First steps towards a formalization of Forcing

Gunther Pagano Sánchez Terraf

November 28, 2018

Contents

theory *Pointed-DC* **imports** *AC*

begin

This proof of DC is from Moschovakis "Notes on Set Theory"

consts *dc-witness* :: $i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i$
primrec
wit0 : *dc-witness*$(0,A,a,s,R) = a$
witrec :*dc-witness*$(succ(n),A,a,s,R) = s`\{x{\in}A.\ \langle dc\text{-}witness(n,A,a,s,R),x\rangle{\in}R\ \}$

lemma *witness-into-A* [*TC*]: $a{\in}A \Longrightarrow n{\in}nat \Longrightarrow$
$(\forall X\ .\ X{\neq}0 \wedge X{\subseteq}A \longrightarrow s`X{\in}X) \Longrightarrow$
$\forall y{\in}A.\ \{x{\in}A.\ \langle y,x\rangle{\in}R\ \} \neq 0 \Longrightarrow$
$dc\text{-}witness(n,\ A,\ a,\ s,\ R){\in}A$
apply (*induct-tac n* ,*simp*+)
apply (*drule-tac x*=*dc-witness*$(x,\ A,\ a,\ s,\ R)$ **in** *bspec*, *assumption*)
apply (*drule-tac x*=$\{xa \in A\ .\ \langle dc\text{-}witness(x,\ A,\ a,\ s,\ R),\ xa\rangle \in R\}$ **in** *spec*)
apply *auto*
done
lemma *witness-related* : $a{\in}A \Longrightarrow n{\in}nat \Longrightarrow$
$(\forall X\ .\ X{\neq}0 \wedge X{\subseteq}A \longrightarrow s`X{\in}X) \Longrightarrow$
$\forall y{\in}A.\ \{x{\in}A.\ \langle y,x\rangle{\in}R\ \} \neq 0 \Longrightarrow$
$\langle dc\text{-}witness(n,\ A,\ a,\ s,\ R),dc\text{-}witness(succ(n),\ A,\ a,\ s,$
$R)\rangle{\in}R$
apply (*frule-tac n*=*n* **and** *s*=*s* **and** *R*=*R* **in** *witness-into-A*, *assumption*+)
apply (*drule-tac x*=*dc-witness*$(n,\ A,\ a,\ s,\ R)$ **in** *bspec*, *assumption*)
apply (*drule-tac x*= $\{x \in A\ .\ \langle dc\text{-}witness(n,\ A,\ a,\ s,\ R),\ x\rangle \in R\}$ **in** *spec*)
apply (*simp*, *blast*)
done

lemma *witness-funtype*: $a{\in}A \Longrightarrow$
$(\forall X\ .\ X{\neq}0 \wedge X{\subseteq}A \longrightarrow s`X{\in}X) \Longrightarrow$
$\forall y{\in}A.\ \{x{\in}A.\ \langle y,x\rangle{\in}R\ \} \neq 0 \Longrightarrow$
$(\lambda n{\in}nat.\ dc\text{-}witness(n,\ A,\ a,\ s,\ R)) \in nat \to A$

apply (*rule-tac* B=$\{dc\text{-}witness(n,\ A,\ a,\ s,\ R).\ n{\in}nat\}$ **in** *fun-weaken-type*)
apply (*rule lam-funtype*)
apply (*blast intro*:*witness-into-A*)
done

lemma *witness-to-fun*: $a{\in}A \Longrightarrow (\forall X\ .\ X{\neq}0 \wedge X{\subseteq}A \longrightarrow s`X{\in}X) \Longrightarrow$
$\forall y{\in}A.\ \{x{\in}A.\ \langle y,x\rangle{\in}R\ \} \neq 0 \Longrightarrow$
$\exists f \in nat{\rightarrow}A.\ \forall n{\in}nat.\ f`n = dc\text{-}witness(n,A,a,s,R)$
apply (*rule-tac* x=$\lambda n{\in}nat.\ dc\text{-}witness(n,A,a,s,R)$ **in** *bexI*, *simp*)
apply (*rule witness-funtype*, *simp*+)
done

theorem *pointed-DC* : $(\forall x{\in}A.\ \exists y{\in}A.\ \langle x,y\rangle{\in}\ R) \Longrightarrow$
$\forall a{\in}A.\ (\exists f \in nat{\rightarrow}A.\ f`0 = a \wedge (\forall n \in nat.\ \langle f`n,f`succ(n)\rangle{\in}R))$
apply (*rule*)
apply (*insert AC-func-Pow*)
apply (*drule allI*)
apply (*drule-tac* x=A **in** *spec*)
apply (*drule-tac* P=$\lambda f\ .\forall x{\in}Pow(A) - \{0\}.\ f\ `\ x \in x$
and A=$Pow(A){-}\{0\}{\rightarrow}\ A$
and Q= $\exists f{\in}nat{\rightarrow}\ A.\ f\ `\ 0 = a \wedge (\forall n{\in}nat.\ \langle f\ `\ n,\ f\ `\ succ(n)\rangle \in R)$
in *bexE*)
prefer *2* **apply** (*assumption*)
apply (*rename-tac* s)
apply (*rule-tac* x=$\lambda n{\in}nat.\ dc\text{-}witness(n,A,a,s,R)$ **in** *bexI*)
prefer *2* **apply** (*blast intro*:*witness-funtype*)
apply (*rule conjI*, *simp*)
apply (*rule ballI*, *rename-tac* m)
apply (*subst beta*, *simp*)+
apply (*rule witness-related*, *auto*)
done

lemma *aux-DC-on-AxNat2* : $\forall x{\in}A{\times}nat.\ \exists y{\in}A.\ \langle x,\langle y,succ(snd(x))\rangle\rangle \in R \Longrightarrow$
$\forall x{\in}A{\times}nat.\ \exists y{\in}A{\times}nat.\ \langle x,y\rangle \in \{\langle a,b\rangle{\in}R.\ snd(b) = succ(snd(a))\}$
apply (*rule ballI*, *erule-tac* x=x **in** *ballE*, *simp-all*)
done

lemma *infer-snd* : $c{\in}\ A{\times}B \Longrightarrow snd(c) = k \Longrightarrow c{=}\langle fst(c),k\rangle$
by *auto*

corollary *DC-on-A-x-nat* :
$(\forall x{\in}A{\times}nat.\ \exists y{\in}A.\ \langle x,\langle y,succ(snd(x))\rangle\rangle \in R) \Longrightarrow$
$\forall a{\in}A.\ (\exists f \in nat{\rightarrow}A.\ f`0 = a \wedge (\forall n \in nat.\ \langle\langle f`n,n\rangle,\langle f`succ(n),succ(n)\rangle\rangle{\in}R))$
apply (*frule aux-DC-on-AxNat2*)
apply (*drule-tac* R=$\{\langle a,b\rangle{\in}R.\ snd(b) = succ(snd(a))\}$ **in** *pointed-DC*)
apply (*rule ballI*)
apply (*rotate-tac*)
apply (*drule-tac* x=$\langle a,0\rangle$ **in** *bspec*, *simp*)

