lemma *perm-comp*:
assumes f: $f \in$ *perm* **and** g: $g \in$ *perm*
shows $(f \circ g) \in$ *perm*
⟨*proof*⟩

lemma *perm-inv*:
assumes f: $f \in$ *perm*
shows $(inv\ f) \in$ *perm*

⟨*proof*⟩

lemma *bij-Rep-perm*: *bij* (*Rep-perm p*)
⟨*proof*⟩

lemma *finite-Rep-perm*: *finite* {*a*. *Rep-perm p a* ≠ *a*}
⟨*proof*⟩

lemma *sort-of-Rep-perm*: *sort-of* (*Rep-perm p a*) = *sort-of a*
⟨*proof*⟩

lemma *Rep-perm-ext*:
Rep-perm p1 = *Rep-perm p2* ⟹ *p1* = *p2*
⟨*proof*⟩

instance *perm* :: *size* ⟨*proof*⟩

2.1 Permutations form a (multiplicative) group

instantiation *perm* :: *group-add*
begin

definition
0 = *Abs-perm id*

definition
− *p* = *Abs-perm* (*inv* (*Rep-perm p*))

definition
p + *q* = *Abs-perm* (*Rep-perm p* ∘ *Rep-perm q*)

definition
(*p1*::*perm*) − *p2* = *p1* + − *p2*

lemma *Rep-perm-0*: *Rep-perm 0* = *id*
⟨*proof*⟩

lemma *Rep-perm-add*:
Rep-perm (*p1* + *p2*) = *Rep-perm p1* ∘ *Rep-perm p2*
⟨*proof*⟩

lemma *Rep-perm-uminus*:
Rep-perm (− *p*) = *inv* (*Rep-perm p*)
⟨*proof*⟩

instance
⟨*proof*⟩

end

3 Implementation of swappings

definition
  swap :: *atom* ⇒ *atom* ⇒ *perm* (*'*(- ⇌ -*'*))
where
  (*a* ⇌ *b*) =
    Abs-perm (*if sort-of a* = *sort-of b*
          then (λ*c*. *if a* = *c then b else if b* = *c then a else c*)
          else id)

lemma *Rep-perm-swap*:
  Rep-perm (*a* ⇌ *b*) =
    (*if sort-of a* = *sort-of b*
     then (λ*c*. *if a* = *c then b else if b* = *c then a else c*)
     else id)
⟨*proof*⟩

lemmas *Rep-perm-simps* =
  Rep-perm-0
  Rep-perm-add
  Rep-perm-uminus
  Rep-perm-swap

lemma *swap-different-sorts* [*simp*]:
  sort-of a ≠ *sort-of b* ⟹ (*a* ⇌ *b*) = *0*
  ⟨*proof*⟩

lemma *swap-cancel*:
  shows (*a* ⇌ *b*) + (*a* ⇌ *b*) = *0*
  and (*a* ⇌ *b*) + (*b* ⇌ *a*) = *0*
  ⟨*proof*⟩

lemma *swap-self* [*simp*]:
  (*a* ⇌ *a*) = *0*
  ⟨*proof*⟩

lemma *minus-swap* [*simp*]:
  − (*a* ⇌ *b*) = (*a* ⇌ *b*)
  ⟨*proof*⟩

lemma *swap-commute*:
  (*a* ⇌ *b*) = (*b* ⇌ *a*)
  ⟨*proof*⟩

lemma *swap-triple*:
  assumes *a* ≠ *b* **and** *c* ≠ *b*
  assumes *sort-of a* = *sort-of b sort-of b* = *sort-of c*
  shows (*a* ⇌ *c*) + (*b* ⇌ *c*) + (*a* ⇌ *c*) = (*a* ⇌ *b*)
  ⟨*proof*⟩

4 Permutation Types

Infix syntax for *permute* has higher precedence than addition, but lower than unary minus.

class *pt* =
  fixes *permute* :: *perm* ⇒ *'a* ⇒ *'a* (- • - [*76*, *75*] *75*)
  assumes *permute-zero* [*simp*]: *0* • *x* = *x*
  assumes *permute-plus* [*simp*]: (*p* + *q*) • *x* = *p* • (*q* • *x*)
begin

lemma *permute-diff* [*simp*]:
  shows (*p* − *q*) • *x* = *p* • − *q* • *x*
  ⟨*proof*⟩

lemma *permute-minus-cancel* [*simp*]:
  shows *p* • − *p* • *x* = *x*
  and − *p* • *p* • *x* = *x*
  ⟨*proof*⟩

lemma *permute-swap-cancel* [*simp*]:
  shows (*a* ⇌ *b*) • (*a* ⇌ *b*) • *x* = *x*
  ⟨*proof*⟩

lemma *permute-swap-cancel2* [*simp*]:
  shows (*a* ⇌ *b*) • (*b* ⇌ *a*) • *x* = *x*
  ⟨*proof*⟩

lemma *inj-permute* [*simp*]:
  shows *inj* (*permute p*)
  ⟨*proof*⟩

lemma *surj-permute* [*simp*]:
  shows *surj* (*permute p*)
  ⟨*proof*⟩

lemma *bij-permute* [*simp*]:
  shows *bij* (*permute p*)
  ⟨*proof*⟩

lemma *inv-permute*:
  shows *inv* (*permute p*) = *permute* (− *p*)
  ⟨*proof*⟩

lemma *permute-minus*:
  shows *permute* (− *p*) = *inv* (*permute p*)
  ⟨*proof*⟩

lemma *permute-eq-iff* [*simp*]:
  shows *p* • *x* = *p* • *y* ⟷ *x* = *y*

⟨*proof*⟩

end

4.1 Permutations for atoms

instantiation *atom* :: *pt*
begin

definition
$p \cdot a = (\mathit{Rep\text{-}perm}\ p)\ a$

instance
⟨*proof*⟩

end

lemma *sort-of-permute* [*simp*]:
shows $\mathit{sort\text{-}of}\ (p \cdot a) = \mathit{sort\text{-}of}\ a$
⟨*proof*⟩

lemma *swap-atom*:
shows $(a \rightleftharpoons b) \cdot c =$
$(\mathit{if}\ \mathit{sort\text{-}of}\ a = \mathit{sort\text{-}of}\ b$
$\mathit{then}\ (\mathit{if}\ c = a\ \mathit{then}\ b\ \mathit{else}\ \mathit{if}\ c = b\ \mathit{then}\ a\ \mathit{else}\ c)\ \mathit{else}\ c)$
⟨*proof*⟩

lemma *swap-atom-simps* [*simp*]:
$\mathit{sort\text{-}of}\ a = \mathit{sort\text{-}of}\ b \Longrightarrow (a \rightleftharpoons b) \cdot a = b$
$\mathit{sort\text{-}of}\ a = \mathit{sort\text{-}of}\ b \Longrightarrow (a \rightleftharpoons b) \cdot b = a$
$c \neq a \Longrightarrow c \neq b \Longrightarrow (a \rightleftharpoons b) \cdot c = c$
⟨*proof*⟩

lemma *perm-eq-iff*:
fixes $p\ q$ :: *perm*
shows $p = q \longleftrightarrow (\forall\, a{::}\mathit{atom}.\ p \cdot a = q \cdot a)$
⟨*proof*⟩

4.2 Permutations for permutations

instantiation *perm* :: *pt*
begin

definition
$p \cdot q = p + q - p$

instance
⟨*proof*⟩

end

lemma *permute-self*:
shows $p \cdot p = p$
⟨*proof*⟩

lemma *pemute-minus-self*:
shows $- p \cdot p = p$
⟨*proof*⟩

4.3 Permutations for functions

instantiation *fun* :: (*pt*, *pt*) *pt*
begin

definition
$p \cdot f = (\lambda x.\ p \cdot (f\ (-\ p \cdot x)))$

instance
⟨*proof*⟩

end

lemma *permute-fun-app-eq*:
shows $p \cdot (f\ x) = (p \cdot f)\ (p \cdot x)$
⟨*proof*⟩

lemma *permute-fun-comp*:
shows $p \cdot f\ = (permute\ p)\ o\ f\ o\ (permute\ (-p))$
⟨*proof*⟩

4.4 Permutations for booleans

instantiation *bool* :: *pt*
begin

definition $p \cdot (b::bool) = b$

instance
⟨*proof*⟩

end

lemma *permute-boolE*:
fixes *P*::*bool*
shows $p \cdot P \Longrightarrow P$
⟨*proof*⟩

lemma *permute-boolI*:
fixes *P*::*bool*
shows $P \Longrightarrow p \cdot P$

⟨*proof*⟩

4.5 Permutations for sets

instantiation *set* :: (*pt*) *pt*
begin

definition
$p \cdot X = \{p \cdot x \mid x.\ x \in X\}$

instance
⟨*proof*⟩

end

lemma *permute-set-eq*:
shows $p \cdot X = \{x.\ -\ p \cdot x \in X\}$
⟨*proof*⟩

lemma *permute-set-eq-image*:
shows $p \cdot X = permute\ p\ `\ X$
⟨*proof*⟩

lemma *permute-set-eq-vimage*:
shows $p \cdot X = permute\ (-\ p)\ -`\ X$
⟨*proof*⟩

lemma *permute-finite* [*simp*]:
shows *finite* $(p \cdot X) =$ *finite* X
⟨*proof*⟩

lemma *swap-set-not-in*:
assumes *a*: $a \notin S\ b \notin S$
shows $(a \rightleftharpoons b) \cdot S = S$
⟨*proof*⟩

lemma *swap-set-in*:
assumes *a*: $a \in S\ b \notin S$ *sort-of* $a =$ *sort-of* b
shows $(a \rightleftharpoons b) \cdot S \neq S$
⟨*proof*⟩

lemma *swap-set-in-eq*:
assumes *a*: $a \in S\ b \notin S$ *sort-of* $a =$ *sort-of* b
shows $(a \rightleftharpoons b) \cdot S = (S - \{a\}) \cup \{b\}$
⟨*proof*⟩

lemma *swap-set-both-in*:
assumes *a*: $a \in S\ b \in S$
shows $(a \rightleftharpoons b) \cdot S = S$

⟨*proof*⟩

lemma *mem-permute-iff*:
shows $(p \cdot x) \in (p \cdot X) \longleftrightarrow x \in X$
⟨*proof*⟩

lemma *empty-eqvt*:
shows $p \cdot \{\} = \{\}$
⟨*proof*⟩

lemma *insert-eqvt*:
shows $p \cdot (\mathit{insert}\ x\ A) = \mathit{insert}\ (p \cdot x)\ (p \cdot A)$
⟨*proof*⟩

4.6 Permutations for *unit*

instantiation *unit* :: *pt*
begin

definition $p \cdot (u{::}\mathit{unit}) = u$

instance
⟨*proof*⟩

end

4.7 Permutations for products

instantiation *prod* :: (*pt*, *pt*) *pt*
begin

primrec
permute-prod
where
Pair-eqvt: $p \cdot (x, y) = (p \cdot x, p \cdot y)$

instance
⟨*proof*⟩

end

4.8 Permutations for sums

instantiation *sum* :: (*pt*, *pt*) *pt*
begin

primrec
permute-sum
where
Inl-eqvt: $p \cdot (\mathit{Inl}\ x) = \mathit{Inl}\ (p \cdot x)$

| *Inr-eqvt*: $p \cdot (Inr\ y) = Inr\ (p \cdot y)$

instance
⟨*proof*⟩

end

4.9 Permutations for *'a list*

instantiation *list* :: (*pt*) *pt*
begin

primrec
permute-list
where
Nil-eqvt: $p \cdot [] = []$
| *Cons-eqvt*: $p \cdot (x \# xs) = p \cdot x \# p \cdot xs$

instance
⟨*proof*⟩

end

lemma *set-eqvt*:
shows $p \cdot (set\ xs) = set\ (p \cdot xs)$
⟨*proof*⟩

4.10 Permutations for *'a option*

instantiation *option* :: (*pt*) *pt*
begin

primrec
permute-option
where
None-eqvt: $p \cdot None = None$
| *Some-eqvt*: $p \cdot (Some\ x) = Some\ (p \cdot x)$

instance
⟨*proof*⟩

end

4.11 Permutations for *'a multiset*

instantiation *multiset* :: (*pt*) *pt*
begin

definition
$p \cdot M = \{\#\ p \cdot x.\ x :\#\ M\ \#\}$

instance
⟨*proof*⟩

end

lemma *permute-multiset* [*simp*]:
fixes $M\ N$::('*a*::*pt*) *multiset*
shows $(p \cdot \{\#\}) = (\{\#\}$ ::('*a*::*pt*) *multiset*)
and $(p \cdot$ *add-mset* $x\ M) =$ *add-mset* $(p \cdot x)\ (p \cdot M)$
and $(p \cdot (M + N)) = (p \cdot M) + (p \cdot N)$
⟨*proof*⟩

4.12 Permutations for *'a fset*

instantiation *fset* :: (*pt*) *pt*
begin

context includes *fset.lifting* **begin**
lift-definition
permute-fset :: *perm* ⇒ *'a fset* ⇒ *'a fset*
is *permute* :: *perm* ⇒ *'a set* ⇒ *'a set* ⟨*proof*⟩
end

context includes *fset.lifting* **begin**
instance
⟨*proof*⟩
end

end

context includes *fset.lifting*
begin
lemma *permute-fset* [*simp*]:
fixes S::('*a*::*pt*) *fset*
shows $(p \cdot \{||\}) = (\{||\}$ ::('*a*::*pt*) *fset*)
and $(p \cdot$ *finsert* $x\ S) =$ *finsert* $(p \cdot x)\ (p \cdot S)$
⟨*proof*⟩

lemma *fset-eqvt*:
shows $p \cdot (fset\ S) = fset\ (p \cdot S)$
⟨*proof*⟩
end

4.13 Permutations for ('*a*, '*b*) *finfun*

instantiation *finfun* :: (*pt*, *pt*) *pt*
begin

lift-definition

permute-finfun :: *perm* ⇒ ('*a*, '*b*) *finfun* ⇒ ('*a*, '*b*) *finfun*
is
permute :: *perm* ⇒ ('*a* ⇒ '*b*) ⇒ ('*a* ⇒ '*b*)
⟨*proof*⟩

instance
⟨*proof*⟩

end

4.14 Permutations for *char*, *nat*, and *int*

instantiation *char* :: *pt*
begin

definition *p* · (*c*::*char*) = *c*

instance
⟨*proof*⟩

end

instantiation *nat* :: *pt*
begin

definition *p* · (*n*::*nat*) = *n*

instance
⟨*proof*⟩

end

instantiation *int* :: *pt*
begin

definition *p* · (*i*::*int*) = *i*

instance
⟨*proof*⟩

end

5 Pure types

Pure types will have always empty support.

class *pure* = *pt* +
assumes *permute-pure*: *p* · *x* = *x*

Types *unit* and *bool* are pure.

instance *unit* :: *pure*
⟨*proof*⟩

instance *bool* :: *pure*
⟨*proof*⟩

Other type constructors preserve purity.

instance *fun* :: (*pure*, *pure*) *pure*
⟨*proof*⟩

instance *set* :: (*pure*) *pure*
⟨*proof*⟩

instance *prod* :: (*pure*, *pure*) *pure*
⟨*proof*⟩

instance *sum* :: (*pure*, *pure*) *pure*
⟨*proof*⟩

instance *list* :: (*pure*) *pure*
⟨*proof*⟩

instance *option* :: (*pure*) *pure*
⟨*proof*⟩

5.1 Types *char*, *nat*, and *int*

instance *char* :: *pure*
⟨*proof*⟩

instance *nat* :: *pure*
⟨*proof*⟩

instance *int* :: *pure*
⟨*proof*⟩

6 Infrastructure for Equivariance and *Perm-simp*

6.1 Basic functions about permutations

⟨*ML*⟩

6.2 Eqvt infrastructure

Setup of the theorem attributes *eqvt* and *eqvt-raw*.

