Types Disproved Being BNFs during the Formalization of the Probabilistic Hierarchy

Johannes Hölzl Andreas Lochbihler Dmitriy Traytel

September 13, 2023

Contents

1 Sets Bounded by a Finite Cardinal > 2 Are Not BNFs

Do not import this theory. It contains an inconsistent axiomatization. The point is to exhibit the particular inconsistency.

typedef $('a, 'k)$ *bset* (- *set*[-] [22, 21] 21) =
$\{A :: 'a\ set.\ |A| <o\ |UNIV :: 'k\ set|\}$
morphisms *set-bset Abs-bset*
by (*rule exI*[*of* - $\{\}$]) (*auto simp*: *card-of-empty4 csum-def*)

setup-lifting *type-definition-bset*

lift-definition *map-bset* ::
$('a \Rightarrow 'b) \Rightarrow 'a\ set['k] \Rightarrow 'b\ set['k]$ **is** *image*
using *card-of-image ordLeq-ordLess-trans* **by** *blast*

inductive *rel-bset* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a, 'k)\ bset \Rightarrow ('b, 'k)\ bset \Rightarrow bool$ **for** R **where**
set-bset $x \subseteq \{(x, y).\ R\ x\ y\} \Longrightarrow$ *rel-bset* R (*map-bset fst* x) (*map-bset snd* x)

We axiomatize the relator commutation property and show that we can deduce *False* from it.

We cannot do this with a locale, since we need the fully polymorphic version of the following axiom.

axiomatization where
inconsistent: *rel-bset* $R1$ *OO* *rel-bset* $R2 \leq$ *rel-bset* ($R1$ *OO* $R2$)

```
bnf ('a, 'k) bset
  map: map-bset
  sets: set-bset
  bd: natLeq +c card-suc ( |UNIV :: 'k set| )
  rel: rel-bset
proof (standard, goal-cases)
  case 1 then show ?case
    by transfer simp
next
  case 2 then show ?case
    apply (rule ext)
    apply transfer
    apply auto
    done
next
  case 3 then show ?case
    apply transfer
    apply (auto simp: image-iff)
    done
next
  case 4 then show ?case
    apply (rule ext)
    apply transfer
    apply simp
    done
next
  case 5 then show ?case by (rule card-order-bd-fun)
next
  case 6 then show ?case by (rule Cinfinite-bd-fun[THEN conjunct1])
next
  case 7 then show ?case by (rule regularCard-bd-fun)
next
  case 8 then show ?case
    by transfer
      (erule ordLess-ordLeq-trans[OF - ordLeq-transitive[OF - ordLeq-csum2]];
        simp add: card-suc-greater ordLess-imp-ordLeq Card-order-card-suc)
next
  case 9 then show ?case by (rule inconsistent) — BAAAAAMMMM
next
  case 10 then show ?case
    by (auto simp: fun-eq-iff intro: rel-bset.intros elim: rel-bset.cases)
qed

lemma card-option-finite[simp]:
  assumes finite (UNIV :: 'k set)
  shows card (UNIV :: 'k option set) = Suc (card (UNIV :: 'k set))
  (is card ?L = Suc (card ?R))
proof −
  have card ?L = Suc (card (?L − {None})) by (rule card.remove) (auto simp:
```

assms)
  also have *card* (*?L* − {*None*}) = *card ?R*
    by (*rule bij-betw-same-card*[*of the*])
      (*auto simp*: *bij-betw-def inj-on-def image-iff intro*!: *bexI*[*of* - *Some x* **for** *x*])
  finally show *?thesis* **.**
qed

datatype ($'a$ :: *enum*) *x* = *A* | *B* $'a$ *option* | *C*

abbreviation *Bs* ≡ *B* ' (*insert None* (*Some* ' *set Enum.enum*))

lemma *UNIV-x*[*simp*]:
  (*UNIV* :: ($'a$ :: *enum*) *x set*) = {*A*, *C*} ∪ *Bs*
  (**is** - = *?R*)
proof (*intro set-eqI iffI*)
  fix *x* :: $'a$ *x* **show** *x* ∈ *?R* **by** (*cases x*) (*auto simp add*: *enum-UNIV*)
qed *simp*

lemma *Collect-split-in-rel*: {(*x*, *y*). *in-rel R x y*} = *R*
  by *auto*

lift-definition *X* :: ($'a$ :: *enum x*, $'a$ *x*) *bset* **is** *insert A Bs*
  by (*subst finite-card-of-iff-card3*) (*auto simp*: *card.insert-remove card-Diff-singleton-if*)

lift-definition *Y* :: ($'a$ :: *enum x*, $'a$ *x*) *bset* **is** *insert C Bs*
  by (*subst finite-card-of-iff-card3*) (*auto simp*: *card.insert-remove card-Diff-singleton-if*)