apply (*erule bexE*, *rename-tac g*)
apply (*rule-tac x*=$\lambda x \in nat.\ fst(g`x)$ **and** $A=nat \to A$ **in** *bexI*, *auto*)
apply (*subgoal-tac* $\forall\, n \in nat.\ g`n = \langle fst(g\ `\ n),\ n\rangle$)
prefer *2* **apply** (*rule ballI*, *rename-tac m*)
apply (*induct-tac m*, *simp*)
apply (*rename-tac d*, *auto*)
apply (*frule-tac A*=*nat* **and** *x*=*d* **in** *bspec*, *simp*)
apply (*rule-tac A*=*A* **and** *B*=*nat* **in** *infer-snd*, *auto*)
apply (*rule-tac a*=$\langle fst(g\ `\ d),\ d\rangle$ **and** $b=g\ `\ d$ **in** *ssubst*, *assumption*)
apply (*subst snd-conv*, *simp*)
done

lemma *aux-sequence-DC* : $\forall\, x \in A.\ \forall\, n \in nat.\ \exists\, y \in A.\ \langle x,y\rangle \in S`n \Longrightarrow$
$\forall\, x \in A \times nat.\ \exists\, y \in A.\ \langle x,\langle y,succ(snd(x))\rangle\rangle \in \{\langle\langle x,n\rangle,\langle y,m\rangle\rangle \in (A \times nat) \times (A \times nat).$
$\langle x,y\rangle \in S`m\ \}$
by *auto*

lemma *sequence-DC*: $\forall\, x \in A.\ \forall\, n \in nat.\ \exists\, y \in A.\ \langle x,y\rangle \in S`n \Longrightarrow$
$\forall\, a \in A.\ (\exists f \in nat \to A.\ f`0 = a \wedge (\forall\, n \in nat.\ \langle f`n, f`succ(n)\rangle \in S`succ(n)))$
apply (*drule aux-sequence-DC*)
apply (*drule DC-on-A-x-nat*, *auto*)
done

end

theory *Forcing-notions* **imports** *Pointed-DC* **begin**

definition *compat-in* :: $i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow o$ **where**
$compat\text{-}in(A,r,p,q) == \exists\, d \in A\ .\ \langle d,p\rangle \in r \wedge \langle d,q\rangle \in r$

lemma *compat-inI* :
$[\![\ d \in A\ ;\ \langle d,p\rangle \in r\ ;\ \langle d,g\rangle \in r\ ]\!] \Longrightarrow compat\text{-}in(A,r,p,g)$
by (*auto simp add*: *compat-in-def*)

lemma *refl-compat*:
$[\![\ refl(A,r)\ ;\ \langle p,q\rangle \in r \mid p=q \mid \langle q,p\rangle \in r\ ;\ p \in A\ ;\ q \in A]\!] \Longrightarrow compat\text{-}in(A,r,p,q)$
by (*auto simp add*: *refl-def compat-inI*)

lemma *chain-compat*:
$refl(A,r) \Longrightarrow linear(A,r) \Longrightarrow\ (\forall\, p \in A.\forall\, q \in A.\ compat\text{-}in(A,r,p,q))$
by (*simp add*: *refl-compat linear-def*)

lemma *subset-fun-image*: $f:N \to P \Longrightarrow f``N \subseteq P$
by (*auto simp add*: *image-fun apply-funtype*)

definition
antichain :: $i \Rightarrow i \Rightarrow i \Rightarrow o$ **where**
$antichain(P,leq,A) == A \subseteq P \wedge (\forall\, p \in A.\forall\, q \in A.(\neg\ compat\text{-}in(P,leq,p,q)))$

definition
ccc :: $i \Rightarrow i \Rightarrow o$ **where**
$ccc(P,leq) == \forall A.\ antichain(P,leq,A) \longrightarrow |A| \leq nat$

locale *forcing-notion* =
fixes *P leq one*
assumes *one-in-P*: $one \in P$
and *leq-preord*: $preorder\text{-}on(P,leq)$
and *one-max*: $\forall p{\in}P.\ \langle p,one\rangle{\in}leq$
begin
definition
dense :: $i{\Rightarrow}o$ **where**
$dense(D) == \forall p{\in}P.\ \exists d{\in}D\ .\ \langle d,p\rangle{\in}leq$

definition
dense-below :: $i{\Rightarrow}i{\Rightarrow}o$ **where**
$dense\text{-}below(D,q) == \forall p{\in}P.\ \langle p,q\rangle{\in}leq \longrightarrow (\exists d{\in}D\ .\ \langle d,p\rangle{\in}leq)$

lemma *P-dense*: $dense(P)$
using *leq-preord*
unfolding *preorder-on-def refl-def dense-def*
by *blast*

definition
increasing :: $i{\Rightarrow}o$ **where**
$increasing(F) == \forall x{\in}F.\ \forall\ p \in P\ .\ \langle x,p\rangle{\in}leq \longrightarrow p{\in}F$

definition
compat :: $i{\Rightarrow}i{\Rightarrow}o$ **where**
$compat(p,q) == compat\text{-}in(P,leq,p,q)$

definition
antichain :: $i{\Rightarrow}o$ **where**
$antichain(A) == A{\subseteq}P \wedge (\forall p{\in}A.\forall q{\in}A.(\neg compat(p,q)))$

definition
filter :: $i{\Rightarrow}o$ **where**
$filter(G) == G{\subseteq}P \wedge increasing(G) \wedge (\forall p{\in}G.\ \forall q{\in}G.\ compat\text{-}in(G,leq,p,q))$

definition
upclosure :: $i{\Rightarrow}i$ **where**
$upclosure(A) == \{p{\in}P.\exists a{\in}A.\langle a,p\rangle{\in}leq\}$

lemma *upclosureI* [*intro*] : $p{\in}P \Longrightarrow a{\in}A \Longrightarrow \langle a,p\rangle{\in}leq \Longrightarrow p{\in}upclosure(A)$
by (*simp add*:*upclosure-def*, *auto*)

lemma *upclosureE* [*elim*] :
$p{\in}upclosure(A) \Longrightarrow (\bigwedge x\ a.\ x{\in}P \Longrightarrow a{\in}A \Longrightarrow \langle a,x\rangle{\in}leq \Longrightarrow R) \Longrightarrow R$

by (*auto simp add*:*upclosure-def*)

lemma *upclosureD* [*dest*] :
$p \in upclosure(A) \Longrightarrow \exists\, a \in A.(\langle a,p\rangle \in leq) \wedge p \in P$
by (*simp add*:*upclosure-def*)