⟨*ML*⟩

lemmas [*eqvt*] =

permute-prod.simps
permute-list.simps
permute-option.simps
permute-sum.simps

empty-eqvt insert-eqvt set-eqvt

permute-fset fset-eqvt

permute-multiset

6.3 *perm-simp* infrastructure

definition
unpermute p = permute (− p)

lemma *eqvt-apply*:
fixes *f* :: *'a::pt* ⇒ *'b::pt*
and *x* :: *'a::pt*
shows *p* ∙ (*f x*) ≡ (*p* ∙ *f*) (*p* ∙ *x*)
⟨*proof*⟩

lemma *eqvt-lambda*:
fixes *f* :: *'a::pt* ⇒ *'b::pt*
shows *p* ∙ *f* ≡ (λ*x*. *p* ∙ (*f* (*unpermute p x*)))
⟨*proof*⟩

lemma *eqvt-bound*:
shows *p* ∙ *unpermute p x* ≡ *x*
⟨*proof*⟩

provides *perm-simp* methods

⟨*ML*⟩

6.3.1 Equivariance for permutations and swapping

lemma *permute-eqvt*:
shows *p* ∙ (*q* ∙ *x*) = (*p* ∙ *q*) ∙ (*p* ∙ *x*)
⟨*proof*⟩

lemma *permute-eqvt-raw* [*eqvt-raw*]:
shows *p* ∙ *permute* ≡ *permute*
⟨*proof*⟩

lemma *zero-perm-eqvt* [*eqvt*]:

shows $p \cdot (0::perm) = 0$
⟨*proof*⟩

lemma *add-perm-eqvt* [*eqvt*]:
fixes p $p1$ $p2$ :: $perm$
shows $p \cdot (p1 + p2) = p \cdot p1 + p \cdot p2$
⟨*proof*⟩

lemma *swap-eqvt* [*eqvt*]:
shows $p \cdot (a \rightleftharpoons b) = (p \cdot a \rightleftharpoons p \cdot b)$
⟨*proof*⟩

lemma *uminus-eqvt* [*eqvt*]:
fixes p $q::perm$
shows $p \cdot (-\ q) = -\ (p \cdot q)$
⟨*proof*⟩

6.3.2 Equivariance of Logical Operators

lemma *eq-eqvt* [*eqvt*]:
shows $p \cdot (x = y) \longleftrightarrow (p \cdot x) = (p \cdot y)$
⟨*proof*⟩

lemma *Not-eqvt* [*eqvt*]:
shows $p \cdot (\neg\ A) \longleftrightarrow \neg\ (p \cdot A)$
⟨*proof*⟩

lemma *conj-eqvt* [*eqvt*]:
shows $p \cdot (A \wedge B) \longleftrightarrow (p \cdot A) \wedge (p \cdot B)$
⟨*proof*⟩

lemma *imp-eqvt* [*eqvt*]:
shows $p \cdot (A \longrightarrow B) \longleftrightarrow (p \cdot A) \longrightarrow (p \cdot B)$
⟨*proof*⟩

declare *imp-eqvt*[*folded HOL.induct-implies-def*, *eqvt*]

lemma *all-eqvt* [*eqvt*]:
shows $p \cdot (\forall x.\ P\ x) = (\forall x.\ (p \cdot P)\ x)$
⟨*proof*⟩

declare *all-eqvt*[*folded HOL.induct-forall-def*, *eqvt*]

lemma *ex-eqvt* [*eqvt*]:
shows $p \cdot (\exists x.\ P\ x) = (\exists x.\ (p \cdot P)\ x)$
⟨*proof*⟩

lemma *ex1-eqvt* [*eqvt*]:
shows $p \cdot (\exists !x.\ P\ x) = (\exists !x.\ (p \cdot P)\ x)$

⟨*proof*⟩

lemma *if-eqvt* [*eqvt*]:
shows $p \cdot (\mathit{if}\ b\ \mathit{then}\ x\ \mathit{else}\ y) = (\mathit{if}\ p \cdot b\ \mathit{then}\ p \cdot x\ \mathit{else}\ p \cdot y)$
⟨*proof*⟩

lemma *True-eqvt* [*eqvt*]:
shows $p \cdot \mathit{True} = \mathit{True}$
⟨*proof*⟩

lemma *False-eqvt* [*eqvt*]:
shows $p \cdot \mathit{False} = \mathit{False}$
⟨*proof*⟩

lemma *disj-eqvt* [*eqvt*]:
shows $p \cdot (A \vee B) \longleftrightarrow (p \cdot A) \vee (p \cdot B)$
⟨*proof*⟩

lemma *all-eqvt2*:
shows $p \cdot (\forall x.\ P\ x) = (\forall x.\ p \cdot P\ (-\ p \cdot x))$
⟨*proof*⟩

lemma *ex-eqvt2*:
shows $p \cdot (\exists x.\ P\ x) = (\exists x.\ p \cdot P\ (-\ p \cdot x))$
⟨*proof*⟩

lemma *ex1-eqvt2*:
shows $p \cdot (\exists !x.\ P\ x) = (\exists !x.\ p \cdot P\ (-\ p \cdot x))$
⟨*proof*⟩

lemma *the-eqvt*:
assumes *unique*: $\exists !x.\ P\ x$
shows $(p \cdot (\mathit{THE}\ x.\ P\ x)) = (\mathit{THE}\ x.\ (p \cdot P)\ x)$
⟨*proof*⟩

lemma *the-eqvt2*:
assumes *unique*: $\exists !x.\ P\ x$
shows $(p \cdot (\mathit{THE}\ x.\ P\ x)) = (\mathit{THE}\ x.\ p \cdot P\ (-\ p \cdot x))$
⟨*proof*⟩

6.3.3 Equivariance of Set operators

lemma *mem-eqvt* [*eqvt*]:
shows $p \cdot (x \in A) \longleftrightarrow (p \cdot x) \in (p \cdot A)$
⟨*proof*⟩

lemma *Collect-eqvt* [*eqvt*]:
shows $p \cdot \{x.\ P\ x\} = \{x.\ (p \cdot P)\ x\}$
⟨*proof*⟩

lemma *Bex-eqvt* [*eqvt*]:
shows $p \cdot (\exists\, x \in S.\ P\ x) = (\exists\, x \in (p \cdot S).\ (p \cdot P)\ x)$
⟨*proof*⟩

lemma *Ball-eqvt* [*eqvt*]:
shows $p \cdot (\forall\, x \in S.\ P\ x) = (\forall\, x \in (p \cdot S).\ (p \cdot P)\ x)$
⟨*proof*⟩

lemma *image-eqvt* [*eqvt*]:
shows $p \cdot (f\ `\ A) = (p \cdot f)\ `\ (p \cdot A)$
⟨*proof*⟩

lemma *Image-eqvt* [*eqvt*]:
shows $p \cdot (R\ ``\ A) = (p \cdot R)\ ``\ (p \cdot A)$
⟨*proof*⟩

lemma *UNIV-eqvt* [*eqvt*]:
shows $p \cdot UNIV = UNIV$
⟨*proof*⟩

lemma *inter-eqvt* [*eqvt*]:
shows $p \cdot (A \cap B) = (p \cdot A) \cap (p \cdot B)$
⟨*proof*⟩

lemma *Inter-eqvt* [*eqvt*]:
shows $p \cdot \bigcap S = \bigcap (p \cdot S)$
⟨*proof*⟩

lemma *union-eqvt* [*eqvt*]:
shows $p \cdot (A \cup B) = (p \cdot A) \cup (p \cdot B)$
⟨*proof*⟩

lemma *Union-eqvt* [*eqvt*]:
shows $p \cdot \bigcup A = \bigcup (p \cdot A)$
⟨*proof*⟩

lemma *Diff-eqvt* [*eqvt*]:
fixes $A\ B :: 'a::pt\ set$
shows $p \cdot (A - B) = (p \cdot A) - (p \cdot B)$
⟨*proof*⟩

lemma *Compl-eqvt* [*eqvt*]:
fixes $A :: 'a::pt\ set$
shows $p \cdot (-\ A) = -\ (p \cdot A)$
⟨*proof*⟩

lemma *subset-eqvt* [*eqvt*]:
shows $p \cdot (S \subseteq T) \longleftrightarrow (p \cdot S) \subseteq (p \cdot T)$

⟨*proof*⟩

lemma *psubset-eqvt* [*eqvt*]:
shows $p \cdot (S \subset T) \longleftrightarrow (p \cdot S) \subset (p \cdot T)$
⟨*proof*⟩

lemma *vimage-eqvt* [*eqvt*]:
shows $p \cdot (f -` A) = (p \cdot f) -` (p \cdot A)$
⟨*proof*⟩

lemma *foldr-eqvt*[*eqvt*]:
$p \cdot foldr\ f\ xs = foldr\ (p \cdot f)\ (p \cdot xs)$
⟨*proof*⟩

lemma *Sigma-eqvt*:
shows $(p \cdot (X \times Y)) = (p \cdot X) \times (p \cdot Y)$
⟨*proof*⟩

In order to prove that lfp is equivariant we need two auxiliary classes which specify that (<=) and Inf are equivariant. Instances for bool and fun are given.

class *le-eqvt* = *pt* +
assumes *le-eqvt* [*eqvt*]: $p \cdot (x \leq y) = ((p \cdot x) \leq (p \cdot (y :: 'a :: \{order, pt\})))$

class *inf-eqvt* = *pt* +
assumes *inf-eqvt* [*eqvt*]: $p \cdot (Inf\ X) = Inf\ (p \cdot (X :: 'a :: \{complete\text{-}lattice, pt\}\ set))$

instantiation *bool* :: *le-eqvt*
begin

instance
⟨*proof*⟩

end

instantiation *fun* :: (*pt*, *le-eqvt*) *le-eqvt*
begin

instance
⟨*proof*⟩

end

instantiation *bool* :: *inf-eqvt*
begin

instance

⟨*proof*⟩

end

instantiation *fun* :: (*pt*, *inf-eqvt*) *inf-eqvt*
begin

instance
⟨*proof*⟩

end

lemma *lfp-eqvt* [*eqvt*]:
  fixes F::$('a \Rightarrow 'b) \Rightarrow ('a$::*pt* $\Rightarrow$ $'b$::{*inf-eqvt*, *le-eqvt*}$)$
  shows $p \cdot (lfp\ F) = lfp\ (p \cdot F)$
⟨*proof*⟩

lemma *finite-eqvt* [*eqvt*]:
  shows $p \cdot finite\ A = finite\ (p \cdot A)$
⟨*proof*⟩

lemma *fun-upd-eqvt*[*eqvt*]:
  shows $p \cdot (f(x := y)) = (p \cdot f)((p \cdot x) := (p \cdot y))$
⟨*proof*⟩

lemma *comp-eqvt* [*eqvt*]:
  shows $p \cdot (f \circ g) = (p \cdot f) \circ (p \cdot g)$
⟨*proof*⟩

6.3.4 Equivariance for product operations

lemma *fst-eqvt* [*eqvt*]:
  shows $p \cdot (fst\ x) = fst\ (p \cdot x)$
  ⟨*proof*⟩

lemma *snd-eqvt* [*eqvt*]:
  shows $p \cdot (snd\ x) = snd\ (p \cdot x)$
  ⟨*proof*⟩

lemma *split-eqvt* [*eqvt*]:
  shows $p \cdot (case\text{-}prod\ P\ x) = case\text{-}prod\ (p \cdot P)\ (p \cdot x)$
  ⟨*proof*⟩

6.3.5 Equivariance for list operations

lemma *append-eqvt* [*eqvt*]:
  shows $p \cdot (xs\ @\ ys) = (p \cdot xs)\ @\ (p \cdot ys)$
  ⟨*proof*⟩

lemma *rev-eqvt* [*eqvt*]:

shows $p \cdot (rev\ xs) = rev\ (p \cdot xs)$
⟨*proof*⟩

lemma *map-eqvt* [*eqvt*]:
shows $p \cdot (map\ f\ xs) = map\ (p \cdot f)\ (p \cdot xs)$
⟨*proof*⟩

lemma *removeAll-eqvt* [*eqvt*]:
shows $p \cdot (removeAll\ x\ xs) = removeAll\ (p \cdot x)\ (p \cdot xs)$
⟨*proof*⟩

lemma *filter-eqvt* [*eqvt*]:
shows $p \cdot (filter\ f\ xs) = filter\ (p \cdot f)\ (p \cdot xs)$
⟨*proof*⟩

lemma *distinct-eqvt* [*eqvt*]:
shows $p \cdot (distinct\ xs) = distinct\ (p \cdot xs)$
⟨*proof*⟩

lemma *length-eqvt* [*eqvt*]:
shows $p \cdot (length\ xs) = length\ (p \cdot xs)$
⟨*proof*⟩

6.3.6 Equivariance for *'a option*

lemma *map-option-eqvt*[*eqvt*]:
shows $p \cdot (map\text{-}option\ f\ x) = map\text{-}option\ (p \cdot f)\ (p \cdot x)$
⟨*proof*⟩

6.3.7 Equivariance for *'a fset*

context includes *fset.lifting* **begin**
lemma *in-fset-eqvt*:
shows $(p \cdot (x\ |\in|\ S)) = ((p \cdot x)\ |\in|\ (p \cdot S))$
⟨*proof*⟩

lemma *union-fset-eqvt* [*eqvt*]:
shows $(p \cdot (S\ |\cup|\ T)) = ((p \cdot S)\ |\cup|\ (p \cdot T))$
⟨*proof*⟩

lemma *inter-fset-eqvt* [*eqvt*]:
shows $(p \cdot (S\ |\cap|\ T)) = ((p \cdot S)\ |\cap|\ (p \cdot T))$
⟨*proof*⟩

lemma *subset-fset-eqvt* [*eqvt*]:
shows $(p \cdot (S\ |\subseteq|\ T)) = ((p \cdot S)\ |\subseteq|\ (p \cdot T))$
⟨*proof*⟩

lemma *map-fset-eqvt* [*eqvt*]:
shows $p \cdot (f\ |`|\ S) = (p \cdot f)\ |`|\ (p \cdot S)$

⟨*proof*⟩
end

6.3.8 Equivariance for ('*a*, '*b*) *finfun*

lemma *finfun-update-eqvt* [*eqvt*]:
shows (*p* ∙ (*finfun-update f a b*)) = *finfun-update* (*p* ∙ *f*) (*p* ∙ *a*) (*p* ∙ *b*)
⟨*proof*⟩

lemma *finfun-const-eqvt* [*eqvt*]:
shows (*p* ∙ (*finfun-const b*)) = *finfun-const* (*p* ∙ *b*)
⟨*proof*⟩

lemma *finfun-apply-eqvt* [*eqvt*]:
shows (*p* ∙ (*finfun-apply f b*)) = *finfun-apply* (*p* ∙ *f*) (*p* ∙ *b*)
⟨*proof*⟩

7 Supp, Freshness and Supports

context *pt*
begin

definition
supp :: '*a* ⇒ *atom set*
where
supp x = {*a*. *infinite* {*b*. (*a* ⇌ *b*) ∙ *x* ≠ *x*}}

definition
fresh :: *atom* ⇒ '*a* ⇒ *bool* (- ♯ - [*55*, *55*] *55*)
where
a ♯ *x* ≡ *a* ∉ *supp x*

end

lemma *supp-conv-fresh*:
shows *supp x* = {*a*. ¬ *a* ♯ *x*}
⟨*proof*⟩

lemma *swap-rel-trans*:
assumes *sort-of a* = *sort-of b*
assumes *sort-of b* = *sort-of c*
assumes (*a* ⇌ *c*) ∙ *x* = *x*
assumes (*b* ⇌ *c*) ∙ *x* = *x*
shows (*a* ⇌ *b*) ∙ *x* = *x*
⟨*proof*⟩

lemma *swap-fresh-fresh*:
assumes *a*: *a* ♯ *x*
and *b*: *b* ♯ *x*

shows $(a \rightleftharpoons b) \cdot x = x$
⟨*proof*⟩

7.1 supp and fresh are equivariant

lemma *supp-eqvt* [*eqvt*]:
shows $p \cdot (supp\ x) = supp\ (p \cdot x)$
⟨*proof*⟩

lemma *fresh-eqvt* [*eqvt*]:
shows $p \cdot (a \sharp x) = (p \cdot a) \sharp (p \cdot x)$
⟨*proof*⟩

lemma *fresh-permute-iff*:
shows $(p \cdot a) \sharp (p \cdot x) \longleftrightarrow a \sharp x$
⟨*proof*⟩

lemma *fresh-permute-left*:
shows $a \sharp p \cdot x \longleftrightarrow -\ p \cdot a \sharp x$
⟨*proof*⟩