lift-definition *Z* :: ($'a$ :: *enum x*, $'a$ *x*) *bset* **is** {*A*, *C*}
  by (*subst finite-card-of-iff-card3*) (*auto simp*: *card.insert-remove card-Diff-singleton-if*)

lift-definition *R* :: ($'a$ *x* × $'a$ *x*, $'a$ :: *enum x*) *bset* **is** *insert* (*A*, *A*) ((λ*B*. (*B*, *C*))
' *Bs*)
  by (*subst finite-card-of-iff-card3*)
   (*auto simp*: *card.insert-remove card-Diff-singleton-if image-iff card-image inj-on-def*)

lift-definition *S* :: ($'a$ *x* × $'a$ *x*, $'a$ :: *enum x*) *bset* **is** *insert* (*C*, *C*) ((λ*B*. (*A*, *B*))
' *Bs*)
  by (*subst finite-card-of-iff-card3*)
   (*auto simp*: *card.insert-remove card-Diff-singleton-if image-iff card-image inj-on-def*)

lift-definition *in-brel* :: ($'a$ × $'b$, $'k$) *bset* ⇒ $'a$ ⇒ $'b$ ⇒ *bool* **is** *in-rel* **.**

lemma *False*
proof −
  have *rel-bset* (*in-brel R*) *X Z*
    unfolding *bset.in-rel mem-Collect-eq*
    apply (*intro exI*[*of* - *R*])
    apply *transfer*
    apply (*auto simp*: *image-iff*)

done
moreover
have *rel-bset* (*in-brel S*) *Z Y*
unfolding *bset.in-rel mem-Collect-eq*
apply (*intro exI*[*of* - *S*])
apply *transfer*
apply (*auto simp*: *image-iff*)
done
ultimately have *rel-bset* (*in-brel R OO in-brel S*) *X Y*
unfolding *bset.rel-compp* **by** *blast*
moreover
have ∗: *insert* (*A*, *A*) ((λ*B*. (*B*, *C*)) ‘ *Bs*) *O insert* (*C*, *C*) ((λ*B*. (*A*, *B*)) ‘ *Bs*)
=
((λ*B*. (*B*, *C*)) ‘ *Bs*) ∪ ((λ*B*. (*A*, *B*)) ‘ *Bs*) (**is** - = *?RS*) **by** *auto*
have ¬ *rel-bset* (*in-brel R OO in-brel S*) *X Y*
unfolding *bset.in-rel mem-Collect-eq*
proof (*transfer*, *safe*, *unfold relcompp-in-rel* ∗ *Collect-split-in-rel*)
fix *Z* :: (′*a* :: *enum x* × ′*a x*) *set*
note *enum-UNIV*[*simp*] *UNIV-option-conv*[*symmetric*, *simp*]
assume *Z* ⊆ *?RS fst* ‘ *Z* = *insert A Bs snd* ‘ *Z* = *insert C Bs*
then have *Z* = *?RS* **unfolding** *fst-eq-Domain snd-eq-Range* **by** *auto*
moreover assume |*Z*| <*o* |*UNIV* :: ′*a x set*|
ultimately show *False* **unfolding** ‹*Z* = *?RS*›
by (*subst* (*asm*) *finite-card-of-iff-card3*, *simp*, *simp*, *subst* (*asm*) *card-Un-disjoint*)
(*auto simp*: *card.insert-remove card-Diff-singleton-if card-image inj-on-def*
split: *if-splits*)
qed
ultimately show *False* **by** *blast*
qed

2 Vardi Systems Are Not a BNF

Do not import this theory. It contains an inconsistent axiomatization. The point is to exhibit the particular inconsistency.

We axiomatize the relator commutation property and show that we can deduce *False* from it.

We cannot do this with a locale, since we need the fully polymorphic version of the following axiom.

axiomatization where
inconsistent: *rel-var R1 S1 OO rel-var R2 S2* ≤ *rel-var* (*R1 OO R2*) (*S1 OO S2*)

bnf (′*a*, ′*b*, ′*k*) *var*
map: *map-var*
sets: *set1-var set2-var*
bd: *bd-pre-var0* :: ′*k var0-pre-var0-bdT rel*
rel: *rel-var*