lemma *upclosure-increasing* :
$A \subseteq P \Longrightarrow increasing(upclosure(A))$
apply (*unfold increasing-def upclosure-def*, *simp*)
apply *clarify*
apply (*rule-tac* $x{=}a$ **in** *bexI*)
apply (*insert leq-preord*, *unfold preorder-on-def*)
apply (*drule conjunct2*, *unfold trans-on-def*)
apply (*drule-tac* $x{=}a$ **in** *bspec*, *fast*)
apply (*drule-tac* $x{=}x$ **in** *bspec*, *assumption*)
apply (*drule-tac* $x{=}p$ **in** *bspec*, *assumption*)
apply (*simp*, *assumption*)
done

lemma *upclosure-in-P*: $A \subseteq P \Longrightarrow upclosure(A) \subseteq P$
apply (*rule subsetI*)
apply (*simp add*:*upclosure-def*)
done

lemma *A-sub-upclosure*: $A \subseteq P \Longrightarrow A \subseteq upclosure(A)$
apply (*rule subsetI*)
apply (*simp add*:*upclosure-def*, *auto*)
apply (*insert leq-preord*, *unfold preorder-on-def refl-def*, *auto*)
done

lemma *elem-upclosure*: $A \subseteq P \Longrightarrow x \in A \Longrightarrow x \in upclosure(A)$
by (*blast dest*:*A-sub-upclosure*)

lemma *closure-compat-filter*:
$A \subseteq P \Longrightarrow (\forall\, p \in A. \forall\, q \in A.\ compat\text{-}in(A,leq,p,q)) \Longrightarrow filter(upclosure(A))$
apply (*unfold filter-def*)
apply (*intro conjI*)
apply (*rule upclosure-in-P*, *assumption*)
apply (*rule upclosure-increasing*, *assumption*)
apply (*unfold compat-in-def*)
apply (*rule ballI*)+
apply (*rename-tac x y*)
apply (*drule upclosureD*)+
apply (*erule bexE*)+
apply (*rename-tac a b*)
apply (*drule-tac* $A{=}A$ **and** $x{=}a$ **in** *bspec*, *assumption*)
apply (*drule-tac* $A{=}A$ **and** $x{=}b$ **in** *bspec*, *assumption*)
apply (*auto*)
apply (*rule-tac* $x{=}d$ **in** *bexI*)

prefer *2* **apply** (*simp add*:*A-sub-upclosure* [*THEN subsetD*])
apply (*insert leq-preord*, *unfold preorder-on-def trans-on-def*, *drule conjunct2*)
apply (*rule conjI*)
apply (*drule-tac x*=*d* **in** *bspec*, *rule-tac A*=*A* **in** *subsetD*, *assumption*+)
apply (*drule-tac x*=*a* **in** *bspec*, *rule-tac A*=*A* **in** *subsetD*, *assumption*+)
apply (*drule-tac x*=*x* **in** *bspec*, *assumption*, *auto*)
done

lemma *aux-RS1*: $f \in N \to P \Longrightarrow n{\in}N \Longrightarrow f`n \in upclosure(f``N)$
apply (*rule-tac elem-upclosure*)
apply (*rule subset-fun-image*, *assumption*)
apply (*simp add*: *image-fun*, *blast*)
done
end

lemma *refl-monot-domain*: $refl(B,r) \Longrightarrow A{\subseteq}B \Longrightarrow refl(A,r)$
apply (*drule subset-iff* [*THEN iffD1*])
apply (*unfold refl-def*)
apply (*blast*)
done

lemma *decr-succ-decr*: $f \in nat \to P \Longrightarrow preorder\text{-}on(P,leq) \Longrightarrow$
$\forall\, n{\in}nat.\ \langle f`succ(n), f`n\rangle \in leq \Longrightarrow$
$n{\in}nat \Longrightarrow m{\in}nat \Longrightarrow n{\le}m \longrightarrow \langle f`m, f`n\rangle \in leq$
apply (*unfold preorder-on-def*, *erule conjE*)
apply (*induct-tac m*, *simp add*:*refl-def*, *rename-tac x*)
apply (*rule impI*)
apply (*case-tac n*$\le$*x*, *simp*)
apply (*drule-tac x*=*x* **in** *bspec*, *assumption*)
apply (*unfold trans-on-def*)
apply (*drule-tac x*=*f`succ*(*x*) **in** *bspec*, *simp*)
apply (*drule-tac x*=*f`x* **in** *bspec*, *simp*)
apply (*drule-tac x*=*f`n* **in** *bspec*, *auto*)
apply (*drule-tac le-succ-iff* [*THEN iffD1*], *simp add*: *refl-def*)
done
lemma *not-le-imp-lt*: $[\![\sim i \le j\ ;\ Ord(i);\ \ Ord(j)\ ]\!] \Longrightarrow\ j{<}i$
by (*simp add*:*not-le-iff-lt*)

lemma *decr-seq-linear*: $refl(P,leq) \Longrightarrow f \in nat \to P \Longrightarrow$
$\forall\, n{\in}nat.\ \langle f`succ(n), f`n\rangle \in leq \Longrightarrow$
$trans[P](leq) \Longrightarrow linear(f``nat, leq)$
apply (*unfold linear-def*)
apply (*rule ball-image-simp* [*THEN iffD2*], *assumption*, *simp*, *rule ballI*)+
apply (*rename-tac y*)
apply (*case-tac x*$\le$*y*)
apply (*drule-tac leq*=*leq* **and** *n*=*x* **and** *m*=*y* **in** *decr-succ-decr*)

apply (*simp add*:*preorder-on-def*)

```
    apply (simp+)
  apply (drule not-le-imp-lt [THEN leI], simp-all)
   apply (drule-tac leq=leq and n=y and m=x in decr-succ-decr)

     apply (simp add:preorder-on-def)

    apply (simp+)
   done


locale countable-generic = forcing-notion +
  fixes 𝒟
  assumes countable-subs-of-P:  𝒟 ∈ nat→Pow(P)
  and     seq-of-denses:        ∀ n ∈ nat. dense(𝒟‘n)

begin

definition
  D-generic :: i⇒o where
  D-generic(G) == filter(G) ∧ (∀ n∈nat.(𝒟‘n)∩G≠0)



lemma RS-relation:
  assumes
       1:  x∈P
           and
       2:  n∈nat
  shows
          ∃ y∈P. ⟨x,y⟩ ∈ (λm∈nat. {⟨x,y⟩∈P*P. ⟨y,x⟩∈leq ∧ y∈𝒟‘(pred(m))})‘n
proof −
  from seq-of-denses and 2 have dense(𝒟 ‘ pred(n)) by (simp)
  with 1 have
          ∃ d∈𝒟 ‘ Arith.pred(n). ⟨d, x⟩ ∈ leq
    unfolding dense-def by (simp)
  then obtain d where
       3:  d ∈ 𝒟 ‘ Arith.pred(n) ∧ ⟨d, x⟩ ∈ leq
    by (rule bexE, simp)
  from countable-subs-of-P have
          𝒟 ‘ Arith.pred(n) ∈ Pow(P)
    using 2 by (blast dest:apply-funtype intro:pred-type)
  then have
          𝒟 ‘ Arith.pred(n) ⊆ P
    by (rule PowD)
  then have
          d ∈ P ∧ ⟨d, x⟩ ∈ leq ∧ d ∈ 𝒟 ‘ Arith.pred(n)
    using 3 by auto
  then show ?thesis using 1 and 2 by auto
qed
```