8 supports

definition
$supports :: atom\ set \Rightarrow 'a::pt \Rightarrow bool$ (**infixl** *supports 80*)
where
$S\ supports\ x \equiv \forall\, a\ b.\ (a \notin S \wedge b \notin S \longrightarrow (a \rightleftharpoons b) \cdot x = x)$

lemma *supp-is-subset*:
fixes $S :: atom\ set$
and $x :: 'a::pt$
assumes *a1*: $S\ supports\ x$
and *a2*: *finite* S
shows $(supp\ x) \subseteq S$
⟨*proof*⟩

lemma *supports-finite*:
fixes $S :: atom\ set$
and $x :: 'a::pt$
assumes *a1*: $S\ supports\ x$
and *a2*: *finite* S
shows *finite* $(supp\ x)$
⟨*proof*⟩

lemma *supp-supports*:
fixes $x :: 'a::pt$
shows $(supp\ x)\ supports\ x$
⟨*proof*⟩

lemma *supports-fresh*:
fixes $x :: 'a::pt$
assumes *a1*: S *supports* x
and *a2*: *finite* S
and *a3*: $a \notin S$
shows $a \sharp x$
⟨*proof*⟩

lemma *supp-is-least-supports*:
fixes $S :: atom\ set$
and $x :: 'a::pt$
assumes *a1*: S *supports* x
and *a2*: *finite* S
and *a3*: $\bigwedge S'.\ finite\ S' \Longrightarrow (S'\ supports\ x) \Longrightarrow S \subseteq S'$
shows $(supp\ x) = S$
⟨*proof*⟩

lemma *subsetCI*:
shows $(\bigwedge x.\ x \in A \Longrightarrow x \notin B \Longrightarrow False) \Longrightarrow A \subseteq B$
⟨*proof*⟩

lemma *finite-supp-unique*:
assumes *a1*: S *supports* x
assumes *a2*: *finite* S
assumes *a3*: $\bigwedge a\ b.\ [\![a \in S;\ b \notin S;\ sort\text{-}of\ a = sort\text{-}of\ b]\!] \Longrightarrow (a \rightleftharpoons b) \cdot x \neq x$
shows $(supp\ x) = S$
⟨*proof*⟩

9 Support w.r.t. relations

This definition is used for unquotient types, where alpha-equivalence does not coincide with equality.

definition
$supp\text{-}rel\ R\ x = \{a.\ infinite\ \{b.\ \neg(R\ ((a \rightleftharpoons b) \cdot x)\ x)\}\}$

10 Finitely-supported types

class $fs = pt +$
assumes *finite-supp*: *finite* $(supp\ x)$

lemma *pure-supp*:
fixes $x::'a::pure$
shows $supp\ x = \{\}$
⟨*proof*⟩

lemma *pure-fresh*:
fixes $x::'a::pure$

shows $a \sharp x$
⟨*proof*⟩

instance *pure* < *fs*
⟨*proof*⟩

10.1 Type *atom* is finitely-supported.

lemma *supp-atom*:
shows *supp* $a = \{a\}$
⟨*proof*⟩

lemma *fresh-atom*:
shows $a \sharp b \longleftrightarrow a \neq b$
⟨*proof*⟩

instance *atom* :: *fs*
⟨*proof*⟩

11 Type *perm* is finitely-supported.

lemma *perm-swap-eq*:
shows $(a \rightleftharpoons b) \cdot p = p \longleftrightarrow (p \cdot (a \rightleftharpoons b)) = (a \rightleftharpoons b)$
⟨*proof*⟩

lemma *supports-perm*:
shows $\{a.\ p \cdot a \neq a\}$ *supports* p
⟨*proof*⟩

lemma *finite-perm-lemma*:
shows *finite* $\{a::atom.\ p \cdot a \neq a\}$
⟨*proof*⟩

lemma *supp-perm*:
shows *supp* $p = \{a.\ p \cdot a \neq a\}$
⟨*proof*⟩

lemma *fresh-perm*:
shows $a \sharp p \longleftrightarrow p \cdot a = a$
⟨*proof*⟩

lemma *supp-swap*:
shows *supp* $(a \rightleftharpoons b) =$ (*if* $a = b \vee$ *sort-of* $a \neq$ *sort-of* b *then* $\{\}$ *else* $\{a, b\}$)
⟨*proof*⟩

lemma *fresh-swap*:
shows $a \sharp (b \rightleftharpoons c) \longleftrightarrow$ (*sort-of* $b \neq$ *sort-of* c) $\vee\ b = c \vee (a \sharp b \wedge a \sharp c)$
⟨*proof*⟩

lemma *fresh-zero-perm*:
shows $a \sharp (0::perm)$
⟨*proof*⟩

lemma *supp-zero-perm*:
shows $supp\ (0::perm) = \{\}$
⟨*proof*⟩

lemma *fresh-plus-perm*:
fixes $p\ q::perm$
assumes $a \sharp p\ a \sharp q$
shows $a \sharp (p + q)$
⟨*proof*⟩

lemma *supp-plus-perm*:
fixes $p\ q::perm$
shows $supp\ (p + q) \subseteq supp\ p \cup supp\ q$
⟨*proof*⟩

lemma *fresh-minus-perm*:
fixes $p::perm$
shows $a \sharp (-\ p) \longleftrightarrow a \sharp p$
⟨*proof*⟩

lemma *supp-minus-perm*:
fixes $p::perm$
shows $supp\ (-\ p) = supp\ p$
⟨*proof*⟩

lemma *plus-perm-eq*:
fixes $p\ q::perm$
assumes *asm*: $supp\ p \cap supp\ q = \{\}$
shows $p + q = q + p$
⟨*proof*⟩

lemma *supp-plus-perm-eq*:
fixes $p\ q::perm$
assumes *asm*: $supp\ p \cap supp\ q = \{\}$
shows $supp\ (p + q) = supp\ p \cup supp\ q$
⟨*proof*⟩

lemma *perm-eq-iff2*:
fixes $p\ q :: perm$
shows $p = q \longleftrightarrow (\forall a::atom \in supp\ p \cup supp\ q.\ p \cdot a = q \cdot a)$
⟨*proof*⟩

instance *perm* :: *fs*
⟨*proof*⟩

12 Finite Support instances for other types

12.1 Type $'a \times 'b$ is finitely-supported.

lemma *supp-Pair*:
shows $supp\ (x,\ y) = supp\ x \cup supp\ y$
⟨*proof*⟩

lemma *fresh-Pair*:
shows $a \sharp (x,\ y) \longleftrightarrow a \sharp x \wedge a \sharp y$
⟨*proof*⟩

lemma *supp-Unit*:
shows $supp\ () = \{\}$
⟨*proof*⟩

lemma *fresh-Unit*:
shows $a \sharp ()$
⟨*proof*⟩

instance *prod* :: (*fs*, *fs*) *fs*
⟨*proof*⟩

12.2 Type $'a + 'b$ is finitely supported

lemma *supp-Inl*:
shows $supp\ (Inl\ x) = supp\ x$
⟨*proof*⟩

lemma *supp-Inr*:
shows $supp\ (Inr\ x) = supp\ x$
⟨*proof*⟩

lemma *fresh-Inl*:
shows $a \sharp Inl\ x \longleftrightarrow a \sharp x$
⟨*proof*⟩

lemma *fresh-Inr*:
shows $a \sharp Inr\ y \longleftrightarrow a \sharp y$
⟨*proof*⟩

instance *sum* :: (*fs*, *fs*) *fs*
⟨*proof*⟩

12.3 Type $'a\ option$ is finitely supported

lemma *supp-None*:
shows $supp\ None = \{\}$
⟨*proof*⟩

lemma *supp-Some*:
shows *supp* (*Some x*) = *supp x*
⟨*proof*⟩

lemma *fresh-None*:
shows *a* ♯ *None*
⟨*proof*⟩

lemma *fresh-Some*:
shows *a* ♯ *Some x* ⟷ *a* ♯ *x*
⟨*proof*⟩

instance *option* :: (*fs*) *fs*
⟨*proof*⟩

12.3.1 Type *'a list* is finitely supported

lemma *supp-Nil*:
shows *supp* [] = {}
⟨*proof*⟩

lemma *fresh-Nil*:
shows *a* ♯ []
⟨*proof*⟩

lemma *supp-Cons*:
shows *supp* (*x* # *xs*) = *supp x* ∪ *supp xs*
⟨*proof*⟩

lemma *fresh-Cons*:
shows *a* ♯ (*x* # *xs*) ⟷ *a* ♯ *x* ∧ *a* ♯ *xs*
⟨*proof*⟩

lemma *supp-append*:
shows *supp* (*xs* @ *ys*) = *supp xs* ∪ *supp ys*
⟨*proof*⟩

lemma *fresh-append*:
shows *a* ♯ (*xs* @ *ys*) ⟷ *a* ♯ *xs* ∧ *a* ♯ *ys*
⟨*proof*⟩

lemma *supp-rev*:
shows *supp* (*rev xs*) = *supp xs*
⟨*proof*⟩

lemma *fresh-rev*:
shows *a* ♯ *rev xs* ⟷ *a* ♯ *xs*
⟨*proof*⟩

lemma *supp-removeAll*:
fixes x::*atom*
shows *supp* (*removeAll x xs*) = *supp xs* − {x}
⟨*proof*⟩

lemma *supp-of-atom-list*:
fixes *as*::*atom list*
shows *supp as* = *set as*
⟨*proof*⟩

instance *list* :: (*fs*) *fs*
⟨*proof*⟩

13 Support and Freshness for Applications

lemma *fresh-conv-MOST*:
shows $a \sharp x \longleftrightarrow (MOST\ b.\ (a \rightleftharpoons b) \cdot x = x)$
⟨*proof*⟩

lemma *fresh-fun-app*:
assumes $a \sharp f$ **and** $a \sharp x$
shows $a \sharp f\ x$
⟨*proof*⟩

lemma *supp-fun-app*:
shows *supp* $(f\ x) \subseteq (supp\ f) \cup (supp\ x)$
⟨*proof*⟩

13.1 Equivariance Predicate *eqvt* and *eqvt-at*

definition
$eqvt\ f \equiv \forall p.\ p \cdot f = f$

lemma *eqvt-boolI*:
fixes f::*bool*
shows *eqvt f*
⟨*proof*⟩

equivariance of a function at a given argument

definition
$eqvt\text{-}at\ f\ x \equiv \forall p.\ p \cdot (f\ x) = f\ (p \cdot x)$

lemma *eqvtI*:
shows $(\bigwedge p.\ p \cdot f \equiv f) \Longrightarrow eqvt\ f$
⟨*proof*⟩

lemma *eqvt-at-perm*:
assumes *eqvt-at f x*
shows *eqvt-at f* $(q \cdot x)$

⟨*proof*⟩

lemma *supp-fun-eqvt*:
assumes *a*: *eqvt f*
shows *supp f* = {}
⟨*proof*⟩

lemma *fresh-fun-eqvt*:
assumes *a*: *eqvt f*
shows *a* ♯ *f*
⟨*proof*⟩

lemma *fresh-fun-eqvt-app*:
assumes *a*: *eqvt f*
shows *a* ♯ *x* ⟹ *a* ♯ *f x*
⟨*proof*⟩

lemma *supp-fun-app-eqvt*:
assumes *a*: *eqvt f*
shows *supp* (*f x*) ⊆ *supp x*
⟨*proof*⟩

lemma *supp-eqvt-at*:
assumes *asm*: *eqvt-at f x*
and *fin*: *finite* (*supp x*)
shows *supp* (*f x*) ⊆ *supp x*
⟨*proof*⟩

lemma *finite-supp-eqvt-at*:
assumes *asm*: *eqvt-at f x*
and *fin*: *finite* (*supp x*)
shows *finite* (*supp* (*f x*))
⟨*proof*⟩

lemma *fresh-eqvt-at*:
assumes *asm*: *eqvt-at f x*
and *fin*: *finite* (*supp x*)
and *fresh*: *a* ♯ *x*
shows *a* ♯ *f x*
⟨*proof*⟩

for handling of freshness of functions

⟨*ML*⟩

13.2 helper functions for *nominal-functions*

lemma *THE-defaultI2*:
assumes ∃!*x*. *P x* ⋀*x*. *P x* ⟹ *Q x*
shows *Q* (*THE-default d P*)
⟨*proof*⟩

lemma *the-default-eqvt*:
assumes *unique*: $\exists!x.\ P\ x$
shows $(p \cdot (\mathit{THE\text{-}default}\ d\ P)) = (\mathit{THE\text{-}default}\ (p \cdot d)\ (p \cdot P))$
⟨*proof*⟩

lemma *fundef-ex1-eqvt*:
fixes $x::'a::pt$
assumes *f-def*: $f == (\lambda x::'a.\ \mathit{THE\text{-}default}\ (d\ x)\ (G\ x))$
assumes *eqvt*: *eqvt* G
assumes *ex1*: $\exists!y.\ G\ x\ y$
shows $(p \cdot (f\ x)) = f\ (p \cdot x)$
⟨*proof*⟩

lemma *fundef-ex1-eqvt-at*:
fixes $x::'a::pt$
assumes *f-def*: $f == (\lambda x::'a.\ \mathit{THE\text{-}default}\ (d\ x)\ (G\ x))$
assumes *eqvt*: *eqvt* G
assumes *ex1*: $\exists!y.\ G\ x\ y$
shows *eqvt-at* $f\ x$
⟨*proof*⟩

lemma *fundef-ex1-prop*:
fixes $x::'a::pt$
assumes *f-def*: $f == (\lambda x::'a.\ \mathit{THE\text{-}default}\ (d\ x)\ (G\ x))$
assumes *P-all*: $\bigwedge x\ y.\ G\ x\ y \Longrightarrow P\ x\ y$
assumes *ex1*: $\exists!y.\ G\ x\ y$
shows $P\ x\ (f\ x)$
⟨*proof*⟩

14 Support of Finite Sets of Finitely Supported Elements

support and freshness for atom sets

lemma *supp-finite-atom-set*:
fixes $S::atom\ set$
assumes *finite* S
shows *supp* $S = S$
⟨*proof*⟩

lemma *supp-cofinite-atom-set*:
fixes $S::atom\ set$
assumes *finite* $(UNIV - S)$
shows *supp* $S = (UNIV - S)$
⟨*proof*⟩

lemma *fresh-finite-atom-set*:
fixes $S::atom\ set$

assumes *finite S*
shows $a \sharp S \longleftrightarrow a \notin S$
⟨*proof*⟩

lemma *fresh-minus-atom-set*:
fixes *S*::*atom set*
assumes *finite S*
shows $a \sharp S - T \longleftrightarrow (a \notin T \longrightarrow a \sharp S)$
⟨*proof*⟩

lemma *Union-supports-set*:
shows $(\bigcup x \in S.\ supp\ x)\ supports\ S$
⟨*proof*⟩

lemma *Union-of-finite-supp-sets*:
fixes *S*::('*a*::*fs set*)
assumes *fin*: *finite S*
shows *finite* $(\bigcup x{\in}S.\ supp\ x)$
⟨*proof*⟩

lemma *Union-included-in-supp*:
fixes *S*::('*a*::*fs set*)
assumes *fin*: *finite S*
shows $(\bigcup x{\in}S.\ supp\ x) \subseteq supp\ S$
⟨*proof*⟩

lemma *supp-of-finite-sets*:
fixes *S*::('*a*::*fs set*)
assumes *fin*: *finite S*
shows $(supp\ S) = (\bigcup x{\in}S.\ supp\ x)$
⟨*proof*⟩

lemma *finite-sets-supp*:
fixes *S*::('*a*::*fs set*)
assumes *finite S*
shows *finite* (*supp S*)
⟨*proof*⟩

lemma *supp-of-finite-union*:
fixes *S T*::('*a*::*fs*) *set*
assumes *fin1*: *finite S*
and *fin2*: *finite T*
shows $supp\ (S \cup T) = supp\ S \cup supp\ T$
⟨*proof*⟩

lemma *fresh-finite-union*:
fixes *S T*::('*a*::*fs*) *set*
assumes *fin1*: *finite S*
and *fin2*: *finite T*

shows $a \sharp (S \cup T) \longleftrightarrow a \sharp S \wedge a \sharp T$
⟨*proof*⟩

lemma *supp-of-finite-insert*:
fixes $S::('a::fs)\ set$
assumes *fin*: *finite S*
shows *supp* (*insert x S*) = *supp x* ∪ *supp S*
⟨*proof*⟩

lemma *fresh-finite-insert*:
fixes $S::('a::fs)\ set$
assumes *fin*: *finite S*
shows $a \sharp (insert\ x\ S) \longleftrightarrow a \sharp x \wedge a \sharp S$
⟨*proof*⟩

lemma *supp-set-empty*:
shows *supp* {} = {}
⟨*proof*⟩

lemma *fresh-set-empty*:
shows $a \sharp \{\}$
⟨*proof*⟩

lemma *supp-set*:
fixes *xs* :: ('*a*::*fs*) *list*
shows *supp* (*set xs*) = *supp xs*
⟨*proof*⟩

lemma *fresh-set*:
fixes *xs* :: ('*a*::*fs*) *list*
shows $a \sharp (set\ xs) \longleftrightarrow a \sharp xs$
⟨*proof*⟩

