

Ramsey's Theorem

Tom Ridge

April 20, 2020

Abstract

The infinite form of Ramsey's Theorem is proved following Boolos and Jeffrey, Chapter 26.

Contents

1 Ramsey's Theorem	1
1.1 Library lemmas	1
1.2 Dependent Choice Variant	2
1.3 Partitions	3
1.4 Ramsey's theorem	3

1 Ramsey's Theorem

```
theory Ramsey
imports Main HOL-Library.Infinite-Set
begin
```

```
declare [[simp-depth-limit = 5]]
```

1.1 Library lemmas

```
lemma infinite-inj-infinite-image: infinite Z  $\implies$  inj-on f Z  $\implies$  infinite (f ` Z)
  apply(rule ccontr)
  apply(simp)
  apply(force dest: finite-imageD)
done
```

```
lemma infinite-dom-finite-rng: [[ infinite A; finite (f ` A) ]]  $\implies$   $\exists b \in f ` A.$ 
infinite {a : A. f a = b}
  apply(rule ccontr) apply(simp)
  apply(subgoal-tac  $\cup ((\lambda b. \{a : A. f a = f b\}) ` A) = A$ ) prefer 2 apply(blast)
  apply(subgoal-tac (UN c : f ` A. {a : A. f a = c}) = (UN b:A. {a : A. f a = f
b})) prefer 2 apply(blast)
  apply(subgoal-tac finite (UN c:f ` A. {a : A. f a = c})) apply(force)
```

apply(*rule finite-UN-I*)
apply(*auto*)
done

lemma *infinite-mem*: $\text{infinite } X \implies \exists x. x \in X$
apply(*rule ccontr*)
apply(*force*)
done

lemma *not-empty-least*: $(Y :: \text{nat set}) \neq \{\}$ $\implies \exists m. m \in Y \wedge (\forall m'. m' \in Y \longrightarrow m \leq m')$
apply(*rule-tac x=LEAST x. x : Y in exI*)
apply(*rule*)
apply(*rule LeastI-ex*) **apply**(*force*)
apply(*rule*)
apply(*rule*)
apply(*rule Least-le*)
apply(*assumption*)
done

1.2 Dependent Choice Variant

—
primrec *choice* :: $('a \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow \text{nat} \Rightarrow 'a$ **where**
choice $P R 0 = (\text{SOME } x. P x)$
| *choice* $P R (\text{Suc } n) = (\text{let } x = \text{choice } P R n \text{ in } \text{SOME } y. P y \wedge R x y)$
—

lemma *dc*:
 $(\forall x y z. R x y \wedge R y z \longrightarrow R x z)$
 $\wedge (\exists x0. P x0)$
 $\wedge (\forall x. P x \longrightarrow (\exists y. P y \wedge R x y))$
 $\longrightarrow (\exists f :: \text{nat} \Rightarrow 'b. (\forall n. P (f n)) \wedge (\forall n m. R (f n) (f (n+m+1))))$

apply(*intro allI impI, elim exE conjE*)
apply(*rule-tac x=choice P R in exI*)
apply(*subgoal-tac* $(\forall n. P (\text{choice } P R n))$) **prefer** 2
apply(*rule, induct-tac n*)
apply(*simp add: Let-def*) **apply**(*rule someI-ex*) **apply**(*blast*)
apply(*simp add: Let-def*) **apply**(*subgoal-tac* $P (\text{SOME } y. P y \wedge R (\text{choice } P R na) y) \wedge R (\text{choice } P R na) (\text{SOME } y. P y \wedge R (\text{choice } P R na) y)$) **apply**(*blast*)
apply(*rule someI-ex*) **apply**(*blast*)
apply(*rule*) **apply**(*assumption*)

apply(*subgoal-tac* $\forall n. R (\text{choice } P R n) (\text{choice } P R (\text{Suc } n))$) **prefer** 2
apply(*rule*)
apply(*simp add: Let-def*)
apply(*subgoal-tac* $P (\text{SOME } y. P y \wedge R (\text{choice } P R n) y) \wedge R (\text{choice } P R n) (\text{SOME } y. P y \wedge R (\text{choice } P R n) y)$) **apply**(*blast*)

apply(*rule someI-ex*) **apply**(*force*)

apply(*rule*) **apply**(*rule*)
apply(*induct-tac m*)
apply(*force*)
apply(*drule-tac x=n+na+1 in spec*) **back**
apply(*force simp del: choice.simps*)
done

