

Equivalents of the Axiom of Choice

Krzysztof Grąbczewski

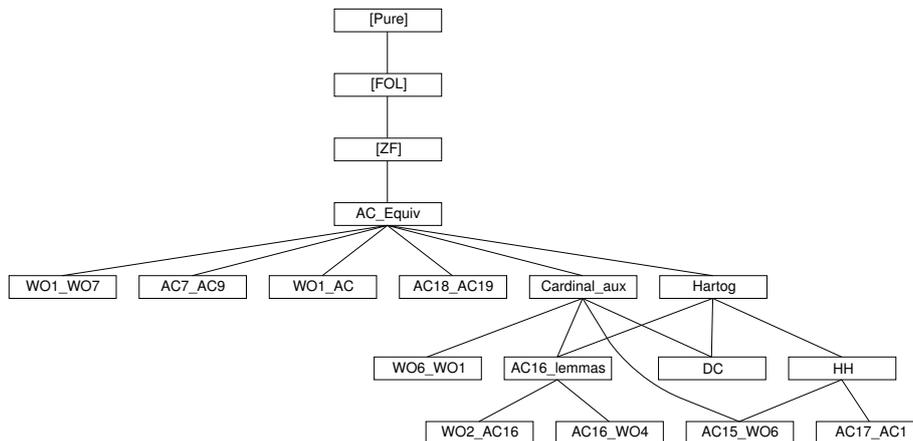
October 1, 2005

Abstract

This development [1] proves the equivalence of seven formulations of the well-ordering theorem and twenty formulations of the axiom of choice. It formalizes the first two chapters of the monograph *Equivalents of the Axiom of Choice* by Rubin and Rubin [2]. Some of this material involves extremely complex techniques.

Contents

0.1	Lemmas useful in each of the three proofs	31
0.2	Lemmas used in the proofs of $AC1 \iff WO2$ and $AC17 \iff AC1$	33
0.3	The proof of $AC1 \iff WO2$	35



```

theory AC_Equiv imports Main begin

constdefs

W01 :: o
  "W01 ==  $\forall A. \exists R. \text{well\_ord}(A,R)$ "

W02 :: o
  "W02 ==  $\forall A. \exists a. \text{Ord}(a) \ \& \ A \approx a$ "

W03 :: o
  "W03 ==  $\forall A. \exists a. \text{Ord}(a) \ \& \ (\exists b. b \subseteq a \ \& \ A \approx b)$ "

W04 :: "i => o"
  "W04(m) ==  $\forall A. \exists a f. \text{Ord}(a) \ \& \ \text{domain}(f)=a \ \& \$ 
     $(\bigcup b < a. f' b) = A \ \& \ (\forall b < a. f' b \lesssim m)$ "

W05 :: o
  "W05 ==  $\exists m \in \text{nat}. 1 \leq m \ \& \ W04(m)$ "

W06 :: o
  "W06 ==  $\forall A. \exists m \in \text{nat}. 1 \leq m \ \& \ (\exists a f. \text{Ord}(a) \ \& \ \text{domain}(f)=a \ \& \$ 
     $(\bigcup b < a. f' b) = A \ \& \ (\forall b < a. f' b \lesssim m))$ "

W07 :: o
  "W07 ==  $\forall A. \text{Finite}(A) \ \leftrightarrow \ (\forall R. \text{well\_ord}(A,R) \ \rightarrow \ \text{well\_ord}(A, \text{converse}(R)))$ "

W08 :: o
  "W08 ==  $\forall A. (\exists f. f \in (\prod X \in A. X)) \ \rightarrow \ (\exists R. \text{well\_ord}(A,R))$ "

pairwise_disjoint :: "i => o"
  "pairwise_disjoint(A) ==  $\forall A1 \in A. \forall A2 \in A. A1 \text{ Int } A2 \neq 0 \ \rightarrow \ A1=A2$ "

sets_of_size_between :: "[i, i, i] => o"
  "sets_of_size_between(A,m,n) ==  $\forall B \in A. m \lesssim B \ \& \ B \lesssim n$ "

AC0 :: o
  "AC0 ==  $\forall A. \exists f. f \in (\prod X \in \text{Pow}(A) - \{0\}. X)$ "

AC1 :: o
  "AC1 ==  $\forall A. 0 \notin A \ \rightarrow \ (\exists f. f \in (\prod X \in A. X))$ "

```

```

AC2 :: o
  "AC2 ==  $\forall A. 0 \notin A \ \& \ \text{pairwise\_disjoint}(A)$ 
      -->  $(\exists C. \forall B \in A. \exists y. B \text{ Int } C = \{y\})$ "
AC3 :: o
  "AC3 ==  $\forall A \ B. \forall f \in A \rightarrow B. \exists g. g \in (\prod x \in \{a \in A. f'a \neq 0\}. f'x)$ "
AC4 :: o
  "AC4 ==  $\forall R \ A \ B. (R \subseteq A * B \rightarrow (\exists f. f \in (\prod x \in \text{domain}(R). R'\{x\})))$ "
AC5 :: o
  "AC5 ==  $\forall A \ B. \forall f \in A \rightarrow B. \exists g \in \text{range}(f) \rightarrow A. \forall x \in \text{domain}(g). f'(g'x) = x$ "
AC6 :: o
  "AC6 ==  $\forall A. 0 \notin A \rightarrow (\prod B \in A. B) \neq 0$ "
AC7 :: o
  "AC7 ==  $\forall A. 0 \notin A \ \& \ (\forall B1 \in A. \forall B2 \in A. B1 \approx B2) \rightarrow (\prod B \in A. B) \neq 0$ "
AC8 :: o
  "AC8 ==  $\forall A. (\forall B \in A. \exists B1 \ B2. B = \langle B1, B2 \rangle \ \& \ B1 \approx B2)$ 
      -->  $(\exists f. \forall B \in A. f'B \in \text{bij}(\text{fst}(B), \text{snd}(B)))$ "
AC9 :: o
  "AC9 ==  $\forall A. (\forall B1 \in A. \forall B2 \in A. B1 \approx B2) \rightarrow$ 
       $(\exists f. \forall B1 \in A. \forall B2 \in A. f'\langle B1, B2 \rangle \in \text{bij}(B1, B2))$ "
AC10 :: "i => o"
  "AC10(n) ==  $\forall A. (\forall B \in A. \sim \text{Finite}(B)) \rightarrow$ 
       $(\exists f. \forall B \in A. (\text{pairwise\_disjoint}(f'B) \ \& \ \text{sets\_of\_size\_between}(f'B, 2, \text{succ}(n)) \ \& \ \text{Union}(f'B) = B))$ "
AC11 :: o
  "AC11 ==  $\exists n \in \text{nat}. 1 \leq n \ \& \ \text{AC10}(n)$ "
AC12 :: o
  "AC12 ==  $\forall A. (\forall B \in A. \sim \text{Finite}(B)) \rightarrow$ 
       $(\exists n \in \text{nat}. 1 \leq n \ \& \ (\exists f. \forall B \in A. (\text{pairwise\_disjoint}(f'B)$ 
&
       $\text{sets\_of\_size\_between}(f'B, 2, \text{succ}(n)) \ \& \ \text{Union}(f'B) = B)))$ "
AC13 :: "i => o"
  "AC13(m) ==  $\forall A. 0 \notin A \rightarrow (\exists f. \forall B \in A. f'B \neq 0 \ \& \ f'B \subseteq B \ \& \ f'B \lesssim m)$ "
AC14 :: o
  "AC14 ==  $\exists m \in \text{nat}. 1 \leq m \ \& \ \text{AC13}(m)$ "

```

```

AC15 :: o
"AC15 ==  $\forall A. 0 \notin A \rightarrow$ 
  ( $\exists m \in \text{nat}. 1 \leq m \ \& \ (\exists f. \forall B \in A. f'B \neq 0 \ \& \ f'B \subseteq B \ \& \ f'B \lesssim m)$ )"

AC16 :: "[i, i] => o"
"AC16(n, k) ==
   $\forall A. \sim \text{Finite}(A) \rightarrow$ 
  ( $\exists T. T \subseteq \{X \in \text{Pow}(A). X \approx_{\text{succ}}(n)\}$ ) &
  ( $\forall X \in \{X \in \text{Pow}(A). X \approx_{\text{succ}}(k)\}. \exists! Y. Y \in T \ \& \ X \subseteq Y$ )")

AC17 :: o
"AC17 ==  $\forall A. \forall g \in (\text{Pow}(A) - \{0\} \rightarrow A) \rightarrow \text{Pow}(A) - \{0\}. \exists f \in \text{Pow}(A) - \{0\} \rightarrow A. f'(g'f) \in g'f$ "

locale AC18 =
  assumes AC18: " $A \neq 0 \ \& \ (\forall a \in A. B(a) \neq 0) \rightarrow$ 
    ( $(\bigcap a \in A. \bigcup b \in B(a). X(a,b)) =$ 
      ( $\bigcup f \in \Pi a \in A. B(a). \bigcap a \in A. X(a, f'a)$ ))"
  — AC18 cannot be expressed within the object-logic

constdefs
AC19 :: o
"AC19 ==  $\forall A. A \neq 0 \ \& \ 0 \notin A \rightarrow ((\bigcap a \in A. \bigcup b \in a. b) =$ 
  ( $\bigcup f \in (\Pi B \in A. B). \bigcap a \in A. f'a$ )")

lemma rvimage_id: "rvimage(A, id(A), r) = r Int A*A"
apply (unfold rvimage_def)
apply (rule equalityI, safe)
apply (drule_tac P = "%a. <id (A) 'xb,a>:r" in id_conv [THEN subst],
  assumption)
apply (drule_tac P = "%a. <a,ya>:r" in id_conv [THEN subst], (assumption+))
apply (fast intro: id_conv [THEN ssubst])
done

lemma ordertype_Int:
  "well_ord(A,r) ==> ordertype(A, r Int A*A) = ordertype(A,r)"
apply (rule_tac P = "%a. ordertype (A,a) =ordertype (A,r) " in rvimage_id
  [THEN subst])
apply (erule id_bij [THEN bij_ordertype_vimage])
done

```

```

lemma lam_sing_bij: "( $\lambda x \in A. \{x\}) \in \text{bij}(A, \{\{x\}. x \in A\})"$ 
apply (rule_tac d = "%z. THE x. z={x}" in lam_bijective)
apply (auto simp add: the_equality)
done

lemma inj_strengthen_type:
  "[| f  $\in$  inj(A, B);  $\forall a. a \in A \implies f'a \in C$  |]  $\implies f \in \text{inj}(A,C)"$ 
by (unfold inj_def, blast intro: Pi_type)

lemma nat_not_Finite: "~ Finite(nat)"
by (unfold Finite_def, blast dest: eqpoll_imp_lepoll ltI lt_not_lepoll)

lemmas le_imp_lepoll = le_imp_subset [THEN subset_imp_lepoll]

lemma ex1_two_eq: "[|  $\exists! x. P(x)$ ;  $P(x)$ ;  $P(y)$  |]  $\implies x=y"$ 
by blast

lemma surj_image_eq: "f  $\in$  surj(A, B)  $\implies f'A = B"$ 
apply (unfold surj_def)
apply (erule CollectE)
apply (rule image_fun [THEN ssubst], assumption, rule subset_refl)
apply (blast dest: apply_type)
done

lemma first_in_B:
  "[| well_ord(Union(A),r);  $0 \notin A$ ;  $B \in A$  |]  $\implies$  (THE b. first(b,B,r))
 $\in B"$ 
by (blast dest!: well_ord_imp_ex1_first
    [THEN theI, THEN first_def [THEN def_imp_iff, THEN
    iffD1]])

lemma ex_choice_fun: "[| well_ord(Union(A), R);  $0 \notin A$  |]  $\implies \exists f. f:(\Pi$ 
 $X \in A. X)"$ 
by (fast elim!: first_in_B intro!: lam_type)

```

```

lemma ex_choice_fun_Pow: "well_ord(A, R) ==> ∃f. f:(Π X ∈ Pow(A)-{0}.
X)"
by (fast elim!: well_ord_subset [THEN ex_choice_fun])

```

```

lemma lepoll_m_imp_domain_lepoll_m:
  "[| m ∈ nat; u ≲ m |] ==> domain(u) ≲ m"
apply (unfold lepoll_def)
apply (erule exE)
apply (rule_tac x = "λx ∈ domain(u). LEAST i. ∃y. <x,y> ∈ u & f'<x,y>
= i"
  in exI)
apply (rule_tac d = "%y. fst (converse(f) ' y) " in lam_injective)
apply (fast intro: LeastI2 nat_into_Ord [THEN Ord_in_Ord]
  inj_is_fun [THEN apply_type])
apply (erule domainE)
apply (frule inj_is_fun [THEN apply_type], assumption)
apply (rule LeastI2)
apply (auto elim!: nat_into_Ord [THEN Ord_in_Ord])
done

```

```

lemma rel_domain_ex1:
  "[| succ(m) ≲ domain(r); r ≲ succ(m); m ∈ nat |] ==> function(r)"
apply (unfold function_def, safe)
apply (rule ccontr)
apply (fast elim!: lepoll_trans [THEN succ_lepoll_natE]
  lepoll_m_imp_domain_lepoll_m [OF _ Diff_sing_lepoll]
  elim: domain_Diff_eq [OF _ not_sym, THEN subst])
done

```

```

lemma rel_is_fun:
  "[| succ(m) ≲ domain(r); r ≲ succ(m); m ∈ nat;
  r ⊆ A*B; A=domain(r) |] ==> r ∈ A->B"
by (simp add: Pi_iff rel_domain_ex1)

end

```

```

theory Cardinal_aux imports AC_Equiv begin

```

```

lemma Diff_lepoll: "[| A ≲ succ(m); B ⊆ A; B≠0 |] ==> A-B ≲ m"

```

```

apply (rule not_emptyE, assumption)
apply (blast intro: lepoll_trans [OF subset_imp_lepoll Diff_sing_lepoll])
done

```

```

lemma lepoll_imp_ex_le_eqpoll:
  "[| A  $\lesssim$  i; Ord(i) |] ==>  $\exists j. j \leq i \ \& \ A \approx j$ "
by (blast intro!: lepoll_cardinal_le well_ord_Memrel
    well_ord_cardinal_eqpoll [THEN eqpoll_sym]
    dest: lepoll_well_ord)

```

```

lemma lesspoll_imp_ex_lt_eqpoll:
  "[| A < i; Ord(i) |] ==>  $\exists j. j < i \ \& \ A \approx j$ "
by (unfold lesspoll_def, blast dest!: lepoll_imp_ex_le_eqpoll elim!: leE)

```

```

lemma Inf_Ord_imp_InfCard_cardinal: "[|  $\sim$ Finite(i); Ord(i) |] ==> InfCard(|i|)"
apply (unfold InfCard_def)
apply (rule conjI)
apply (rule Card_cardinal)
apply (rule Card_nat
  [THEN Card_def [THEN def_imp_iff, THEN iffD1, THEN ssubst]])
  — rewriting would loop!
apply (rule well_ord_Memrel [THEN well_ord_lepoll_imp_Card_le], assumption)

apply (rule nat_le_infinite_Ord [THEN le_imp_lepoll], assumption+)
done

```

An alternative and more general proof goes like this: A and B are both well-ordered (because they are injected into an ordinal), either A lepoll B or B lepoll A. Also both are equipollent to their cardinalities, so (if A and B are infinite) then $A \cup B \leq \max(|A|, |B|) \leq \max(|A|, |B|) \leq i$. In fact, the correctly strengthened version of this theorem appears below.

```

lemma Un_lepoll_Inf_Ord_weak:
  "[| A  $\approx$  i; B  $\approx$  i;  $\neg$  Finite(i); Ord(i) |] ==> A  $\cup$  B  $\lesssim$  i"
apply (rule Un_lepoll_sum [THEN lepoll_trans])
apply (rule lepoll_imp_sum_lepoll_prod [THEN lepoll_trans])
apply (erule eqpoll_trans [THEN eqpoll_imp_lepoll])
apply (erule eqpoll_sym)
apply (rule subset_imp_lepoll [THEN lepoll_trans, THEN lepoll_trans])

apply (rule nat_2I [THEN OrdmemD], rule Ord_nat)

```

```

apply (rule nat_le_infinite_Ord [THEN le_imp_lepoll], assumption+)
apply (erule eqpoll_sym [THEN eqpoll_imp_lepoll])
apply (erule prod_eqpoll_cong [THEN eqpoll_imp_lepoll, THEN lepoll_trans],
      assumption)
apply (rule eqpoll_imp_lepoll)
apply (rule well_ord_Memrel [THEN well_ord_InfCard_square_eq], assumption)

apply (rule Inf_Ord_imp_InfCard_cardinal, assumption+)
done

```

```

lemma Un_eqpoll_Inf_Ord:
  "[| A  $\approx$  i; B  $\approx$  i;  $\sim$ Finite(i); Ord(i) |] ==> A Un B  $\approx$  i"
apply (rule eqpollI)
apply (blast intro: Un_lepoll_Inf_Ord_weak)
apply (erule eqpoll_sym [THEN eqpoll_imp_lepoll, THEN lepoll_trans])
apply (rule Un_upper1 [THEN subset_imp_lepoll])
done

```

```

lemma paired_bij: "?f  $\in$  bij({{y,z}. y  $\in$  x}, x)"
apply (rule RepFun_bijective)
apply (simp add: doubleton_eq_iff, blast)
done

```

```

lemma paired_eqpoll: "{{y,z}. y  $\in$  x}  $\approx$  x"
by (unfold eqpoll_def, fast intro!: paired_bij)

```

```

lemma ex_eqpoll_disjoint: " $\exists$ B. B  $\approx$  A & B Int C = 0"
by (fast intro!: paired_eqpoll equalsOI elim: mem_asym)