14.1 Type *'a multiset* is finitely supported

lemma *set-mset-eqvt* [*eqvt*]:
shows $p \cdot (set\text{-}mset\ M) = set\text{-}mset\ (p \cdot M)$
⟨*proof*⟩

lemma *supp-set-mset*:
shows *supp* (*set-mset M*) ⊆ *supp M*
⟨*proof*⟩

lemma *Union-finite-multiset*:
fixes $M::'a::fs\ multiset$
shows *finite* ($\bigcup \{supp\ x \mid x.\ x \in\# M\}$)
⟨*proof*⟩

lemma *Union-supports-multiset*:

shows $\bigcup\{supp\ x \mid x.\ x \in\#\ M\}$ *supports* M
⟨*proof*⟩

lemma *Union-included-multiset*:
fixes M::('a::*fs multiset*)
shows $(\bigcup\{supp\ x \mid x.\ x \in\#\ M\}) \subseteq supp\ M$
⟨*proof*⟩

lemma *supp-of-multisets*:
fixes M::('a::*fs multiset*)
shows $(supp\ M) = (\bigcup\{supp\ x \mid x.\ x \in\#\ M\})$
⟨*proof*⟩

lemma *multisets-supp-finite*:
fixes M::('a::*fs multiset*)
shows *finite* $(supp\ M)$
⟨*proof*⟩

lemma *supp-of-multiset-union*:
fixes $M\ N$::('a::*fs*) *multiset*
shows $supp\ (M + N) = supp\ M \cup supp\ N$
⟨*proof*⟩

lemma *supp-empty-mset* [*simp*]:
shows $supp\ \{\#\} = \{\}$
⟨*proof*⟩

instance *multiset* :: (*fs*) *fs*
⟨*proof*⟩

14.2 Type '*a fset* is finitely supported

lemma *supp-fset* [*simp*]:
shows $supp\ (fset\ S) = supp\ S$
⟨*proof*⟩

lemma *supp-empty-fset* [*simp*]:
shows $supp\ \{||\} = \{\}$
⟨*proof*⟩

lemma *fresh-empty-fset*:
shows $a \sharp \{||\}$
⟨*proof*⟩

lemma *supp-finsert* [*simp*]:
fixes x::'a::*fs*
and S::'a *fset*
shows $supp\ (finsert\ x\ S) = supp\ x \cup supp\ S$
⟨*proof*⟩

lemma *fresh-finsert*:
fixes $x::'a::fs$
and $S::'a\ fset$
shows $a \sharp\ finsert\ x\ S \longleftrightarrow a \sharp\ x \wedge a \sharp\ S$
⟨*proof*⟩

lemma *fset-finite-supp*:
fixes $S::('a::fs)\ fset$
shows *finite* (*supp* S)
⟨*proof*⟩

lemma *supp-union-fset*:
fixes $S\ T::'a::fs\ fset$
shows $supp\ (S\ |\cup|\ T) = supp\ S \cup supp\ T$
⟨*proof*⟩

lemma *fresh-union-fset*:
fixes $S\ T::'a::fs\ fset$
shows $a \sharp\ S\ |\cup|\ T \longleftrightarrow a \sharp\ S \wedge a \sharp\ T$
⟨*proof*⟩

instance *fset* :: (*fs*) *fs*
⟨*proof*⟩

14.3 Type ('a, 'b) *finfun* is finitely supported

lemma *fresh-finfun-const*:
shows $a \sharp\ (finfun\text{-}const\ b) \longleftrightarrow a \sharp\ b$
⟨*proof*⟩

lemma *fresh-finfun-update*:
shows $[\![a \sharp\ f;\ a \sharp\ x;\ a \sharp\ y]\!] \Longrightarrow a \sharp\ finfun\text{-}update\ f\ x\ y$
⟨*proof*⟩

lemma *supp-finfun-const*:
shows $supp\ (finfun\text{-}const\ b) = supp(b)$
⟨*proof*⟩

lemma *supp-finfun-update*:
shows $supp\ (finfun\text{-}update\ f\ x\ y) \subseteq supp(f,\ x,\ y)$
⟨*proof*⟩

instance *finfun* :: (*fs*, *fs*) *fs*
⟨*proof*⟩

15 Freshness and Fresh-Star

lemma *fresh-Unit-elim*:

shows $(a \sharp () \Longrightarrow PROP\ C) \equiv PROP\ C$
⟨*proof*⟩

lemma *fresh-Pair-elim*:
shows $(a \sharp (x, y) \Longrightarrow PROP\ C) \equiv (a \sharp x \Longrightarrow a \sharp y \Longrightarrow PROP\ C)$
⟨*proof*⟩

lemma [*simp*]:
shows $a \sharp x1 \Longrightarrow a \sharp x2 \Longrightarrow a \sharp (x1, x2)$
⟨*proof*⟩

lemma *fresh-PairD*:
shows $a \sharp (x, y) \Longrightarrow a \sharp x$
and $a \sharp (x, y) \Longrightarrow a \sharp y$
⟨*proof*⟩

⟨*ML*⟩

The fresh-star generalisation of fresh is used in strong induction principles.

definition
fresh-star :: *atom set* ⇒ *'a::pt* ⇒ *bool* (- ♯* - [*80*,*80*] *80*)
where
$as \sharp* x \equiv \forall a \in as.\ a \sharp x$

lemma *fresh-star-supp-conv*:
shows $supp\ x \sharp* y \Longrightarrow supp\ y \sharp* x$
⟨*proof*⟩

lemma *fresh-star-perm-set-conv*:
fixes *p::perm*
assumes *fresh*: $as \sharp* p$
and *fin*: *finite as*
shows $supp\ p \sharp* as$
⟨*proof*⟩

lemma *fresh-star-atom-set-conv*:
assumes *fresh*: $as \sharp* bs$
and *fin*: *finite as finite bs*
shows $bs \sharp* as$
⟨*proof*⟩

lemma *atom-fresh-star-disjoint*:
assumes *fin*: *finite bs*
shows $as \sharp* bs \longleftrightarrow (as \cap bs = \{\})$

⟨*proof*⟩

lemma *fresh-star-Pair*:
shows $as \sharp* (x, y) = (as \sharp* x \wedge as \sharp* y)$
⟨*proof*⟩

lemma *fresh-star-list*:
shows $as \sharp* (xs @ ys) \longleftrightarrow as \sharp* xs \wedge as \sharp* ys$
and $as \sharp* (x \# xs) \longleftrightarrow as \sharp* x \wedge as \sharp* xs$
and $as \sharp* []$
⟨*proof*⟩

lemma *fresh-star-set*:
fixes $xs::('a::fs)\ list$
shows $as \sharp* set\ xs \longleftrightarrow as \sharp* xs$
⟨*proof*⟩

lemma *fresh-star-singleton*:
fixes $a::atom$
shows $as \sharp* \{a\} \longleftrightarrow as \sharp* a$
⟨*proof*⟩

lemma *fresh-star-fset*:
fixes $xs::('a::fs)\ list$
shows $as \sharp* fset\ S \longleftrightarrow as \sharp* S$
⟨*proof*⟩

lemma *fresh-star-Un*:
shows $(as \cup bs) \sharp* x = (as \sharp* x \wedge bs \sharp* x)$
⟨*proof*⟩

lemma *fresh-star-insert*:
shows $(insert\ a\ as) \sharp* x = (a \sharp x \wedge as \sharp* x)$
⟨*proof*⟩

lemma *fresh-star-Un-elim*:
$((as \cup bs) \sharp* x \Longrightarrow PROP\ C) \equiv (as \sharp* x \Longrightarrow bs \sharp* x \Longrightarrow PROP\ C)$
⟨*proof*⟩

lemma *fresh-star-insert-elim*:
$(insert\ a\ as \sharp* x \Longrightarrow PROP\ C) \equiv (a \sharp x \Longrightarrow as \sharp* x \Longrightarrow PROP\ C)$
⟨*proof*⟩

lemma *fresh-star-empty-elim*:
$(\{\} \sharp* x \Longrightarrow PROP\ C) \equiv PROP\ C$
⟨*proof*⟩

lemma *fresh-star-Unit-elim*:
shows $(a \sharp* () \Longrightarrow PROP\ C) \equiv PROP\ C$

⟨*proof*⟩

lemma *fresh-star-Pair-elim*:
shows $(a \sharp* (x, y) \Longrightarrow PROP\ C) \equiv (a \sharp* x \Longrightarrow a \sharp* y \Longrightarrow PROP\ C)$
⟨*proof*⟩

lemma *fresh-star-zero*:
shows $as \sharp* (0::perm)$
⟨*proof*⟩

lemma *fresh-star-plus*:
fixes $p\ q::perm$
shows $[\![a \sharp* p;\ \ a \sharp* q]\!] \Longrightarrow a \sharp* (p + q)$
⟨*proof*⟩

lemma *fresh-star-permute-iff*:
shows $(p \cdot a) \sharp* (p \cdot x) \longleftrightarrow a \sharp* x$
⟨*proof*⟩

lemma *fresh-star-eqvt* [*eqvt*]:
shows $p \cdot (as \sharp* x) \longleftrightarrow (p \cdot as) \sharp* (p \cdot x)$
⟨*proof*⟩

16 Induction principle for permutations

lemma *smaller-supp*:
assumes *a*: $a \in supp\ p$
shows $supp\ ((p \cdot a \rightleftharpoons a) + p) \subset supp\ p$
⟨*proof*⟩

lemma *perm-struct-induct*[*consumes 1*, *case-names zero swap*]:
assumes *S*: $supp\ p \subseteq S$
and *zero*: $P\ 0$
and *swap*: $\bigwedge p\ a\ b.\ [\![P\ p;\ supp\ p \subseteq S;\ a \in S;\ b \in S;\ a \neq b;\ sort\text{-}of\ a = sort\text{-}of\ b]\!] \Longrightarrow P\ ((a \rightleftharpoons b) + p)$
shows $P\ p$
⟨*proof*⟩

lemma *perm-simple-struct-induct*[*case-names zero swap*]:
assumes *zero*: $P\ 0$
and *swap*: $\bigwedge p\ a\ b.\ [\![P\ p;\ a \neq b;\ sort\text{-}of\ a = sort\text{-}of\ b]\!] \Longrightarrow P\ ((a \rightleftharpoons b) + p)$
shows $P\ p$
⟨*proof*⟩

lemma *perm-struct-induct2*[*consumes 1*, *case-names zero swap plus*]:
assumes *S*: $supp\ p \subseteq S$
assumes *zero*: $P\ 0$
assumes *swap*: $\bigwedge a\ b.\ [\![sort\text{-}of\ a = sort\text{-}of\ b;\ a \neq b;\ a \in S;\ b \in S]\!] \Longrightarrow P\ (a \rightleftharpoons$

b)
assumes *plus*: $\bigwedge p1\ p2.\ [\![P\ p1;\ P\ p2;\ supp\ p1 \subseteq S;\ supp\ p2 \subseteq S]\!] \Longrightarrow P\ (p1 + p2)$
shows $P\ p$
⟨*proof*⟩

lemma *perm-simple-struct-induct2*[*case-names zero swap plus*]:
assumes *zero*: $P\ 0$
assumes *swap*: $\bigwedge a\ b.\ [\![sort\text{-}of\ a = sort\text{-}of\ b;\ a \neq b]\!] \Longrightarrow P\ (a \rightleftharpoons b)$
assumes *plus*: $\bigwedge p1\ p2.\ [\![P\ p1;\ P\ p2]\!] \Longrightarrow P\ (p1 + p2)$
shows $P\ p$
⟨*proof*⟩

lemma *supp-perm-singleton*:
fixes *p*::*perm*
shows $supp\ p \subseteq \{b\} \longleftrightarrow p = 0$
⟨*proof*⟩

lemma *supp-perm-pair*:
fixes *p*::*perm*
shows $supp\ p \subseteq \{a, b\} \longleftrightarrow p = 0 \vee p = (b \rightleftharpoons a)$
⟨*proof*⟩

lemma *supp-perm-eq*:
assumes $(supp\ x)\ \sharp*\ p$
shows $p \cdot x = x$
⟨*proof*⟩

same lemma as above, but proved with a different induction principle

lemma *supp-perm-eq-test*:
assumes $(supp\ x)\ \sharp*\ p$
shows $p \cdot x = x$
⟨*proof*⟩

lemma *perm-supp-eq*:
assumes *a*: $(supp\ p)\ \sharp*\ x$
shows $p \cdot x = x$
⟨*proof*⟩

lemma *supp-perm-perm-eq*:
assumes *a*: $\forall a \in supp\ x.\ p \cdot a = q \cdot a$
shows $p \cdot x = q \cdot x$
⟨*proof*⟩

disagreement set

definition
$dset :: perm \Rightarrow perm \Rightarrow atom\ set$
where
$dset\ p\ q = \{a::atom.\ p \cdot a \neq q \cdot a\}$

lemma *ds-fresh*:
assumes *dset p q* $\sharp*$ *x*
shows $p \cdot x = q \cdot x$
⟨*proof*⟩

lemma *atom-set-perm-eq*:
assumes *a*: *as* $\sharp*$ *p*
shows $p \cdot as = as$
⟨*proof*⟩

17 Avoiding of atom sets

For every set of atoms, there is another set of atoms avoiding a finitely supported c and there is a permutation which 'translates' between both sets.

lemma *at-set-avoiding-aux*:
fixes *Xs*::*atom set*
and *As*::*atom set*
assumes *b*: $Xs \subseteq As$
and *c*: *finite As*
shows $\exists p.\ (p \cdot Xs) \cap As = \{\} \wedge (supp\ p) = (Xs \cup (p \cdot Xs))$
⟨*proof*⟩

lemma *at-set-avoiding*:
assumes *a*: *finite Xs*
and *b*: *finite* (*supp c*)
obtains *p*::*perm* **where** $(p \cdot Xs)\sharp*c$ **and** $(supp\ p) = (Xs \cup (p \cdot Xs))$
⟨*proof*⟩

lemma *at-set-avoiding1*:
assumes *finite xs*
and *finite* (*supp c*)
shows $\exists p.\ (p \cdot xs)\ \sharp*\ c$
⟨*proof*⟩

lemma *at-set-avoiding2*:
assumes *finite xs*
and *finite* (*supp c*) *finite* (*supp x*)
and *xs* $\sharp*$ *x*
shows $\exists p.\ (p \cdot xs)\ \sharp*\ c \wedge supp\ x\ \sharp*\ p$
⟨*proof*⟩

lemma *at-set-avoiding3*:
assumes *finite xs*
and *finite* (*supp c*) *finite* (*supp x*)
and *xs* $\sharp*$ *x*
shows $\exists p.\ (p \cdot xs)\ \sharp*\ c \wedge supp\ x\ \sharp*\ p \wedge supp\ p = xs \cup (p \cdot xs)$

⟨*proof*⟩

lemma *at-set-avoiding2-atom*:
assumes *finite* (*supp c*) *finite* (*supp x*)
and *b*: *a* ♯ *x*
shows ∃ *p*. (*p* ∙ *a*) ♯ *c* ∧ *supp x* ♯* *p*
⟨*proof*⟩

18 Renaming permutations

lemma *set-renaming-perm*:
assumes *b*: *finite bs*
shows ∃ *q*. (∀ *b* ∈ *bs*. *q* ∙ *b* = *p* ∙ *b*) ∧ *supp q* ⊆ *bs* ∪ (*p* ∙ *bs*)
⟨*proof*⟩

lemma *set-renaming-perm2*:
shows ∃ *q*. (∀ *b* ∈ *bs*. *q* ∙ *b* = *p* ∙ *b*) ∧ *supp q* ⊆ *bs* ∪ (*p* ∙ *bs*)
⟨*proof*⟩

lemma *list-renaming-perm*:
shows ∃ *q*. (∀ *b* ∈ *set bs*. *q* ∙ *b* = *p* ∙ *b*) ∧ *supp q* ⊆ *set bs* ∪ (*p* ∙ *set bs*)
⟨*proof*⟩

19 Concrete Atoms Types

Class *at-base* allows types containing multiple sorts of atoms. Class *at* only allows types with a single sort.