1.3 Partitions

definition

part :: *nat* \Rightarrow *nat* \Rightarrow '*a set* \Rightarrow ('*a set* \Rightarrow *nat*) \Rightarrow *bool* **where**
part r s Y f = ($\forall X. X \subseteq Y \wedge \text{finite } X \wedge \text{card } X = r \longrightarrow f X < s$)

lemma *part*: [*infinite YY*; *part (Suc n) s YY f*; *yy : YY*] \implies *part n s (YY*
- {yy}) ($\lambda u. f (\text{insert } yy u)$)
apply(*simp add: part-def*)
apply(*intro allI impI, elim exE conjE*)
apply(*drule-tac x=insert yy X in spec*)
apply(*force simp: card-Diff-singleton-if*)
done

lemma *part-subset*: *part (Suc n) s YY f* $\implies Y \subseteq YY \implies$ *part (Suc n) s Y f*
apply(*simp add: part-def*)
apply(*blast*)
done

1.4 Ramsey's theorem

lemma *ramsey*:

$\forall (s::\text{nat}) (r::\text{nat}) (YY::'\text{a set}) (f::'\text{a set} \Rightarrow \text{nat}).$
infinite YY
 $\wedge (\forall X. X \subseteq YY \wedge \text{finite } X \wedge \text{card } X = r \longrightarrow f X < s)$
 $\longrightarrow (\exists Y' t'.$
 $Y' \subseteq YY$
 $\wedge \text{infinite } Y'$
 $\wedge t' < s$
 $\wedge (\forall X. X \subseteq Y' \wedge \text{finite } X \wedge \text{card } X = r \longrightarrow f X = t')$)
apply(*simp add: part-def[symmetric] del: ex-simps*)
apply(*rule, rule, rule-tac nat.induct*)

apply(*intro allI impI*)
apply(*rule-tac x=YY in exI*)
apply(*rule-tac x=f {} in exI*)
apply(*force simp: part-def*)

apply(*intro allI impI*) **apply**(*elim exE conjE*)

apply(*subgoal-tac*

$\exists g.$

($\forall m::\text{nat. let } (y, Y, t) = (g\ m) \text{ in}$

$y \in YY \wedge y \notin Y$

$\wedge Y \subseteq YY \wedge \text{infinite } Y$

$\wedge t < s$

$\wedge (\forall X. X \subseteq Y \wedge \text{finite } X \wedge \text{card } X = \text{nat} \longrightarrow (f \circ \text{insert } y) X = t)$

$\wedge (\forall m\ m'.$

$\text{let } (y, Y, t) = (g\ m) \text{ in}$

$\text{let } (y', Y', t') = (g\ (m+m'+1)) \text{ in}$

$y' : Y$

$\wedge Y' \subseteq Y$

)

)

prefer 2

apply(*cut-tac*

$P = \lambda gn.$

$\text{let } (y, Y, t) = (gn) \text{ in}$

$y \in YY \wedge y \notin Y$

$\wedge Y \subseteq YY \wedge \text{infinite } Y$

$\wedge t < s$

$\wedge (\forall X. X \subseteq Y \wedge \text{finite } X \wedge \text{card } X = \text{nat} \longrightarrow (f \circ \text{insert } y) X = t)$

and $R = \lambda gn\ gn'.$

$\text{let } (y, Y, t) = (gn) \text{ in}$

$\text{let } (y', Y', t') = (gn') \text{ in}$

$y' : Y$

$\wedge Y' \subseteq Y$

in *dc*)

apply(*erule impE*)

apply(*intro conjI*)

apply(*intro allI impI, elim conjE*) **apply**(*simp add: Let-def split-beta*) **ap-**

ply(*blast*)

apply(*subgoal-tac* $\exists yy. yy \in YY$) **prefer** 2 **apply**(*rule infinite-mem*) **ap-**

ply(*assumption*)

apply(*elim exE conjE*)

apply(*drule-tac* $x=YY - \{yy\}$ **in** *spec*) **apply**(*drule-tac* $x=f \circ \text{insert } yy$ **in** *spec*)

apply(*erule impE*) **apply**(*simp*) **apply**(*rule part*) **apply**(*assumption+*)

apply(*elim exE conjE*)

apply(*rule-tac* $x=(yy, Y', t')$ **in** *exI*) **apply**(*simp*) **apply**(*blast*)

apply(*intro allI impI*)

apply(*case-tac* x) **apply**(*rename-tac* $yx\ b\ Yx\ tx$)

apply(*subgoal-tac* $\exists yx'. yx' \in Yx$) **prefer** 2 **apply**(*rule infinite-mem*) **ap-**

ply(*force*)
apply(*elim exE conjE*)
apply(*drule-tac x=Yx - {yx'} in spec*)
apply(*drule-tac x=f ◦ insert yx' in spec*)
apply(*erule impE*) **apply**(*simp*) **apply**(*elim exE conjE*) **apply**(*rule part*)
apply(*assumption+*)
apply(*rule part-subset*) **apply**(*assumption*) **apply**(*assumption*) **apply**(*assumption*)
apply(*elim exE conjE*)
apply(*rule-tac x=(yx', Y', t') in exI*) **apply**(*simp*) **apply**(*blast*)
apply(*assumption*)