theorem *rasiowa-sikorski*:
$p{\in}P \Longrightarrow \exists\, G.\ p{\in}G \wedge D\text{-}generic(G)$
proof −
assume
Eq1: $p{\in}P$
let
$?S{=}(\lambda m{\in}nat.\ \{\langle x,y\rangle{\in}P{*}P.\ \langle y,x\rangle{\in}leq \wedge\ y{\in}\mathcal{D}`(pred(m))\})$
from *RS-relation* **have**
$\forall\, x{\in}P.\ \forall\, n{\in}nat.\ \exists\, y{\in}P.\ \langle x,y\rangle \in\ ?S`n$
by (*auto*)
with *sequence-DC* **have**
$\forall\, a{\in}P.\ (\exists f \in nat{\to}P.\ f`0 = a \wedge (\forall\, n \in nat.\ \langle f`n,f`succ(n)\rangle{\in}?S`succ(n)))$
by (*blast*)
then obtain f **where**
Eq2: $f : nat{\to}P$
and
Eq3: $f\ `\ 0 = p \wedge$
$(\forall\, n{\in}nat.$
$f\ `\ n \in P \wedge f\ `\ succ(n) \in P \wedge \langle f\ `\ succ(n),\ f\ `\ n\rangle \in leq \wedge$
$f\ `\ succ(n) \in \mathcal{D}\ `\ n)$
using *Eq1* **by** (*auto*)
then have
Eq4: $f``nat\ \subseteq P$
by (*simp add*:*subset-fun-image*)
with *leq-preord* **have**
Eq5: $refl(f``nat,\ leq) \wedge trans[P](leq)$
unfolding *preorder-on-def* **by** (*blast intro*:*refl-monot-domain*)
from *Eq3* **have**
$\forall\, n{\in}nat.\ \ \langle f\ `\ succ(n),\ f\ `\ n\rangle \in leq$
by (*simp*)
with *Eq2* **and** *Eq5* **and** *leq-preord* **and** *decr-seq-linear* **have**
Eq6: $linear(f``nat,\ leq)$
unfolding *preorder-on-def* **by** (*blast*)
with *Eq5* **and** *chain-compat* **have**
$(\forall\, p{\in}f``nat.\forall\, q{\in}f``nat.\ compat\text{-}in(f``nat,leq,p,q))$
by (*auto*)
then have
fil: $filter(upclosure(f``nat))$
(**is** $filter(?G)$)
using *closure-compat-filter* **and** *Eq4* **by** *simp*
have
gen: $\forall\, n{\in}nat.\ \mathcal{D}\ `\ n \cap\ ?G \neq\ 0$
proof
fix n
assume
$n{\in}nat$
with *Eq2* **and** *Eq3* **have**
$f`succ(n) \in\ ?G \wedge f`succ(n) \in \mathcal{D}\ `\ n$

```
    using aux-RS1 by simp
  then show
        𝒟 ‘ n ∩ ?G ≠ 0
    by blast
 qed
 from Eq3 and Eq2 have
        p ∈ ?G
  using aux-RS1 by auto
 with gen and fil show ?thesis
  unfolding D-generic-def by auto
qed

end

end

theory Forcing-data imports Forcing-notions begin

lemma lam-codomain: ∀ n∈N. (λx∈N. b(x))‘n ∈ B ⟹ (λx∈N. b(x)) : N→B
 apply (rule fun-weaken-type)
  apply (subgoal-tac (λx∈N. b(x)) : N → {b(x).x∈N}, assumption)
  apply (auto simp add:lam-funtype)
 done

locale forcing-data = forcing-notion +
 fixes M enum
 assumes M-countable:     enum∈bij(nat,M)
    and P-in-M:          P ∈ M
    and leq-in-M:        leq ∈ M
    and trans-M:         Transset(M)

begin
definition
 M-generic :: i⇒o where
 M-generic(G) == filter(G) ∧ (∀ D∈M. D⊆P ∧ dense(D)⟶D∩G≠0)

declare iff-trans [trans]

lemma generic-filter-existence:
 p∈P ⟹ ∃ G. p∈G ∧ M-generic(G)
proof −
 assume
       Eq1: p∈P
 let
           ?D=λn∈nat. (if (enum‘n⊆P ∧ dense(enum‘n)) then enum‘n else P)
 have
       Eq2: ∀ n∈nat. ?D‘n ∈ Pow(P)
   by auto
 then have
```

Eq3: *?D*:*nat*→*Pow*(*P*)
by (*rule lam-codomain*)
have
Eq4: ∀ *n*∈*nat*. *dense*(*?D*‘*n*)
proof
show
dense(*?D*‘*n*)
if *Eq5*: *n*∈*nat* **for** *n*
proof −
have
dense(*?D*‘*n*)
⟷ *dense*(*if enum* ‘ *n* ⊆ *P* ∧ *dense*(*enum* ‘ *n*) *then enum* ‘ *n else P*)
using *Eq5* **by** *simp*
also have
... ⟷ (¬(*enum* ‘ *n* ⊆ *P* ∧ *dense*(*enum* ‘ *n*)) ⟶ *dense*(*P*))
using *split-if* **by** *simp*
finally show *?thesis*
using *P-dense* **and** *Eq5* **by** *auto*
qed
qed
from *Eq3* **and** *Eq4* **interpret**
cg: *countable-generic P leq one ?D*
by (*unfold-locales*, *auto*)
from *cg.rasiowa-sikorski* **and** *Eq1* **obtain** *G* **where**
Eq6: *p*∈*G* ∧ *filter*(*G*) ∧ (∀ *n*∈*nat*.(*?D*‘*n*)∩*G*≠*0*)
unfolding *cg.D-generic-def* **by** *blast*
then have
Eq7: (∀ *D*∈*M*. *D*⊆*P* ∧ *dense*(*D*)⟶*D*∩*G*≠*0*)
proof (*intro ballI impI*)
show
D ∩ *G* ≠ *0*
if *Eq8*: *D*∈*M* **and**
Eq9: *D* ⊆ *P* ∧ *dense*(*D*) **for** *D*
proof −
from *M-countable* **and** *bij-is-surj* **have**
∀ *y*∈*M*. ∃ *x*∈*nat*. *enum*‘*x*= *y*
unfolding *surj-def* **by** (*simp*)
with *Eq8* **obtain** *n* **where**
Eq10: *n*∈*nat* ∧ *enum*‘*n* = *D*
by *auto*
with *Eq9* **and** *if-P* **have**
Eq11: *?D*‘*n* = *D*
by (*simp*)
with *Eq6* **and** *Eq10* **show**
D∩*G*≠*0*
by *auto*
qed
with *Eq6* **have**
Eq12: ∃ *G*. *filter*(*G*) ∧ (∀ *D*∈*M*. *D*⊆*P* ∧ *dense*(*D*)⟶*D*∩*G*≠*0*)

