Recursion Theorem

Georgy Dunaev

March 17, 2025

Abstract

This document contains a proof of the recursion theorem. This is a mechanization of the proof of the recursion theorem from the text *Introduction to Set Theory*, by Karel Hrbacek and Thomas Jech. This implementation may be used as the basis for a model of Peano Arithmetic in ZF. While recursion and the natural numbers are already available in ZF, this clean development is much easier to follow.

Contents

1 Recursion Submission

Recursion Theorem is proved in the following document. It also contains the addition on natural numbers. The development is done in the context of Zermelo-Fraenkel set theory.

theory *recursion*
  imports *ZF*
begin

2 Basic Set Theory

Useful lemmas about sets, functions and natural numbers

lemma *pisubsig* : ‹*Pi*(*A*,*P*)⊆*Pow*(*Sigma*(*A*,*P*))›
⟨*proof*⟩

lemma *apparg*:
fixes *f A B*
assumes *T0*:‹*f*:*A*→*B*›
assumes *T1*:‹*f* ‘ *a* = *b*›
assumes *T2*:‹*a* ∈ *A*›
shows ‹⟨*a*, *b*⟩ ∈ *f*›
⟨*proof*⟩

theorem *nat-induct-bound* :
assumes *H0*:‹*P*(*0*)›
assumes *H1*:‹!!*x*. *x*∈*nat* ⟹ *P*(*x*) ⟹ *P*(*succ*(*x*))›
shows ‹∀ *n*∈*nat*. *P*(*n*)›
⟨*proof*⟩

theorem *nat-Tr* : ‹∀ *n*∈*nat*. *m*∈*n* ⟶ *m*∈*nat*›
⟨*proof*⟩

theorem *zeroleq* : ‹∀ *n*∈*nat*. *0*∈*n* ∨ *0*=*n*›
⟨*proof*⟩

theorem *JH2-1ii* : ‹*m*∈*succ*(*n*) ⟹ *m*∈*n*∨*m*=*n*›
⟨*proof*⟩

theorem *nat-transitive*:‹∀ *n*∈*nat*. ∀ *k*. ∀ *m*. *k* ∈ *m* ∧ *m* ∈ *n* ⟶ *k* ∈ *n*›
⟨*proof*⟩

theorem *nat-xninx* : ‹∀ *n*∈*nat*. ¬(*n*∈*n*)›
⟨*proof*⟩

theorem *nat-asym* : ‹∀ *n*∈*nat*. ∀ *m*. ¬(*n*∈*m* ∧ *m*∈*n*)›
⟨*proof*⟩

theorem *zerolesucc* :‹∀ *n*∈*nat*. *0* ∈ *succ*(*n*)›
⟨*proof*⟩

theorem *succ-le* : ‹∀ *n*∈*nat*. *succ*(*m*)∈*succ*(*n*) ⟶ *m*∈*n*›
⟨*proof*⟩

theorem *succ-le2* : ‹∀ *n*∈*nat*. ∀ *m*. *succ*(*m*)∈*succ*(*n*) ⟶ *m*∈*n*›
⟨*proof*⟩

theorem *le-succ* : ‹∀ *n*∈*nat*. *m*∈*n* ⟶ *succ*(*m*)∈*succ*(*n*)›

⟨*proof*⟩

theorem *nat-linord*:‹∀ *n*∈*nat*. ∀ *m*∈*nat*. *m*∈*n*∨*m*=*n*∨*n*∈*m*›
⟨*proof*⟩

lemma *tgb*:
assumes *knat*: ‹*k*∈*nat*›
assumes *D*: ‹*t* ∈ *k* → *A*›
shows ‹*t* ∈ *Pow*(*nat* × *A*)›
⟨*proof*⟩

3 Compatible set

Union of compatible set of functions is a function.

definition *compat* :: ‹[*i*,*i*]⇒*o*›
where *compat*(*f1*,*f2*) == ∀ *x*.∀ *y1*.∀ *y2*.⟨*x*,*y1*⟩ ∈ *f1* ∧ ⟨*x*,*y2*⟩ ∈ *f2* ⟶ *y1*=*y2*

lemma *compatI* [*intro*]:
assumes *H*:‹⋀*x y1 y2*.⟦⟨*x*,*y1*⟩ ∈ *f1*; ⟨*x*,*y2*⟩ ∈ *f2*⟧⟹*y1*=*y2*›
shows ‹*compat*(*f1*,*f2*)›
⟨*proof*⟩