```

```

lemma Un_lepoll_Inf_Ord:
  "[| A  $\lesssim$  i; B  $\lesssim$  i;  $\sim$ Finite(i); Ord(i) |] ==> A Un B  $\lesssim$  i"
apply (rule_tac A1 = i and C1 = i in ex_eqpoll_disjoint [THEN exE])
apply (erule conjE)
apply (drule lepoll_trans)
apply (erule eqpoll_sym [THEN eqpoll_imp_lepoll])
apply (rule Un_lepoll_Un [THEN lepoll_trans], (assumption+))
apply (blast intro: eqpoll_refl Un_eqpoll_Inf_Ord eqpoll_imp_lepoll)
done

```

```

lemma Least_in_Ord: "[| P(i); i  $\in$  j; Ord(j) |] ==> (LEAST i. P(i))  $\in$ 
j"
apply (erule Least_le [THEN leE])
apply (erule Ord_in_Ord, assumption)
apply (erule ltE)
apply (fast dest: OrdmemD)
apply (erule subst_elem, assumption)
done

```

```

lemma Diff_first_lepoll:
  "[| well_ord(x,r); y ⊆ x; y ≲ succ(n); n ∈ nat |]
  ==> y - {THE b. first(b,y,r)} ≲ n"
apply (case_tac "y=0", simp add: empty_lepollI)
apply (fast intro!: Diff_sing_lepoll the_first_in)
done

lemma UN_subset_split:
  "(⋃ x ∈ X. P(x)) ⊆ (⋃ x ∈ X. P(x)-Q(x)) Un (⋃ x ∈ X. Q(x))"
by blast

lemma UN_sing_lepoll: "Ord(a) ==> (⋃ x ∈ a. {P(x)}) ≲ a"
apply (unfold lepoll_def)
apply (rule_tac x = "λz ∈ (⋃ x ∈ a. {P (x) }) . (LEAST i. P (i) =z)
" in exI)
apply (rule_tac d = "%z. P (z) " in lam_injective)
apply (fast intro!: Least_in_Ord)
apply (fast intro: LeastI elim!: Ord_in_Ord)
done

lemma UN_fun_lepoll_lemma [rule_format]:
  "[| well_ord(T, R); ~Finite(a); Ord(a); n ∈ nat |]
  ==> ∀f. (∀b ∈ a. f`b ≲ n & f`b ⊆ T) --> (⋃ b ∈ a. f`b) ≲ a"
apply (induct_tac "n")
apply (rule allI)
apply (rule impI)
apply (rule_tac b = "⋃ b ∈ a. f`b" in subst)
apply (rule_tac [2] empty_lepollI)
apply (rule equalsOI [symmetric], clarify)
apply (fast dest: lepoll_0_is_0 [THEN subst])
apply (rule allI)
apply (rule impI)
apply (erule_tac x = "λx ∈ a. f`x - {THE b. first (b,f`x,R) }" in allE)
apply (erule impE, simp)
apply (fast intro!: Diff_first_lepoll, simp)
apply (rule UN_subset_split [THEN subset_imp_lepoll, THEN lepoll_trans])
apply (fast intro: Un_lepoll_Inf_Ord UN_sing_lepoll)
done

lemma UN_fun_lepoll:
  "[| ∀b ∈ a. f`b ≲ n & f`b ⊆ T; well_ord(T, R);
  ~Finite(a); Ord(a); n ∈ nat |] ==> (⋃ b ∈ a. f`b) ≲ a"
by (blast intro: UN_fun_lepoll_lemma)

lemma UN_lepoll:
  "[| ∀b ∈ a. F(b) ≲ n & F(b) ⊆ T; well_ord(T, R);
  ~Finite(a); Ord(a); n ∈ nat |]
  ==> (⋃ b ∈ a. F(b)) ≲ a"
apply (rule rev_mp)

```

```

apply (rule_tac f="λb ∈ a. F (b)" in UN_fun_lepoll)
apply auto
done

lemma UN_eq_UN_Diffs:
  "Ord(a) ==> (⋃ b ∈ a. F(b)) = (⋃ b ∈ a. F(b) - (⋃ c ∈ b. F(c)))"
apply (rule equalityI)
  prefer 2 apply fast
apply (rule subsetI)
apply (erule UN_E)
apply (rule UN_I)
  apply (rule_tac P = "%z. x ∈ F (z) " in Least_in_Ord, (assumption+))
apply (rule DiffI, best intro: Ord_in_Ord LeastI, clarify)
apply (erule_tac P = "%z. x ∈ F (z) " and i = c in less_LeastE)
apply (blast intro: Ord_Least ltI)
done

lemma lepoll_imp_eqpoll_subset:
  "a ≲ X ==> ∃Y. Y ⊆ X & a ≈ Y"
apply (unfold lepoll_def eqpoll_def, clarify)
apply (blast intro: restrict_bij
  dest: inj_is_fun [THEN fun_is_rel, THEN image_subset])
done

lemma Diff_lesspoll_eqpoll_Card_lemma:
  "[| A ≈ a; ~Finite(a); Card(a); B < a; A-B < a |] ==> P"
apply (elim lesspoll_imp_ex_lt_eqpoll [THEN exE] Card_is_Ord conjE)
apply (frule_tac j=xa in Un_upper1_le [OF lt_Ord lt_Ord], assumption)
apply (frule_tac j=xa in Un_upper2_le [OF lt_Ord lt_Ord], assumption)
apply (drule Un_least_lt, assumption)
apply (drule eqpoll_imp_lepoll [THEN lepoll_trans],
  rule le_imp_lepoll, assumption)+
apply (case_tac "Finite(x Un xa)")

finite case
  apply (drule Finite_Un [OF lepoll_Finite lepoll_Finite], assumption+)

  apply (drule subset_Un_Diff [THEN subset_imp_lepoll, THEN lepoll_Finite])
  apply (fast dest: eqpoll_sym [THEN eqpoll_imp_lepoll, THEN lepoll_Finite])

infinite case
  apply (drule Un_lepoll_Inf_Ord, (assumption+))
  apply (blast intro: le_Ord2)
  apply (drule lesspoll_trans1
    [OF subset_Un_Diff [THEN subset_imp_lepoll, THEN lepoll_trans]

```

```

        lt_Card_imp_lespoll], assumption+)
apply (simp add: lesspoll_def)
done

lemma Diff_lespoll_eqpoll_Card:
  "[| A ≈ a; ~Finite(a); Card(a); B < a |] ==> A - B ≈ a"
apply (rule ccontr)
apply (rule Diff_lespoll_eqpoll_Card_lemma, (assumption+))
apply (blast intro: lesspoll_def [THEN def_imp_iff, THEN iffD2]
        subset_imp_lepoll eqpoll_imp_lepoll lepoll_trans)
done

end

theory W06_W01 imports Cardinal_aux begin

constdefs

  NN  :: "i => i"
      "NN(y) == {m ∈ nat. ∃a. ∃f. Ord(a) & domain(f)=a &
                (⋃ b<a. f`b) = y & (∀ b<a. f`b ≲ m)}"

  uu  :: "[i, i, i, i] => i"
      "uu(f, beta, gamma, delta) == (f`beta * f`gamma) Int f`delta"

  vv1 :: "[i, i, i] => i"
      "vv1(f,m,b) ==
        let g = LEAST g. (∃ d. Ord(d) & (domain(uu(f,b,g,d)) ≠ 0 &
                                     domain(uu(f,b,g,d)) ≲ m));
        d = LEAST d. domain(uu(f,b,g,d)) ≠ 0 &
                    domain(uu(f,b,g,d)) ≲ m
        in if f`b ≠ 0 then domain(uu(f,b,g,d)) else 0"

  ww1 :: "[i, i, i] => i"
      "ww1(f,m,b) == f`b - vv1(f,m,b)"

  gg1 :: "[i, i, i] => i"
      "gg1(f,a,m) == λb ∈ a++a. if b<a then vv1(f,m,b) else ww1(f,m,b--a)"

  vv2 :: "[i, i, i, i] => i"
      "vv2(f,b,g,s) ==
        if f`g ≠ 0 then {uu(f, b, g, LEAST d. uu(f,b,g,d) ≠ 0)`s}
        else 0"

```

```

ww2 :: "[i, i, i, i] => i"
      "ww2(f,b,g,s) == f'g - vv2(f,b,g,s)"

gg2 :: "[i, i, i, i] => i"
      "gg2(f,a,b,s) ==
        λg ∈ a++a. if g<a then vv2(f,b,g,s) else ww2(f,b,g--a,s)"

lemma W02_W03: "W02 ==> W03"
by (unfold W02_def W03_def, fast)

lemma W03_W01: "W03 ==> W01"
apply (unfold eqpoll_def W01_def W03_def)
apply (intro allI)
apply (drule_tac x=A in spec)
apply (blast intro: bij_is_inj well_ord_rvimage
          well_ord_Memrel [THEN well_ord_subset])
done

lemma W01_W02: "W01 ==> W02"
apply (unfold eqpoll_def W01_def W02_def)
apply (blast intro!: Ord_ordertype ordermap_bij)
done

lemma lam_sets: "f ∈ A->B ==> (λx ∈ A. {f'x}): A -> {{b}. b ∈ B}"
by (fast intro!: lam_type apply_type)

lemma surj_imp_eq_: "f ∈ surj(A,B) ==> (∪ a ∈ A. {f'a}) = B"
apply (unfold surj_def)
apply (fast elim!: apply_type)
done

lemma surj_imp_eq: "[| f ∈ surj(A,B); Ord(A) |] ==> (∪ a<A. {f'a}) = B"
by (fast dest!: surj_imp_eq_ intro!: ltI elim!: ltE)

lemma W01_W04: "W01 ==> W04(1)"
apply (unfold W01_def W04_def)
apply (rule allI)
apply (erule_tac x = A in allE)
apply (erule exE)
apply (intro exI conjI)
apply (erule Ord_ordertype)

```

```

apply (erule ordermap_bij [THEN bij_converse_bij, THEN bij_is_fun, THEN
lam_sets, THEN domain_of_fun])
apply (simp_all add: singleton_eqpoll_1 eqpoll_imp_lepoll Ord_ordertype
ordermap_bij [THEN bij_converse_bij, THEN bij_is_surj, THEN surj_imp_eq]
ltD)
done

```

```

lemma W04_mono: "[| m ≤ n; W04(m) |] ==> W04(n)"
apply (unfold W04_def)
apply (blast dest!: spec intro: lepoll_trans [OF _ le_imp_lepoll])
done

```

```

lemma W04_W05: "[| m ∈ nat; 1 < m; W04(m) |] ==> W05"
by (unfold W04_def W05_def, blast)

```

```

lemma W05_W06: "W05 ==> W06"
by (unfold W04_def W05_def W06_def, blast)

```

```

lemma lt_oadd_odiff_disj:
  "[| k < i++j; Ord(i); Ord(j) |]
   ==> k < i | (~ k < i & k = i ++ (k--i) & (k--i) < j)"
apply (rule_tac i = k and j = i in Ord_linear2)
prefer 4
  apply (drule odiff_lt_mono2, assumption)
  apply (simp add: oadd_odiff_inverse odiff_oadd_inverse)
apply (auto elim!: lt_Ord)
done

```

```

lemma domain_uu_subset: "domain(uu(f,b,g,d)) ⊆ f`b"
by (unfold uu_def, blast)

```

```

lemma quant_domain_uu_lepoll_m:
  "∀ b<a. f' b ≲ m ==> ∀ b<a. ∀ g<a. ∀ d<a. domain(uu(f,b,g,d)) ≲ m"
by (blast intro: domain_uu_subset [THEN subset_imp_lepoll] lepoll_trans)

```

```

lemma uu_subset1: "uu(f,b,g,d) ⊆ f' b * f' g"
by (unfold uu_def, blast)

```

```

lemma uu_subset2: "uu(f,b,g,d) ⊆ f' d"
by (unfold uu_def, blast)

```

```

lemma uu_lepoll_m: "[| ∀ b<a. f' b ≲ m; d<a |] ==> uu(f,b,g,d) ≲ m"
by (blast intro: uu_subset2 [THEN subset_imp_lepoll] lepoll_trans)

```

```

lemma cases:
  "∀ b<a. ∀ g<a. ∀ d<a. u(f,b,g,d) ≲ m
  ==> (∀ b<a. f' b ≠ 0 -->
      (∃ g<a. ∃ d<a. u(f,b,g,d) ≠ 0 & u(f,b,g,d) < m))
  | (∃ b<a. f' b ≠ 0 & (∀ g<a. ∀ d<a. u(f,b,g,d) ≠ 0 -->
      u(f,b,g,d) ≈ m))"

```

```

apply (unfold lesspoll_def)
apply (blast del: equalityI)
done

```

```

lemma UN_oadd: "Ord(a) ==> (⋃ b<a++a. C(b)) = (⋃ b<a. C(b) Un C(a++b))"
by (blast intro: ltI lt_oadd1 oadd_lt_mono2 dest!: lt_oadd_disj)

```

```

lemma vv1_subset: "vv1(f,m,b) ⊆ f' b"
by (simp add: vv1_def Let_def domain_uu_subset)

```

```

lemma UN_gg1_eq:
  "[| Ord(a); m ∈ nat |] ==> (⋃ b<a++a. gg1(f,a,m)'b) = (⋃ b<a. f' b)"
by (simp add: gg1_def UN_oadd lt_oadd1 oadd_le_self [THEN le_imp_not_lt]

      lt_Ord odiff_oadd_inverse ltD vv1_subset [THEN Diff_partition]
      ww1_def)

```

```

lemma domain_gg1: "domain(gg1(f,a,m)) = a++a"
by (simp add: lam_funtype [THEN domain_of_fun] gg1_def)

lemma nested_LeastI:
  "[| P(a, b); Ord(a); Ord(b);
    Least_a = (LEAST a.  $\exists x. \text{Ord}(x) \ \& \ P(a, x)$ ) |]
  ==> P(Least_a, LEAST b. P(Least_a, b))"
apply (erule ssubst)
apply (rule_tac Q = "%z. P (z, LEAST b. P (z, b))" in LeastI2)
apply (fast elim!: LeastI)+
done

lemmas nested_Least_instance =
  nested_LeastI [of "%g d. domain(uu(f,b,g,d))  $\neq 0$  &
    domain(uu(f,b,g,d))  $\lesssim m$ ",
    standard]

lemma gg1_lepoll_m:
  "[| Ord(a); m  $\in$  nat;
     $\forall b < a. f' b \neq 0 \ \rightarrow$ 
      ( $\exists g < a. \exists d < a. \text{domain}(uu(f,b,g,d)) \neq 0 \ \&$ 
         $\text{domain}(uu(f,b,g,d)) \lesssim m$ );
     $\forall b < a. f' b \lesssim \text{succ}(m); b < a++a$  |]
  ==> gg1(f,a,m)'b  $\lesssim m$ "
apply (simp add: gg1_def empty_lepollI)
apply (safe dest!: lt_oadd_odiff_disj)

  apply (simp add: vv1_def Let_def empty_lepollI)
  apply (fast intro: nested_Least_instance [THEN conjunct2]
    elim!: lt_Ord)

apply (simp add: ww1_def empty_lepollI)
apply (case_tac "f' (b--a) = 0", simp add: empty_lepollI)
apply (rule Diff_lepoll, blast)
apply (rule vv1_subset)
apply (drule ospec [THEN mp], assumption+)
apply (elim oexE conjE)
apply (simp add: vv1_def Let_def lt_Ord nested_Least_instance [THEN conjunct1])
done

```

```

lemma ex_d_uu_not_empty:
  "[| b<a; g<a; f' b ≠ 0; f' g ≠ 0;
    y*y ⊆ y; (⋃ b<a. f' b)=y |]
  ==> ∃ d<a. uu(f,b,g,d) ≠ 0"
by (unfold uu_def, blast)

lemma uu_not_empty:
  "[| b<a; g<a; f' b ≠ 0; f' g ≠ 0; y*y ⊆ y; (⋃ b<a. f' b)=y |]
  ==> uu(f,b,g,LEAST d. (uu(f,b,g,d) ≠ 0)) ≠ 0"
apply (drule ex_d_uu_not_empty, assumption+)
apply (fast elim!: LeastI lt_Ord)
done

lemma not_empty_rel_imp_domain: "[| r ⊆ A*B; r ≠ 0 |] ==> domain(r) ≠ 0"
by blast

lemma Least_uu_not_empty_lt_a:
  "[| b<a; g<a; f' b ≠ 0; f' g ≠ 0; y*y ⊆ y; (⋃ b<a. f' b)=y |]
  ==> (LEAST d. uu(f,b,g,d) ≠ 0) < a"
apply (erule ex_d_uu_not_empty [THEN oexE], assumption+)
apply (blast intro: Least_le [THEN lt_trans1] lt_Ord)
done

lemma subset_Diff_sing: "[| B ⊆ A; a ∉ B |] ==> B ⊆ A-{a}"
by blast

lemma supset_lepoll_imp_eq:
  "[| A ≲ m; m ≲ B; B ⊆ A; m ∈ nat |] ==> A=B"
apply (erule natE)
apply (fast dest!: lepoll_0_is_0 intro!: equalityI)
apply (safe intro!: equalityI)
apply (rule ccontr)
apply (rule succ_lepoll_natE)
  apply (erule lepoll_trans)
  apply (rule lepoll_trans)
  apply (erule subset_Diff_sing [THEN subset_imp_lepoll], assumption)
  apply (rule Diff_sing_lepoll, assumption+)
done

lemma uu_Least_is_fun:
  "[| ∀ g<a. ∀ d<a. domain(uu(f, b, g, d)) ≠ 0 -->
    domain(uu(f, b, g, d)) ≈ succ(m);
    ∀ b<a. f' b ≲ succ(m); y*y ⊆ y;
    (⋃ b<a. f' b)=y; b<a; g<a; d<a;
    f' b ≠ 0; f' g ≠ 0; m ∈ nat; s ∈ f' b |]