by *auto*
qed
with *Eq6* **show** *?thesis*
unfolding *M-generic-def* **by** *auto*
qed
end

end

theory *Names* **imports** *Forcing-data* **begin**

lemma *transD* : *Transset*(*M*) $\Longrightarrow$ $y \in M \Longrightarrow y \subseteq M$
by (*unfold Transset-def*, *blast*)

definition
SepReplace :: [*i*, *i*⇒*i*, *i*⇒ *o*] ⇒*i* **where**
SepReplace(*A*,*b*,*Q*) == {*y* . *x*∈*A*, *y*=*b*(*x*) ∧ *Q*(*x*)}

syntax
-*SepReplace* :: [*i*, *pttrn*, *i*, *o*] => *i* ((*1*{- ../ - ∈ -, -}))
translations
{*b* .. *x*∈*A*, *Q*} => *CONST SepReplace*(*A*, λ*x*. *b*, λ*x*. *Q*)

lemma *Sep-and-Replace*: {*b*(*x*) .. *x*∈*A*, *P*(*x*) } = {*b*(*x*) . *x*∈{*y*∈*A*. *P*(*y*)}}
by (*auto simp add*:*SepReplace-def*)

lemma *SepReplace-subset* : *A*⊆*A*′$\Longrightarrow$ {*b* .. *x*∈*A*, *Q*}⊆{*b* .. *x*∈*A*′, *Q*}
by (*auto simp add*:*SepReplace-def*)

lemma *SepReplace-dom-implies* :
∀ *x*∈*A*. *b*(*x*) = *b*′(*x*)$\Longrightarrow$ {*b*(*x*) .. *x*∈*A*, *Q*(*x*)}={*b*′(*x*) .. *x*∈*A*, *Q*(*x*)}
by (*simp add*:*SepReplace-def*)

lemma *SepReplace-pred-implies* :
∀ *x*. *Q*(*x*)⟶ *b*(*x*) = *b*′(*x*)$\Longrightarrow$ {*b*(*x*) .. *x*∈*A*, *Q*(*x*)}={*b*′(*x*) .. *x*∈*A*, *Q*(*x*)}
by (*force simp add*:*SepReplace-def*)

The well founded relation on which *val* is defined

definition
ed :: [*i*,*i*] ⇒ *o* **where**
ed(*x*,*y*) == *x* ∈ *domain*(*y*)

definition
edrel :: *i* ⇒ *i* **where**
edrel(*A*) == {<*x*,*y*> ∈ *A*∗*A* . *x* ∈ *domain*(*y*)}

lemma *edrelI* [*intro*!]: *x*∈*A* $\Longrightarrow$ *y*∈*A* $\Longrightarrow$ *x* ∈ *domain*(*y*) $\Longrightarrow$ <*x*,*y*>∈*edrel*(*A*)

by (*simp add*:*edrel-def*)

lemma *edrelD* [*dest*] : $<x,y>\in edrel(A) \Longrightarrow x \in domain(y)$
by (*simp add*:*edrel-def*)

lemma *edrel-trans*: $Transset(A) \Longrightarrow y\in A \Longrightarrow x \in domain(y) \Longrightarrow <x,y>\in edrel(A)$
by (*rule edrelI*, *auto simp add*:*Transset-def domain-def Pair-def*)

lemma *edrel-trans-iff*: $Transset(A) \Longrightarrow y\in A \Longrightarrow x \in domain(y) \longleftrightarrow <x,y>\in edrel(A)$
by (*auto simp add*: *edrel-trans*, *auto simp add*:*Transset-def Pair-def*)

lemma *edrel-domain*: $x\in M \Longrightarrow edrel(eclose(M)) -`` \{x\} = domain(x)$
apply (*rule equalityI*, *auto* , *subgoal-tac* $Transset(eclose(M))$, *rule vimageI*)
apply (*auto simp add*: *edrelI Transset-eclose*)
apply (*rename-tac y z*)
apply (*rule-tac* $A=\{y\}$ **in** *ecloseD*)
apply (*rule-tac* $A=\langle y, z\rangle$ **in** *ecloseD*)
apply (*rule-tac* $A=x$ **in** *ecloseD*)
apply (*tactic* ⟪ *distinct-subgoals-tac* ⟫)
apply (*auto simp add*: *Pair-def arg-into-eclose*)
done

lemma *domain-trans*: $Transset(A) \Longrightarrow y\in A \Longrightarrow x \in domain(y) \Longrightarrow x\in A$
by (*auto simp add*: *Transset-def domain-def Pair-def*)

lemma *edrel-sub-memrel*: $edrel(A) \subseteq trancl(Memrel(eclose(A)))$
proof
let
$?r=trancl(Memrel(eclose(A)))$
fix *z*
assume
$z\in edrel(A)$
then obtain *x y* **where**
$x\in A\ y\in A\ z=<x,y>\ x\in domain(y)$
unfolding *edrel-def* **by** (*auto*)
then obtain *u v* **where**
Eq1: $x\in u\ u\in v\ v\in y$
unfolding *domain-def Pair-def* **by** *auto*
with ‹$x\in A$› ‹$y\in A$› **have**
$x\in eclose(A)\ y\in eclose(A)$
using *arg-into-eclose* **by** *auto*
moreover with ‹$v\in y$› **have**
$v\in eclose(A)$
using *ecloseD* **by** *simp*
moreover with ‹$u\in v$› **have**
$u\in eclose(A)$
using *ecloseD arg-into-eclose* **by** *simp*
ultimately have
$<x,u>\in ?r\ <u,v>\in ?r\ <v,y>\in ?r$

using *Eq1 r-into-trancl* **by** *auto*
then have
$<x,y>\in ?r$
by (*rule-tac trancl-trans*; *simp*)+
with ‹$z=<x,y>$› **show**
$z\in ?r$ **by** *simp*
qed

lemma *wf-edrel* : *wf*(*edrel*(*A*))
apply (*rule-tac wf-subset* [*of trancl*(*Memrel*(*eclose*(*A*)))])
apply (*auto simp add*:*edrel-sub-memrel wf-trancl wf-Memrel*)
done

lemma *eq-sub-trans* : $x=y \Longrightarrow y\subseteq z \Longrightarrow x\subseteq z$
$x\subseteq y \Longrightarrow y=z \Longrightarrow x\subseteq z$
by *simp-all*

lemma *SepReplace-iff* [*simp*]: $y\in\{b(x)\ ..\ x\in A,\ P(x)\} \longleftrightarrow (\exists\, x\in A.\ y=b(x)\ \&\ P(x))$
by (*auto simp add*:*SepReplace-def*)