proof (*standard*, *goal-cases*)
  case *1* **then show** *?case*
    by *transfer* (*auto simp add*: *var0.map-id*)
next
  case *2* **then show** *?case*
    apply (*rule ext*)
    apply *transfer*
    apply (*auto simp add*: *var0.map-comp*)
    done
next
  case *3* **then show** *?case*
    apply *transfer*
    apply (*subst var0.map-cong0*)
    apply *assumption*
    apply *assumption*
    apply *auto*
    done
next
  case *4* **then show** *?case*
    apply (*rule ext*)
    apply *transfer*
    apply (*simp add*: *var0.set-map0*)
    done
next
  case *5* **then show** *?case*
    apply (*rule ext*)
    apply *transfer*
    apply (*simp add*: *var0.set-map0*)
    done
next
  case *6* **then show** *?case* **by** (*rule var0.bd-card-order*)
next
  case *7* **then show** *?case*
    by (*simp add*: *var0.bd-cinfinite*)
next
  case *8* **then show** *?case* **by** (*rule var0.bd-regularCard*)
next
  case (*9 x*) **then show** *?case*
    unfolding *subset-eq set1-var-def* **by** (*simp add*: *var0.set-bd*(*1*))
next
  case (*10 x*) **then show** *?case*
    unfolding *subset-eq set2-var-def* **by** (*simp add*: *var0.set-bd*(*2*))
next
  case *11* **then show** *?case* **by** (*rule inconsistent*) — BAAAAAMMMM
next
  case *12* **then show** *?case*
      unfolding *rel-var.simps*[*abs-def*] **by** (*auto simp*: *fun-eq-iff*)
qed

lift-definition *X* :: (*bool*, *'b*, *'k*) *var* **is** *BPS* (*binsert* (*True*, *undefined*) (*binsert* (*False*, *undefined*) *bempty*))**.**

lift-definition *Y* :: (*bool*, *'b*, *'k*) *var* **is** *PMF* (*pmf-of-set* {(*True*, *undefined*), (*False*, *undefined*)})**.**

lift-definition *Z* :: (*bool*, *'b*, *'k*) *var* **is** *PMF* (*return-pmf* (*True*, *undefined*))**.**

lift-definition *Z'* :: (*bool*, *'b*, *'k*) *var* **is** *BPS* (*bsingleton* (*True*, *undefined*))**.**

lift-definition *C* :: (*bool*×*bool*, *'b*×*'b*, *'k*) *var* **is**
BPS (*binsert* ((*True*, *True*), (*undefined*, *undefined*)) (*binsert* ((*False*, *True*), (*undefined*, *undefined*)) *bempty*))**.**

lift-definition *C'* :: (*bool*×*bool*, *'b*×*'b*, *'k*) *var* **is**
PMF (*map-pmf* (λ((*a*, *b*), (*c*, *d*)). ((*a*,*c*), (*b*,*d*))) (*pair-pmf* (*return-pmf* (*True*, *undefined*)) (*pmf-of-set* {(*True*, *undefined*), (*False*, *undefined*)})))**.**

lemma *Z-eq-Z'*: *Z* = *Z'*
by *transfer auto*

lemma *False*
proof −
have [*simp*]: ⋀*x*. *pmf-of-set* {(*True*, *undefined*), (*False*, *undefined*)} ≠ *return-pmf* *x*
by (*auto simp*: *pmf-eq-iff split*: *split-indicator*)
have [*simp*]: ⋀*x*. *binsert* (*True*, *undefined*) (*binsert* (*False*, *undefined*) *bempty*) ≠ *bsingleton* *x*
unfolding *bsingleton-def* **by** *transfer auto*

define *R* **where** *R a b* = *b* **for** *a b* :: *bool*
have *rel-var R* (=) *X Z'*
unfolding *R-def var.in-rel mem-Collect-eq subset-eq*
apply (*intro exI*[*of - C*])
apply *transfer*
apply (*auto simp*: *set-bset binsert.rep-eq fsts.simps snds.simps bempty.rep-eq bsingleton-def*)
done
moreover
define *S* **where** *S a b* = *a* **for** *a b* :: *bool*
have *rel-var S* (=) *Z Y*
unfolding *S-def var.in-rel mem-Collect-eq subset-eq*
apply (*intro exI*[*of - C'*])
apply *transfer*
apply (*auto simp*: *fsts.simps snds.simps pmf.map-comp comp-def split-beta map-fst-pair-pmf map-snd-pair-pmf*)
done
ultimately have *rel-var* (*R OO S*) ((=) *OO* (=)) *X Y* (**is** *rel-var ?R ?S X Y*)
unfolding *var.rel-compp* **unfolding** *Z-eq-Z'* **by** *blast*

moreover have ¬ *rel-var ?R ?S X Y*
unfolding *var.in-rel mem-Collect-eq subset-eq*
apply (*auto simp*: *split-beta*)
apply *transfer′*
apply (*auto elim*!: *var-eq.cases*)
apply (*case-tac* [!] *z*)
apply (*auto simp add*: *snds.simps*)
done
ultimately show *False*
by *auto*
qed