```

```

    ==> uu(f, b, g, LEAST d. uu(f,b,g,d)≠0) ∈ f' b -> f' g"
  apply (drule_tac x2=g in ospec [THEN ospec, THEN mp])
    apply (rule_tac [3] not_empty_rel_imp_domain [OF uu_subset1 uu_not_empty])
      apply (rule_tac [2] Least_uu_not_empty_lt_a, assumption+)
  apply (rule rel_is_fun)
    apply (erule eqpoll_sym [THEN eqpoll_imp_lepoll])
    apply (erule uu_lepoll_m)
    apply (rule Least_uu_not_empty_lt_a, assumption+)
  apply (rule uu_subset1)
  apply (rule supset_lepoll_imp_eq [OF _ eqpoll_sym [THEN eqpoll_imp_lepoll]])
  apply (fast intro!: domain_uu_subset)+
done

```

```

lemma vv2_subset:
  "[| ∀g<a. ∀d<a. domain(uu(f, b, g, d))≠0 -->
    domain(uu(f, b, g, d)) ≈ succ(m);
    ∀b<a. f' b ≲ succ(m); y*y ⊆ y;
    (∪ b<a. f' b)=y; b<a; g<a; m ∈ nat; s ∈ f' b |]
  ==> vv2(f,b,g,s) ⊆ f' g"
  apply (simp add: vv2_def)
  apply (blast intro: uu_Least_is_fun [THEN apply_type])
done

```

```

lemma UN_gg2_eq:
  "[| ∀g<a. ∀d<a. domain(uu(f,b,g,d)) ≠ 0 -->
    domain(uu(f,b,g,d)) ≈ succ(m);
    ∀b<a. f' b ≲ succ(m); y*y ⊆ y;
    (∪ b<a. f' b)=y; Ord(a); m ∈ nat; s ∈ f' b; b<a |]
  ==> (∪ g<a++a. gg2(f,a,b,s) ' g) = y"
  apply (unfold gg2_def)
  apply (drule sym)
  apply (simp add: ltD UN_oadd oadd_le_self [THEN le_imp_not_lt]
    lt_Ord odiff_oadd_inverse ww2_def
    vv2_subset [THEN Diff_partition])
done

```

```

lemma domain_gg2: "domain(gg2(f,a,b,s)) = a++a"
by (simp add: lam_funtype [THEN domain_of_fun] gg2_def)

```

```

lemma vv2_lepoll: "[| m ∈ nat; m≠0 |] ==> vv2(f,b,g,s) ≲ m"
  apply (unfold vv2_def)

```

```

apply (simp add: empty_lepollI)
apply (fast dest!: le_imp_subset [THEN subset_imp_lepoll, THEN lepoll_0_is_0]

      intro!: singleton_eqpoll_1 [THEN eqpoll_imp_lepoll, THEN lepoll_trans]
      not_lt_imp_le [THEN le_imp_subset, THEN subset_imp_lepoll]
      nat_into_Ord nat_1I)
done

lemma ww2_lepoll:
  "[|  $\forall b < a. f' b \lesssim \text{succ}(m)$ ;  $g < a$ ;  $m \in \text{nat}$ ;  $\text{vv2}(f, b, g, d) \subseteq f' g$  |]

  ==>  $\text{ww2}(f, b, g, d) \lesssim m$ "
apply (unfold ww2_def)
apply (case_tac "f'g = 0")
apply (simp add: empty_lepollI)
apply (drule ospec, assumption)
apply (rule Diff_lepoll, assumption+)
apply (simp add: vv2_def not_emptyI)
done

lemma gg2_lepoll_m:
  "[|  $\forall g < a. \forall d < a. \text{domain}(\text{uu}(f, b, g, d)) \neq 0 \text{ -->}$ 
       $\text{domain}(\text{uu}(f, b, g, d)) \approx \text{succ}(m)$ ;
       $\forall b < a. f' b \lesssim \text{succ}(m)$ ;  $y * y \subseteq y$ ;
       $(\bigcup b < a. f' b) = y$ ;  $b < a$ ;  $s \in f' b$ ;  $m \in \text{nat}$ ;  $m \neq 0$ ;  $g < a + a$  |]

  ==>  $\text{gg2}(f, a, b, s) \text{ ' } g \lesssim m$ "
apply (simp add: gg2_def empty_lepollI)
apply (safe elim!: lt_Ord2 dest!: lt_oadd_odiff_disj)
  apply (simp add: vv2_lepoll)
apply (simp add: ww2_lepoll vv2_subset)
done

lemma lemma_ii: "[|  $\text{succ}(m) \in \text{NN}(y)$ ;  $y * y \subseteq y$ ;  $m \in \text{nat}$ ;  $m \neq 0$  |] ==>
 $m \in \text{NN}(y)$ "
apply (unfold NN_def)
apply (elim CollectE exE conjE)
apply (rule quant_domain_uu_lepoll_m [THEN cases, THEN disjE], assumption)

apply (simp add: lesspoll_succ_iff)
apply (rule_tac x = "a+a" in exI)
apply (fast intro!: Ord_oadd domain_gg1 UN_gg1_eq gg1_lepoll_m)

apply (elim oexE conjE)
apply (rule_tac A = "f'?B" in not_emptyE, assumption)
apply (rule CollectI)

```

```

apply (erule succ_natD)
apply (rule_tac x = "a++a" in exI)
apply (rule_tac x = "gg2 (f,a,b,x) " in exI)

apply (simp add: Ord_oadd domain_gg2 UN_gg2_eq gg2_lepoll_m)
done

```

```

lemma z_n_subset_z_succ_n:
  " $\forall n \in \text{nat}. \text{rec}(n, x, \%k r. r \text{ Un } r*r) \subseteq \text{rec}(\text{succ}(n), x, \%k r. r \text{ Un } r*r)$ "
by (fast intro: rec_succ [THEN ssubst])

```

```

lemma le_subsets:
  " $[| \forall n \in \text{nat}. f(n) \leq f(\text{succ}(n)); n \leq m; n \in \text{nat}; m \in \text{nat} |]$ 
 $\implies f(n) \leq f(m)$ "
apply (erule_tac P = "n ≤ m" in rev_mp)
apply (rule_tac P = "%z. n ≤ z --> f (n) ⊆ f (z) " in nat_induct)
apply (auto simp add: le_iff)
done

```

```

lemma le_imp_rec_subset:
  " $[| n \leq m; m \in \text{nat} |]$ 
 $\implies \text{rec}(n, x, \%k r. r \text{ Un } r*r) \subseteq \text{rec}(m, x, \%k r. r \text{ Un } r*r)$ "
apply (rule z_n_subset_z_succ_n [THEN le_subsets])
apply (blast intro: lt_nat_in_nat)+
done

```

```

lemma lemma_iv: " $\exists y. x \text{ Un } y*y \subseteq y$ "
apply (rule_tac x = " $\bigcup n \in \text{nat}. \text{rec} (n, x, \%k r. r \text{ Un } r*r)$  " in exI)
apply safe
apply (rule nat_0I [THEN UN_I], simp)
apply (rule_tac a = "succ (n Un na) " in UN_I)
apply (erule Un_nat_type [THEN nat_succI], assumption)
apply (auto intro: le_imp_rec_subset [THEN subsetD]
  intro!: Un_upper1_le Un_upper2_le Un_nat_type
  elim!: nat_into_Ord)
done

```

```
lemma W06_imp_NN_not_empty: "W06 ==> NN(y) ≠ 0"
by (unfold W06_def NN_def, clarify, blast)
```

```
lemma lemma1:
  "[| (⋃ b<a. f' b)=y; x ∈ y; ∀ b<a. f' b ≲ 1; Ord(a) |] ==> ∃ c<a. f' c
= {x}"
by (fast elim!: lepoll_1_is_sing)
```

```
lemma lemma2:
  "[| (⋃ b<a. f' b)=y; x ∈ y; ∀ b<a. f' b ≲ 1; Ord(a) |]
==> f' (LEAST i. f' i = {x}) = {x}"
apply (drule lemma1, assumption+)
apply (fast elim!: lt_Ord intro: LeastI)
done
```

```
lemma NN_imp_ex_inj: "1 ∈ NN(y) ==> ∃ a f. Ord(a) & f ∈ inj(y, a)"
apply (unfold NN_def)
apply (elim CollectE exE conjE)
apply (rule_tac x = a in exI)
apply (rule_tac x = "λx ∈ y. LEAST i. f' i = {x}" in exI)
apply (rule conjI, assumption)
apply (rule_tac d = "%i. THE x. x ∈ f' i" in lam_injective)
apply (drule lemma1, assumption+)
apply (fast elim!: Least_le [THEN lt_trans1, THEN ltD] lt_Ord)
apply (rule lemma2 [THEN ssubst], assumption+, blast)
done
```

```
lemma y_well_ord: "[| y*y ⊆ y; 1 ∈ NN(y) |] ==> ∃ r. well_ord(y, r)"
apply (drule NN_imp_ex_inj)
apply (fast elim!: well_ord_rvimage [OF _ well_ord_Memrel])
done
```

```

lemma rev_induct_lemma [rule_format]:
  "[| n ∈ nat; !!m. [| m ∈ nat; m≠0; P(succ(m)) |] ==> P(m) |]
   ==> n≠0 --> P(n) --> P(1)"
by (erule nat_induct, blast+)

```

```

lemma rev_induct:
  "[| n ∈ nat; P(n); n≠0;
   !!m. [| m ∈ nat; m≠0; P(succ(m)) |] ==> P(m) |]
   ==> P(1)"
by (rule rev_induct_lemma, blast+)

```

```

lemma NN_into_nat: "n ∈ NN(y) ==> n ∈ nat"
by (simp add: NN_def)

```

```

lemma lemma3: "[| n ∈ NN(y); y*y ⊆ y; n≠0 |] ==> 1 ∈ NN(y)"
apply (rule rev_induct [OF NN_into_nat], assumption+)
apply (rule lemma_ii, assumption+)
done

```

```

lemma NN_y_0: "0 ∈ NN(y) ==> y=0"
apply (unfold NN_def)
apply (fast intro!: equalityI dest!: lepoll_0_is_0 elim: subst)
done

```

```

lemma W06_imp_W01: "W06 ==> W01"
apply (unfold W01_def)
apply (rule allI)
apply (case_tac "A=0")
apply (fast intro!: well_ord_Memrel nat_0I [THEN nat_into_Ord])
apply (rule_tac x1 = A in lemma_iv [THEN revcut_rl])
apply (erule exE)
apply (drule W06_imp_NN_not_empty)
apply (erule Un_subset_iff [THEN iffD1, THEN conjE])
apply (erule_tac A = "NN (y) " in not_emptyE)
apply (frule y_well_ord)
  apply (fast intro!: lemma3 dest!: NN_y_0 elim!: not_emptyE)
apply (fast elim: well_ord_subset)
done

```

```

end

```

```

theory W01_W07 imports AC_Equiv begin

constdefs
  LEMMA :: o
    "LEMMA ==
      $\forall X. \sim\text{Finite}(X) \rightarrow (\exists R. \text{well\_ord}(X,R) \ \& \ \sim\text{well\_ord}(X,\text{converse}(R)))"$ 

lemma W07_iff_LEMMA: "W07 <-> LEMMA"
apply (unfold W07_def LEMMA_def)
apply (blast intro: Finite_well_ord_converse)
done

lemma LEMMA_imp_W01: "LEMMA ==> W01"
apply (unfold W01_def LEMMA_def Finite_def eqpoll_def)
apply (blast intro!: well_ord_rvimage [OF bij_is_inj nat_implies_well_ord])
done

lemma converse_Memrel_not_wf_on:
  "[| Ord(a);  $\sim\text{Finite}(a)$  |] ==>  $\sim\text{wf}[a](\text{converse}(\text{Memrel}(a)))"$ 
apply (unfold wf_on_def wf_def)
apply (drule nat_le_infinite_Ord [THEN le_imp_subset], assumption)
apply (rule notI)
apply (erule_tac x = nat in allE, blast)
done

```

```

lemma converse_Memrel_not_well_ord:
  "[| Ord(a); ~Finite(a) |] ==> ~well_ord(a, converse(Memrel(a)))"
apply (unfold well_ord_def)
apply (blast dest: converse_Memrel_not_wf_on)
done

lemma well_ord_rvimage_ordertype:
  "well_ord(A,r) ==>
   rvimage (ordertype(A,r), converse(ordermap(A,r)),r) =
   Memrel(ordertype(A,r))"
by (blast intro: ordertype_ord_iso [THEN ord_iso_sym] ord_iso_rvimage_eq
    Memrel_type [THEN subset_Int_iff [THEN iffD1]] trans)

lemma well_ord_converse_Memrel:
  "[| well_ord(A,r); well_ord(A, converse(r)) |]
   ==> well_ord(ordertype(A,r), converse(Memrel(ordertype(A,r))))"

apply (subst well_ord_rvimage_ordertype [symmetric], assumption)
apply (rule rvimage_converse [THEN subst])
apply (blast intro: ordertype_ord_iso ord_iso_sym ord_iso_is_bij
    bij_is_inj well_ord_rvimage)
done

lemma W01_imp_LEMMA: "W01 ==> LEMMA"
apply (unfold W01_def LEMMA_def, clarify)
apply (blast dest: well_ord_converse_Memrel
    Ord_ordertype [THEN converse_Memrel_not_well_ord]
    intro: ordertype_ord_iso ord_iso_is_bij bij_is_inj lepoll_Finite
    lepoll_def [THEN def_imp_iff, THEN iffD2] )
done

lemma W01_iff_W07: "W01 <-> W07"
apply (simp add: W07_iff_LEMMA)
apply (blast intro: LEMMA_imp_W01 W01_imp_LEMMA)
done

lemma W01_W08: "W01 ==> W08"
by (unfold W01_def W08_def, fast)

lemma W08_W01: "W08 ==> W01"
apply (unfold W01_def W08_def)

```

```

apply (rule allI)
apply (erule_tac x = "{x}. x ∈ A" in allE)
apply (erule impE)
  apply (rule_tac x = "λa ∈ {x}. x ∈ A. THE x. a={x}" in exI)
  apply (force intro!: lam_type simp add: singleton_eq_iff the_equality)
apply (blast intro: lam_sing_bij bij_is_inj well_ord_rvimage)
done

end

```

```

theory AC7_AC9 imports AC_Equiv begin

```

```

lemma Sigma_fun_space_not0: "[| 0 ∉ A; B ∈ A |] ==> (nat->Union(A)) *
B ≠ 0"
by (blast dest!: Sigma_empty_iff [THEN iffD1] Union_empty_iff [THEN iffD1])

```

```

lemma inj_lemma:
  "C ∈ A ==> (λg ∈ (nat->Union(A))*C.
    (λn ∈ nat. if(n=0, snd(g), fst(g)‘(n #- 1)))
    ∈ inj((nat->Union(A))*C, (nat->Union(A)) ) )"
apply (unfold inj_def)
apply (rule CollectI)
apply (fast intro!: lam_type if_type apply_type fst_type snd_type, auto)

apply (rule fun_extension, assumption+)
apply (drule lam_eqE [OF _ nat_succI], assumption, simp)
apply (drule lam_eqE [OF _ nat_0I], simp)
done

```

```

lemma Sigma_fun_space_eqpoll:
  "[| C ∈ A; 0 ∉ A |] ==> (nat->Union(A)) * C ≈ (nat->Union(A))"
apply (rule eqpollI)
apply (simp add: lepoll_def)
apply (fast intro!: inj_lemma)
apply (fast elim!: prod_lepoll_self not_sym [THEN not_emptyE] subst_elem

      elim: swap)
done

```

```
lemma AC6_AC7: "AC6 ==> AC7"
by (unfold AC6_def AC7_def, blast)
```

```
lemma lemma1_1: "y ∈ (Π B ∈ A. Y*B) ==> (λB ∈ A. snd(y'B)) ∈ (Π B
∈ A. B)"
by (fast intro!: lam_type snd_type apply_type)
```

```
lemma lemma1_2:
  "y ∈ (Π B ∈ {Y*C. C ∈ A}. B) ==> (λB ∈ A. y'(Y*B)) ∈ (Π B ∈ A.
Y*B)"
apply (fast intro!: lam_type apply_type)
done
```

```
lemma AC7_AC6_lemma1:
  "(Π B ∈ {(nat->Union(A))*C. C ∈ A}. B) ≠ 0 ==> (Π B ∈ A. B) ≠
0"
by (fast intro!: equals0I lemma1_1 lemma1_2)
```

```
lemma AC7_AC6_lemma2: "0 ∉ A ==> 0 ∉ {(nat -> Union(A)) * C. C ∈ A}"
by (blast dest: Sigma_fun_space_not0)
```

```
lemma AC7_AC6: "AC7 ==> AC6"
apply (unfold AC6_def AC7_def)
apply (rule allI)
apply (rule impI)
apply (case_tac "A=0", simp)
apply (rule AC7_AC6_lemma1)
apply (erule allE)
apply (blast del: notI
  intro!: AC7_AC6_lemma2 intro: eqpoll_sym eqpoll_trans
  Sigma_fun_space_eqpoll)
done
```

```
lemma AC1_AC8_lemma1:
  "∀ B ∈ A. ∃ B1 B2. B=<B1,B2> & B1 ≈ B2"
```

```

    ==> 0 ∉ { bij(fst(B),snd(B)). B ∈ A }"
  apply (unfold eqpoll_def, auto)
done

lemma AC1_AC8_lemma2:
  "[| f ∈ (∏ X ∈ RepFun(A,p). X); D ∈ A |] ==> (λx ∈ A. f'p(x))'D
  ∈ p(D)"
  apply (simp, fast elim!: apply_type)
done

lemma AC1_AC8: "AC1 ==> AC8"
  apply (unfold AC1_def AC8_def)
  apply (fast dest: AC1_AC8_lemma1 AC1_AC8_lemma2)
done

lemma AC8_AC9_lemma:
  "∀ B1 ∈ A. ∀ B2 ∈ A. B1 ≈ B2
  ==> ∀ B ∈ A*A. ∃ B1 B2. B=<B1,B2> & B1 ≈ B2"
  by fast

lemma AC8_AC9: "AC8 ==> AC9"
  apply (unfold AC8_def AC9_def)
  apply (intro allI impI)
  apply (erule allE)
  apply (erule impE, erule AC8_AC9_lemma, force)
done

lemma snd_lepoll_SigmaI: "b ∈ B ==> X ≲ B × X"
  by (blast intro: lepoll_trans prod_lepoll_self eqpoll_imp_lepoll
    prod_commute_eqpoll)

lemma nat_lepoll_lemma:
  "[| 0 ∉ A; B ∈ A |] ==> nat ≲ ((nat → Union(A)) × B) × nat"

