Types Proved Being BNFs during the Formalization of the Probabilistic Hierarchy

Johannes Hölzl Andreas Lochbihler Dmitriy Traytel

September 13, 2023

Contents

1 Nonempty Sets Strictly Bounded by an Infinite Cardinal

theory *Nonempty-Bounded-Set*
imports
HOL−Cardinals.Bounded-Set
begin

typedef ('*a*, '*k*) *nebset* (- *set*![-] [*22*, *21*] *21*) =
{*A* :: '*a set*. *A* ≠ {} ∧ |*A*| <*o natLeq* +*c* |*UNIV* :: '*k set*|}
morphisms *set-nebset Abs-nebset*
apply (*rule exI*[*of* - {*undefined*}], *simp*)
apply (*rule Cfinite-ordLess-Cinfinite*)
apply (*auto simp*: *cfinite-def cinfinite-csum natLeq-cinfinite Card-order-csum*)
done

setup-lifting *type-definition-nebset*

lift-definition *map-nebset* ::
('*a* ⇒ '*b*) ⇒ '*a set*!['*k*] ⇒ '*b set*!['*k*] **is** *image*
using *card-of-image ordLeq-ordLess-trans* **by** *blast*

lift-definition *rel-nebset* ::
('*a* ⇒ '*b* ⇒ *bool*) ⇒ '*a set*!['*k*] ⇒ '*b set*!['*k*] ⇒ *bool* **is** *rel-set* **.**

lift-definition *nebinsert* :: '*a* ⇒ '*a set*!['*k*] ⇒ '*a set*!['*k*] **is** *insert*
using *infinite-card-of-insert ordIso-ordLess-trans Field-card-of Field-natLeq UNIV-Plus-UNIV csum-def finite-Plus-UNIV-iff finite-insert finite-ordLess-infinite2 infinite-UNIV-nat insert-not-empty* **by** *metis*

```
lift-definition nebsingleton :: 'a ⇒ 'a set![′k] is λa. {a}
  apply simp
  apply (rule Cfinite-ordLess-Cinfinite)
  apply (auto simp: cfinite-def cinfinite-csum natLeq-cinfinite Card-order-csum)
  done

lemma set-nebset-to-set-nebset: A ≠ {} ⟹ |A| <o natLeq +c |UNIV :: 'k set|
⟹
  set-nebset (the-inv set-nebset A :: 'a set!['k]) = A
  apply (rule f-the-inv-into-f[unfolded inj-on-def])
  apply (simp add: set-nebset-inject range-eqI Abs-nebset-inverse[symmetric])
  apply (rule range-eqI Abs-nebset-inverse[symmetric] CollectI)+
  apply blast
  done

lemma rel-nebset-aux-infinite:
  fixes a :: 'a set!['k] and b :: 'b set!['k]
  shows (∀ t ∈ set-nebset a. ∃ u ∈ set-nebset b. R t u) ∧ (∀ u ∈ set-nebset b. ∃ t ∈
set-nebset a. R t u) ⟷
   ((BNF-Def.Grp {a. set-nebset a ⊆ {(a, b). R a b}} (map-nebset fst))⁻¹⁻¹ OO
    BNF-Def.Grp {a. set-nebset a ⊆ {(a, b). R a b}} (map-nebset snd)) a b (is ?L
⟷ ?R)
proof
  assume ?L
  define R' :: ('a × 'b) set!['k]
      where R' = the-inv set-nebset (Collect (case-prod R) ∩ (set-nebset a ×
set-nebset b))
    (is - = the-inv set-nebset ?L')
  from ‹?L› have ?L' ≠ {} by transfer auto
  moreover
  have |?L'| <o natLeq +c |UNIV :: 'k set|
    unfolding csum-def Field-natLeq
    by (intro ordLeq-ordLess-trans[OF card-of-mono1[OF Int-lower2]]
      card-of-Times-ordLess-infinite)
      (simp, (transfer, simp add: csum-def Field-natLeq)+)
  ultimately have *: set-nebset R' = ?L' unfolding R'-def by (intro set-nebset-to-set-nebset)
  show ?R unfolding Grp-def relcompp.simps conversep.simps
  proof (intro CollectI case-prodI exI[of - a] exI[of - b] exI[of - R'] conjI refl)
    from * show a = map-nebset fst R' using conjunct1[OF ‹?L›]
      by (transfer, auto simp add: image-def Int-def split: prod.splits)
    from * show b = map-nebset snd R' using conjunct2[OF ‹?L›]
      by (transfer, auto simp add: image-def Int-def split: prod.splits)
  qed (auto simp add: *)
next
  assume ?R thus ?L unfolding Grp-def relcompp.simps conversep.simps
    by transfer force
qed

bnf 'a set!['k]
```