class *at-base* = *pt* +
fixes *atom* :: *'a* ⇒ *atom*
assumes *atom-eq-iff* [*simp*]: *atom a* = *atom b* ⟷ *a* = *b*
assumes *atom-eqvt*: *p* ∙ (*atom a*) = *atom* (*p* ∙ *a*)

declare *atom-eqvt* [*eqvt*]

class *at* = *at-base* +
assumes *sort-of-atom-eq* [*simp*]: *sort-of* (*atom a*) = *sort-of* (*atom b*)

lemma *sort-ineq* [*simp*]:
assumes *sort-of* (*atom a*) ≠ *sort-of* (*atom b*)
shows *atom a* ≠ *atom b*
⟨*proof*⟩

lemma *supp-at-base*:
fixes *a*::*'a*::*at-base*
shows *supp a* = {*atom a*}
⟨*proof*⟩

lemma *fresh-at-base*:

shows *sort-of a* $\neq$ *sort-of* (*atom b*) $\Longrightarrow$ *a* $\sharp$ *b*
and *a* $\sharp$ *b* $\longleftrightarrow$ *a* $\neq$ *atom b*
⟨*proof*⟩

lemma *fresh-ineq-at-base* [*simp*]:
shows *a* $\neq$ *atom b* $\Longrightarrow$ *a* $\sharp$ *b*
⟨*proof*⟩

lemma *fresh-atom-at-base* [*simp*]:
fixes *b*::′*a*::*at-base*
shows *a* $\sharp$ *atom b* $\longleftrightarrow$ *a* $\sharp$ *b*
⟨*proof*⟩

lemma *fresh-star-atom-at-base*:
fixes *b*::′*a*::*at-base*
shows *as* $\sharp*$ *atom b* $\longleftrightarrow$ *as* $\sharp*$ *b*
⟨*proof*⟩

lemma *if-fresh-at-base* [*simp*]:
shows *atom a* $\sharp$ *x* $\Longrightarrow$ *P* (*if a* = *x then t else s*) = *P s*
and *atom a* $\sharp$ *x* $\Longrightarrow$ *P* (*if x* = *a then t else s*) = *P s*
⟨*proof*⟩

⟨*ML*⟩

instance *at-base* < *fs*
⟨*proof*⟩

lemma *at-base-infinite* [*simp*]:
shows *infinite* (*UNIV* :: ′*a*::*at-base set*) (**is** *infinite ?U*)
⟨*proof*⟩

lemma *swap-at-base-simps* [*simp*]:
fixes *x y*::′*a*::*at-base*
shows *sort-of* (*atom x*) = *sort-of* (*atom y*) $\Longrightarrow$ (*atom x* $\rightleftharpoons$ *atom y*) $\cdot$ *x* = *y*
and *sort-of* (*atom x*) = *sort-of* (*atom y*) $\Longrightarrow$ (*atom x* $\rightleftharpoons$ *atom y*) $\cdot$ *y* = *x*
and *atom x* $\neq$ *a* $\Longrightarrow$ *atom x* $\neq$ *b* $\Longrightarrow$ (*a* $\rightleftharpoons$ *b*) $\cdot$ *x* = *x*
⟨*proof*⟩

lemma *obtain-at-base*:
assumes *X*: *finite X*
obtains *a*::′*a*::*at-base* **where** *atom a* $\notin$ *X*
⟨*proof*⟩

lemma *obtain-fresh′*:

assumes *fin*: *finite* (*supp x*)
obtains *a*::*'a*::*at-base* **where** *atom a* ♯ *x*
⟨*proof*⟩

lemma *obtain-fresh*:
fixes *x*::*'b*::*fs*
obtains *a*::*'a*::*at-base* **where** *atom a* ♯ *x*
⟨*proof*⟩

lemma *supp-finite-set-at-base*:
assumes *a*: *finite S*
shows *supp S* = *atom* ‘ *S*
⟨*proof*⟩

lemma *fresh-finite-set-at-base*:
fixes *a*::*'a*::*at-base*
assumes *a*: *finite S*
shows *atom a* ♯ *S* ⟷ *a* ∉ *S*
⟨*proof*⟩

lemma *fresh-at-base-permute-iff* [*simp*]:
fixes *a*::*'a*::*at-base*
shows *atom* (*p* ∙ *a*) ♯ *p* ∙ *x* ⟷ *atom a* ♯ *x*
⟨*proof*⟩

lemma *fresh-at-base-permI*:
shows *atom a* ♯ *p* ⟹ *p* ∙ *a* = *a*
⟨*proof*⟩

20 Infrastructure for concrete atom types

definition
flip :: *'a*::*at-base* ⇒ *'a* ⇒ *perm* (*'*(- ↔ -*'*))
where
(*a* ↔ *b*) = (*atom a* ⇌ *atom b*)

lemma *flip-fresh-fresh*:
assumes *atom a* ♯ *x atom b* ♯ *x*
shows (*a* ↔ *b*) ∙ *x* = *x*
⟨*proof*⟩

lemma *flip-self* [*simp*]: (*a* ↔ *a*) = *0*
⟨*proof*⟩

lemma *flip-commute*: (*a* ↔ *b*) = (*b* ↔ *a*)
⟨*proof*⟩

lemma *minus-flip* [*simp*]: $-(a \leftrightarrow b) = (a \leftrightarrow b)$
⟨*proof*⟩

lemma *add-flip-cancel*: $(a \leftrightarrow b) + (a \leftrightarrow b) = 0$
⟨*proof*⟩

lemma *permute-flip-cancel* [*simp*]: $(a \leftrightarrow b) \cdot (a \leftrightarrow b) \cdot x = x$
⟨*proof*⟩

lemma *permute-flip-cancel2* [*simp*]: $(a \leftrightarrow b) \cdot (b \leftrightarrow a) \cdot x = x$
⟨*proof*⟩

lemma *flip-eqvt* [*eqvt*]:
shows $p \cdot (a \leftrightarrow b) = (p \cdot a \leftrightarrow p \cdot b)$
⟨*proof*⟩

lemma *flip-at-base-simps* [*simp*]:
shows *sort-of* (*atom a*) = *sort-of* (*atom b*) $\Longrightarrow (a \leftrightarrow b) \cdot a = b$
and *sort-of* (*atom a*) = *sort-of* (*atom b*) $\Longrightarrow (a \leftrightarrow b) \cdot b = a$
and $[\![a \neq c;\ b \neq c]\!] \Longrightarrow (a \leftrightarrow b) \cdot c = c$
and *sort-of* (*atom a*) ≠ *sort-of* (*atom b*) $\Longrightarrow (a \leftrightarrow b) \cdot x = x$
⟨*proof*⟩

the following two lemmas do not hold for *at-base*, only for single sort atoms from at

lemma *flip-triple*:
fixes *a b c*::′*a*::*at*
assumes $a \neq b$ **and** $c \neq b$
shows $(a \leftrightarrow c) + (b \leftrightarrow c) + (a \leftrightarrow c) = (a \leftrightarrow b)$
⟨*proof*⟩

lemma *permute-flip-at*:
fixes *a b c*::′*a*::*at*
shows $(a \leftrightarrow b) \cdot c = (\textit{if } c = a \textit{ then } b \textit{ else if } c = b \textit{ then } a \textit{ else } c)$
⟨*proof*⟩

lemma *flip-at-simps* [*simp*]:
fixes *a b*::′*a*::*at*
shows $(a \leftrightarrow b) \cdot a = b$
and $(a \leftrightarrow b) \cdot b = a$
⟨*proof*⟩

20.1 Syntax for coercing at-elements to the atom-type

syntax
-*atom-constrain* :: *logic* ⇒ *type* ⇒ *logic* (-:::- [*4*, *0*] *3*)

translations
-*atom-constrain a t* => *CONST atom* (-*constrain a t*)

20.2 A lemma for proving instances of class *at*.

⟨*ML*⟩

New atom types are defined as subtypes of *atom*.

lemma *exists-eq-simple-sort*:
shows $\exists a.\ a \in \{a.\ sort\text{-}of\ a = s\}$
⟨*proof*⟩

lemma *exists-eq-sort*:
shows $\exists a.\ a \in \{a.\ sort\text{-}of\ a \in range\ sort\text{-}fun\}$
⟨*proof*⟩

lemma *at-base-class*:
fixes *sort-fun* :: $'b \Rightarrow atom\text{-}sort$
fixes *Rep* :: $'a \Rightarrow atom$ **and** *Abs* :: $atom \Rightarrow 'a$
assumes *type*: *type-definition Rep Abs* $\{a.\ sort\text{-}of\ a \in range\ sort\text{-}fun\}$
assumes *atom-def*: $\bigwedge a.\ atom\ a = Rep\ a$
assumes *permute-def*: $\bigwedge p\ a.\ p \cdot a = Abs\ (p \cdot Rep\ a)$
shows *OFCLASS*($'a$, *at-base-class*)
⟨*proof*⟩

lemma *at-class*:
fixes *s* :: *atom-sort*
fixes *Rep* :: $'a \Rightarrow atom$ **and** *Abs* :: $atom \Rightarrow 'a$
assumes *type*: *type-definition Rep Abs* $\{a.\ sort\text{-}of\ a = s\}$
assumes *atom-def*: $\bigwedge a.\ atom\ a = Rep\ a$
assumes *permute-def*: $\bigwedge p\ a.\ p \cdot a = Abs\ (p \cdot Rep\ a)$
shows *OFCLASS*($'a$, *at-class*)
⟨*proof*⟩

lemma *at-class-sort*:
fixes *s* :: *atom-sort*
fixes *Rep* :: $'a \Rightarrow atom$ **and** *Abs* :: $atom \Rightarrow 'a$
fixes $a{::}'a$
assumes *type*: *type-definition Rep Abs* $\{a.\ sort\text{-}of\ a = s\}$
assumes *atom-def*: $\bigwedge a.\ atom\ a = Rep\ a$
shows *sort-of* $(atom\ a) = s$
⟨*proof*⟩

⟨*ML*⟩

21 Library functions for the nominal infrastructure

⟨*ML*⟩

22 The freshness lemma according to Andy Pitts

lemma *freshness-lemma*:
fixes $h :: 'a::at \Rightarrow 'b::pt$
assumes a: $\exists\, a.\ atom\ a \sharp (h,\ h\ a)$
shows $\exists\, x.\ \forall\, a.\ atom\ a \sharp h \longrightarrow h\ a = x$
⟨*proof*⟩

lemma *freshness-lemma-unique*:
fixes $h :: 'a::at \Rightarrow 'b::pt$
assumes a: $\exists\, a.\ atom\ a \sharp (h,\ h\ a)$
shows $\exists!x.\ \forall\, a.\ atom\ a \sharp h \longrightarrow h\ a = x$
⟨*proof*⟩

packaging the freshness lemma into a function

definition
$Fresh :: ('a::at \Rightarrow 'b::pt) \Rightarrow 'b$
where
$Fresh\ h = (THE\ x.\ \forall\, a.\ atom\ a \sharp h \longrightarrow h\ a = x)$

lemma *Fresh-apply*:
fixes $h :: 'a::at \Rightarrow 'b::pt$
assumes a: $\exists\, a.\ atom\ a \sharp (h,\ h\ a)$
assumes b: $atom\ a \sharp h$
shows $Fresh\ h = h\ a$
⟨*proof*⟩

lemma *Fresh-apply'*:
fixes $h :: 'a::at \Rightarrow 'b::pt$
assumes a: $atom\ a \sharp h\ atom\ a \sharp h\ a$
shows $Fresh\ h = h\ a$
⟨*proof*⟩

⟨*ML*⟩

lemma *Fresh-eqvt*:
fixes $h :: 'a::at \Rightarrow 'b::pt$
assumes a: $\exists\, a.\ atom\ a \sharp (h,\ h\ a)$
shows $p \cdot (Fresh\ h) = Fresh\ (p \cdot h)$
⟨*proof*⟩

lemma *Fresh-supports*:
fixes $h :: 'a::at \Rightarrow 'b::pt$
assumes a: $\exists\, a.\ atom\ a \sharp (h,\ h\ a)$
shows $(supp\ h)\ supports\ (Fresh\ h)$
⟨*proof*⟩

notation *Fresh* (**binder** *FRESH* *10*)

lemma *FRESH-f-iff*:
fixes $P :: 'a::at \Rightarrow 'b::pure$
fixes $f :: 'b \Rightarrow 'c::pure$
assumes P: *finite* (*supp* P)
shows (*FRESH* x. f (P x)) = f (*FRESH* x. P x)
⟨*proof*⟩

lemma *FRESH-binop-iff*:
fixes $P :: 'a::at \Rightarrow 'b::pure$
fixes $Q :: 'a::at \Rightarrow 'c::pure$
fixes *binop* $:: 'b \Rightarrow 'c \Rightarrow 'd::pure$
assumes P: *finite* (*supp* P)
and Q: *finite* (*supp* Q)
shows (*FRESH* x. *binop* (P x) (Q x)) = *binop* (*FRESH* x. P x) (*FRESH* x. Q x)
⟨*proof*⟩

lemma *FRESH-conj-iff*:
fixes P $Q :: 'a::at \Rightarrow bool$
assumes P: *finite* (*supp* P) **and** Q: *finite* (*supp* Q)
shows (*FRESH* x. P $x \wedge$ Q x) $\longleftrightarrow$ (*FRESH* x. P x) $\wedge$ (*FRESH* x. Q x)
⟨*proof*⟩

lemma *FRESH-disj-iff*:
fixes P $Q :: 'a::at \Rightarrow bool$
assumes P: *finite* (*supp* P) **and** Q: *finite* (*supp* Q)
shows (*FRESH* x. P $x \vee$ Q x) $\longleftrightarrow$ (*FRESH* x. P x) $\vee$ (*FRESH* x. Q x)
⟨*proof*⟩

23 Automation for creating concrete atom types

At the moment only single-sort concrete atoms are supported.

⟨*ML*⟩

24 Automatic equivariance procedure for inductive definitions

⟨*ML*⟩

end
theory *Nominal2-Abs*
imports *Nominal2-Base*
HOL−Library.Quotient-List
HOL−Library.Quotient-Product
begin

25 Abstractions

fun
alpha-set
where
alpha-set[*simp del*]:
alpha-set (*bs*, *x*) *R f p* (*cs*, *y*) ⟷
f x − *bs* = *f y* − *cs* ∧
(*f x* − *bs*) ♯* *p* ∧
R (*p* ∙ *x*) *y* ∧
p ∙ *bs* = *cs*

fun
alpha-res
where
alpha-res[*simp del*]:
alpha-res (*bs*, *x*) *R f p* (*cs*, *y*) ⟷
f x − *bs* = *f y* − *cs* ∧
(*f x* − *bs*) ♯* *p* ∧
R (*p* ∙ *x*) *y*

fun
alpha-lst
where
alpha-lst[*simp del*]:
alpha-lst (*bs*, *x*) *R f p* (*cs*, *y*) ⟷
f x − *set bs* = *f y* − *set cs* ∧
(*f x* − *set bs*) ♯* *p* ∧
R (*p* ∙ *x*) *y* ∧
p ∙ *bs* = *cs*

lemmas *alphas* = *alpha-set.simps alpha-res.simps alpha-lst.simps*

notation
alpha-set (- ≈*set* - - - - [*100*, *100*, *100*, *100*, *100*] *100*) **and**
alpha-res (- ≈*res* - - - - [*100*, *100*, *100*, *100*, *100*] *100*) **and**
alpha-lst (- ≈*lst* - - - - [*100*, *100*, *100*, *100*, *100*] *100*)

26 Mono

lemma [*mono*]:
shows *R1* ≤ *R2* ⟹ *alpha-set bs R1* ≤ *alpha-set bs R2*
and *R1* ≤ *R2* ⟹ *alpha-res bs R1* ≤ *alpha-res bs R2*
and *R1* ≤ *R2* ⟹ *alpha-lst cs R1* ≤ *alpha-lst cs R2*
⟨*proof*⟩

27 Equivariance

lemma *alpha-eqvt*[*eqvt*]:
shows $(bs, x) \approx set\ R\ f\ q\ (cs, y) \Longrightarrow (p \cdot bs, p \cdot x) \approx set\ (p \cdot R)\ (p \cdot f)\ (p \cdot q)\ (p \cdot cs, p \cdot y)$
and $(bs, x) \approx res\ R\ f\ q\ (cs, y) \Longrightarrow (p \cdot bs, p \cdot x) \approx res\ (p \cdot R)\ (p \cdot f)\ (p \cdot q)\ (p \cdot cs, p \cdot y)$
and $(ds, x) \approx lst\ R\ f\ q\ (es, y) \Longrightarrow (p \cdot ds, p \cdot x) \approx lst\ (p \cdot R)\ (p \cdot f)\ (p \cdot q)\ (p \cdot es, p \cdot y)$
⟨*proof*⟩