```

```

by (blast dest: Sigma_fun_space_not0 intro: snd_lepoll_SigmaI)

lemma AC9_AC1_lemma1:
  "[| 0∉A; A≠0;
    C = {(nat->Union(A))*B}*nat. B ∈ A} Un
      {cons(0,((nat->Union(A))*B)*nat). B ∈ A};
    B1 ∈ C; B2 ∈ C |]
  ==> B1 ≈ B2"
by (blast intro!: nat_lepoll_lemma Sigma_fun_space_eqpoll
    nat_cons_eqpoll [THEN eqpoll_trans]
    prod_eqpoll_cong [OF _ eqpoll_refl]
    intro: eqpoll_trans eqpoll_sym )

lemma AC9_AC1_lemma2:
  "∀B1 ∈ {(F*B)*N. B ∈ A} Un {cons(0,(F*B)*N). B ∈ A}.
  ∀B2 ∈ {(F*B)*N. B ∈ A} Un {cons(0,(F*B)*N). B ∈ A}.
  f'⟨B1,B2⟩ ∈ bij(B1, B2)
  ==> (λB ∈ A. snd(fst((f'⟨cons(0,(F*B)*N), (F*B)*N⟩)'0))) ∈ (Π X
  ∈ A. X)"
apply (intro lam_type snd_type fst_type)
apply (rule apply_type [OF _ consI1])
apply (fast intro!: fun_weaken_type bij_is_fun)
done

lemma AC9_AC1: "AC9 ==> AC1"
apply (unfold AC1_def AC9_def)
apply (intro allI impI)
apply (erule allE)
apply (case_tac "A~=0")
apply (blast dest: AC9_AC1_lemma1 AC9_AC1_lemma2, force)
done

end

theory W01_AC imports AC_Equiv begin

theorem W01_AC1: "W01 ==> AC1"
by (unfold AC1_def W01_def, fast elim!: ex_choice_fun)

```

```

lemma lemma1: "[| W01;  $\forall B \in A. \exists C \in D(B). P(C,B)$  |] ==>  $\exists f. \forall B \in A. P(f'B,B)$ "
apply (unfold W01_def)
apply (erule_tac x = "Union ({C  $\in$  D (B) . P (C,B) }. B  $\in$  A)" in allE)
apply (erule exE, drule ex_choice_fun, fast)
apply (erule exE)
apply (rule_tac x = " $\lambda x \in A. f'\{C \in D (x) . P (C,x)\}$ " in exI)
apply (simp, blast dest!: apply_type [OF _ RepFunI])
done

lemma lemma2_1: "[|  $\sim$ Finite(B); W01 |] ==>  $|B| + |B| \approx B$ "
apply (unfold W01_def)
apply (rule eqpoll_trans)
prefer 2 apply (fast elim!: well_ord_cardinal_eqpoll)
apply (rule eqpoll_sym [THEN eqpoll_trans])
apply (fast elim!: well_ord_cardinal_eqpoll)
apply (drule spec [of _ B])
apply (clarify dest!: eqpoll_imp_Finite_iff [OF well_ord_cardinal_eqpoll])

apply (simp add: cadd_def [symmetric]
      eqpoll_refl InfCard_cdouble_eq Card_cardinal Inf_Card_is_InfCard)

done

lemma lemma2_2:
  " $f \in \text{bij}(D+D, B) \implies \{f'\text{Inl}(i), f'\text{Inr}(i)\}. i \in D\} \in \text{Pow}(\text{Pow}(B))$ "
by (fast elim!: bij_is_fun [THEN apply_type])

lemma lemma2_3:
  " $f \in \text{bij}(D+D, B) \implies \text{pairwise\_disjoint}(\{f'\text{Inl}(i), f'\text{Inr}(i)\}. i \in D\})$ "
apply (unfold pairwise_disjoint_def)
apply (blast dest: bij_is_inj [THEN inj_apply_equality])
done

lemma lemma2_4:
  "[|  $f \in \text{bij}(D+D, B)$ ;  $1 \leq n$  |]
  ==>  $\text{sets\_of\_size\_between}(\{f'\text{Inl}(i), f'\text{Inr}(i)\}. i \in D\}, 2, \text{succ}(n))$ "
apply (simp (no_asm_simp) add: sets_of_size_between_def succ_def)
apply (blast intro!: cons_lepoll_cong
      intro: singleton_eqpoll_1 [THEN eqpoll_imp_lepoll]
      le_imp_subset [THEN subset_imp_lepoll] lepoll_trans
      dest: bij_is_inj [THEN inj_apply_equality] elim!: mem_irrefl)
done

lemma lemma2_5:
  " $f \in \text{bij}(D+D, B) \implies \text{Union}(\{f'\text{Inl}(i), f'\text{Inr}(i)\}. i \in D\})=B$ "

```

```

apply (unfold bij_def surj_def)
apply (fast elim!: inj_is_fun [THEN apply_type])
done

lemma lemma2:
  "[| W01; ~Finite(B); 1 ≤ n |]
  ==> ∃ C ∈ Pow(Pow(B)). pairwise_disjoint(C) &
    sets_of_size_between(C, 2, succ(n)) &
    Union(C)=B"
apply (drule lemma2_1 [THEN eqpoll_def [THEN def_imp_iff, THEN iffD1]],
      assumption)
apply (blast intro!: lemma2_2 lemma2_3 lemma2_4 lemma2_5)
done

theorem W01_AC10: "[| W01; 1 ≤ n |] ==> AC10(n)"
apply (unfold AC10_def)
apply (fast intro!: lemma1 elim!: lemma2)
done

end

```

```

theory Hartog imports AC_Equiv begin

```

```

constdefs

```

```

  Hartog :: "i => i"
  "Hartog(X) == LEAST i. ~ i ≲ X"

```

```

lemma Ords_in_set: "∀ a. Ord(a) --> a ∈ X ==> P"
apply (rule_tac X1 = "{y ∈ X. Ord(y) }" in ON_class [THEN revcut_rl])
apply fast
done

```

```

lemma Ord_lepoll_imp_ex_well_ord:
  "[| Ord(a); a ≲ X |]
  ==> ∃ Y. Y ⊆ X & (∃ R. well_ord(Y,R) & ordertype(Y,R)=a)"
apply (unfold lepoll_def)
apply (erule exE)
apply (intro exI conjI)
  apply (erule inj_is_fun [THEN fun_is_rel, THEN image_subset])
  apply (rule well_ord_rvimage [OF bij_is_inj well_ord_Memrel])
  apply (erule restrict_bij [THEN bij_converse_bij])
apply (rule subset_refl, assumption)
apply (rule trans)
apply (rule bij_ordertype_vimage)
apply (erule restrict_bij [THEN bij_converse_bij])

```

```

apply (rule subset_refl)
apply (erule well_ord_Memrel)
apply (erule ordertype_Memrel)
done

lemma Ord_lepoll_imp_eq_ordertype:
  "[| Ord(a); a  $\lesssim$  X |] ==>  $\exists Y. Y \subseteq X \ \& \ (\exists R. R \subseteq X*X \ \& \ \text{ordertype}(Y,R)=a)$ "
apply (drule Ord_lepoll_imp_ex_well_ord, assumption, clarify)
apply (intro exI conjI)
apply (erule_tac [3] ordertype_Int, auto)
done

lemma Ords_lepoll_set_lemma:
  " $(\forall a. \text{Ord}(a) \rightarrow a \lesssim X) \implies$ 
 $\forall a. \text{Ord}(a) \rightarrow$ 
 $a \in \{b. Z \in \text{Pow}(X)*\text{Pow}(X*X), \exists Y R. Z=\langle Y,R \rangle \ \& \ \text{ordertype}(Y,R)=b\}$ "
apply (intro allI impI)
apply (elim allE impE, assumption)
apply (blast dest!: Ord_lepoll_imp_eq_ordertype intro: sym)
done

lemma Ords_lepoll_set: " $\forall a. \text{Ord}(a) \rightarrow a \lesssim X \implies P$ "
by (erule Ords_lepoll_set_lemma [THEN Ords_in_set])

lemma ex_Ord_not_lepoll: " $\exists a. \text{Ord}(a) \ \& \ \sim a \lesssim X$ "
apply (rule ccontr)
apply (best intro: Ords_lepoll_set)
done

lemma not_Hartog_lepoll_self: " $\sim \text{Hartog}(A) \lesssim A$ "
apply (unfold Hartog_def)
apply (rule ex_Ord_not_lepoll [THEN exE])
apply (rule LeastI, auto)
done

lemmas Hartog_lepoll_selfE = not_Hartog_lepoll_self [THEN notE, standard]

lemma Ord_Hartog: " $\text{Ord}(\text{Hartog}(A))$ "
by (unfold Hartog_def, rule Ord_Least)

lemma less_HartogE1: " $[| i < \text{Hartog}(A); \sim i \lesssim A |] \implies P$ "
by (unfold Hartog_def, fast elim: less_LeastE)

lemma less_HartogE: " $[| i < \text{Hartog}(A); i \approx \text{Hartog}(A) |] \implies P$ "
by (blast intro: less_HartogE1 eqpoll_sym eqpoll_imp_lepoll
  lepoll_trans [THEN Hartog_lepoll_selfE])

lemma Card_Hartog: " $\text{Card}(\text{Hartog}(A))$ "
by (fast intro!: CardI Ord_Hartog elim: less_HartogE)

```

end

theory *HH* imports *AC_Equiv Hartog* begin

constdefs

```
HH :: "[i, i, i] => i"
      "HH(f,x,a) == transrec(a, %b r. let z = x - ( $\bigcup c \in b. r'c$ )
                                in if f'z  $\in$  Pow(z)-{0} then f'z else
{x})"
```

0.1 Lemmas useful in each of the three proofs

lemma *HH_def_satisfies_eq*:

```
"HH(f,x,a) = (let z = x - ( $\bigcup b \in a. HH(f,x,b)$ )
              in if f'z  $\in$  Pow(z)-{0} then f'z else {x})"
by (rule HH_def [THEN def_transrec, THEN trans], simp)
```

lemma *HH_values*: " $HH(f,x,a) \in Pow(x)-\{0\} \mid HH(f,x,a)=\{x\}$ "

```
apply (rule HH_def_satisfies_eq [THEN ssubst])
apply (simp add: Let_def Diff_subset [THEN PowI], fast)
done
```

lemma *subset_imp_Diff_eq*:

```
" $B \subseteq A \implies X - (\bigcup a \in A. P(a)) = X - (\bigcup a \in A - B. P(a)) - (\bigcup b \in B. P(b))"$ 
by fast
```

lemma *Ord_DiffE*: " $[\mid c \in a - b; b < a \mid] \implies c = b \mid b < c \ \& \ c < a$ "

```
apply (erule ltE)
apply (drule Ord_linear [of _ c])
apply (fast elim: Ord_in_Ord)
apply (fast intro!: ltI intro: Ord_in_Ord)
done
```

lemma *Diff_UN_eq_self*: " $(\forall y. y \in A \implies P(y) = \{x\}) \implies x - (\bigcup y \in A. P(y)) = x$ "

```
by (simp, fast elim!: mem_irrefl)
```

lemma *HH_eq*: " $x - (\bigcup b \in a. HH(f,x,b)) = x - (\bigcup b \in a1. HH(f,x,b)) \implies HH(f,x,a) = HH(f,x,a1)$ "

```
apply (subst HH_def_satisfies_eq [of _ _ a1])
apply (rule HH_def_satisfies_eq [THEN trans], simp)
done
```

lemma *HH_is_x_gt_too*: " $[\mid HH(f,x,b)=\{x\}; b < a \mid] \implies HH(f,x,a)=\{x\}$ "

```
apply (rule_tac P = " $b < a$ " in impE)
```

```

prefer 2 apply assumption+
apply (erule lt_Ord2 [THEN trans_induct])
apply (rule impI)
apply (rule HH_eq [THEN trans])
prefer 2 apply assumption+
apply (rule leI [THEN le_imp_subset, THEN subset_imp_Diff_eq, THEN ssubst],

      assumption)
apply (rule_tac t = "%z. z-?X" in subst_context)
apply (rule Diff_UN_eq_self)
apply (drule Ord_DiffE, assumption)
apply (fast elim: ltE, auto)
done

lemma HH_subset_x_lt_too:
  "[| HH(f,x,a) ∈ Pow(x)-{0}; b<a |] ==> HH(f,x,b) ∈ Pow(x)-{0}"
apply (rule HH_values [THEN disjE], assumption)
apply (drule HH_is_x_gt_too, assumption)
apply (drule subst, assumption)
apply (fast elim!: mem_irrefl)
done

lemma HH_subset_x_imp_subset_Diff_UN:
  "HH(f,x,a) ∈ Pow(x)-{0} ==> HH(f,x,a) ∈ Pow(x - (⋃ b ∈ a. HH(f,x,b)))-{0}"
apply (drule HH_def_satisfies_eq [THEN subst])
apply (rule HH_def_satisfies_eq [THEN ssubst])
apply (simp add: Let_def Diff_subset [THEN PowI])
apply (drule split_if [THEN iffD1])
apply (fast elim!: mem_irrefl)
done

lemma HH_eq_arg_lt:
  "[| HH(f,x,v)=HH(f,x,w); HH(f,x,v) ∈ Pow(x)-{0}; v ∈ w |] ==> P"
apply (frule_tac P = "%y. y ∈ Pow (x) -{0}" in subst, assumption)
apply (drule_tac a = w in HH_subset_x_imp_subset_Diff_UN)
apply (drule subst_elem, assumption)
apply (fast intro!: singleton_iff [THEN iffD2] equalsOI)
done

lemma HH_eq_imp_arg_eq:
  "[| HH(f,x,v)=HH(f,x,w); HH(f,x,w) ∈ Pow(x)-{0}; Ord(v); Ord(w) |] ==>
v=w"
apply (rule_tac j = w in Ord_linear_lt)
apply (simp_all (no_asm_simp))
  apply (drule subst_elem, assumption)
  apply (blast dest: ltD HH_eq_arg_lt)
apply (blast dest: HH_eq_arg_lt [OF sym] ltD)
done

```

```

lemma HH_subset_x_imp_lepoll:
  "[| HH(f, x, i) ∈ Pow(x)-{0}; Ord(i) |] ==> i lepoll Pow(x)-{0}"
apply (unfold lepoll_def inj_def)
apply (rule_tac x = "λj ∈ i. HH (f, x, j) " in exI)
apply (simp (no_asm_simp))
apply (fast del: DiffE
        elim!: HH_eq_imp_arg_eq Ord_in_Ord HH_subset_x_lt_too
        intro!: lam_type ballI ltI intro: bexI)
done

```

```

lemma HH_Hartog_is_x: "HH(f, x, Hartog(Pow(x)-{0})) = {x}"
apply (rule HH_values [THEN disjE])
prefer 2 apply assumption
apply (fast del: DiffE
        intro!: Ord_Hartog
        dest!: HH_subset_x_imp_lepoll
        elim!: Hartog_lepoll_selfE)
done

```

```

lemma HH_Least_eq_x: "HH(f, x, LEAST i. HH(f, x, i) = {x}) = {x}"
by (fast intro!: Ord_Hartog HH_Hartog_is_x LeastI)

```

```

lemma less_Least_subset_x:
  "a ∈ (LEAST i. HH(f,x,i)={x}) ==> HH(f,x,a) ∈ Pow(x)-{0}"
apply (rule HH_values [THEN disjE], assumption)
apply (rule less_LeastE)
apply (erule_tac [2] ltI [OF _ Ord_Least], assumption)
done

```

0.2 Lemmas used in the proofs of AC1 ==_i WO2 and AC17 ==_i AC1

```

lemma lam_Least_HH_inj_Pow:
  "(λa ∈ (LEAST i. HH(f,x,i)={x}). HH(f,x,a))
   ∈ inj(LEAST i. HH(f,x,i)={x}, Pow(x)-{0})"
apply (unfold inj_def, simp)
apply (fast intro!: lam_type dest: less_Least_subset_x
        elim!: HH_eq_imp_arg_eq Ord_Least [THEN Ord_in_Ord])
done

```

```

lemma lam_Least_HH_inj:
  "∀a ∈ (LEAST i. HH(f,x,i)={x}). ∃z ∈ x. HH(f,x,a) = {z}
   ==> (λa ∈ (LEAST i. HH(f,x,i)={x}). HH(f,x,a))
   ∈ inj(LEAST i. HH(f,x,i)={x}, {{y}. y ∈ x})"
by (rule lam_Least_HH_inj_Pow [THEN inj_strengthen_type], simp)

```

```

lemma lam_surj_sing:
  "[| x - (⋃a ∈ A. F(a)) = 0; ∀a ∈ A. ∃z ∈ x. F(a) = {z} |]"