map: *map-nebset*
sets: *set-nebset*
bd: *natLeq* +*c* *card-suc* |*UNIV* :: '*k set*|
rel: *rel-nebset*
proof −
show *map-nebset id* = *id* **by** (*rule ext*, *transfer*) *simp*
next
fix *f g*
show *map-nebset* (*f o g*) = *map-nebset f o map-nebset g* **by** (*rule ext*, *transfer*) *auto*
next
fix *X f g*
assume $\bigwedge z.\ z \in$ *set-nebset X* $\Longrightarrow f\ z = g\ z$
then show *map-nebset f X* = *map-nebset g X* **by** *transfer force*
next
fix *f*
show *set-nebset* ∘ *map-nebset f* = (\`) *f* ∘ *set-nebset* **by** (*rule ext*, *transfer*) *auto*
next
fix *X* :: '*a set*!['*k*]
show |*set-nebset X*| <*o natLeq* +*c card-suc* |*UNIV* :: '*k set*|
by *transfer*
(*elim conjE ordLess-ordLeq-trans csum-mono1*;
simp add: *card-suc-greater ordLess-imp-ordLeq Card-order-card-suc csum-mono2*)
next
fix *R S*
show *rel-nebset R OO rel-nebset S* ≤ *rel-nebset* (*R OO S*)
by (*rule predicate2I*, *transfer*) (*auto simp*: *rel-set-OO*[*symmetric*])
next
fix *R* :: '*a* ⇒ '*b* ⇒ *bool*
show *rel-nebset R* = ((λ*x y*. ∃ *z*. *set-nebset z* ⊆ {(*x*, *y*). *R x y*} ∧
map-nebset fst z = *x* ∧ *map-nebset snd z* = *y*) :: '*a set*!['*k*] ⇒ '*b set*!['*k*] ⇒ *bool*)
by (*simp add*: *rel-nebset-def map-fun-def o-def rel-set-def*
rel-nebset-aux-infinite[*unfolded OO-Grp-alt*])
qed (*simp-all add*: *card-order-bd-fun Cinfinite-bd-fun regularCard-bd-fun*)

lemma *map-nebset-nebinsert*[*simp*]: *map-nebset f* (*nebinsert x X*) = *nebinsert* (*f x*) (*map-nebset f X*)
by *transfer auto*

lemma *map-nebset-nebsingleton*: *map-nebset f* (*nebsingleton x*) = *nebsingleton* (*f x*)
by *transfer auto*

lemma *nebsingleton-inj*[*simp*]: *nebsingleton x* = *nebsingleton y* ⟷ *x* = *y*
by *transfer auto*

lemma *rel-nebsingleton*[*simp*]:
rel-nebset R (*nebsingleton x1*) (*nebsingleton x2*) = *R x1 x2*
by *transfer* (*auto simp*: *rel-set-def*)

lemma *rel-nebset-nebsingleton*[*simp*]:
rel-nebset R (*nebsingleton x1*) *X* = (∀ *x2*∈*set-nebset X*. *R x1 x2*)
rel-nebset R X (*nebsingleton x2*) = (∀ *x1*∈*set-nebset X*. *R x1 x2*)
by (*transfer*, *force simp add*: *rel-set-def*)+

lemma *rel-nebset-False*[*simp*]: *rel-nebset* (λ*x y*. *False*) *x y* = *False*
by *transfer* (*auto simp*: *rel-set-def*)

lemmas *set-nebset-nebsingleton*[*simp*] = *nebsingleton.rep-eq*

lemma *nebinsert-absorb*[*simp*]: *nebinsert a* (*nebinsert a x*) = *nebinsert a x*
by *transfer simp*

lift-definition *bset-of-nebset* :: ′*a set*![′*k*] ⇒ ′*a set*[′*k*] **is** λ*X*. *X* **by** (*rule conjunct2*)

lemma *rel-bset-bset-of-nebset*[*simp*]:
rel-bset R (*bset-of-nebset X*) (*bset-of-nebset Y*) = *rel-nebset R X Y*
by *transfer* (*rule refl*)

lemma *rel-nebset-conj*[*simp*]:
rel-nebset (λ*x y*. *P* ∧ *Q x y*) *x y* ⟷ *P* ∧ *rel-nebset Q x y*
rel-nebset (λ*x y*. *Q x y* ∧ *P*) *x y* ⟷ *P* ∧ *rel-nebset Q x y*
by (*transfer*, *auto simp*: *rel-set-def*)+

lemma *set-bset-empty*[*simp*]: *set-bset X* = {} ⟷ *X* = *bempty*
by *transfer simp*

end