28 Equivalence

lemma *alpha-refl*:
assumes *a*: $R\ x\ x$
shows $(bs, x) \approx set\ R\ f\ 0\ (bs, x)$
and $(bs, x) \approx res\ R\ f\ 0\ (bs, x)$
and $(cs, x) \approx lst\ R\ f\ 0\ (cs, x)$
⟨*proof*⟩

lemma *alpha-sym*:
assumes *a*: $R\ (p \cdot x)\ y \Longrightarrow R\ (-\ p \cdot y)\ x$
shows $(bs, x) \approx set\ R\ f\ p\ (cs, y) \Longrightarrow (cs, y) \approx set\ R\ f\ (-\ p)\ (bs, x)$
and $(bs, x) \approx res\ R\ f\ p\ (cs, y) \Longrightarrow (cs, y) \approx res\ R\ f\ (-\ p)\ (bs, x)$
and $(ds, x) \approx lst\ R\ f\ p\ (es, y) \Longrightarrow (es, y) \approx lst\ R\ f\ (-\ p)\ (ds, x)$
⟨*proof*⟩

lemma *alpha-trans*:
assumes *a*: $[\![R\ (p \cdot x)\ y;\ R\ (q \cdot y)\ z]\!] \Longrightarrow R\ ((q + p) \cdot x)\ z$
shows $[\![(bs, x) \approx set\ R\ f\ p\ (cs, y);\ (cs, y) \approx set\ R\ f\ q\ (ds, z)]\!] \Longrightarrow (bs, x) \approx set\ R\ f\ (q + p)\ (ds, z)$
and $[\![(bs, x) \approx res\ R\ f\ p\ (cs, y);\ (cs, y) \approx res\ R\ f\ q\ (ds, z)]\!] \Longrightarrow (bs, x) \approx res\ R\ f\ (q + p)\ (ds, z)$
and $[\![(es, x) \approx lst\ R\ f\ p\ (gs, y);\ (gs, y) \approx lst\ R\ f\ q\ (hs, z)]\!] \Longrightarrow (es, x) \approx lst\ R\ f\ (q + p)\ (hs, z)$
⟨*proof*⟩

lemma *alpha-sym-eqvt*:
assumes *a*: $R\ (p \cdot x)\ y \Longrightarrow R\ y\ (p \cdot x)$
and *b*: $p \cdot R = R$
shows $(bs, x) \approx set\ R\ f\ p\ (cs, y) \Longrightarrow (cs, y) \approx set\ R\ f\ (-\ p)\ (bs, x)$
and $(bs, x) \approx res\ R\ f\ p\ (cs, y) \Longrightarrow (cs, y) \approx res\ R\ f\ (-\ p)\ (bs, x)$
and $(ds, x) \approx lst\ R\ f\ p\ (es, y) \Longrightarrow (es, y) \approx lst\ R\ f\ (-\ p)\ (ds, x)$
⟨*proof*⟩

lemma *alpha-set-trans-eqvt*:
assumes *b*: $(cs, y) \approx set\ R\ f\ q\ (ds, z)$
and *a*: $(bs, x) \approx set\ R\ f\ p\ (cs, y)$
and *d*: $q \cdot R = R$

and c: $[\![R\ (p \cdot x)\ y;\ R\ y\ (-\ q \cdot z)]\!] \Longrightarrow R\ (p \cdot x)\ (-\ q \cdot z)$
shows $(bs,\ x) \approx set\ R\ f\ (q + p)\ (ds,\ z)$
⟨*proof*⟩

lemma *alpha-res-trans-eqvt*:
assumes b: $(cs,\ y) \approx res\ R\ f\ q\ (ds,\ z)$
and a: $(bs,\ x) \approx res\ R\ f\ p\ (cs,\ y)$
and d: $q \cdot R = R$
and c: $[\![R\ (p \cdot x)\ y;\ R\ y\ (-\ q \cdot z)]\!] \Longrightarrow R\ (p \cdot x)\ (-\ q \cdot z)$
shows $(bs,\ x) \approx res\ R\ f\ (q + p)\ (ds,\ z)$
⟨*proof*⟩

lemma *alpha-lst-trans-eqvt*:
assumes b: $(cs,\ y) \approx lst\ R\ f\ q\ (ds,\ z)$
and a: $(bs,\ x) \approx lst\ R\ f\ p\ (cs,\ y)$
and d: $q \cdot R = R$
and c: $[\![R\ (p \cdot x)\ y;\ R\ y\ (-\ q \cdot z)]\!] \Longrightarrow R\ (p \cdot x)\ (-\ q \cdot z)$
shows $(bs,\ x) \approx lst\ R\ f\ (q + p)\ (ds,\ z)$
⟨*proof*⟩

lemmas *alpha-trans-eqvt* = *alpha-set-trans-eqvt alpha-res-trans-eqvt alpha-lst-trans-eqvt*

29 General Abstractions

fun
alpha-abs-set
where
[*simp del*]:
$alpha\text{-}abs\text{-}set\ (bs,\ x)\ (cs,\ y) \longleftrightarrow (\exists\, p.\ (bs,\ x) \approx set\ ((=))\ supp\ p\ (cs,\ y))$

fun
alpha-abs-lst
where
[*simp del*]:
$alpha\text{-}abs\text{-}lst\ (bs,\ x)\ (cs,\ y) \longleftrightarrow (\exists\, p.\ (bs,\ x) \approx lst\ ((=))\ supp\ p\ (cs,\ y))$

fun
alpha-abs-res
where
[*simp del*]:
$alpha\text{-}abs\text{-}res\ (bs,\ x)\ (cs,\ y) \longleftrightarrow (\exists\, p.\ (bs,\ x) \approx res\ ((=))\ supp\ p\ (cs,\ y))$

notation
alpha-abs-set (**infix** $\approx abs'\text{-}set$ *50*) **and**
alpha-abs-lst (**infix** $\approx abs'\text{-}lst$ *50*) **and**
alpha-abs-res (**infix** $\approx abs'\text{-}res$ *50*)

lemmas *alphas-abs* = *alpha-abs-set.simps alpha-abs-res.simps alpha-abs-lst.simps*

lemma *alphas-abs-refl*:
shows $(bs, x) \approx$*abs-set* (bs, x)
and $(bs, x) \approx$*abs-res* (bs, x)
and $(cs, x) \approx$*abs-lst* (cs, x)
⟨*proof*⟩

lemma *alphas-abs-sym*:
shows $(bs, x) \approx$*abs-set* $(cs, y) \Longrightarrow (cs, y) \approx$*abs-set* (bs, x)
and $(bs, x) \approx$*abs-res* $(cs, y) \Longrightarrow (cs, y) \approx$*abs-res* (bs, x)
and $(ds, x) \approx$*abs-lst* $(es, y) \Longrightarrow (es, y) \approx$*abs-lst* (ds, x)
⟨*proof*⟩

lemma *alphas-abs-trans*:
shows $[\![(bs, x) \approx$*abs-set* $(cs, y); (cs, y) \approx$*abs-set* $(ds, z)]\!] \Longrightarrow (bs, x) \approx$*abs-set* (ds, z)
and $[\![(bs, x) \approx$*abs-res* $(cs, y); (cs, y) \approx$*abs-res* $(ds, z)]\!] \Longrightarrow (bs, x) \approx$*abs-res* (ds, z)
and $[\![(es, x) \approx$*abs-lst* $(gs, y); (gs, y) \approx$*abs-lst* $(hs, z)]\!] \Longrightarrow (es, x) \approx$*abs-lst* (hs, z)
⟨*proof*⟩

lemma *alphas-abs-eqvt*:
shows $(bs, x) \approx$*abs-set* $(cs, y) \Longrightarrow (p \cdot bs, p \cdot x) \approx$*abs-set* $(p \cdot cs, p \cdot y)$
and $(bs, x) \approx$*abs-res* $(cs, y) \Longrightarrow (p \cdot bs, p \cdot x) \approx$*abs-res* $(p \cdot cs, p \cdot y)$
and $(ds, x) \approx$*abs-lst* $(es, y) \Longrightarrow (p \cdot ds, p \cdot x) \approx$*abs-lst* $(p \cdot es, p \cdot y)$
⟨*proof*⟩

30 Strengthening the equivalence

lemma *disjoint-right-eq*:
assumes *a*: $A \cup B1 = A \cup B2$
and *b*: $A \cap B1 = \{\}$ $A \cap B2 = \{\}$
shows $B1 = B2$
⟨*proof*⟩

lemma *supp-property-res*:
assumes *a*: $(as, x) \approx$*res* $(=)$ *supp* p (as', x')
shows $p \cdot (supp\ x \cap as) = supp\ x' \cap as'$
⟨*proof*⟩

lemma *alpha-abs-res-stronger1-aux*:
assumes *asm*: $(as, x) \approx$*res* $(=)$ *supp* p' (as', x')
shows $\exists p.\ (as, x) \approx$*res* $(=)$ *supp* p $(as', x') \wedge supp\ p \subseteq (supp\ x \cap as) \cup (supp\ x' \cap as')$
⟨*proof*⟩

lemma *alpha-abs-res-minimal*:
assumes *asm*: $(as, x) \approx$*res* $(=)$ *supp* p (as', x')

shows $(as \cap supp\ x,\ x) \approx res\ (=)\ supp\ p\ (as' \cap supp\ x',\ x')$
⟨*proof*⟩

lemma *alpha-abs-res-abs-set*:
assumes *asm*: $(as,\ x) \approx res\ (=)\ supp\ p\ (as',\ x')$
shows $(as \cap supp\ x,\ x) \approx set\ (=)\ supp\ p\ (as' \cap supp\ x',\ x')$
⟨*proof*⟩

lemma *alpha-abs-set-abs-res*:
assumes *asm*: $(as \cap supp\ x,\ x) \approx set\ (=)\ supp\ p\ (as' \cap supp\ x',\ x')$
shows $(as,\ x) \approx res\ (=)\ supp\ p\ (as',\ x')$
⟨*proof*⟩

lemma *alpha-abs-res-stronger1*:
assumes *asm*: $(as,\ x) \approx res\ (=)\ supp\ p'\ (as',\ x')$
shows $\exists\, p.\ (as,\ x) \approx res\ (=)\ supp\ p\ (as',\ x') \wedge supp\ p \subseteq as \cup as'$
⟨*proof*⟩

lemma *alpha-abs-set-stronger1*:
assumes *asm*: $(as,\ x) \approx set\ (=)\ supp\ p'\ (as',\ x')$
shows $\exists\, p.\ (as,\ x) \approx set\ (=)\ supp\ p\ (as',\ x') \wedge supp\ p \subseteq as \cup as'$
⟨*proof*⟩

lemma *alpha-abs-lst-stronger1*:
assumes *asm*: $(as,\ x) \approx lst\ (=)\ supp\ p'\ (as',\ x')$
shows $\exists\, p.\ (as,\ x) \approx lst\ (=)\ supp\ p\ (as',\ x') \wedge supp\ p \subseteq set\ as \cup set\ as'$
⟨*proof*⟩

lemma *alphas-abs-stronger*:
shows $(as,\ x) \approx abs\text{-}set\ (as',\ x') \longleftrightarrow (\exists\, p.\ (as,\ x) \approx set\ (=)\ supp\ p\ (as',\ x') \wedge supp\ p \subseteq as \cup as')$
and $(as,\ x) \approx abs\text{-}res\ (as',\ x') \longleftrightarrow (\exists\, p.\ (as,\ x) \approx res\ (=)\ supp\ p\ (as',\ x') \wedge supp\ p \subseteq as \cup as')$
and $(bs,\ x) \approx abs\text{-}lst\ (bs',\ x') \longleftrightarrow$
$(\exists\, p.\ (bs,\ x) \approx lst\ (=)\ supp\ p\ (bs',\ x') \wedge supp\ p \subseteq set\ bs \cup set\ bs')$
⟨*proof*⟩

lemma *alpha-res-alpha-set*:
$(bs,\ x) \approx res\ (=)\ supp\ p\ (cs,\ y) \longleftrightarrow (bs \cap supp\ x,\ x) \approx set\ (=)\ supp\ p\ (cs \cap supp\ y,\ y)$
⟨*proof*⟩

31 Quotient types

quotient-type
$'a\ abs\text{-}set = (atom\ set \times 'a::pt)\ /\ alpha\text{-}abs\text{-}set$
⟨*proof*⟩

quotient-type

'b abs-res = (*atom set* × *'b*::*pt*) / *alpha-abs-res*
⟨*proof*⟩

quotient-type
'c abs-lst = (*atom list* × *'c*::*pt*) / *alpha-abs-lst*
⟨*proof*⟩

quotient-definition
Abs-set ([-]*set*. - [60, 60] 60)
where
Abs-set::*atom set* ⇒ (*'a*::*pt*) ⇒ *'a abs-set*
is
Pair::*atom set* ⇒ (*'a*::*pt*) ⇒ (*atom set* × *'a*) ⟨*proof*⟩

quotient-definition
Abs-res ([-]*res*. - [60, 60] 60)
where
Abs-res::*atom set* ⇒ (*'a*::*pt*) ⇒ *'a abs-res*
is
Pair::*atom set* ⇒ (*'a*::*pt*) ⇒ (*atom set* × *'a*) ⟨*proof*⟩

quotient-definition
Abs-lst ([-]*lst*. - [60, 60] 60)
where
Abs-lst::*atom list* ⇒ (*'a*::*pt*) ⇒ *'a abs-lst*
is
Pair::*atom list* ⇒ (*'a*::*pt*) ⇒ (*atom list* × *'a*) ⟨*proof*⟩

lemma [*quot-respect*]:
shows ((=) ===> (=) ===> *alpha-abs-set*) *Pair Pair*
and ((=) ===> (=) ===> *alpha-abs-res*) *Pair Pair*
and ((=) ===> (=) ===> *alpha-abs-lst*) *Pair Pair*
⟨*proof*⟩

lemma [*quot-respect*]:
shows ((=) ===> *alpha-abs-set* ===> *alpha-abs-set*) *permute permute*
and ((=) ===> *alpha-abs-res* ===> *alpha-abs-res*) *permute permute*
and ((=) ===> *alpha-abs-lst* ===> *alpha-abs-lst*) *permute permute*
⟨*proof*⟩

lemma *Abs-eq-iff*:
shows [*bs*]*set*. *x* = [*bs'*]*set*. *y* ⟷ (∃ *p*. (*bs*, *x*) ≈*set* (=) *supp p* (*bs'*, *y*))
and [*bs*]*res*. *x* = [*bs'*]*res*. *y* ⟷ (∃ *p*. (*bs*, *x*) ≈*res* (=) *supp p* (*bs'*, *y*))
and [*cs*]*lst*. *x* = [*cs'*]*lst*. *y* ⟷ (∃ *p*. (*cs*, *x*) ≈*lst* (=) *supp p* (*cs'*, *y*))
⟨*proof*⟩

lemma *Abs-eq-iff2*:
shows [*bs*]*set*. *x* = [*bs'*]*set*. *y* ⟷ (∃ *p*. (*bs*, *x*) ≈*set* ((=)) *supp p* (*bs'*, *y*) ∧ *supp p* ⊆ *bs* ∪ *bs'*)

and $[bs]res.\ x = [bs']res.\ y \longleftrightarrow (\exists\, p.\ (bs,\ x) \approx res\ ((=))\ supp\ p\ (bs',\ y) \wedge supp\ p \subseteq bs \cup bs')$
and $[cs]lst.\ x = [cs']lst.\ y \longleftrightarrow (\exists\, p.\ (cs,\ x) \approx lst\ ((=))\ supp\ p\ (cs',\ y) \wedge supp\ p \subseteq set\ cs \cup set\ cs')$
⟨*proof*⟩

lemma *Abs-eq-res-set*:
shows $[bs]res.\ x = [cs]res.\ y \longleftrightarrow [bs \cap supp\ x]set.\ x = [cs \cap supp\ y]set.\ y$
⟨*proof*⟩

lemma *Abs-eq-res-supp*:
assumes *asm*: $supp\ x \subseteq bs$
shows $[as]res.\ x = [as \cap bs]res.\ x$
⟨*proof*⟩

lemma *Abs-exhausts*[*cases type*]:
shows $(\bigwedge as\ (x::'a::pt).\ y1 = [as]set.\ x \Longrightarrow P1) \Longrightarrow P1$
and $(\bigwedge as\ (x::'a::pt).\ y2 = [as]res.\ x \Longrightarrow P2) \Longrightarrow P2$
and $(\bigwedge bs\ (x::'a::pt).\ y3 = [bs]lst.\ x \Longrightarrow P3) \Longrightarrow P3$
⟨*proof*⟩