```

```

      ==> ( $\lambda a \in A. F(a) \in \text{surj}(A, \{y. y \in x\})$ )"
apply (simp add: surj_def lam_type Diff_eq_0_iff)
apply (blast elim: equalityE)
done

lemma not_emptyI2: " $y \in \text{Pow}(x) - \{0\} \implies x \neq 0$ "
by auto

lemma f_subset_imp_HH_subset:
  " $f'(x - (\bigcup j \in i. \text{HH}(f,x,j))) \in \text{Pow}(x - (\bigcup j \in i. \text{HH}(f,x,j))) - \{0\}$ "
  ==>  $\text{HH}(f, x, i) \in \text{Pow}(x) - \{0\}$ "
apply (rule HH_def_satisfies_eq [THEN ssubst])
apply (simp add: Let_def Diff_subset [THEN PowI] not_emptyI2 [THEN if_P],
fast)
done

lemma f_subsets_imp_UN_HH_eq_x:
  " $\forall z \in \text{Pow}(x) - \{0\}. f'z \in \text{Pow}(z) - \{0\}$ "
  ==>  $x - (\bigcup j \in (\text{LEAST } i. \text{HH}(f,x,i) = \{x\}). \text{HH}(f,x,j)) = 0$ "
apply (case_tac "?P  $\in \{0\}$ ", fast)
apply (drule Diff_subset [THEN PowI, THEN DiffI])
apply (drule bspec, assumption)
apply (drule f_subset_imp_HH_subset)
apply (blast dest!: subst_elem [OF _ HH_Least_eq_x [symmetric]]
elim!: mem_irrefl)
done

lemma HH_values2: " $\text{HH}(f,x,i) = f'(x - (\bigcup j \in i. \text{HH}(f,x,j))) \mid \text{HH}(f,x,i) = \{x\}$ "
apply (rule HH_def_satisfies_eq [THEN ssubst])
apply (simp add: Let_def Diff_subset [THEN PowI])
done

lemma HH_subset_imp_eq:
  " $\text{HH}(f,x,i) \in \text{Pow}(x) - \{0\} \implies \text{HH}(f,x,i) = f'(x - (\bigcup j \in i. \text{HH}(f,x,j)))$ "
apply (rule HH_values2 [THEN disjE], assumption)
apply (fast elim!: equalityE mem_irrefl dest!: singleton_subsetD)
done

lemma f_sing_imp_HH_sing:
  " $[ \mid f \in (\text{Pow}(x) - \{0\}) \rightarrow \{z\}. z \in x;$ "
   $a \in (\text{LEAST } i. \text{HH}(f,x,i) = \{x\}) \mid ] \implies \exists z \in x. \text{HH}(f,x,a) = \{z\}$ "
apply (drule less_Least_subset_x)
apply (frule HH_subset_imp_eq)
apply (drule apply_type)
apply (rule Diff_subset [THEN PowI, THEN DiffI])
apply (fast dest!: HH_subset_x_imp_subset_Diff_UN [THEN not_emptyI2],
force)
done

```

```

lemma f_sing_lam_bij:
  "[| x - ( $\bigcup j \in (\text{LEAST } i. \text{HH}(f,x,i)=\{x\}). \text{HH}(f,x,j)) = 0;$ 
    f  $\in (\text{Pow}(x)-\{0\}) \rightarrow \{\{z\}. z \in x\}$  |]
  ==> ( $\lambda a \in (\text{LEAST } i. \text{HH}(f,x,i)=\{x\}). \text{HH}(f,x,a)$ )
     $\in \text{bij}(\text{LEAST } i. \text{HH}(f,x,i)=\{x\}, \{\{y\}. y \in x\})"$ 
apply (unfold bij_def)
apply (fast intro!: lam_Least_HH_inj lam_surj_sing f_sing_imp_HH_sing)
done

lemma lam_singI:
  "f  $\in (\prod X \in \text{Pow}(x)-\{0\}. F(X))$ 
  ==> ( $\lambda X \in \text{Pow}(x)-\{0\}. \{f'X\} \in (\prod X \in \text{Pow}(x)-\{0\}. \{\{z\}. z \in F(X)\})"$ 
by (fast del: DiffI DiffE
    intro!: lam_type singleton_eq_iff [THEN iffD2] dest: apply_type)

lemmas bij_Least_HH_x =
  comp_bij [OF f_sing_lam_bij [OF _ lam_singI]
    lam_sing_bij [THEN bij_converse_bij], standard]

```

0.3 The proof of AC1 ==_j WO2

```

lemma bijection:
  "f  $\in (\prod X \in \text{Pow}(x) - \{0\}. X)$ 
  ==>  $\exists g. g \in \text{bij}(x, \text{LEAST } i. \text{HH}(\lambda X \in \text{Pow}(x)-\{0\}. \{f'X\}, x, i) =$ 
 $\{x\})"$ 
apply (rule exI)
apply (rule bij_Least_HH_x [THEN bij_converse_bij])
apply (rule f_subsets_imp_UN_HH_eq_x)
apply (intro ballI apply_type)
apply (fast intro: lam_type apply_type del: DiffE, assumption)
apply (fast intro: Pi_weaken_type)
done

lemma AC1_WO2: "AC1 ==> WO2"
apply (unfold AC1_def WO2_def eqpoll_def)
apply (intro allI)
apply (drule_tac x = "Pow(A) - {0}" in spec)
apply (blast dest: bijection)
done

end

```

```

theory AC15_WO6 imports HH Cardinal_aux begin

```

```

lemma lepoll_Sigma: "A≠0 ==> B ≲ A*B"
apply (unfold lepoll_def)
apply (erule not_emptyE)
apply (rule_tac x = "λz ∈ B. <x,z>" in exI)
apply (fast intro!: snd_conv lam_injective)
done

lemma cons_times_nat_not_Finite:
  "0∉A ==> ∀B ∈ {cons(0,x*nat). x ∈ A}. ~Finite(B)"
apply clarify
apply (rule nat_not_Finite [THEN notE] )
apply (subgoal_tac "x ~ 0")
  apply (blast intro: lepoll_Sigma [THEN lepoll_Finite])+
done

lemma lemma1: "[| Union(C)=A; a ∈ A |] ==> ∃B ∈ C. a ∈ B & B ⊆ A"
by fast

lemma lemma2:
  "[| pairwise_disjoint(A); B ∈ A; C ∈ A; a ∈ B; a ∈ C |] ==>
B=C"
by (unfold pairwise_disjoint_def, blast)

lemma lemma3:
  "∀B ∈ {cons(0, x*nat). x ∈ A}. pairwise_disjoint(f`B) &
sets_of_size_between(f`B, 2, n) & Union(f`B)=B
==> ∀B ∈ A. ∃! u. u ∈ f`cons(0, B*nat) & u ⊆ cons(0, B*nat) &

0 ∈ u & 2 ≲ u & u ≲ n"
apply (unfold sets_of_size_between_def)
apply (rule ballI)
apply (erule_tac x="cons(0, B*nat)" in ballE)
  apply (blast dest: lemma1 intro!: lemma2, blast)
done

lemma lemma4: "[| A ≲ i; Ord(i) |] ==> {P(a). a ∈ A} ≲ i"
apply (unfold lepoll_def)
apply (erule exE)
apply (rule_tac x = "λx ∈ RepFun(A,P). LEAST j. ∃a∈A. x=P(a) & f`a=j"

```

```

      in exI)
    apply (rule_tac d = "%y. P (converse (f) 'y) " in lam_injective)
    apply (erule RepFunE)
    apply (frule inj_is_fun [THEN apply_type], assumption)
    apply (fast intro: LeastI2 elim!: Ord_in_Ord inj_is_fun [THEN apply_type])
    apply (erule RepFunE)
    apply (rule LeastI2)
      apply fast
      apply (fast elim!: Ord_in_Ord inj_is_fun [THEN apply_type])
    apply (fast elim: sym left_inverse [THEN ssubst])
  done

```

```

lemma lemma5_1:
  "[| B ∈ A; 2 ≲ u(B) |] ==> (λx ∈ A. {fst(x). x ∈ u(x)-{0}}) 'B ≠
  0"
  apply simp
  apply (fast dest: lepoll_Diff_sing
    elim: lepoll_trans [THEN succ_lepoll_natE] ssubst
    intro!: lepoll_refl)
done

```

```

lemma lemma5_2:
  "[| B ∈ A; u(B) ⊆ cons(0, B*nat) |]
  ==> (λx ∈ A. {fst(x). x ∈ u(x)-{0}}) 'B ⊆ B"
  apply auto
done

```

```

lemma lemma5_3:
  "[| n ∈ nat; B ∈ A; 0 ∈ u(B); u(B) ≲ succ(n) |]
  ==> (λx ∈ A. {fst(x). x ∈ u(x)-{0}}) 'B ≲ n"
  apply simp
  apply (fast elim!: Diff_lepoll [THEN lemma4 [OF _ nat_into_Ord]])
done

```

```

lemma ex_fun_AC13_AC15:
  "[| ∀B ∈ {cons(0, x*nat). x ∈ A}.
    pairwise_disjoint(f'B) &
    sets_of_size_between(f'B, 2, succ(n)) & Union(f'B)=B;

    n ∈ nat |]
  ==> ∃f. ∀B ∈ A. f'B ≠ 0 & f'B ⊆ B & f'B ≲ n"
  by (fast del: subsetI notI
    dest!: lemma3 theI intro!: lemma5_1 lemma5_2 lemma5_3)

```

```

theorem AC10_AC11: "[| n ∈ nat; 1 ≤ n; AC10(n) |] ==> AC11"
by (unfold AC10_def AC11_def, blast)

```

```

theorem AC11_AC12: "AC11 ==> AC12"
by (unfold AC10_def AC11_def AC11_def AC12_def, blast)

```

```

theorem AC12_AC15: "AC12 ==> AC15"
apply (unfold AC12_def AC15_def)
apply (blast del: ballI
      intro!: cons_times_nat_not_Finite ex_fun_AC13_AC15)
done

```

```

lemma OUN_eq_UN: "Ord(x) ==> (⋃ a < x. F(a)) = (⋃ a ∈ x. F(a))"
by (fast intro!: ltI dest!: ltD)

```

```

lemma AC15_W06_aux1:
  "∀ x ∈ Pow(A) - {0}. f'x ≠ 0 & f'x ⊆ x & f'x ≲ m
  ==> (⋃ i < LEAST x. HH(f,A,x)={A}. HH(f,A,i)) = A"
apply (simp add: Ord_Least [THEN OUN_eq_UN])
apply (rule equalityI)
apply (fast dest!: less_Least_subset_x)
apply (blast del: subsetI
      intro!: f_subsets_imp_UN_HH_eq_x [THEN Diff_eq_0_iff [THEN
iffD1]])
done

```

```

lemma AC15_W06_aux2:
  "∀ x ∈ Pow(A) - {0}. f'x ≠ 0 & f'x ⊆ x & f'x ≲ m
  ==> ∀ x < (LEAST x. HH(f,A,x)={A}). HH(f,A,x) ≲ m"
apply (rule oallI)
apply (drule ltD [THEN less_Least_subset_x])
apply (frule HH_subset_imp_eq)
apply (erule ssubst)

```

```

apply (blast dest!: HH_subset_x_imp_subset_Diff_UN [THEN not_emptyI2])

done

theorem AC15_W06: "AC15 ==> W06"
apply (unfold AC15_def W06_def)
apply (rule allI)
apply (erule_tac x = "Pow (A) -{0}" in allE)
apply (erule impE, fast)
apply (elim bexE conjE exE)
apply (rule bexI)
  apply (rule conjI, assumption)
  apply (rule_tac x = "LEAST i. HH (f,A,i) ={A}" in exI)
  apply (rule_tac x = "\lambda j \in (LEAST i. HH (f,A,i) ={A}) . HH (f,A,j) "
in exI)
  apply (simp_all add: ltD)
apply (fast intro!: Ord_Least lam_type [THEN domain_of_fun]
      elim!: less_Least_subset_x AC15_W06_aux1 AC15_W06_aux2)

done

```

```

theorem AC10_AC13: "[| n \in nat; 1 \le n; AC10(n) |] ==> AC13(n)"
apply (unfold AC10_def AC13_def, safe)
apply (erule allE)
apply (erule impE [OF _ cons_times_nat_not_Finite], assumption)
apply (fast elim!: impE [OF _ cons_times_nat_not_Finite]
      dest!: ex_fun_AC13_AC15)

done

```

```

lemma AC1_AC13: "AC1 ==> AC13(1)"
apply (unfold AC1_def AC13_def)
apply (rule allI)
apply (erule allE)
apply (rule impI)
apply (drule mp, assumption)
apply (elim exE)
apply (rule_tac x = " $\lambda x \in A. \{f'x\}$ " in exI)
apply (simp add: singleton_eqpoll_1 [THEN eqpoll_imp_lepoll])
done

```

```

lemma AC13_mono: "[| m ≤ n; AC13(m) |] ==> AC13(n)"
apply (unfold AC13_def)
apply (drule le_imp_lepoll)
apply (fast elim!: lepoll_trans)
done

```

```

theorem AC13_AC14: "[| n ∈ nat; 1 ≤ n; AC13(n) |] ==> AC14"
by (unfold AC13_def AC14_def, auto)

```

```

theorem AC14_AC15: "AC14 ==> AC15"
by (unfold AC13_def AC14_def AC15_def, fast)

```

```

lemma lemma_aux: "[| A ≠ 0; A ≲ 1 |] ==> ∃ a. A = {a}"
by (fast elim!: not_emptyE lepoll_1_is_sing)

```

```

lemma AC13_AC1_lemma:
  "∀B ∈ A. f(B) ≠ 0 & f(B) ≤ B & f(B) ≲ 1
   ==> (λx ∈ A. THE y. f(x) = {y}) ∈ (Π X ∈ A. X)"
apply (rule lam_type)
apply (drule bspec, assumption)
apply (elim conjE)
apply (erule lemma_aux [THEN exE], assumption)
apply (simp add: the_equality)
done

theorem AC13_AC1: "AC13(1) ==> AC1"
apply (unfold AC13_def AC1_def)
apply (fast elim!: AC13_AC1_lemma)
done

theorem AC11_AC14: "AC11 ==> AC14"
apply (unfold AC11_def AC14_def)
apply (fast intro!: AC10_AC13)
done

end

theory AC16_lemmas imports AC_Equiv Hartog Cardinal_aux begin

lemma cons_Diff_eq: "a ∉ A ==> cons(a,A) - {a} = A"
by fast

lemma nat_1_lepoll_iff: "1 ≲ X <-> (∃ x. x ∈ X)"
apply (unfold lepoll_def)
apply (rule iffI)
apply (fast intro: inj_is_fun [THEN apply_type])
apply (erule exE)
apply (rule_tac x = "λa ∈ 1. x" in exI)
apply (fast intro!: lam_injective)
done

lemma eqpoll_1_iff_singleton: "X ≈ 1 <-> (∃ x. X = {x})"
apply (rule iffI)
apply (erule eqpollE)
apply (drule nat_1_lepoll_iff [THEN iffD1])
apply (fast intro!: lepoll_1_is_sing)

```

```

apply (fast intro!: singleton_eqpoll_1)
done

lemma cons_eqpoll_succ: "[| x≈n; y∉x |] ==> cons(y,x)≈succ(n)"
apply (unfold succ_def)
apply (fast elim!: cons_eqpoll_cong mem_irrefl)
done

lemma subsets_eqpoll_1_eq: "{Y ∈ Pow(X). Y≈1} = {{x}. x ∈ X}"
apply (rule equalityI)
apply (rule subsetI)
apply (erule CollectE)
apply (drule eqpoll_1_iff_singleton [THEN iffD1])
apply (fast intro!: RepFunI)
apply (rule subsetI)
apply (erule RepFunE)
apply (rule CollectI, fast)
apply (fast intro!: singleton_eqpoll_1)
done

lemma eqpoll_RepFun_sing: "X≈{{x}. x ∈ X}"
apply (unfold eqpoll_def bij_def)
apply (rule_tac x = "λx ∈ X. {x}" in exI)
apply (rule IntI)
apply (unfold inj_def surj_def, simp)
apply (fast intro!: lam_type RepFunI intro: singleton_eq_iff [THEN iffD1],
simp)
apply (fast intro!: lam_type)
done

lemma subsets_eqpoll_1_eqpoll: "{Y ∈ Pow(X). Y≈1}≈X"
apply (rule subsets_eqpoll_1_eq [THEN ssubst])
apply (rule eqpoll_RepFun_sing [THEN eqpoll_sym])
done

lemma InfCard_Least_in:
  "[| InfCard(x); y ⊆ x; y ≈ succ(z) |] ==> (LEAST i. i ∈ y) ∈ y"
apply (erule eqpoll_sym [THEN eqpoll_imp_lepoll,
  THEN succ_lepoll_imp_not_empty, THEN not_emptyE])
apply (fast intro: LeastI
  dest!: InfCard_is_Card [THEN Card_is_Ord]
  elim: Ord_in_Ord)
done

lemma subsets_lepoll_lemma1:
  "[| InfCard(x); n ∈ nat |]
  ==> {y ∈ Pow(x). y≈succ(succ(n))} ≲ x*{y ∈ Pow(x). y≈succ(n)}"
apply (unfold lepoll_def)
apply (rule_tac x = "λy ∈ {y ∈ Pow(x) . y≈succ (succ (n))}."