lemma *compatD*:
assumes *H*: ‹*compat*(*f1*,*f2*)›
shows ‹⋀*x y1 y2*.⟦⟨*x*,*y1*⟩ ∈ *f1*; ⟨*x*,*y2*⟩ ∈ *f2*⟧⟹*y1*=*y2*›
⟨*proof*⟩

lemma *compatE*:
assumes *H*: ‹*compat*(*f1*,*f2*)›
and *W*:‹(⋀*x y1 y2*.⟦⟨*x*,*y1*⟩ ∈ *f1*; ⟨*x*,*y2*⟩ ∈ *f2*⟧⟹*y1*=*y2*) ⟹ *E*›
shows ‹*E*›
⟨*proof*⟩

definition *compatset* :: ‹*i*⇒*o*›
where *compatset*(*S*) == ∀ *f1*∈*S*.∀ *f2*∈*S*. *compat*(*f1*,*f2*)

lemma *compatsetI* [*intro*] :
assumes *1*:‹⋀*f1 f2*. ⟦*f1*∈*S*;*f2*∈*S*⟧ ⟹ *compat*(*f1*,*f2*)›
shows ‹*compatset*(*S*)›
⟨*proof*⟩

lemma *compatsetD*:
assumes *H*: ‹*compatset*(*S*)›
shows ‹⋀*f1 f2*.⟦*f1*∈*S*; *f2*∈*S*⟧⟹*compat*(*f1*,*f2*)›
⟨*proof*⟩

lemma *compatsetE*:

assumes *H*: ‹*compatset*(*S*)›
and *W*:‹(⋀*f1 f2*.⟦*f1*∈*S*; *f2*∈*S*⟧⟹*compat*(*f1*,*f2*)) ⟹ *E*›
shows ‹*E*›
⟨*proof*⟩

theorem *upairI1* : ‹*a* ∈ {*a*, *b*}›
⟨*proof*⟩

theorem *upairI2* : ‹*b* ∈ {*a*, *b*}›
⟨*proof*⟩

theorem *sinup* : ‹{*x*} ∈ ⟨*x*, *xa*⟩›
⟨*proof*⟩

theorem *compatsetunionfun* :
fixes *S*
assumes *H0*:‹*compatset*(*S*)›
shows ‹*function*(⋃ *S*)›
⟨*proof*⟩

theorem *mkel* :
assumes *1*:‹*A*›
assumes *2*:‹*A*⟹*B*›
shows ‹*B*›
⟨*proof*⟩

theorem *valofunion* :
fixes *S*
assumes *H0*:‹*compatset*(*S*)›
assumes *W*:‹*f*∈*S*›
assumes *Q*:‹*f*:*A*→*B*›
assumes *T*:‹*a*∈*A*›
assumes *P*:‹*f* ‘ *a* = *v*›
shows *N*:‹(⋃ *S*)‘*a* = *v*›
⟨*proof*⟩

4 Partial computation

definition *satpc* :: ‹[*i*,*i*,*i*] ⇒ *o* ›
where ‹*satpc*(*t*,*α*,*g*) == ∀ *n* ∈ *α* . *t*‘*succ*(*n*) = *g* ‘ <*t*‘*n*, *n*>›

m-step computation based on *a* and *g*

definition *partcomp* :: ‹[*i*,*i*,*i*,*i*,*i*]⇒*o*›
where ‹*partcomp*(*A*,*t*,*m*,*a*,*g*) == (*t*:*succ*(*m*)→*A*) ∧ (*t*‘*0*=*a*) ∧ *satpc*(*t*,*m*,*g*)›

lemma *partcompI* [*intro*]:
assumes *H1*:‹(*t*:*succ*(*m*)→*A*)›
assumes *H2*:‹(*t*‘*0*=*a*)›
assumes *H3*:‹*satpc*(*t*,*m*,*g*)›

shows ‹*partcomp*(*A*,*t*,*m*,*a*,*g*)›
⟨*proof*⟩

lemma *partcompD1*: ‹*partcomp*(*A*,*t*,*m*,*a*,*g*) ⟹ *t* ∈ *succ*(*m*) → *A*›
⟨*proof*⟩

lemma *partcompD2*: ‹*partcomp*(*A*,*t*,*m*,*a*,*g*) ⟹ (*t*‘*0*=*a*)›
⟨*proof*⟩

lemma *partcompD3*: ‹*partcomp*(*A*,*t*,*m*,*a*,*g*) ⟹ *satpc*(*t*,*m*,*g*)›
⟨*proof*⟩

lemma *partcompE* [*elim*] :
assumes *1*:‹*partcomp*(*A*,*t*,*m*,*a*,*g*)›
and *2*:‹⟦(*t*:*succ*(*m*)→*A*) ; (*t*‘*0*=*a*) ; *satpc*(*t*,*m*,*g*)⟧ ⟹ *E*›
shows ‹*E*›
⟨*proof*⟩