instantiation *abs-set* :: (*pt*) *pt*
begin

quotient-definition
$permute\text{-}abs\text{-}set::perm \Rightarrow ('a::pt\ abs\text{-}set) \Rightarrow 'a\ abs\text{-}set$
is
$permute::\ perm \Rightarrow (atom\ set \times 'a::pt) \Rightarrow (atom\ set \times 'a::pt)$
⟨*proof*⟩

lemma *permute-Abs-set*[*simp*]:
fixes $x::'a::pt$
shows $(p \cdot ([as]set.\ x)) = [p \cdot as]set.\ (p \cdot x)$
⟨*proof*⟩

instance
⟨*proof*⟩

end

instantiation *abs-res* :: (*pt*) *pt*
begin

quotient-definition
$permute\text{-}abs\text{-}res::perm \Rightarrow ('a::pt\ abs\text{-}res) \Rightarrow 'a\ abs\text{-}res$
is
$permute::\ perm \Rightarrow (atom\ set \times 'a::pt) \Rightarrow (atom\ set \times 'a::pt)$
⟨*proof*⟩

lemma *permute-Abs-res*[*simp*]:
fixes $x::'a::pt$
shows $(p \cdot ([as]res.\ x)) = [p \cdot as]res.\ (p \cdot x)$
⟨*proof*⟩

instance
⟨*proof*⟩

end

instantiation *abs-lst* :: (*pt*) *pt*
begin

quotient-definition
permute-abs-lst::*perm* ⇒ ($'a::pt$ *abs-lst*) ⇒ $'a$ *abs-lst*
is
permute:: *perm* ⇒ (*atom list* × $'a::pt$) ⇒ (*atom list* × $'a::pt$)
⟨*proof*⟩

lemma *permute-Abs-lst*[*simp*]:
fixes $x::'a::pt$
shows $(p \cdot ([as]lst.\ x)) = [p \cdot as]lst.\ (p \cdot x)$
⟨*proof*⟩

instance
⟨*proof*⟩

end

lemmas *permute-Abs*[*eqvt*] = *permute-Abs-set permute-Abs-res permute-Abs-lst*

lemma *Abs-swap1*:
assumes *a1*: $a \notin (supp\ x) - bs$
and *a2*: $b \notin (supp\ x) - bs$
shows $[bs]set.\ x = [(a \rightleftharpoons b) \cdot bs]set.\ ((a \rightleftharpoons b) \cdot x)$
and $[bs]res.\ x = [(a \rightleftharpoons b) \cdot bs]res.\ ((a \rightleftharpoons b) \cdot x)$
⟨*proof*⟩

lemma *Abs-swap2*:
assumes *a1*: $a \notin (supp\ x) - (set\ bs)$
and *a2*: $b \notin (supp\ x) - (set\ bs)$
shows $[bs]lst.\ x = [(a \rightleftharpoons b) \cdot bs]lst.\ ((a \rightleftharpoons b) \cdot x)$
⟨*proof*⟩

lemma *Abs-supports*:
shows $((supp\ x) - as)\ supports\ ([as]set.\ x)$
and $((supp\ x) - as)\ supports\ ([as]res.\ x)$

and $((supp\ x) - set\ bs)\ supports\ ([bs]lst.\ x)$
⟨*proof*⟩

function
$supp\text{-}set :: ('a::pt)\ abs\text{-}set \Rightarrow atom\ set$ **and**
$supp\text{-}res :: ('a::pt)\ abs\text{-}res \Rightarrow atom\ set$ **and**
$supp\text{-}lst :: ('a::pt)\ abs\text{-}lst \Rightarrow atom\ set$
where
$supp\text{-}set\ ([as]set.\ x) = supp\ x - as$
| $supp\text{-}res\ ([as]res.\ x) = supp\ x - as$
| $supp\text{-}lst\ (Abs\text{-}lst\ cs\ x) = (supp\ x) - (set\ cs)$
⟨*proof*⟩

termination
⟨*proof*⟩

lemma *supp-funs-eqvt*[*eqvt*]:
shows $(p \cdot supp\text{-}set\ x) = supp\text{-}set\ (p \cdot x)$
and $(p \cdot supp\text{-}res\ y) = supp\text{-}res\ (p \cdot y)$
and $(p \cdot supp\text{-}lst\ z) = supp\text{-}lst\ (p \cdot z)$
⟨*proof*⟩

lemma *Abs-fresh-aux*:
shows $a \sharp [bs]set.\ x \Longrightarrow a \sharp supp\text{-}set\ ([bs]set.\ x)$
and $a \sharp [bs]res.\ x \Longrightarrow a \sharp supp\text{-}res\ ([bs]res.\ x)$
and $a \sharp [cs]lst.\ x \Longrightarrow a \sharp supp\text{-}lst\ ([cs]lst.\ x)$
⟨*proof*⟩

lemma *Abs-supp-subset1*:
assumes *a*: $finite\ (supp\ x)$
shows $(supp\ x) - as \subseteq supp\ ([as]set.\ x)$
and $(supp\ x) - as \subseteq supp\ ([as]res.\ x)$
and $(supp\ x) - (set\ bs) \subseteq supp\ ([bs]lst.\ x)$
⟨*proof*⟩

lemma *Abs-supp-subset2*:
assumes *a*: $finite\ (supp\ x)$
shows $supp\ ([as]set.\ x) \subseteq (supp\ x) - as$
and $supp\ ([as]res.\ x) \subseteq (supp\ x) - as$
and $supp\ ([bs]lst.\ x) \subseteq (supp\ x) - (set\ bs)$
⟨*proof*⟩

lemma *Abs-finite-supp*:
assumes *a*: $finite\ (supp\ x)$
shows $supp\ ([as]set.\ x) = (supp\ x) - as$
and $supp\ ([as]res.\ x) = (supp\ x) - as$
and $supp\ ([bs]lst.\ x) = (supp\ x) - (set\ bs)$
⟨*proof*⟩

lemma *supp-Abs*:
fixes $x::'a::fs$
shows $supp\ ([as]set.\ x) = (supp\ x) - as$
and $supp\ ([as]res.\ x) = (supp\ x) - as$
and $supp\ ([bs]lst.\ x) = (supp\ x) - (set\ bs)$
⟨*proof*⟩

instance *abs-set* :: (*fs*) *fs*
⟨*proof*⟩

instance *abs-res* :: (*fs*) *fs*
⟨*proof*⟩

instance *abs-lst* :: (*fs*) *fs*
⟨*proof*⟩

lemma *Abs-fresh-iff*:
fixes $x::'a::fs$
shows $a\ \sharp\ [bs]set.\ x \longleftrightarrow a \in bs \vee (a \notin bs \wedge a\ \sharp\ x)$
and $a\ \sharp\ [bs]res.\ x \longleftrightarrow a \in bs \vee (a \notin bs \wedge a\ \sharp\ x)$
and $a\ \sharp\ [cs]lst.\ x \longleftrightarrow a \in (set\ cs) \vee (a \notin (set\ cs) \wedge a\ \sharp\ x)$
⟨*proof*⟩

lemma *Abs-fresh-star-iff*:
fixes $x::'a::fs$
shows $as\ \sharp*\ ([bs]set.\ x) \longleftrightarrow (as - bs)\ \sharp*\ x$
and $as\ \sharp*\ ([bs]res.\ x) \longleftrightarrow (as - bs)\ \sharp*\ x$
and $as\ \sharp*\ ([cs]lst.\ x) \longleftrightarrow (as - set\ cs)\ \sharp*\ x$
⟨*proof*⟩

lemma *Abs-fresh-star*:
fixes $x::'a::fs$
shows $as \subseteq as' \Longrightarrow as\ \sharp*\ ([as']set.\ x)$
and $as \subseteq as' \Longrightarrow as\ \sharp*\ ([as']res.\ x)$
and $bs \subseteq set\ bs' \Longrightarrow bs\ \sharp*\ ([bs']lst.\ x)$
⟨*proof*⟩

lemma *Abs-fresh-star2*:
fixes $x::'a::fs$
shows $as \cap bs = \{\} \Longrightarrow as\ \sharp*\ ([bs]set.\ x) \longleftrightarrow as\ \sharp*\ x$
and $as \cap bs = \{\} \Longrightarrow as\ \sharp*\ ([bs]res.\ x) \longleftrightarrow as\ \sharp*\ x$
and $cs \cap set\ ds = \{\} \Longrightarrow cs\ \sharp*\ ([ds]lst.\ x) \longleftrightarrow cs\ \sharp*\ x$
⟨*proof*⟩

32 Abstractions of single atoms

lemma *Abs1-eq*:
fixes $x\ y::'a::fs$
shows $[\{atom\ a\}]set.\ x = [\{atom\ a\}]set.\ y \longleftrightarrow x = y$

and $[\{atom\ a\}]res.\ x = [\{atom\ a\}]res.\ y \longleftrightarrow x = y$
and $[[atom\ a]]lst.\ x = [[atom\ a]]lst.\ y \longleftrightarrow x = y$
⟨*proof*⟩

lemma *Abs1-eq-iff-fresh*:
fixes $x\ y::'a::fs$
and $a\ b\ c::'b::at$
assumes $atom\ c \sharp (a, b, x, y)$
shows $[\{atom\ a\}]set.\ x = [\{atom\ b\}]set.\ y \longleftrightarrow (a \leftrightarrow c) \cdot x = (b \leftrightarrow c) \cdot y$
and $[\{atom\ a\}]res.\ x = [\{atom\ b\}]res.\ y \longleftrightarrow (a \leftrightarrow c) \cdot x = (b \leftrightarrow c) \cdot y$
and $[[atom\ a]]lst.\ x = [[atom\ b]]lst.\ y \longleftrightarrow (a \leftrightarrow c) \cdot x = (b \leftrightarrow c) \cdot y$
⟨*proof*⟩

lemma *Abs1-eq-iff-all*:
fixes $x\ y::'a::fs$
and $z::'c::fs$
and $a\ b::'b::at$
shows $[\{atom\ a\}]set.\ x = [\{atom\ b\}]set.\ y \longleftrightarrow (\forall c.\ atom\ c \sharp z \longrightarrow atom\ c \sharp (a, b, x, y) \longrightarrow (a \leftrightarrow c) \cdot x = (b \leftrightarrow c) \cdot y)$
and $[\{atom\ a\}]res.\ x = [\{atom\ b\}]res.\ y \longleftrightarrow (\forall c.\ atom\ c \sharp z \longrightarrow atom\ c \sharp (a, b, x, y) \longrightarrow (a \leftrightarrow c) \cdot x = (b \leftrightarrow c) \cdot y)$
and $[[atom\ a]]lst.\ x = [[atom\ b]]lst.\ y \longleftrightarrow (\forall c.\ atom\ c \sharp z \longrightarrow atom\ c \sharp (a, b, x, y) \longrightarrow (a \leftrightarrow c) \cdot x = (b \leftrightarrow c) \cdot y)$
⟨*proof*⟩

lemma *Abs1-eq-iff*:
fixes $x\ y::'a::fs$
and $a\ b::'b::at$
shows $[\{atom\ a\}]set.\ x = [\{atom\ b\}]set.\ y \longleftrightarrow (a = b \wedge x = y) \vee (a \neq b \wedge x = (a \leftrightarrow b) \cdot y \wedge atom\ a \sharp y)$
and $[\{atom\ a\}]res.\ x = [\{atom\ b\}]res.\ y \longleftrightarrow (a = b \wedge x = y) \vee (a \neq b \wedge x = (a \leftrightarrow b) \cdot y \wedge atom\ a \sharp y)$
and $[[atom\ a]]lst.\ x = [[atom\ b]]lst.\ y \longleftrightarrow (a = b \wedge x = y) \vee (a \neq b \wedge x = (a \leftrightarrow b) \cdot y \wedge atom\ a \sharp y)$
⟨*proof*⟩

lemma *Abs1-eq-iff'*:
fixes $x::'a::fs$
and $a\ b::'b::at$
shows $[\{atom\ a\}]set.\ x = [\{atom\ b\}]set.\ y \longleftrightarrow (a = b \wedge x = y) \vee (a \neq b \wedge (b \leftrightarrow a) \cdot x = y \wedge atom\ b \sharp x)$
and $[\{atom\ a\}]res.\ x = [\{atom\ b\}]res.\ y \longleftrightarrow (a = b \wedge x = y) \vee (a \neq b \wedge (b \leftrightarrow a) \cdot x = y \wedge atom\ b \sharp x)$
and $[[atom\ a]]lst.\ x = [[atom\ b]]lst.\ y \longleftrightarrow (a = b \wedge x = y) \vee (a \neq b \wedge (b \leftrightarrow a) \cdot x = y \wedge atom\ b \sharp x)$
⟨*proof*⟩

⟨*ML*⟩

32.1 Renaming of bodies of abstractions

lemma *Abs-rename-set*:
fixes $x::'a::fs$
assumes a: $(p \cdot bs) \sharp* x$

shows $\exists q.\ [bs]set.\ x = [p \cdot bs]set.\ (q \cdot x) \wedge q \cdot bs = p \cdot bs$
⟨*proof*⟩

lemma *Abs-rename-res*:
fixes $x::'a::fs$
assumes a: $(p \cdot bs) \sharp* x$

shows $\exists q.\ [bs]res.\ x = [p \cdot bs]res.\ (q \cdot x) \wedge q \cdot bs = p \cdot bs$
⟨*proof*⟩

lemma *Abs-rename-lst*:
fixes $x::'a::fs$
assumes a: $(p \cdot (set\ bs)) \sharp* x$
shows $\exists q.\ [bs]lst.\ x = [p \cdot bs]lst.\ (q \cdot x) \wedge q \cdot bs = p \cdot bs$
⟨*proof*⟩

for deep recursive binders

lemma *Abs-rename-set′*:
fixes $x::'a::fs$
assumes a: $(p \cdot bs) \sharp* x$

shows $\exists q.\ [bs]set.\ x = [q \cdot bs]set.\ (q \cdot x) \wedge q \cdot bs = p \cdot bs$
⟨*proof*⟩

lemma *Abs-rename-res′*:
fixes $x::'a::fs$
assumes a: $(p \cdot bs) \sharp* x$

shows $\exists q.\ [bs]res.\ x = [q \cdot bs]res.\ (q \cdot x) \wedge q \cdot bs = p \cdot bs$
⟨*proof*⟩

lemma *Abs-rename-lst′*:
fixes $x::'a::fs$
assumes a: $(p \cdot (set\ bs)) \sharp* x$
shows $\exists q.\ [bs]lst.\ x = [q \cdot bs]lst.\ (q \cdot x) \wedge q \cdot bs = p \cdot bs$
⟨*proof*⟩

33 Infrastructure for building tuples of relations and functions

fun
prod-fv :: $('a \Rightarrow atom\ set) \Rightarrow ('b \Rightarrow atom\ set) \Rightarrow ('a \times 'b) \Rightarrow atom\ set$
where

prod-fv fv1 fv2 (x, y) = fv1 x ∪ fv2 y

definition
prod-alpha :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ (*'b* ⇒ *'b* ⇒ *bool*) ⇒ (*'a* × *'b* ⇒ *'a* × *'b* ⇒ *bool*)
where
prod-alpha = rel-prod

lemma [*quot-respect*]:
shows ((*R1* ===> (=)) ===> (*R2* ===> (=)) ===> *rel-prod R1 R2* ===> (=)) *prod-fv prod-fv*
⟨*proof*⟩

lemma [*quot-preserve*]:
assumes *q1*: *Quotient3 R1 abs1 rep1*
and *q2*: *Quotient3 R2 abs2 rep2*
shows ((*abs1* ---> *id*) ---> (*abs2* ---> *id*) ---> *map-prod rep1 rep2* ---> *id*) *prod-fv = prod-fv*
⟨*proof*⟩

lemma [*mono*]:
shows *A <= B* ⟹ *C <= D* ==> *prod-alpha A C <= prod-alpha B D*
⟨*proof*⟩

lemma [*eqvt*]:
shows *p · prod-alpha A B x y = prod-alpha (p · A) (p · B) (p · x) (p · y)*
⟨*proof*⟩

lemma [*eqvt*]:
shows *p · prod-fv A B (x, y) = prod-fv (p · A) (p · B) (p · x, p · y)*
⟨*proof*⟩

lemma *prod-fv-supp*:
shows *prod-fv supp supp = supp*
⟨*proof*⟩

lemma *prod-alpha-eq*:
shows *prod-alpha* ((=)) ((=)) = ((=))
⟨*proof*⟩

end
theory *Nominal2-FCB*
imports *Nominal2-Abs*
begin

A tactic which solves all trivial cases in function definitions, and leaves the others unchanged.