apply(*elim exE conjE*)

apply(*subgoal-tac ∃ s'. s' < s ∧ infinite {n. (λn. let (y, Y, t) = g n in t) n = s'}*) **prefer** 2

apply(*subgoal-tac ∃ s' ∈ ((λn. let (Y, y, t) = g n in t) 'UNIV). infinite {n : UNIV. (let (Y, y, t) = g n in t) = s'}*) **prefer** 2
apply(*rule infinite-dom-finite-rng*) **apply**(*simp*)
apply(*simp (no-asm) add: finite-nat-iff-bounded*)
apply(*rule-tac x=s in exI*)
apply(*rule*)
apply(*simp add: image-iff*) **apply**(*elim exE conjE*)
apply(*drule-tac x=xa in spec*) **apply**(*force simp add: Let-def split-beta*)
apply(*elim bexE conjE*) **apply**(*rule-tac x=s' in exI*) **apply**(*simp*)
apply(*simp add: image-iff*) **apply**(*elim exE conjE*) **apply**(*drule-tac x=x in spec*) **apply**(*force simp: Let-def split-beta*)

apply(*elim exE conjE*)
apply(*rule-tac x=(λn. let (y, Y, t) = g n in y) ' {n. (λn. let (y, Y, t) = g n in t) n = s'}*) **in exI**
apply(*rule-tac x=s' in exI*)

apply(*subgoal-tac inj (λn. let (y, Y, t) = g n in y)*) **prefer** 2
apply(*simp add: inj-on-def*)

apply(*subgoal-tac ∀ x y. x < y ∧ (let (y, Y, t) = g x in y) = (let (y, Y, t) = g y in y) → x = y*)
apply(*intro allI impI*)
apply(*subgoal-tac x < y | x = y | y < x*) **prefer** 2 **apply**(*arith*)
apply(*elim disjE*)
apply(*drule-tac x=x in spec*) **back back** **apply**(*drule-tac x=y in spec*) **back**
back **apply**(*force simp: Let-def*)
apply(*assumption*)
apply(*drule-tac x=y in spec*) **back back** **apply**(*drule-tac x=x in spec*) **back**
back **apply**(*force simp: Let-def*)
apply(*intro allI impI*)
apply(*drule-tac x=x in spec*) **apply**(*drule-tac x=x in spec*) **apply**(*drule-tac x=y-(Suc x) in spec*) **apply**(*force simp: Let-def*)

```

apply(intro allI conjI)

apply(drule-tac x=xa in spec)
apply(force simp add: Let-def split-beta)

apply(rule infinite-inj-infinite-image) apply(assumption)
apply(simp add: inj-on-def)

apply(simp)

apply(intro allI impI, elim exE conjE)
apply(simp add: subset-image-iff) apply(elim exE conjE)
apply(subgoal-tac  $\exists a. a \in AA \wedge (\forall a'. a' \in AA \longrightarrow a \leq a')$ ) prefer 2 apply(rule
not-empty-least) apply(force)
apply(elim exE conjE)
apply(case-tac g a rule: prod.exhaust)
apply(rename-tac b) apply(case-tac b rule: prod.exhaust) apply(rename-tac ya
b Ya ta)
apply(subgoal-tac ya : X) prefer 2 apply(force intro!: rev-image-eqI simp:
Let-def)
apply(drule-tac s=X in sym)
apply(subgoal-tac f X = (f o insert ya) (X - {ya})) apply(simp)
apply(drule-tac x=a in spec)
apply(simp add: Let-def) apply(elim exE conjE)
apply(drule-tac x=X-{ya} in spec) back apply(erule impE)
apply(simp)
apply(rule)
apply(rule)
apply(drule-tac t=X in sym) apply(simp)
apply(simp add: image-iff) apply(elim bexE exE conjE) apply(rename-tac a')
apply(subgoal-tac a'  $\neq$  a) prefer 2 apply(force)
apply(drule-tac x=a in spec)
apply(drule-tac x=a'-Suc a in spec) back
apply(simp add: Let-def split-beta) apply(case-tac g a' rule: prod.exhaust) ap-
ply(case-tac ba rule: prod.exhaust) apply(rename-tac ya' ba Ya' ta') apply(simp
add: Let-def split-beta)
apply(drule-tac x=a' in spec) apply(erule impE) apply(force)
apply(force)
apply(force simp add: card-Diff-singleton-if)
apply(subgoal-tac ta = s') apply(simp) apply(force)
apply(simp) apply(rule-tac f=f in arg-cong) apply(force)
done

end

```