```

```

      <LEAST i. i ∈ y, y-{LEAST i. i ∈ y}>" in exI)
apply (rule_tac d = "%z. cons (fst(z), snd(z))" in lam_injective)
  apply (blast intro!: Diff_sing_eqpoll intro: InfCard_Least_in)
apply (simp, blast intro: InfCard_Least_in)
done

lemma set_of_Ord_succ_Union: "(∀y ∈ z. Ord(y)) ==> z ⊆ succ(Union(z))"
apply (rule subsetI)
apply (case_tac "∀y ∈ z. y ⊆ x", blast )
apply (simp, erule bexE)
apply (rule_tac i=y and j=x in Ord_linear_le)
apply (blast dest: le_imp_subset elim: leE ltE)+
done

lemma subset_not_mem: "j ⊆ i ==> i ∉ j"
by (fast elim!: mem_irrefl)

lemma succ_Union_not_mem:
  "(!y. y ∈ z ==> Ord(y)) ==> succ(Union(z)) ∉ z"
apply (rule set_of_Ord_succ_Union [THEN subset_not_mem], blast)
done

lemma Union_cons_eq_succ_Union:
  "Union(cons(succ(Union(z)),z)) = succ(Union(z))"
by fast

lemma Un_Ord_disj: "[| Ord(i); Ord(j) |] ==> i Un j = i | i Un j = j"
by (fast dest!: le_imp_subset elim: Ord_linear_le)

lemma Union_eq_Un: "x ∈ X ==> Union(X) = x Un Union(X-{x})"
by fast

lemma Union_in_lemma [rule_format]:
  "n ∈ nat ==> ∀z. (∀y ∈ z. Ord(y)) & z ≈ n & z ≠ 0 --> Union(z) ∈ z"
apply (induct_tac "n")
apply (fast dest!: eqpoll_imp_lepoll [THEN lepoll_0_is_0])
apply (intro allI impI)
apply (erule natE)
apply (fast dest!: eqpoll_1_iff_singleton [THEN iffD1]
  intro!: Union_singleton, clarify)
apply (elim not_emptyE)
apply (erule_tac x = "z-{xb}" in allE)
apply (erule impE)
apply (fast elim!: Diff_sing_eqpoll
  Diff_sing_eqpoll [THEN eqpoll_succ_imp_not_empty])
apply (subgoal_tac "xb ∪ ⋃ (z - {xb}) ∈ z")
apply (simp add: Union_eq_Un [symmetric])
apply (frule bspec, assumption)

```

```

apply (drule bspec)
apply (erule Diff_subset [THEN subsetD])
apply (drule Un_Ord_disj, assumption, auto)
done

lemma Union_in: "[|  $\forall x \in z. \text{Ord}(x); z \approx n; z \neq 0; n \in \text{nat}$  |] ==> Union(z)
 $\in z$ "
by (blast intro: Union_in_lemma)

lemma succ_Union_in_x:
  "[| InfCard(x); z  $\in$  Pow(x); z  $\approx$  n; n  $\in$  nat |] ==> succ(Union(z))
 $\in x$ "
apply (rule Limit_has_succ [THEN ltE])
prefer 3 apply assumption
apply (erule InfCard_is_Limit)
apply (case_tac "z=0")
apply (simp, fast intro!: InfCard_is_Limit [THEN Limit_has_0])
apply (rule ltI [OF PowD [THEN subsetD] InfCard_is_Card [THEN Card_is_Ord]],
assumption)
apply (blast intro: Union_in
          InfCard_is_Card [THEN Card_is_Ord, THEN Ord_in_Ord])
done

lemma succ_lepoll_succ_succ:
  "[| InfCard(x); n  $\in$  nat |]
  ==> {y  $\in$  Pow(x). y  $\approx$  succ(n)}  $\lesssim$  {y  $\in$  Pow(x). y  $\approx$  succ(succ(n))}"
apply (unfold lepoll_def)
apply (rule_tac x = " $\lambda z \in \{y \in \text{Pow}(x). y \approx \text{succ}(n)\}. \text{cons}(\text{succ}(\text{Union}(z)), z)$ "
  in exI)
apply (rule_tac d = "%z. z-{Union (z) }" in lam_injective)
apply (blast intro!: succ_Union_in_x succ_Union_not_mem
  intro: cons_eqpoll_succ Ord_in_Ord
  dest!: InfCard_is_Card [THEN Card_is_Ord])
apply (simp only: Union_cons_eq_succ_Union)
apply (rule cons_Diff_eq)
apply (fast dest!: InfCard_is_Card [THEN Card_is_Ord]
  elim: Ord_in_Ord
  intro!: succ_Union_not_mem)
done

lemma subsets_eqpoll_X:
  "[| InfCard(X); n  $\in$  nat |] ==> {Y  $\in$  Pow(X). Y  $\approx$  succ(n)}  $\approx$  X"
apply (induct_tac "n")
apply (rule subsets_eqpoll_1_eqpoll)
apply (rule eqpollI)
apply (rule subsets_lepoll_lemma1 [THEN lepoll_trans], assumption+)
apply (rule eqpoll_trans [THEN eqpoll_imp_lepoll])
apply (erule eqpoll_refl [THEN prod_eqpoll_cong])

```

```

apply (erule InfCard_square_eqpoll)
apply (fast elim: eqpoll_sym [THEN eqpoll_imp_lepoll, THEN lepoll_trans]

      intro!: succ_lepoll_succ_succ)
done

lemma image_vimage_eq:
  "[| f ∈ surj(A,B); y ⊆ B |] ==> f``(converse(f)``y) = y"
apply (unfold surj_def)
apply (fast dest: apply_equality2 elim: apply_iff [THEN iffD2])
done

lemma vimage_image_eq: "[| f ∈ inj(A,B); y ⊆ A |] ==> converse(f)``(f``y)
= y"
by (fast elim!: inj_is_fun [THEN apply_Pair] dest: inj_equality)

lemma subsets_eqpoll:
  "A ≈ B ==> {Y ∈ Pow(A). Y ≈ n} ≈ {Y ∈ Pow(B). Y ≈ n}"
apply (unfold eqpoll_def)
apply (erule exE)
apply (rule_tac x = "λX ∈ {Y ∈ Pow (A) . ∃f. f ∈ bij (Y, n) }. f``X"
in exI)
apply (rule_tac d = "%Z. converse (f) ``Z" in lam_bijective)
apply (fast intro!: bij_is_inj [THEN restrict_bij, THEN bij_converse_bij,

      THEN comp_bij]
      elim!: bij_is_fun [THEN fun_is_rel, THEN image_subset])
apply (blast intro!: bij_is_inj [THEN restrict_bij]
      comp_bij bij_converse_bij
      bij_is_fun [THEN fun_is_rel, THEN image_subset])
apply (fast elim!: bij_is_inj [THEN vimage_image_eq])
apply (fast elim!: bij_is_surj [THEN image_vimage_eq])
done

lemma W02_imp_ex_Card: "W02 ==> ∃ a. Card(a) & X ≈ a"
apply (unfold W02_def)
apply (drule spec [of _ X])
apply (blast intro: Card_cardinal eqpoll_trans
      well_ord_Memrel [THEN well_ord_cardinal_eqpoll, THEN eqpoll_sym])
done

lemma lepoll_infinite: "[| X ≲ Y; ~Finite(X) |] ==> ~Finite(Y)"
by (blast intro: lepoll_Finite)

lemma infinite_Card_is_InfCard: "[| ~Finite(X); Card(X) |] ==> InfCard(X)"
apply (unfold InfCard_def)
apply (fast elim!: Card_is_Ord [THEN nat_le_infinite_Ord])
done

```

```

lemma W02_infinite_subsets_eqpoll_X: "[| W02; n ∈ nat; ~Finite(X) |]
    ==> {Y ∈ Pow(X). Y≈succ(n)}≈X"
apply (drule W02_imp_ex_Card)
apply (elim allE exE conjE)
apply (frule eqpoll_imp_lepoll [THEN lepoll_infinite], assumption)
apply (drule infinite_Card_is_InfCard, assumption)
apply (blast intro: subsets_eqpoll subsets_eqpoll_X eqpoll_sym eqpoll_trans)

done

lemma well_ord_imp_ex_Card: "well_ord(X,R) ==> ∃a. Card(a) & X≈a"
by (fast elim!: well_ord_cardinal_eqpoll [THEN eqpoll_sym]
    intro!: Card_cardinal)

lemma well_ord_infinite_subsets_eqpoll_X:
    "[| well_ord(X,R); n ∈ nat; ~Finite(X) |] ==> {Y ∈ Pow(X). Y≈succ(n)}≈X"
apply (drule well_ord_imp_ex_Card)
apply (elim allE exE conjE)
apply (frule eqpoll_imp_lepoll [THEN lepoll_infinite], assumption)
apply (drule infinite_Card_is_InfCard, assumption)
apply (blast intro: subsets_eqpoll subsets_eqpoll_X eqpoll_sym eqpoll_trans)

done

end

theory W02_AC16 imports AC_Equiv AC16_lemmas Cardinal_aux begin

constdefs
  recfunAC16 :: "[i,i,i,i] => i"
    "recfunAC16(f,h,i,a) ==
      transrec2(i, 0,
        %g r. if (∃y ∈ r. h'g ⊆ y) then r
              else r Un {f'(LEAST i. h'g ⊆ f'i &
                (∀b<a. (h'b ⊆ f'i --> (∀t ∈ r. ~ h'b ⊆ t)))})}"

lemma recfunAC16_0: "recfunAC16(f,h,0,a) = 0"
by (simp add: recfunAC16_def)

lemma recfunAC16_succ:

```

```

"recfunAC16(f,h,succ(i),a) =
  (if ( $\exists y \in \text{recfunAC16}(f,h,i,a). h \text{ ' } i \subseteq y$ ) then  $\text{recfunAC16}(f,h,i,a)$ 

    else  $\text{recfunAC16}(f,h,i,a)$  Un
      {f ' (LEAST j. h ' i  $\subseteq$  f ' j &
        ( $\forall b < a. (h \text{ ' } b \subseteq f \text{ ' } j$ 
          --> ( $\forall t \in \text{recfunAC16}(f,h,i,a). \sim h \text{ ' } b \subseteq t$ ))))})}"
apply (simp add: recfunAC16_def)
done

lemma recfunAC16_Limit: "Limit(i)
  ==>  $\text{recfunAC16}(f,h,i,a) = (\bigcup_{j < i}. \text{recfunAC16}(f,h,j,a))$ "
by (simp add: recfunAC16_def transrec2_Limit)

lemma transrec2_mono_lemma [rule_format]:
  "[| !!g r. r  $\subseteq$  B(g,r); Ord(i) |]
  ==>  $j < i \text{ --> } \text{transrec2}(j, 0, B) \subseteq \text{transrec2}(i, 0, B)$ "
apply (erule trans_induct)
apply (rule Ord_cases, assumption+, fast)
apply (simp (no_asm_simp))
apply (blast elim!: leE)
apply (simp add: transrec2_Limit)
apply (blast intro: OUN_I ltI Ord_in_Ord [THEN le_refl]
  elim!: Limit_has_succ [THEN ltE])
done

lemma transrec2_mono:
  "[| !!g r. r  $\subseteq$  B(g,r);  $j \leq i$  |]
  ==>  $\text{transrec2}(j, 0, B) \subseteq \text{transrec2}(i, 0, B)$ "
apply (erule leE)
apply (rule transrec2_mono_lemma)
apply (auto intro: lt_Ord2 )
done

lemma recfunAC16_mono:
  " $i \leq j \text{ ==> } \text{recfunAC16}(f, g, i, a) \subseteq \text{recfunAC16}(f, g, j, a)$ "
apply (unfold recfunAC16_def)
apply (rule transrec2_mono, auto)
done

```

```

lemma lemma3_1:
  "[|  $\forall y < x. \forall z < a. z < y \mid (\exists Y \in F(y). f(z) \leq Y) \rightarrow (\exists! Y. Y \in F(y) \& f(z) \leq Y)$ ;
     $\forall i j. i \leq j \rightarrow F(i) \subseteq F(j); j \leq i; i < x; z < a;$ 
     $V \in F(i); f(z) \leq V; W \in F(j); f(z) \leq W \mid]$ 
  ==>  $V = W$ "
apply (erule asm_rl allE impE)+
apply (drule subsetD, assumption, blast)
done

```

```

lemma lemma3:
  "[|  $\forall y < x. \forall z < a. z < y \mid (\exists Y \in F(y). f(z) \leq Y) \rightarrow (\exists! Y. Y \in F(y) \& f(z) \leq Y)$ ;
     $\forall i j. i \leq j \rightarrow F(i) \subseteq F(j); i < x; j < x; z < a;$ 
     $V \in F(i); f(z) \leq V; W \in F(j); f(z) \leq W \mid]$ 
  ==>  $V = W$ "
apply (rule_tac j=j in Ord_linear_le [OF lt_Ord lt_Ord], assumption+)
apply (erule lemma3_1 [symmetric], assumption+)
apply (erule lemma3_1, assumption+)
done

```

```

lemma lemma4:
  "[|  $\forall y < x. F(y) \subseteq X \&$ 
     $(\forall x < a. x < y \mid (\exists Y \in F(y). h(x) \subseteq Y) \rightarrow$ 
     $(\exists! Y. Y \in F(y) \& h(x) \subseteq Y));$ 
     $x < a \mid]$ 
  ==>  $\forall y < x. \forall z < a. z < y \mid (\exists Y \in F(y). h(z) \subseteq Y) \rightarrow$ 
     $(\exists! Y. Y \in F(y) \& h(z) \subseteq Y)$ "
apply (intro oallI impI)
apply (drule ospec, assumption, clarify)
apply (blast elim!: oallE )
done

```

```

lemma lemma5:
  "[|  $\forall y < x. F(y) \subseteq X \&$ 
     $(\forall x < a. x < y \mid (\exists Y \in F(y). h(x) \subseteq Y) \rightarrow$ 
     $(\exists! Y. Y \in F(y) \& h(x) \subseteq Y));$ 
     $x < a; \text{Limit}(x); \forall i j. i \leq j \rightarrow F(i) \subseteq F(j) \mid]$ 
  ==>  $(\bigcup x < x. F(x)) \subseteq X \&$ 
     $(\forall xa < a. xa < x \mid (\exists x \in \bigcup x < x. F(x). h(xa) \subseteq x)$ 
     $\rightarrow (\exists! Y. Y \in (\bigcup x < x. F(x)) \& h(xa) \subseteq Y))$ "
apply (rule conjI)
apply (rule subsetI)
apply (erule OUN_E)

```

```

apply (drule ospec, assumption, fast)
apply (drule lemma4, assumption)
apply (rule oallI)
apply (rule impI)
apply (erule disjE)
apply (frule ospec, erule Limit_has_succ, assumption)
apply (drule_tac A = a and x = xa in ospec, assumption)
apply (erule impE, rule le_refl [THEN disjI1], erule lt_Ord)
apply (blast intro: lemma3 Limit_has_succ)
apply (blast intro: lemma3)
done

```

```

lemma dbl_Diff_eqpoll_Card:
  "[| A≈a; Card(a); ~Finite(a); B<a; C<a |] ==> A - B - C≈a"
by (blast intro: Diff_lesspoll_eqpoll_Card)

```

```

lemma Finite_lesspoll_infinite_Ord:
  "[| Finite(X); ~Finite(a); Ord(a) |] ==> X<a"
apply (unfold lesspoll_def)
apply (rule conjI)
apply (drule nat_le_infinite_Ord [THEN le_imp_lepoll], assumption)
apply (unfold Finite_def)
apply (blast intro: leI [THEN le_imp_subset, THEN subset_imp_lepoll]
      ltI eqpoll_imp_lepoll lepoll_trans)
apply (blast intro: eqpoll_sym [THEN eqpoll_trans])
done

```

```

lemma Union_lesspoll:
  "[| ∀x ∈ X. x lepoll n & x ⊆ T; well_ord(T, R); X lepoll b;
    b<a; ~Finite(a); Card(a); n ∈ nat |]
  ==> Union(X) <a"
apply (case_tac "Finite (X)")
apply (blast intro: Card_is_Ord Finite_lesspoll_infinite_Ord)

```

```

                                lepoll_nat_imp_Finite Finite_Union)
apply (drule lepoll_imp_ex_le_eqpoll)
apply (erule lt_Ord)
apply (elim exE conjE)
apply (frule eqpoll_imp_lepoll [THEN lepoll_infinite], assumption)
apply (erule eqpoll_sym [THEN eqpoll_def [THEN def_imp_iff, THEN iffD1],
                                THEN exE])
apply (frule bij_is_surj [THEN surj_image_eq])
apply (drule image_fun [OF bij_is_fun subset_refl])
apply (drule sym [THEN trans], assumption)
apply (blast intro: lt_Ord UN_lepoll lt_Card_imp_lesspoll
                                lt_trans1 lesspoll_trans1)
done

lemma Un_sing_eq_cons: "A Un {a} = cons(a, A)"
by fast

lemma Un_lepoll_succ: "A lepoll B ==> A Un {a} lepoll succ(B)"
apply (simp add: Un_sing_eq_cons succ_def)
apply (blast elim!: mem_irrefl intro: cons_lepoll_cong)
done

lemma Diff_UN_succ_empty: "Ord(a) ==> F(a) - ( $\bigcup_{b < \text{succ}(a)} F(b)$ ) = 0"
by (fast intro!: le_refl)

lemma Diff_UN_succ_subset: "Ord(a) ==> F(a) Un X - ( $\bigcup_{b < \text{succ}(a)} F(b)$ )
 $\subseteq X$ "
by blast

lemma recfunAC16_Diff_lepoll_1:
  "Ord(x)
  ==> recfunAC16(f, g, x, a) - ( $\bigcup_{i < x} \text{recfunAC16}(f, g, i, a)$ ) lepoll
  1"
apply (erule Ord_cases)
  apply (simp add: recfunAC16_0 empty_subsetI [THEN subset_imp_lepoll])

prefer 2 apply (simp add: recfunAC16_Limit Diff_cancel
                                empty_subsetI [THEN subset_imp_lepoll])

apply (simp add: recfunAC16_succ
                                Diff_UN_succ_empty [of _ "%j. recfunAC16(f,g,j,a)"]
                                empty_subsetI [THEN subset_imp_lepoll])
apply (best intro: Diff_UN_succ_subset [THEN subset_imp_lepoll]
                                singleton_eqpoll_1 [THEN eqpoll_imp_lepoll] lepoll_trans)
done