⟨*ML*⟩

lemma *Abs-lst1-fcb*:
fixes $x\ y :: 'a :: at$
and $S\ T :: 'b :: fs$
assumes *e*: $[[atom\ x]]lst.\ T = [[atom\ y]]lst.\ S$
and *f1*: $[\![x \neq y;\ atom\ y \sharp T;\ atom\ x \sharp (y \leftrightarrow x) \cdot T]\!] \Longrightarrow atom\ x \sharp f\ x\ T$
and *f2*: $[\![x \neq y;\ atom\ y \sharp T;\ atom\ x \sharp (y \leftrightarrow x) \cdot T]\!] \Longrightarrow atom\ y \sharp f\ x\ T$
and *p*: $[\![S = (x \leftrightarrow y) \cdot T;\ x \neq y;\ atom\ y \sharp T;\ atom\ x \sharp S]\!]$
$\Longrightarrow (x \leftrightarrow y) \cdot (f\ x\ T) = f\ y\ S$
shows $f\ x\ T = f\ y\ S$
⟨*proof*⟩

lemma *Abs-lst-fcb*:
fixes $xs\ ys :: 'a :: fs$
and $S\ T :: 'b :: fs$
assumes *e*: $(Abs\text{-}lst\ (ba\ xs)\ T) = (Abs\text{-}lst\ (ba\ ys)\ S)$
and *f1*: $\bigwedge x.\ x \in set\ (ba\ xs) \Longrightarrow x \sharp f\ xs\ T$
and *f2*: $\bigwedge x.\ [\![supp\ T - set\ (ba\ xs) = supp\ S - set\ (ba\ ys);\ x \in set\ (ba\ ys)]\!]$
$\Longrightarrow x \sharp f\ xs\ T$
and *eqv*: $\bigwedge p.\ [\![p \cdot T = S;\ p \cdot ba\ xs = ba\ ys;\ supp\ p \subseteq set\ (ba\ xs) \cup set\ (ba\ ys)]\!]$
$\Longrightarrow p \cdot (f\ xs\ T) = f\ ys\ S$
shows $f\ xs\ T = f\ ys\ S$
⟨*proof*⟩

lemma *Abs-set-fcb*:
fixes $xs\ ys :: 'a :: fs$
and $S\ T :: 'b :: fs$
assumes *e*: $(Abs\text{-}set\ (ba\ xs)\ T) = (Abs\text{-}set\ (ba\ ys)\ S)$
and *f1*: $\bigwedge x.\ x \in ba\ xs \Longrightarrow x \sharp f\ xs\ T$
and *f2*: $\bigwedge x.\ [\![supp\ T - ba\ xs = supp\ S - ba\ ys;\ x \in ba\ ys]\!] \Longrightarrow x \sharp f\ xs\ T$
and *eqv*: $\bigwedge p.\ [\![p \cdot T = S;\ p \cdot ba\ xs = ba\ ys;\ supp\ p \subseteq ba\ xs \cup ba\ ys]\!] \Longrightarrow p \cdot (f\ xs\ T) = f\ ys\ S$
shows $f\ xs\ T = f\ ys\ S$
⟨*proof*⟩

lemma *Abs-res-fcb*:
fixes $xs\ ys :: ('a :: at\text{-}base)\ set$
and $S\ T :: 'b :: fs$
assumes *e*: $(Abs\text{-}res\ (atom\ `\ xs)\ T) = (Abs\text{-}res\ (atom\ `\ ys)\ S)$
and *f1*: $\bigwedge x.\ x \in atom\ `\ xs \Longrightarrow x \in supp\ T \Longrightarrow x \sharp f\ xs\ T$
and *f2*: $\bigwedge x.\ [\![supp\ T - atom\ `\ xs = supp\ S - atom\ `\ ys;\ x \in atom\ `\ ys;\ x \in supp\ S]\!] \Longrightarrow x \sharp f\ xs\ T$
and *eqv*: $\bigwedge p.\ [\![p \cdot T = S;\ supp\ p \subseteq atom\ `\ xs \cap supp\ T \cup atom\ `\ ys \cap supp\ S;\ p \cdot (atom\ `\ xs \cap supp\ T) = atom\ `\ ys \cap supp\ S]\!] \Longrightarrow p \cdot (f\ xs\ T) = f\ ys\ S$
shows $f\ xs\ T = f\ ys\ S$
⟨*proof*⟩

lemma *Abs-set-fcb2*:
fixes *as bs* :: *atom set*
and *x y* :: *'b* :: *fs*
and *c*::*'c*::*fs*
assumes *eq*: [*as*]*set*. *x* = [*bs*]*set*. *y*
and *fin*: *finite as finite bs*
and *fcb1*: *as* ♯* *f as x c*
and *fresh1*: *as* ♯* *c*
and *fresh2*: *bs* ♯* *c*
and *perm1*: ⋀*p*. *supp p* ♯* *c* ⟹ *p* ∙ (*f as x c*) = *f* (*p* ∙ *as*) (*p* ∙ *x*) *c*
and *perm2*: ⋀*p*. *supp p* ♯* *c* ⟹ *p* ∙ (*f bs y c*) = *f* (*p* ∙ *bs*) (*p* ∙ *y*) *c*
shows *f as x c* = *f bs y c*
⟨*proof*⟩

lemma *Abs-res-fcb2*:
fixes *as bs* :: *atom set*
and *x y* :: *'b* :: *fs*
and *c*::*'c*::*fs*
assumes *eq*: [*as*]*res*. *x* = [*bs*]*res*. *y*
and *fin*: *finite as finite bs*
and *fcb1*: (*as* ∩ *supp x*) ♯* *f* (*as* ∩ *supp x*) *x c*
and *fresh1*: *as* ♯* *c*
and *fresh2*: *bs* ♯* *c*
and *perm1*: ⋀*p*. *supp p* ♯* *c* ⟹ *p* ∙ (*f* (*as* ∩ *supp x*) *x c*) = *f* (*p* ∙ (*as* ∩ *supp x*)) (*p* ∙ *x*) *c*
and *perm2*: ⋀*p*. *supp p* ♯* *c* ⟹ *p* ∙ (*f* (*bs* ∩ *supp y*) *y c*) = *f* (*p* ∙ (*bs* ∩ *supp y*)) (*p* ∙ *y*) *c*
shows *f* (*as* ∩ *supp x*) *x c* = *f* (*bs* ∩ *supp y*) *y c*
⟨*proof*⟩

lemma *Abs-lst-fcb2*:
fixes *as bs* :: *atom list*
and *x y* :: *'b* :: *fs*
and *c*::*'c*::*fs*
assumes *eq*: [*as*]*lst*. *x* = [*bs*]*lst*. *y*
and *fcb1*: (*set as*) ♯* *f as x c*
and *fresh1*: *set as* ♯* *c*
and *fresh2*: *set bs* ♯* *c*
and *perm1*: ⋀*p*. *supp p* ♯* *c* ⟹ *p* ∙ (*f as x c*) = *f* (*p* ∙ *as*) (*p* ∙ *x*) *c*
and *perm2*: ⋀*p*. *supp p* ♯* *c* ⟹ *p* ∙ (*f bs y c*) = *f* (*p* ∙ *bs*) (*p* ∙ *y*) *c*
shows *f as x c* = *f bs y c*
⟨*proof*⟩

lemma *Abs-lst1-fcb2*:
fixes *a b* :: *atom*
and *x y* :: *'b* :: *fs*
and *c*::*'c* :: *fs*
assumes *e*: [[*a*]]*lst*. *x* = [[*b*]]*lst*. *y*

and *fcb1*: $a \sharp f\ a\ x\ c$
and *fresh*: $\{a, b\} \sharp* c$
and *perm1*: $\bigwedge p.\ supp\ p \sharp* c \Longrightarrow p \cdot (f\ a\ x\ c) = f\ (p \cdot a)\ (p \cdot x)\ c$
and *perm2*: $\bigwedge p.\ supp\ p \sharp* c \Longrightarrow p \cdot (f\ b\ y\ c) = f\ (p \cdot b)\ (p \cdot y)\ c$
shows $f\ a\ x\ c = f\ b\ y\ c$
⟨*proof*⟩

lemma *Abs-lst1-fcb2'*:
fixes $a\ b :: 'a::at\text{-}base$
and $x\ y :: 'b :: fs$
and $c::'c :: fs$
assumes *e*: $[[atom\ a]]lst.\ x = [[atom\ b]]lst.\ y$
and *fcb1*: $atom\ a \sharp f\ a\ x\ c$
and *fresh*: $\{atom\ a, atom\ b\} \sharp* c$
and *perm1*: $\bigwedge p.\ supp\ p \sharp* c \Longrightarrow p \cdot (f\ a\ x\ c) = f\ (p \cdot a)\ (p \cdot x)\ c$
and *perm2*: $\bigwedge p.\ supp\ p \sharp* c \Longrightarrow p \cdot (f\ b\ y\ c) = f\ (p \cdot b)\ (p \cdot y)\ c$
shows $f\ a\ x\ c = f\ b\ y\ c$
⟨*proof*⟩

end
theory *Nominal2*
imports
Nominal2-Base Nominal2-Abs Nominal2-FCB
keywords
nominal-datatype :: *thy-defn* **and**
nominal-function nominal-inductive nominal-termination :: *thy-goal-defn* **and**
avoids binds
begin

⟨*ML*⟩

34 Interface for *nominal-datatype*

⟨*ML*⟩
Infrastructure for adding *-raw* to types and terms

⟨*ML*⟩

35 Preparing and parsing of the specification

⟨*ML*⟩
associates every SOME with the index in the list; drops NONEs

⟨*ML*⟩
adds an empty binding clause for every argument that is not already part of a binding clause

⟨*ML*⟩

end

theory *Atoms*
imports *Nominal2-Base*
begin

36 *nat-of* is an example of a function without finite support

lemma *not-fresh-nat-of*:
shows $\neg$ *a* $\sharp$ *nat-of*
⟨*proof*⟩

lemma *supp-nat-of*:
shows *supp nat-of* = *UNIV*
⟨*proof*⟩

37 Manual instantiation of class *at*.

typedef *name* = {*a*. *sort-of a* = *Sort* ''*name*'' []}
⟨*proof*⟩

instantiation *name* :: *at*
begin

definition
p · *a* = *Abs-name* (*p* · *Rep-name a*)

definition
atom a = *Rep-name a*

instance
⟨*proof*⟩

end

lemma *sort-of-atom-name*:
shows *sort-of* (*atom* (*a*::*name*)) = *Sort* ''*name*'' []
⟨*proof*⟩

Custom syntax for concrete atoms of type at

term *a*:::*name*

38 Automatic instantiation of class *at*.

atom-decl *name2*

lemma
sort-of (*atom* (*a*::*name2*)) ≠ *sort-of* (*atom* (*b*::*name*))
⟨*proof*⟩

example swappings

lemma
fixes *a b*::*atom*
assumes *sort-of a* = *sort-of b*
shows (*a* ⇌ *b*) ∙ (*a*, *b*) = (*b*, *a*)
⟨*proof*⟩

lemma
fixes *a b*::*name2*
shows (*a* ↔ *b*) ∙ (*a*, *b*) = (*b*, *a*)
⟨*proof*⟩

39 An example for multiple-sort atoms

datatype *ty* =
TVar string
| *Fun ty ty* (- → -)

primrec
sort-of-ty::*ty* ⇒ *atom-sort*
where
sort-of-ty (*TVar s*) = *Sort* ''*TVar*'' [*Sort s* []]
| *sort-of-ty* (*Fun ty1 ty2*) = *Sort* ''*Fun*'' [*sort-of-ty ty1*, *sort-of-ty ty2*]

lemma *sort-of-ty-eq-iff*:
shows *sort-of-ty x* = *sort-of-ty y* ⟷ *x* = *y*
⟨*proof*⟩

declare *sort-of-ty.simps* [*simp del*]

typedef *var* = {*a*. *sort-of a* ∈ *range sort-of-ty*}
⟨*proof*⟩

instantiation *var* :: *at-base*
begin

definition
p ∙ *a* = *Abs-var* (*p* ∙ *Rep-var a*)

definition
atom a = *Rep-var a*

instance
⟨*proof*⟩

end

Constructor for variables.

definition
Var :: *nat* ⇒ *ty* ⇒ *var*
where
Var x t = *Abs-var* (*Atom* (*sort-of-ty t*) *x*)

lemma *Var-eq-iff* [*simp*]:
shows *Var x s* = *Var y t* ⟷ *x* = *y* ∧ *s* = *t*
⟨*proof*⟩

lemma *sort-of-atom-var* [*simp*]:
sort-of (*atom* (*Var n ty*)) = *sort-of-ty ty*
⟨*proof*⟩

lemma
assumes *α* ≠ *β*
shows (*Var x α* ↔ *Var y α*) · (*Var x α*, *Var x β*) = (*Var y α*, *Var x β*)
⟨*proof*⟩

Projecting out the type component of a variable.

definition
ty-of :: *var* ⇒ *ty*
where
ty-of x = *inv sort-of-ty* (*sort-of* (*atom x*))

Functions *Var*/*ty-of* satisfy many of the same properties as *Atom*/*sort-of*.

lemma *ty-of-Var* [*simp*]:
shows *ty-of* (*Var x t*) = *t*
⟨*proof*⟩

lemma *ty-of-permute* [*simp*]:
shows *ty-of* (*p* · *x*) = *ty-of x*
⟨*proof*⟩

40 Tests with subtyping and automatic coercions

declare [[*coercion-enabled*]]

atom-decl *var1*
atom-decl *var2*

declare [[*coercion atom*::*var1*⇒*atom*]]

declare [[*coercion atom*::*var2*⇒*atom*]]

lemma

fixes *a*::*var1* **and** *b*::*var2*
shows $a \sharp t \wedge b \sharp t$
⟨*proof*⟩

lemma
fixes *as*::*var1 set*
shows *atom* ‘ *as* $\sharp*$ *t*

⟨*proof*⟩

end

theory *Eqvt*
imports *Nominal2-Base*
begin

declare [[*trace-eqvt* = *false*]]

lemma
fixes *B*::′*a*::*pt*
shows $p \cdot (B = C)$
⟨*proof*⟩

lemma
fixes *B*::*bool*
shows $p \cdot (B = C)$
⟨*proof*⟩

lemma
fixes *B*::*bool*
shows $p \cdot (A \longrightarrow B = C)$
⟨*proof*⟩

lemma
shows $p \cdot (\lambda(x::'a::pt).\ A \longrightarrow (B::'a \Rightarrow bool)\ x = C) = foo$
⟨*proof*⟩

lemma
shows $p \cdot (\lambda B::bool.\ A \longrightarrow (B = C)) = foo$
⟨*proof*⟩

lemma
shows $p \cdot (\lambda x\ y.\ \exists z.\ x = z \wedge x = y \longrightarrow z \neq x) = foo$
⟨*proof*⟩

lemma
shows $p \cdot (\lambda f\ x.\ f\ (g\ (f\ x))) = foo$
⟨*proof*⟩

lemma
fixes $p\ q$::*perm*
and x::$'a$::*pt*
shows $p \cdot (q \cdot x) = foo$
⟨*proof*⟩

lemma
fixes $p\ q\ r$::*perm*
and x::$'a$::*pt*
shows $p \cdot (q \cdot r \cdot x) = foo$
⟨*proof*⟩

lemma
fixes $p\ r$::*perm*
shows $p \cdot (\lambda q$::*perm*. $q \cdot (r \cdot x)) = foo$
⟨*proof*⟩

lemma
fixes $C\ D$::*bool*
shows $B\ (p \cdot (C = D))$
⟨*proof*⟩

declare [[*trace-eqvt* = *false*]]

there is no raw eqvt-rule for The

lemma $p \cdot (THE\ x.\ P\ x) = foo$
⟨*proof*⟩

lemma
fixes $P :: (('b \Rightarrow bool) \Rightarrow ('b$::*pt*$)) \Rightarrow ('a$::*pt*$)$
shows $p \cdot (P\ The) = foo$
⟨*proof*⟩

lemma
fixes $P :: ('a$::*pt*$) \Rightarrow ('b$::*pt*$) \Rightarrow bool$
shows $p \cdot (\lambda(a,\ b).\ P\ a\ b) = (\lambda(a,\ b).\ (p \cdot P)\ a\ b)$
⟨*proof*⟩

thm *eqvts*
thm *eqvts-raw*

⟨*ML*⟩

end