context *forcing-data*
begin

definition
Hcheck :: [*i*,*i*] ⇒ *i* **where**
Hcheck(*z*,*f*) == { <*f*‘*y*,*one*> . *y* ∈ *z*}

definition
check :: *i* ⇒ *i* **where**
check(*x*) == *wfrec*(*Memrel*(*eclose*({*x*})), *x* , *Hcheck*)

lemma *aux-def-check*:
(λ*x*∈*y*. *wfrec*(*Memrel*(*eclose*({*y*})), *x*, *Hcheck*)) =
(λ*x*∈*y*. *wfrec*(*Memrel*(*eclose*({*x*})), *x*, *Hcheck*))
apply (*rule lam-cong*)
defer *1* **apply** (*rule wfrec-eclose-eq*)
defer *1* **apply** (*rule ecloseD*, *auto simp add*: *arg-into-eclose*)
done

lemma *def-check* : *check*(*y*) = { <*check*(*w*),*one*> . *w* ∈ *y*}
proof −
let
$?r=\lambda y.\ Memrel(eclose(\{y\}))$
from *wf-Memrel* **have**
wfr: ∀ *w* . *wf*(*?r*(*w*)) **..**
with *wfrec* [*of ?r*(*y*) *y Hcheck*] **have**

check(*y*)=
Hcheck(*y*, λ*x*∈*?r*(*y*) −‘‘ {*y*}. *wfrec*(*?r*(*y*), *x*, *Hcheck*))
by (*simp add*:*check-def*)
also have
... = *Hcheck*(*y*, λ*x*∈*y*. *wfrec*(*?r*(*y*), *x*, *Hcheck*))
by (*simp add*:*under-Memrel-eclose arg-into-eclose*)
also have
... = *Hcheck*(*y*, λ*x*∈*y*. *check*(*x*))
using *aux-def-check* **by** (*simp add*:*check-def*)
finally show *?thesis* **by** (*simp add*:*Hcheck-def*)
qed

definition
Hv :: *i*⇒*i*⇒*i*⇒*i* **where**
Hv(*G*,*x*,*f*) == { *f*‘*y* .. *y*∈ *domain*(*x*), ∃ *p*∈*P*. <*y*,*p*> ∈ *x* ∧ *p* ∈ *G* }

definition
val :: *i*⇒*i*⇒*i* **where**
val(*G*,τ) == *wfrec*(*edrel*(*eclose*(*M*)), τ ,*Hv*(*G*))

definition
GenExt :: *i*⇒*i* (*M*[-])
where *GenExt*(*G*)== {*val*(*G*,τ). τ ∈ *M*}

lemma *def-val*: *x*∈*M* ⟹ *val*(*G*,*x*) = {*val*(*G*,*t*) .. *t*∈*domain*(*x*) , ∃ *p*∈*P* . ⟨*t*, *p*⟩∈*x* ∧ *p* ∈ *G* }
proof −
assume
asm: *x*∈*M*
let
?r=*edrel*(*eclose*(*M*))
let
?f=λ*z*∈*?r*−‘‘{*x*}. *wfrec*(*?r*,*z*,*Hv*(*G*))
have
∀ τ. *wf*(*?r*)
by (*simp add*: *wf-edrel*)
with *wfrec* [*of ?r x Hv*(*G*)] **have**
val(*G*,*x*) = *Hv*(*G*,*x*,*?f*)
by (*simp add*:*val-def*)
also have
... = *Hv*(*G*,*x*,λ*z*∈*domain*(*x*). *wfrec*(*?r*,*z*,*Hv*(*G*)))
using *asm* **and** *edrel-domain* **by** (*simp*)
also have
... = *Hv*(*G*,*x*,λ*z*∈*domain*(*x*). *val*(*G*,*z*))
by (*simp add*:*val-def*)
finally show *?thesis* **by** (*simp add*:*Hv-def SepReplace-def*)
qed

lemma *elem-of-val*: ⟦ *x*∈*val*(*G*,π) ; π ∈ *M* ⟧ ⟹ ∃ ϑ∈*domain*(π). *val*(*G*,ϑ) = *x*

by (*subst* (*asm*) *def-val*,*auto*)

lemma *elem-of-val-pair*: ⟦ $x \in val(G,\pi)$; $\pi \in M$ ⟧ ⟹ $\exists \vartheta.\ \exists p \in G.\ \ <\vartheta,p> \in \pi \wedge val(G,\vartheta) = x$
by (*subst* (*asm*) *def-val*,*auto*)

lemma *GenExtD*:
$x \in M[G] \Longrightarrow \exists \tau \in M.\ x = val(G,\tau)$
by (*simp add*:*GenExt-def*)

lemma *GenExtI*:
$x \in M \Longrightarrow val(G,x) \in M[G]$
by (*auto simp add*: *GenExt-def*)

lemma *val-of-name* :
$\{x \in A \times P.\ Q(x)\} \in M \Longrightarrow$
$val(G,\{x \in A \times P.\ Q(x)\}) = \{val(G,t)\ ..\ t \in A\ ,\ \exists p \in P\ .\ \ Q(<t,p>) \wedge p \in G\ \}$
proof −
let
$?n = \{x \in A \times P.\ Q(x)\}$ **and**
$?r = edrel(eclose(M))$
assume
asm: $?n \in M$
let
$?f = \lambda z \in ?r - `` \{?n\}.\ val(G,z)$
have
$\forall \tau.\ wf(?r)$
by (*simp add*: *wf-edrel*)
with *val-def* **have**
$val(G,?n) = Hv(G,?n,?f)$
by (*rule-tac def-wfrec* [*of* - *?r Hv*(*G*)], *simp-all*)
also have
$... = \{?f`t\ ..\ t \in domain(?n),\ \exists p \in P\ .\ <t,p> \in ?n \wedge p \in G\}$
unfolding *Hv-def* **by** *simp*
also have
$... = \{\ (if\ t \in ?r - `` \{?n\}\ then\ val(G,t)\ else\ 0)\ ..\ t \in domain(?n),\ \exists p \in P\ .\ <t,p> \in ?n \wedge p \in G\}$
by (*simp*)
also have
Eq1: $... = \{\ val(G,t)\ ..\ t \in domain(?n),\ \exists p \in P\ .\ <t,p> \in ?n \wedge p \in G\}$
proof −
from *edrel-domain* **and** *asm* **have**
$domain(?n) \subseteq ?r - `` \{?n\}$
by *simp*
then have
$\forall t \in domain(?n).\ (if\ t \in ?r - `` \{?n\}\ then\ val(G,t)\ else\ 0) = val(G,t)$
by *auto*
then show
$\{\ (if\ t \in ?r - `` \{?n\}\ then\ val(G,t)\ else\ 0)\ ..\ t \in domain(?n),\ \exists p \in P\ .$