If we add ordered pair in the middle of partial computation then it will not change.

lemma *addmiddle*:

assumes *mnat*:‹*m*∈*nat*›
assumes *F*:‹*partcomp*(*A*,*t*,*m*,*a*,*g*)›
assumes *xinm*:‹*x*∈*m*›
shows ‹*cons*(⟨*succ*(*x*), *g* ‘ ⟨*t* ‘ *x*, *x*⟩⟩, *t*) = *t*›
⟨*proof*⟩

5 Set of functions

It is denoted as F on page 48 in "Introduction to Set Theory".

definition *pcs* :: ‹[*i*,*i*,*i*]⇒*i*›
where ‹*pcs*(*A*,*a*,*g*) == {*t*∈*Pow*(*nat*∗*A*). ∃ *m*∈*nat*. *partcomp*(*A*,*t*,*m*,*a*,*g*)}›

lemma *pcs-uniq* :
assumes *F1*:‹*m1*∈*nat*›
assumes *F2*:‹*m2*∈*nat*›
assumes *H1*: ‹*partcomp*(*A*,*f1*,*m1*,*a*,*g*)›
assumes *H2*: ‹*partcomp*(*A*,*f2*,*m2*,*a*,*g*)›
shows ‹∀ *n*∈*nat*. *n*∈*succ*(*m1*) ∧ *n*∈*succ*(*m2*) ⟶ *f1*‘*n* = *f2*‘*n*›
⟨*proof*⟩

lemma *domainsubsetfunc* :
assumes *Q*:‹*f1*⊆*f2*›
shows ‹*domain*(*f1*)⊆*domain*(*f2*)›
⟨*proof*⟩

lemma *natdomfunc*:

assumes *1*:‹$q \in A$›
assumes *J0*:‹$f1 \in Pow(nat \times A)$›
assumes *U*:‹$m1 \in domain(f1)$›
shows ‹$m1 \in nat$›
⟨*proof*⟩

lemma *pcs-lem* :
assumes *1*:‹$q \in A$›
shows ‹$compatset(pcs(A,\ a,\ g))$›
⟨*proof*⟩

theorem *fuissu* : ‹$f \in X \mathrel{->} Y \Longrightarrow f \subseteq X \times Y$›
⟨*proof*⟩

theorem *recuniq* :
fixes *f*
assumes *H0*:‹$f \in nat \mathrel{->} A \wedge f\ `\ 0 = a \wedge satpc(f,\ nat,\ g)$›
fixes *t*
assumes *H1*:‹$t \in nat \mathrel{->} A \wedge t\ `\ 0 = a \wedge satpc(t,\ nat,\ g)$›
fixes *x*
shows ‹$f = t$›
⟨*proof*⟩

6 Lemmas for recursion theorem

locale *recthm* =
fixes *A* :: *i*
and *a* :: *i*
and *g* :: *i*
assumes *hyp1* : ‹$a \in A$›
and *hyp2* : ‹$g : ((A * nat) \to A)$›
begin

lemma *l3*:‹$function(\bigcup pcs(A,\ a,\ g))$›
⟨*proof*⟩

lemma *l1* : ‹$\bigcup pcs(A,\ a,\ g) \subseteq nat \times A$›
⟨*proof*⟩

lemma *le1*:
assumes *H*:‹$x \in 1$›
shows ‹$x = 0$›
⟨*proof*⟩

lemma *lsinglfun* : ‹$function(\{\langle 0,\ a\rangle\})$›
⟨*proof*⟩

lemma *singlsatpc*:‹$satpc(\{\langle 0,\ a\rangle\},\ 0,\ g)$›
⟨*proof*⟩

lemma *zerostep* :
shows ‹*partcomp*(*A*, {⟨*0*, *a*⟩}, *0*, *a*, *g*)›
⟨*proof*⟩

lemma *zainupcs* : ‹⟨*0*, *a*⟩ ∈ ⋃ *pcs*(*A*, *a*, *g*)›
⟨*proof*⟩

lemma *l2′*: ‹*0* ∈ *domain*(⋃ *pcs*(*A*, *a*, *g*))›
⟨*proof*⟩

Push an ordered pair to the end of partial computation t and obtain another partial computation.