```

```

lemma in_Least_Diff:
  "[| z ∈ F(x); Ord(x) |]"
  ==> z ∈ F(LEAST i. z ∈ F(i)) - (⋃j<(LEAST i. z ∈ F(i)). F(j))"
by (fast elim: less_LeastE elim!: LeastI)

lemma Least_eq_imp_ex:
  "[| (LEAST i. w ∈ F(i)) = (LEAST i. z ∈ F(i));
    w ∈ (⋃i<a. F(i)); z ∈ (⋃i<a. F(i)) |]"
  ==> ∃b<a. w ∈ (F(b) - (⋃c<b. F(c))) & z ∈ (F(b) - (⋃c<b. F(c)))"
apply (elim OUN_E)
apply (drule in_Least_Diff, erule lt_Ord)
apply (frule in_Least_Diff, erule lt_Ord)
apply (rule oexI, force)
apply (blast intro: lt_Ord Least_le [THEN lt_trans1])
done

lemma two_in_lepoll_1: "[| A lepoll 1; a ∈ A; b ∈ A |] ==> a=b"
by (fast dest!: lepoll_1_is_sing)

lemma UN_lepoll_index:
  "[| ∀i<a. F(i)-(⋃j<i. F(j)) lepoll 1; Limit(a) |]"
  ==> (⋃x<a. F(x)) lepoll a"
apply (rule lepoll_def [THEN def_imp_iff [THEN iffD2]])
apply (rule_tac x = "λz ∈ (⋃x<a. F(x)). LEAST i. z ∈ F(i)" in exI)
apply (unfold inj_def)
apply (rule CollectI)
apply (rule lam_type)
apply (erule OUN_E)
apply (erule Least_in_Ord)
apply (erule ltD)
apply (erule lt_Ord2)
apply (intro ballI)
apply (simp (no_asm_simp))
apply (rule impI)
apply (drule Least_eq_imp_ex, assumption+)
apply (fast elim!: two_in_lepoll_1)
done

lemma recfunAC16_lepoll_index: "Ord(y) ==> recfunAC16(f, h, y, a) lepoll
y"
apply (erule trans_induct3)

apply (simp (no_asm_simp) add: recfunAC16_0 lepoll_refl)

apply (simp (no_asm_simp) add: recfunAC16_succ)

```

```

apply (blast dest!: succI1 [THEN rev_bspec]
      intro: subset_succI [THEN subset_imp_lepoll] Un_lepoll_succ
      lepoll_trans)
apply (simp (no_asm_simp) add: recfunAC16_Limit)
apply (blast intro: lt_Ord [THEN recfunAC16_Diff_lepoll_1] UN_lepoll_index)
done

```

```

lemma Union_recfunAC16_lesspoll:
  "[| recfunAC16(f,g,y,a)  $\subseteq$  {X  $\in$  Pow(A). X $\approx$ n};
    A $\approx$ a; y<a;  $\sim$ Finite(a); Card(a); n  $\in$  nat |]
  ==> Union(recfunAC16(f,g,y,a)) <a"
apply (erule eqpoll_def [THEN def_imp_iff, THEN iffD1, THEN exE])
apply (rule_tac T=A in Union_lesspoll, simp_all)
apply (blast intro!: eqpoll_imp_lepoll)
apply (blast intro: bij_is_inj Card_is_Ord [THEN well_ord_Memrel]
      well_ord_rvimage)
apply (erule lt_Ord [THEN recfunAC16_lepoll_index])
done

```

```

lemma dbl_Diff_eqpoll:
  "[| recfunAC16(f, h, y, a)  $\subseteq$  {X  $\in$  Pow(A) . X $\approx$ succ(k #+ m)};
    Card(a);  $\sim$  Finite(a); A $\approx$ a;
    k  $\in$  nat; y<a;
    h  $\in$  bij(a, {Y  $\in$  Pow(A). Y $\approx$ succ(k)}) |]
  ==> A - Union(recfunAC16(f, h, y, a)) - h'y $\approx$ a"
apply (rule dbl_Diff_eqpoll_Card, simp_all)
apply (simp add: Union_recfunAC16_lesspoll)
apply (rule Finite_lesspoll_infinite_Ord)
apply (rule Finite_def [THEN def_imp_iff, THEN iffD2])
apply (blast dest: ltD bij_is_fun [THEN apply_type], assumption)
apply (blast intro: Card_is_Ord)
done

```

```

lemmas disj_Un_eqpoll_nat_sum =
  eqpoll_trans [THEN eqpoll_trans,
               OF disj_Un_eqpoll_sum sum_eqpoll_cong nat_sum_eqpoll_sum,
               standard]

```

```

lemma Un_in_Collect: "[| x  $\in$  Pow(A - B - h'i); x $\approx$ m;
  h  $\in$  bij(a, {x  $\in$  Pow(A) . x $\approx$ k}); i<a; k  $\in$  nat; m  $\in$  nat |]
  ==> h ' i Un x  $\in$  {x  $\in$  Pow(A) . x $\approx$ k #+ m}"
by (blast intro: disj_Un_eqpoll_nat_sum
    dest: ltD bij_is_fun [THEN apply_type])

```

```

lemma lemma6:
  "[|  $\forall y < \text{succ}(j). F(y) \leq X$  &  $(\forall x < a. x < y \mid P(x,y) \rightarrow Q(x,y))$ ;  $\text{succ}(j) < a$ 
  |]
  ==>  $F(j) \leq X$  &  $(\forall x < a. x < j \mid P(x,j) \rightarrow Q(x,j))"$ 
by (blast intro!: lt_Ord succI1 [THEN ltI, THEN lt_Ord, THEN le_refl])

```

```

lemma lemma7:
  "[|  $\forall x < a. x < j \mid P(x,j) \rightarrow Q(x,j)$ ;  $\text{succ}(j) < a$  |]
  ==>  $P(j,j) \rightarrow (\forall x < a. x \leq j \mid P(x,j) \rightarrow Q(x,j))"$ 
by (fast elim!: leE)

```

```

lemma ex_subset_eqpoll:
  "[|  $A \approx a$ ;  $\sim \text{Finite}(a)$ ;  $\text{Ord}(a)$ ;  $m \in \text{nat}$  |] ==>  $\exists X \in \text{Pow}(A). X \approx^m$ "
apply (rule lepoll_imp_eqpoll_subset [of m A, THEN exE])
  apply (rule lepoll_trans, rule leI [THEN le_imp_lepoll])
  apply (blast intro: lt_trans2 [OF ltI nat_le_infinite_Ord] Ord_nat)
apply (erule eqpoll_sym [THEN eqpoll_imp_lepoll])
apply (fast elim!: eqpoll_sym)
done

```

```

lemma subset_Un_disjoint: "[|  $A \subseteq B \cup C$ ;  $A \cap C = 0$  |] ==>  $A \subseteq B$ "
by blast

```

```

lemma Int_empty:
  "[|  $X \in \text{Pow}(A - \text{Union}(B) - C)$ ;  $T \in B$ ;  $F \subseteq T$  |] ==>  $F \cap X = 0$ "
by blast

```

```

lemma subset_imp_eq_lemma:
  " $m \in \text{nat} \implies \forall A B. A \subseteq B$  &  $m \text{ lepoll } A$  &  $B \text{ lepoll } m \rightarrow A=B$ "
apply (induct_tac "m")

```

```

apply (fast dest!: lepoll_0_is_0)
apply (intro allI impI)
apply (elim conjE)
apply (rule succ_lepoll_imp_not_empty [THEN not_emptyE], assumption)
apply (frule subsetD [THEN Diff_sing_lepoll], assumption+)
apply (frule lepoll_Diff_sing)
apply (erule allE impE)+
apply (rule conjI)
prefer 2 apply fast
apply fast
apply (blast elim: equalityE)
done

lemma subset_imp_eq: "[| A ⊆ B; m lepoll A; B lepoll m; m ∈ nat |] ==>
A=B"
by (blast dest!: subset_imp_eq_lemma)

lemma bij_imp_arg_eq:
  "[| f ∈ bij(a, {Y ∈ X. Y≈succ(k)}); k ∈ nat; f' b ⊆ f'y; b<a; y<a
|]
  ==> b=y"
apply (drule subset_imp_eq)
apply (erule_tac [3] nat_succI)
apply (unfold bij_def inj_def)
apply (blast intro: eqpoll_sym eqpoll_imp_lepoll
  dest: ltD apply_type)+
done

lemma ex_next_set:
  "[| recfunAC16(f, h, y, a) ⊆ {X ∈ Pow(A) . X≈succ(k #+ m)};
  Card(a); ~ Finite(a); A≈a;
  k ∈ nat; m ∈ nat; y<a;
  h ∈ bij(a, {Y ∈ Pow(A). Y≈succ(k)});
  ~ (∃ Y ∈ recfunAC16(f, h, y, a). h'y ⊆ Y) |]
  ==> ∃ X ∈ {Y ∈ Pow(A). Y≈succ(k #+ m)}. h'y ⊆ X &
  (∀ b<a. h'b ⊆ X -->
  (∀ T ∈ recfunAC16(f, h, y, a). ~ h'b ⊆ T))"
apply (erule_tac m1=m in dbl_Diff_eqpoll [THEN ex_subset_eqpoll, THEN
bexE],
  assumption+)
apply (erule Card_is_Ord, assumption)
apply (frule Un_in_Collect, (erule asm_rl nat_succI)+)
apply (erule CollectE)
apply (rule rev_bexI, simp)
apply (rule conjI, blast)
apply (intro ballI impI oallI notI)

```

```

apply (drule subset_Un_disjoint, rule Int_empty, assumption+)
apply (blast dest: bij_imp_arg_eq)
done

```

```

lemma ex_next_Ord:
  "[| recfunAC16(f, h, y, a)  $\subseteq$  {X  $\in$  Pow(A) . X $\approx$ succ(k #+ m)};
    Card(a);  $\sim$  Finite(a); A $\approx$ a;
    k  $\in$  nat; m  $\in$  nat; y<a;
    h  $\in$  bij(a, {Y  $\in$  Pow(A). Y $\approx$ succ(k)});
    f  $\in$  bij(a, {Y  $\in$  Pow(A). Y $\approx$ succ(k #+ m)});
     $\sim$  ( $\exists$ Y  $\in$  recfunAC16(f, h, y, a). h'y  $\subseteq$  Y) |]
  ==>  $\exists$ c<a. h'y  $\subseteq$  f'c &
    ( $\forall$ b<a. h'b  $\subseteq$  f'c -->
      ( $\forall$ T  $\in$  recfunAC16(f, h, y, a).  $\sim$  h'b  $\subseteq$  T))"
apply (drule ex_next_set, assumption+)
apply (erule bexE)
apply (rule_tac x="converse(f)'X" in oexI)
apply (simp add: right_inverse_bij)
apply (blast intro: bij_converse_bij bij_is_fun [THEN apply_type] ltI
      Card_is_Ord)
done

```

```

lemma lemma8:
  "[|  $\forall$ x<a. x<j | ( $\exists$ xa  $\in$  F(j). P(x, xa))
    --> ( $\exists!$  Y. Y  $\in$  F(j) & P(x, Y)); F(j)  $\subseteq$  X;
    L  $\in$  X; P(j, L) & ( $\forall$ x<a. P(x, L) --> ( $\forall$ xa  $\in$  F(j).  $\sim$ P(x, xa)))
  |]
  ==> F(j) Un {L}  $\subseteq$  X &
    ( $\forall$ x<a. x $\leq$ j | ( $\exists$ xa  $\in$  (F(j) Un {L}). P(x, xa)) -->
      ( $\exists!$  Y. Y  $\in$  (F(j) Un {L}) & P(x, Y)))"
apply (rule conjI)
apply (fast intro!: singleton_subsetI)
apply (rule oallI)
apply (blast elim!: leE oallE)
done

```

```

lemma main_induct:
  "[| b < a; f ∈ bij(a, {Y ∈ Pow(A) . Y ≈ succ(k #+ m)});
    h ∈ bij(a, {Y ∈ Pow(A) . Y ≈ succ(k)});
    ~Finite(a); Card(a); A ≈ a; k ∈ nat; m ∈ nat |]
  ==> recfunAC16(f, h, b, a) ⊆ {X ∈ Pow(A) . X ≈ succ(k #+ m)} &
    (∀ x < a. x < b | (∃ Y ∈ recfunAC16(f, h, b, a). h ` x ⊆ Y) -->
      (∃ ! Y. Y ∈ recfunAC16(f, h, b, a) & h ` x ⊆ Y))"
  apply (erule lt_induct)
  apply (frule lt_Ord)
  apply (erule Ord_cases)

  apply (simp add: recfunAC16_0)

  prefer 2 apply (simp add: recfunAC16_Limit)
    apply (rule lemma5, assumption+)
    apply (blast dest!: recfunAC16_mono)

  apply clarify
  apply (erule lemma6 [THEN conjE], assumption)
  apply (simp (no_asm_simp) split del: split_if add: recfunAC16_succ)
  apply (rule conjI [THEN split_if [THEN iffD2]])
    apply (simp, erule lemma7, assumption)
  apply (rule impI)
  apply (rule ex_next_Ord [THEN oexE],
    assumption+, rule le_refl [THEN lt_trans], assumption+)
  apply (erule lemma8, assumption)
    apply (rule bij_is_fun [THEN apply_type], assumption)
    apply (erule Least_le [THEN lt_trans2, THEN ltD])
      apply (erule lt_Ord)
      apply (erule succ_leI)
  apply (erule LeastI)
  apply (erule lt_Ord)
done

```

```

lemma lemma_simp_induct:
  "[| ∀ b. b < a --> F(b) ⊆ S & (∀ x < a. (x < b | (∃ Y ∈ F(b). f ` x ⊆ Y))
    --> (∃ ! Y. Y ∈ F(b) & f ` x ⊆ Y));
    f ∈ a -> f `` (a); Limit(a);
    ∀ i j. i ≤ j --> F(i) ⊆ F(j) |]
  ==> (⋃ j < a. F(j)) ⊆ S &

```

```

      ( $\forall x \in f^{-1}a. \exists! Y. Y \in (\bigcup_{j < a} F(j)) \ \& \ x \subseteq Y$ )"
apply (rule conjI)
apply (rule subsetI)
apply (erule OUN_E, blast)
apply (rule ballI)
apply (erule imageE)
apply (drule ltI, erule Limit_is_Ord)
apply (drule Limit_has_succ, assumption)
apply (frule_tac x1="succ(xa)" in spec [THEN mp], assumption)
apply (erule conjE)
apply (drule ospec)

apply (erule leI [THEN succ_leE])
apply (erule impE)
apply (fast elim!: leI [THEN succ_leE, THEN lt_Ord, THEN le_refl])
apply (drule apply_equality, assumption)
apply (elim conjE ex1E)

apply (rule ex1I, blast)
apply (elim conjE OUN_E)
apply (erule_tac i="succ(xa)" and j=aa
      in Ord_linear_le [OF lt_Ord lt_Ord], assumption)
prefer 2
apply (drule spec [THEN spec, THEN mp, THEN subsetD], assumption+, blast)

apply (drule_tac x1=aa in spec [THEN mp], assumption)
apply (frule succ_leE)
apply (drule spec [THEN spec, THEN mp, THEN subsetD], assumption+, blast)

done

```

```

theorem W02_AC16: "[| W02; 0 < m; k ∈ nat; m ∈ nat |] ==> AC16(k #+ m, k)"
apply (unfold AC16_def)
apply (rule allI)
apply (rule impI)
apply (frule W02_infinite_subsets_eqpoll_X, assumption+)
apply (frule_tac n="k #+ m" in W02_infinite_subsets_eqpoll_X, simp, simp)

apply (frule W02_imp_ex_Card)
apply (elim exE conjE)
apply (drule eqpoll_trans [THEN eqpoll_sym,
      THEN eqpoll_def [THEN def_imp_iff, THEN iffD1]],
      assumption)
apply (drule eqpoll_trans [THEN eqpoll_sym,

```

```

                                THEN eqpoll_def [THEN def_imp_iff, THEN iffD1]],

    assumption+)
  apply (elim exE)
  apply (rename_tac h)
  apply (rule_tac x = "⋃ j<a. recfunAC16 (h,f,j,a) " in exI)
  apply (rule_tac P="%z. ?Y & (∀ x ∈ z. ?Z (x))"
    in bij_is_surj [THEN surj_image_eq, THEN subst], assumption)
  apply (rule lemma_simp_induct)
  apply (blast del: conjI notI
    intro!: main_induct eqpoll_imp_lepoll [THEN lepoll_infinite]
  )
  apply (blast intro: bij_is_fun [THEN surj_image, THEN surj_is_fun])
  apply (erule eqpoll_imp_lepoll [THEN lepoll_infinite,
    THEN infinite_Card_is_InfCard,
    THEN InfCard_is_Limit],
    assumption+)
  apply (blast dest!: recfunAC16_mono)
done

end

```

```

theory AC16_W04 imports AC16_lemmas begin

```

```

lemma lemma1:
  "[| Finite(A); 0<m; m ∈ nat |]
  ==> ∃ a f. Ord(a) & domain(f) = a &
    (⋃ b<a. f ` b) = A & (∀ b<a. f ` b ⋚ m)"
  apply (unfold Finite_def)
  apply (erule bexE)
  apply (drule eqpoll_sym [THEN eqpoll_def [THEN def_imp_iff, THEN iffD1]])
  apply (erule exE)
  apply (rule_tac x = n in exI)
  apply (rule_tac x = "λi ∈ n. {f ` i}" in exI)
  apply (simp add: ltD bij_def surj_def)
  apply (fast intro!: ltI nat_into_Ord lam_funtype [THEN domain_of_fun]
    singleton_eqpoll_1 [THEN eqpoll_imp_lepoll, THEN lepoll_trans]
    nat_1_lepoll_iff [THEN iffD2]
    elim!: apply_type ltE)
done