$<t,p> \in ?n \wedge p \in G\} =$
$\{\ val(G,t)\ ..\ t \in domain(?n), \exists\, p \in P\ .\ <t,p> \in ?n \wedge p \in G\}$
by *auto*
qed
also have
$... = \{\ val(G,t)\ ..\ t \in A, \exists\, p \in P\ .\ <t,p> \in ?n \wedge p \in G\}$
by *force*
finally show
$val(G,?n)\ = \{\ val(G,t)\ ..\ t \in A, \exists\, p \in P\ .\ Q(<t,p>) \wedge p \in G\}$
by *auto*
qed

lemma *valcheck* : $y \in M \Longrightarrow one \in G \Longrightarrow \forall\, x \in M.\ check(x) \in M \Longrightarrow val(G,check(y)) = y$
proof (*induct rule*:*eps-induct*)
case (*1 y*)
then show *?case*
proof −
from *def-check* **have**
Eq1: $check(y) = \{\ \langle check(w),\ one\rangle\ .\ w \in y\}$ (**is** - = *?C*) **.**
with *1* **have**
Eq2: $?C \in M$
by *auto*
with *1 transD subsetD trans-M* **have**
w-in-M : $\forall\ w \in y\ .\ w \in M$ **by** *force*
from *Eq1* **have**
$val(G,check(y)) = val(G, \{\langle check(w),\ one\rangle\ .\ w \in y\})$
by *simp*
also have
$... = \{val(G,t)\ ..\ t \in domain(?C)\ , \exists\, p \in P\ .\ \langle t,\ p\rangle \in ?C \wedge p \in G\ \}$
using *def-val* **and** *Eq2* **by** *simp*
also have
$... = \ \{val(G,t)\ ..\ t \in domain(?C)\ , \exists\, w \in y.\ t = check(w)\ \}$
using *1* **and** *one-in-P* **by** *simp*
also have
$... = \{val(G,check(w))\ .\ w \in y\ \}$
by *force*
also have
$... = y$
using *1* **and** *w-in-M* **by** *simp*
finally show $val(G,check(y)) = y$
using *1* **by** *simp*
qed
qed

lemma *trans-Gen-Ext'* :
assumes $vc \in M[G]$
$y \in vc$

shows
$y \in M[G]$
proof −
from ‹$vc \in M[G]$› **have**
$\exists\, c \in M.\ vc = val(G,c)$
by (*blast dest*:*GenExtD*)
then obtain c **where**
$c \in M\ vc = val(G,c)$ **by** *auto*
with ‹$vc \in M[G]$› **have**
$val(G,c) \in M[G]$ **by** *simp*
from ‹$y \in vc$› **and** ‹$vc = val(G,c)$› **have**
$y \in val(G,c)$ **by** *simp*
with ‹$c \in M$› **and** *elem-of-val* **have**
$\exists\, \vartheta \in domain(c).\ val(G,\vartheta) = y$ **by** *simp*
then obtain ϑ **where**
$\vartheta \in domain(c)\ val(G,\vartheta) = y$ **by** *auto*
from ‹$\vartheta \in domain(c)$› *trans-M* ‹$c \in M$› *domain-trans* **have**
$\vartheta \in M$ **by** *simp*
then have
$val(G,\vartheta) \in M[G]$
by (*blast intro*:*GenExtI*)
with ‹$val(G,\vartheta) = y$› **show** *?thesis* **by** *simp*
qed

lemma *trans-Gen-Ext*:
$Transset(M[G])$
by (*auto simp add*: *Transset-def trans-Gen-Ext'*)

lemma *val-of-elem*:
assumes
$<\vartheta,p> \in \pi\ \pi \in M\ p \in G\ p \in P$
shows
$val(G,\vartheta) \in val(G,\pi)$
proof −
from ‹$\pi \in M$› **have**
1:$val(G,\pi) = \{val(G,t)\ ..\ t \in domain(\pi)\ ,\ \exists\, p \in P\ .\ \langle t,\ p\rangle \in \pi \wedge p \in G\ \}$
using *def-val* **by** *simp*
from ‹$<\vartheta,p> \in \pi$› **have** $\vartheta \in domain(\pi)$ **by** *auto*
with ‹$p \in G$› ‹$p \in P$› ‹$\vartheta \in domain(\pi)$› ‹$<\vartheta,p> \in \pi$› **have**
$val(G,\vartheta) \in \{val(G,t)\ ..\ t \in domain(\pi)\ ,\ \exists\, p \in P\ .\ \langle t,\ p\rangle \in \pi \wedge p \in G\ \}$
by *auto*
with *1* **show** *?thesis* **by** *simp*
qed

end

definition
$upair :: [i=>o,i,i,i] => o$ **where**

$upair(M,a,b,z) == a \in z \ \&\ b \in z \ \&\ (\forall\, x[M].\ x{\in}z \longrightarrow x = a \mid x = b)$

definition
$upair\text{-}ax :: (i{=>}o) => o$ **where**
$upair\text{-}ax(M) == \forall\, x[M].\ \forall\, y[M].\ \exists\, z[M].\ upair(M,x,y,z)$

definition
$univalent :: [i{=>}o,\ i,\ [i,i]{=>}o] => o$ **where**
$univalent(M,A,P) ==$
$\forall\, x[M].\ x{\in}A \longrightarrow (\forall\, y[M].\ \forall\, z[M].\ P(x,y)\ \&\ P(x,z) \longrightarrow y{=}z)$

definition
$strong\text{-}replacement :: [i{=>}o,\ [i,i]{=>}o] => o$ **where**
$strong\text{-}replacement(M,P) ==$
$\forall\, A[M].\ univalent(M,A,P) \longrightarrow$
$(\exists\, Y[M].\ \forall\, b[M].\ b \in Y \longleftrightarrow (\exists\, x[M].\ x{\in}A\ \&\ P(x,b)))$

lemma *univalent-triv* [*intro*,*simp*]:
$univalent(M,\ A,\ \lambda x\ y.\ y = f(x))$
by (*simp add*: *univalent-def*)

locale *M-extra-assms* = *forcing-data* +
assumes
$check\text{-}in\text{-}M : \bigwedge x.\ x \in M \Longrightarrow check(x) \in M$
and *upair-ax*: $upair\text{-}ax(\#\#M)$
and *repl-check-pair* : $strong\text{-}replacement(\#\#M,\lambda p\ y.\ y = {<}check(p),p{>})$

begin

lemma *Transset-intf* :
$Transset(M) \Longrightarrow\ y{\in}x \Longrightarrow x \in M \Longrightarrow y \in M$
by (*simp add*: *Transset-def*,*auto*)

lemma *upair-abs* [*simp*]:
$z{\in}M ==> upair(\#\#M,a,b,z) \longleftrightarrow z{=}\{a,b\}$
apply (*simp add*: *upair-def*)
apply (*insert trans-M*)
apply (*blast intro*: *Transset-intf*)
done

lemma *upairM* : $x \in M \Longrightarrow y \in M \Longrightarrow \{x,y\} \in M$
by(*insert upair-ax*, *auto simp add*: *upair-ax-def*)