lemma *shortlem* :
assumes *mnat*:‹*m*∈*nat*›
assumes *F*:‹*partcomp*(*A*,*t*,*m*,*a*,*g*)›
shows ‹*partcomp*(*A*,*cons*(⟨*succ*(*m*), *g* ‘ <*t*‘*m*, *m*>⟩, *t*),*succ*(*m*),*a*,*g*)›
⟨*proof*⟩

lemma *l2*:‹*nat* ⊆ *domain*(⋃ *pcs*(*A*, *a*, *g*))›
⟨*proof*⟩

lemma *useful* : ‹∀ *m*∈*nat*. ∃ *t*. *partcomp*(*A*,*t*,*m*,*a*,*g*)›
⟨*proof*⟩

lemma *l4* : ‹(⋃ *pcs*(*A*,*a*,*g*)) ∈ *nat* –> *A*›
⟨*proof*⟩

lemma *l5*: ‹(⋃ *pcs*(*A*, *a*, *g*)) ‘ *0* = *a*›
⟨*proof*⟩

lemma *ballE2*:
assumes ‹∀ *x*∈*AA*. *P*(*x*)›
assumes ‹*x*∈*AA*›
assumes ‹*P*(*x*) ==> *Q*›
shows *Q*
⟨*proof*⟩

Recall that *satpc*(*t*,*α*,*g*) == ∀ *n* ∈ *α* . *t*‘*succ*(*n*) = *g* ‘ <*t*‘*n*, *n*> *partcomp*(*A*,*t*,*m*,*a*,*g*) == (*t*:*succ*(*m*)→*A*) ∧ (*t*‘*0*=*a*) ∧ *satpc*(*t*,*m*,*g*) *pcs*(*A*,*a*,*g*) == {*t*∈*Pow*(*nat*∗*A*). ∃ *m*. *partcomp*(*A*,*t*,*m*,*a*,*g*)}

lemma *l6new*: ‹*satpc*(⋃ *pcs*(*A*, *a*, *g*), *nat*, *g*)›
⟨*proof*⟩

7 Recursion theorem

theorem *recursionthm*:
shows ‹∃ !*f*. ((*f* ∈ (*nat*→*A*)) ∧ ((*f*‘*0*) = *a*) ∧ *satpc*(*f*,*nat*,*g*))›

⟨*proof*⟩

end

8 Lemmas for addition

Let's define function t(x) = (a+x). Firstly we need to define a function g:$nat \times nat \to nat$, such that $g`\langle t`n,\ n\rangle = t`succ(n) = a + (n + 1) = (a + n) + 1 = (t`n) + 1$ So $g`\langle a,\ b\rangle = a + 1$ and $g(p) = succ(pr1(p))$ and $satpc(t,\alpha,g) \Longleftrightarrow \forall\, n \in \alpha\ .\ t`succ(n) = succ(t`n)$.

definition *addg* :: ‹*i*›
  where *addg-def* : ‹*addg* == $\lambda x \in (nat*nat).\ succ(fst(x))$›

lemma *addgfun*: ‹*function*(*addg*)›
  ⟨*proof*⟩

lemma *addgsubpow* : ‹*addg* ∈ *Pow*((*nat* × *nat*) × *nat*)›
⟨*proof*⟩

lemma *addgdom* : ‹*nat* × *nat* ⊆ *domain*(*addg*)›
⟨*proof*⟩

lemma *plussucc*:
  assumes *F*:‹*f* ∈ (*nat*→*nat*)›
  assumes *H*:‹*satpc*(*f*,*nat*,*addg*)›
  shows ‹∀ *n* ∈ *nat* . *f*`*succ*(*n*) = *succ*(*f*`*n*)›
⟨*proof*⟩

9 Definition of addition

Theorem that addition of natural numbers exists and unique in some sense. Due to theorem 'plussucc' the term $satpc(f,nat,addg)$ can be replaced here with $\forall\, n \in nat\ .\ f`succ(n) = succ(f`n)$.

theorem *addition*:
  assumes ‹*a*∈*nat*›
  shows
 ‹∃!*f*. ((*f* ∈ (*nat*→*nat*)) ∧ ((*f*`*0*) = *a*) ∧ *satpc*(*f*,*nat*,*addg*))›
⟨*proof*⟩

end