lemma *pairM* : $x \in\ M \Longrightarrow y \in M \Longrightarrow {<}x,y{>} \in M$
by(*unfold Pair-def*, *rule upairM*,(*rule upairM*,*simp*+)+)

lemma *univalent-Replace-iff*:
[| $A{\in}M$; *univalent*($\#\#M$,A,Q);
!!x y. [| $x{\in}A$; $Q(x,y)$ |] ==> $y{\in}M$ |]
==> $u \in$ *Replace*(A,Q) $\longleftrightarrow$ ($\exists\, x.\ x{\in}A\ \&\ Q(x,u)$)
apply (*simp add*: *Replace-iff univalent-def*)
apply (*insert trans-M*)
apply (*blast dest*: *Transset-intf*)
done

lemma *strong-replacement-closed* [*intro*,*simp*]:
[| *strong-replacement*($\#\#M$,Q); $A{\in}M$; *univalent*($\#\#M$,A,Q);
!!x y. [| $x{\in}A$; $Q(x,y)$ |] ==> $y{\in}M$ |] ==> (*Replace*(A,Q)$\in M$)
apply (*simp add*: *strong-replacement-def*)
apply (*drule-tac* $x{=}A$ **in** *bspec*, *safe*)
apply (*subgoal-tac* *Replace*(A,Q) = Y)
apply *simp*
apply (*rule equality-iffI*)
apply (*simp add*: *univalent-Replace-iff*)
apply (*insert trans-M*)
apply (*blast dest*: *Transset-intf*)
done

lemma *P-sub-M* : $P \subseteq M$
by (*simp add*: *P-in-M trans-M transD*)

definition
G-dot :: i **where**
G-dot == {<*check*(p),p> . $p{\in}P$}

lemma *G-dot-in-M* :
G-dot $\in M$
proof −
have *0*:*G-dot* = { y . $p{\in}P$, y = <*check*(p),p> }
unfolding *G-dot-def* **by** *auto*
from *P-in-M check-in-M pairM P-sub-M* **have** $\bigwedge$ p . $p{\in}P \Longrightarrow$ <*check*(p),p> $\in$ M
by *auto*
then have
1:$\bigwedge x$ y. $[\![$ $x{\in}P$; y = <*check*(x),x> $]\!] \Longrightarrow$ $y{\in}M$
by *simp*
then have
$\forall\, A{\in}M.(\exists\, Y{\in}M.\ \forall\, b{\in}M.\ b \in Y \longleftrightarrow (\exists\, x{\in}M.\ x{\in}A\ \&\ b = <check(x),x>))$
using *repl-check-pair* **unfolding** *strong-replacement-def* **by** *simp*
then have
$(\exists\, Y{\in}M.\ \forall\, b{\in}M.\ b \in Y \longleftrightarrow (\exists\, x{\in}M.\ x{\in}P\ \&\ b = <check(x),x>))$
using *P-in-M* **by** *simp*
with *1 repl-check-pair P-in-M strong-replacement-closed* **have**

```
    { y . p∈P , y = <check(p),p> } ∈ M by simp
  then show ?thesis using 0 by simp
qed

lemma val-G-dot :
  assumes G ⊆ P
    one ∈ G
  shows val(G,G-dot) = G
proof (intro equalityI subsetI)
  fix x
  assume x∈val(G,G-dot)
  then have
      ∃ϑ. ∃ p∈G.  <ϑ,p>∈G-dot ∧ val(G,ϑ) = x
    using G-dot-in-M elem-of-val-pair by simp
  then obtain r p where
    p∈G <r,p> ∈ G-dot val(G,r) = x
    by auto
  then have
    r = check(p)
    unfolding G-dot-def by simp
  with ‹one∈G› ‹G⊆P› ‹p∈G› ‹val(G,r) = x› show
      x ∈ G
    using valcheck P-sub-M  check-in-M by auto
next
  fix p
  assume p∈G
  have ∀ q∈P. <check(q),q> ∈ G-dot
    unfolding G-dot-def by simp
  with ‹p∈G› ‹G⊆P› have
    val(G,check(p)) ∈ val(G,G-dot)
    using val-of-elem G-dot-in-M by blast
  with ‹p∈G› ‹G⊆P› ‹one∈G› show
    p ∈ val(G,G-dot)
    using P-sub-M check-in-M valcheck by auto
qed


lemma G-in-Gen-Ext :
  assumes G ⊆ P
    one ∈ G
  shows   G ∈ M[G]
proof −
  from G-dot-in-M have
    val(G,G-dot) ∈ M[G]
    by (auto intro:GenExtI)
  with assms val-G-dot
  show ?thesis by simp
qed
```

end

end

theory *Gen-ext-pair* **imports** *Names Forcing-data* **begin**

context *M-extra-assms*
begin

lemma *one-in-M* : $one \in M$
  by (*insert trans-M*,*insert one-in-P*,*insert P-in-M*,*rule Transset-intf*)

lemma *pairs-in-M* :
  $[\![\ a \in M\ ;\ b \in M\ ;\ c \in M\ ;\ d \in M\ ]\!] \Longrightarrow \{\langle a,c\rangle,\langle b,d\rangle\} \in M$
  by (*unfold Pair-def*, ((*rule upairM*)+,*assumption*+)+)

lemma *sigma-in-M* :
  $one \in G \Longrightarrow \tau \in M \Longrightarrow \varrho \in M \Longrightarrow \{\langle \tau,one\rangle,\langle \varrho,one\rangle\} \in M$
  by (*rule pairs-in-M*,*simp-all add*: *upair-ax-def one-in-M*)

lemma *valsigma* :
  $one \in G \Longrightarrow \{\langle \tau,one\rangle,\langle \varrho,one\rangle\} \in M \Longrightarrow$
   $val(G,\{\langle \tau,one\rangle,\langle \varrho,one\rangle\}) = \{val(G,\tau),val(G,\varrho)\}$
  apply (*insert one-in-P*)
  apply (*rule trans*)
   apply (*rule def-val*,*assumption*)
  apply (*auto simp add*: *Sep-and-Replace*)
done

lemma *pairing-axiom* :
  $one \in G \Longrightarrow upair\text{-}ax(\#\#M[G])$
  apply (*simp add*: *upair-ax-def*)
  apply (*rule ballI*)+
  apply (*drule GenExtD*)+
  apply (*rule bexE*,*assumption*)
  apply (*rule-tac* $A{=}M$ **and** $P{=}\lambda w.\ y{=}val(G,w)$ **in** *bexE*,*assumption*)
  apply (*rename-tac x y* τ ϱ)
  apply (*rule-tac* $x{=}val(G,\{\langle \tau,one\rangle,\langle \varrho,one\rangle\})$ **in** *bexI*)
   apply (*subst valsigma*,*assumption*+)
   defer *1* **apply** (*simp add*: *upair-def*)
   apply (*rule GenExtI*)
   apply (*insert sigma-in-M*,*simp-all add*: *upair-ax-def*)
done

end
end