```

```
lemmas well_ord_paired = paired_bij [THEN bij_is_inj, THEN well_ord_rvimage]
```

```
lemma lepoll_trans1: "[| A  $\lesssim$  B;  $\sim$  A  $\lesssim$  C |] ==>  $\sim$  B  $\lesssim$  C"  
by (blast intro: lepoll_trans)
```

```
lemmas lepoll_paired = paired_eqpoll [THEN eqpoll_sym, THEN eqpoll_imp_lepoll]
```

```
lemma lemma2: " $\exists y R. \text{well\_ord}(y,R) \ \& \ x \ \text{Int} \ y = 0 \ \& \ \sim y \lesssim z \ \& \ \sim \text{Finite}(y) "$   
apply (rule_tac x = "{a,x}. a  $\in$  nat Un Hartog(z) }" in exI)  
apply (rule well_ord_Un [OF Ord_nat [THEN well_ord_Memrel]  
Ord_Hartog [THEN well_ord_Memrel], THEN exE])  
apply (blast intro!: Ord_Hartog well_ord_Memrel well_ord_paired  
lepoll_trans1 [OF _ not_Hartog_lepoll_self]  
lepoll_trans [OF subset_imp_lepoll lepoll_paired])  
elim!: nat_not_Finite [THEN notE]  
elim: mem_asym  
dest!: Un_upper1 [THEN subset_imp_lepoll, THEN lepoll_Finite]  
lepoll_paired [THEN lepoll_Finite])  
done
```

```
lemma infinite_Un: " $\sim \text{Finite}(B) ==> \sim \text{Finite}(A \ \text{Un} \ B) "$   
by (blast intro: subset_Finite)
```

```
lemma succ_not_lepoll_lemma:  
"[|  $\sim(\exists x \in A. f'x=y)$ ;  $f \in \text{inj}(A, B)$ ;  $y \in B$  |]  
==> ( $\lambda a \in \text{succ}(A). \text{if}(a=A, y, f'a) \in \text{inj}(\text{succ}(A), B) "$   
apply (rule_tac d = "%z. if (z=y, A, converse (f) 'z) " in lam_injective)
```

```

apply (force simp add: inj_is_fun [THEN apply_type])

apply (simp (no_asm_simp))
apply force
done

lemma succ_not_lepoll_imp_eqpoll: "[| ~A ≈ B; A ≲ B |] ==> succ(A)
  ≲ B"
apply (unfold lepoll_def eqpoll_def bij_def surj_def)
apply (fast elim!: succ_not_lepoll_lemma inj_is_fun)
done

lemmas ordertype_eqpoll =
  ordermap_bij [THEN exI [THEN eqpoll_def [THEN def_imp_iff, THEN
iffD2]]]

lemma cons_cons_subset:
  "[| a ⊆ y; b ∈ y-a; u ∈ x |] ==> cons(b, cons(u, a)) ∈ Pow(x Un
y)"
by fast

lemma cons_cons_eqpoll:
  "[| a ≈ k; a ⊆ y; b ∈ y-a; u ∈ x; x Int y = 0 |]
  ==> cons(b, cons(u, a)) ≈ succ(succ(k))"
by (fast intro!: cons_eqpoll_succ)

lemma set_eq_cons:
  "[| succ(k) ≈ A; k ≈ B; B ⊆ A; a ∈ A-B; k ∈ nat |] ==> A = cons(a,
B)"
apply (rule equalityI)
prefer 2 apply fast
apply (rule Diff_eq_0_iff [THEN iffD1])
apply (rule equalsOI)
apply (drule eqpoll_sym [THEN eqpoll_imp_lepoll])
apply (drule eqpoll_sym [THEN cons_eqpoll_succ], fast)
apply (drule cons_eqpoll_succ, fast)
apply (fast elim!: lepoll_trans [THEN lepoll_trans, THEN succ_lepoll_natE,
OF eqpoll_sym [THEN eqpoll_imp_lepoll] subset_imp_lepoll])
done

lemma cons_eqE: "[| cons(x,a) = cons(y,a); x ∉ a |] ==> x = y "
by (fast elim!: equalityE)

lemma eq_imp_Int_eq: "A = B ==> A Int C = B Int C"

```

by blast

```
lemma eqpoll_sum_imp_Diff_lepoll_lemma [rule_format]:
  "[| k ∈ nat; m ∈ nat |]
   ==> ∀A B. A ≈ k #+ m & k ≲ B & B ⊆ A --> A-B ≲ m"
apply (induct_tac "k")
apply (simp add: add_0)
apply (blast intro: eqpoll_imp_lepoll lepoll_trans
              Diff_subset [THEN subset_imp_lepoll])
apply (intro allI impI)
apply (rule succ_lepoll_imp_not_empty [THEN not_emptyE], fast)
apply (erule_tac x = "A - {xa}" in allE)
apply (erule_tac x = "B - {xa}" in allE)
apply (erule impE)
apply (simp add: add_succ)
apply (fast intro!: Diff_sing_eqpoll lepoll_Diff_sing)
apply (subgoal_tac "A - {xa} - (B - {xa}) = A - B", simp)
apply blast
done
```

```
lemma eqpoll_sum_imp_Diff_lepoll:
  "[| A ≈ succ(k #+ m); B ⊆ A; succ(k) ≲ B; k ∈ nat; m ∈ nat |]
   ==> A-B ≲ m"
apply (simp only: add_succ [symmetric])
apply (blast intro: eqpoll_sum_imp_Diff_lepoll_lemma)
done
```

```
lemma eqpoll_sum_imp_Diff_eqpoll_lemma [rule_format]:
  "[| k ∈ nat; m ∈ nat |]
   ==> ∀A B. A ≈ k #+ m & k ≈ B & B ⊆ A --> A-B ≈ m"
apply (induct_tac "k")
apply (force dest!: eqpoll_sym [THEN eqpoll_imp_lepoll, THEN lepoll_0_is_0])
apply (intro allI impI)
apply (rule succ_lepoll_imp_not_empty [THEN not_emptyE])
apply (fast elim!: eqpoll_imp_lepoll)
apply (erule_tac x = "A - {xa}" in allE)
apply (erule_tac x = "B - {xa}" in allE)
apply (erule impE)
apply (force intro: eqpoll_sym intro!: Diff_sing_eqpoll)
apply (subgoal_tac "A - {xa} - (B - {xa}) = A - B", simp)
```

```

apply blast
done

lemma eqpoll_sum_imp_Diff_eqpoll:
  "[| A ≈ succ(k #+ m); B ⊆ A; succ(k) ≈ B; k ∈ nat; m ∈ nat |]

  ==> A-B ≈ m"
apply (simp only: add_succ [symmetric])
apply (blast intro: eqpoll_sum_imp_Diff_eqpoll_lemma)
done

lemma subsets_lepoll_0_eq_unit: "{x ∈ Pow(X). x ≲ 0} = {0}"
by (fast dest!: lepoll_0_is_0 intro!: lepoll_refl)

lemma subsets_lepoll_succ:
  "n ∈ nat ==> {z ∈ Pow(y). z ≲ succ(n)} =
  {z ∈ Pow(y). z ≲ n} Un {z ∈ Pow(y). z ≈ succ(n)}"
by (blast intro: leI le_imp_lepoll nat_into_Ord
  lepoll_trans eqpoll_imp_lepoll
  dest!: lepoll_succ_disj)

lemma Int_empty:
  "n ∈ nat ==> {z ∈ Pow(y). z ≲ n} Int {z ∈ Pow(y). z ≈ succ(n)}
= 0"
by (blast intro: eqpoll_sym [THEN eqpoll_imp_lepoll, THEN lepoll_trans]
  succ_lepoll_natE)

locale (open) AC16 =
  fixes x and y and k and l and m and t_n and R and MM and LL and
  GG and s
  defines k_def: "k == succ(1)"
  and MM_def: "MM == {v ∈ t_n. succ(k) ≲ v Int y}"
  and LL_def: "LL == {v Int y. v ∈ MM}"
  and GG_def: "GG == λv ∈ LL. (THE w. w ∈ MM & v ⊆ w) - v"
  and s_def: "s(u) == {v ∈ t_n. u ∈ v & k ≲ v Int y}"
  assumes all_ex: "∀z ∈ {z ∈ Pow(x Un y) . z ≈ succ(k)}.
  ∃! w. w ∈ t_n & z ⊆ w "
  and disjoint[iff]: "x Int y = 0"
  and "includes": "t_n ⊆ {v ∈ Pow(x Un y). v ≈ succ(k #+ m)}"
  and WO_R[iff]: "well_ord(y,R)"
  and lnat[iff]: "l ∈ nat"
  and mnat[iff]: "m ∈ nat"

```

```

and mpos[iff]:      "0<m"
and Infinite[iff]: "~ Finite(y)"
and noLepoll:      "~ y ≲ {v ∈ Pow(x). v ≈ m}"

lemma (in AC16) knat [iff]: "k ∈ nat"
by (simp add: k_def)

lemma (in AC16) Diff_Finite_eqpoll: "[| l ≈ a; a ⊆ y |] ==> y - a ≈
y"
apply (insert WO_R Infinite lnat)
apply (rule eqpoll_trans)
apply (rule Diff_lesspoll_eqpoll_Card)
apply (erule well_ord_cardinal_eqpoll [THEN eqpoll_sym])
apply (blast intro: lesspoll_trans1
intro!: Card_cardinal
Card_cardinal [THEN Card_is_Ord, THEN nat_le_infinite_Ord,
THEN le_imp_lepoll]
dest: well_ord_cardinal_eqpoll
eqpoll_sym eqpoll_imp_lepoll
n_lesspoll_nat [THEN lesspoll_trans2]
well_ord_cardinal_eqpoll [THEN eqpoll_sym,
THEN eqpoll_imp_lepoll, THEN lepoll_infinite])+)
done

lemma (in AC16) s_subset: "s(u) ⊆ t_n"
by (unfold s_def, blast)

lemma (in AC16) sI:
"[| w ∈ t_n; cons(b,cons(u,a)) ⊆ w; a ⊆ y; b ∈ y-a; l ≈ a |]
==> w ∈ s(u)"
apply (unfold s_def succ_def k_def)
apply (blast intro!: eqpoll_imp_lepoll [THEN cons_lepoll_cong]
intro: subset_imp_lepoll lepoll_trans)
done

lemma (in AC16) in_s_imp_u_in: "v ∈ s(u) ==> u ∈ v"
by (unfold s_def, blast)

lemma (in AC16) ex1_superset_a:
"[| l ≈ a; a ⊆ y; b ∈ y - a; u ∈ x |]"

```

```

      ==>  $\exists! c. c \in s(u) \ \& \ a \subseteq c \ \& \ b \in c$ "
apply (rule all_ex [simplified k_def, THEN ballE])
  apply (erule ex1E)
  apply (rule_tac a = w in ex1I, blast intro: sI)
  apply (blast dest: s_subset [THEN subsetD] in_s_imp_u_in)
apply (blast del: PowI
      intro!: cons_cons_subset eqpoll_sym [THEN cons_cons_eqpoll])
done

```

```

lemma (in AC16) the_eq_cons:
  "[|  $\forall v \in s(u). \text{succ}(l) \approx v \text{ Int } y$ ;
     $l \approx a$ ;  $a \subseteq y$ ;  $b \in y - a$ ;  $u \in x$  |]
  ==> (THE  $c. c \in s(u) \ \& \ a \subseteq c \ \& \ b \in c$ ) Int  $y = \text{cons}(b, a)$ "
apply (frule ex1_superset_a [THEN theI], assumption+)
apply (rule set_eq_cons)
apply (fast+)
done

```

```

lemma (in AC16) y_lepoll_subset_s:
  "[|  $\forall v \in s(u). \text{succ}(l) \approx v \text{ Int } y$ ;
     $l \approx a$ ;  $a \subseteq y$ ;  $u \in x$  |]
  ==>  $y \lesssim \{v \in s(u). a \subseteq v\}$ "
apply (rule Diff_Finite_eqpoll [THEN eqpoll_sym, THEN eqpoll_imp_lepoll,
      THEN lepoll_trans], fast+)
apply (rule_tac f3 = " $\lambda b \in y - a. \text{THE } c. c \in s(u) \ \& \ a \subseteq c \ \& \ b \in c$ "

```

```

      in exI [THEN lepoll_def [THEN def_imp_iff, THEN iffD2]])
apply (simp add: inj_def)
apply (rule conjI)
apply (rule lam_type)
apply (frule ex1_superset_a [THEN theI], fast+, clarify)
apply (rule cons_eqE [of _ a])
apply (drule_tac A = "THE  $c. ?P(c)$ " and C = y in eq_imp_Int_eq)
apply (simp_all add: the_eq_cons)
done

```

```

lemma (in AC16) x_imp_not_y [dest]: " $a \in x \implies a \notin y$ "
by (blast dest: disjoint [THEN equalityD1, THEN subsetD, OF IntI])

```

```

lemma (in AC16) w_Int_eq_w_Diff:
  " $w \subseteq x \ \text{Un } y \implies w \text{ Int } (x - \{u\}) = w - \text{cons}(u, w \text{ Int } y)$ "

```

```

by blast

lemma (in AC16) w_Int_eqpoll_m:
  "[| w ∈ {v ∈ s(u). a ⊆ v};
    l ≈ a; u ∈ x;
    ∀v ∈ s(u). succ(l) ≈ v Int y |]
  ==> w Int (x - {u}) ≈ m"
apply (erule CollectE)
apply (subst w_Int_eq_w_Diff)
apply (fast dest!: s_subset [THEN subsetD]
      "includes" [simplified k_def, THEN subsetD])
apply (blast dest: s_subset [THEN subsetD]
      "includes" [simplified k_def, THEN subsetD]
      dest: eqpoll_sym [THEN cons_eqpoll_succ, THEN eqpoll_sym]

      in_s_imp_u_in
      intro!: eqpoll_sum_imp_Diff_eqpoll)
done

lemma (in AC16) eqpoll_m_not_empty: "a ≈ m ==> a ≠ 0"
apply (insert mpos)
apply (fast elim!: zero_lt_natE dest!: eqpoll_succ_imp_not_empty)
done

lemma (in AC16) cons_cons_in:
  "[| z ∈ xa Int (x - {u}); l ≈ a; a ⊆ y; u ∈ x |]
  ==> ∃! w. w ∈ t_n & cons(z, cons(u, a)) ⊆ w"
apply (rule all_ex [THEN bspec])
apply (unfold k_def)
apply (fast intro!: cons_eqpoll_succ elim: eqpoll_sym)
done

lemma (in AC16) subset_s_lepoll_w:
  "[| ∀v ∈ s(u). succ(l) ≈ v Int y; a ⊆ y; l ≈ a; u ∈ x |]
  ==> {v ∈ s(u). a ⊆ v} ≲ {v ∈ Pow(x). v ≈ m}"
apply (rule_tac f3 = "λw ∈ {v ∈ s (u) . a ⊆ v}. w Int (x - {u})"
  in exI [THEN lepoll_def [THEN def_imp_iff, THEN iffD2]])
apply (simp add: inj_def)
apply (intro conjI lam_type CollectI)
  apply fast
  apply (blast intro: w_Int_eqpoll_m)
apply (intro ballI impI)

```

```

apply (rule w_Int_eqpoll_m [THEN eqpoll_m_not_empty, THEN not_emptyE])
apply (blast, assumption+)
apply (drule equalityD1 [THEN subsetD], assumption)
apply (frule cons_cons_in, assumption+)
apply (blast dest: ex1_two_eq intro: s_subset [THEN subsetD] in_s_imp_u_in)+
done

```

```

lemma (in AC16) well_ord_subsets_eqpoll_n:
  "n ∈ nat ==> ∃S. well_ord({z ∈ Pow(y) . z ≈ succ(n)}, S)"
apply (rule WO_R [THEN well_ord_infinite_subsets_eqpoll_X,
  THEN eqpoll_def [THEN def_imp_iff, THEN iffD1], THEN
  exE])
apply (fast intro: bij_is_inj [THEN well_ord_rvimage])+
done

```

```

lemma (in AC16) well_ord_subsets_lepoll_n:
  "n ∈ nat ==> ∃R. well_ord({z ∈ Pow(y). z ≲ n}, R)"
apply (induct_tac "n")
apply (force intro!: well_ord_unit simp add: subsets_lepoll_0_eq_unit)
apply (erule exE)
apply (rule well_ord_subsets_eqpoll_n [THEN exE], assumption)
apply (simp add: subsets_lepoll_succ)
apply (drule well_ord_radd, assumption)
apply (erule Int_empty [THEN disj_Un_eqpoll_sum,
  THEN eqpoll_def [THEN def_imp_iff, THEN iffD1], THEN
  exE])
apply (fast elim!: bij_is_inj [THEN well_ord_rvimage])
done

```

```

lemma (in AC16) LL_subset: "LL ⊆ {z ∈ Pow(y). z ≲ succ(k #+ m)}"
apply (unfold LL_def MM_def)
apply (insert "includes")
apply (blast intro: subset_imp_lepoll eqpoll_imp_lepoll lepoll_trans)
done

```

```

lemma (in AC16) well_ord_LL: "∃S. well_ord(LL,S)"
apply (rule well_ord_subsets_lepoll_n [THEN exE, of "succ(k#+m)"])
apply simp
apply (blast intro: well_ord_subset [OF _ LL_subset])
done

```

```

lemma (in AC16) unique_superset_in_MM:
  "v ∈ LL ==> ∃! w. w ∈ MM & v ⊆ w"
apply (unfold MM_def LL_def, safe, fast)
apply (rule lepoll_imp_eqpoll_subset [THEN exE], assumption)
apply (rule_tac x = x in all_ex [THEN ballE])
apply (blast intro: eqpoll_sym)+
done

```

```

lemma (in AC16) Int_in_LL: "w ∈ MM ==> w Int y ∈ LL"
by (unfold LL_def, fast)

```

```

lemma (in AC16) in_LL_eq_Int:
  "v ∈ LL ==> v = (THE x. x ∈ MM & v ⊆ x) Int y"
apply (unfold LL_def, clarify)
apply (subst unique_superset_in_MM [THEN the_equality2])
apply (auto simp add: Int_in_LL)
done

```

```

lemma (in AC16) unique_superset1: "a ∈ LL ==> (THE x. x ∈ MM ∧ a ⊆
x) ∈ MM"
by (erule unique_superset_in_MM [THEN theI, THEN conjunct1])

```

```

lemma (in AC16) the_in_MM_subset:
  "v ∈ LL ==> (THE x. x ∈ MM & v ⊆ x) ⊆ x Un y"
apply (drule unique_superset1)
apply (unfold MM_def)
apply (fast dest!: unique_superset1 "includes" [THEN subsetD])
done

```

```

lemma (in AC16) GG_subset: "v ∈ LL ==> GG ' v ⊆ x"
apply (unfold GG_def)
apply (frule the_in_MM_subset)
apply (frule in_LL_eq_Int)
apply (force elim: equalityE)
done

```

```

lemma (in AC16) nat_lepoll_ordertype: "nat ≲ ordertype(y, R)"
apply (rule nat_le_infinite_Ord [THEN le_imp_lepoll])

```

```

  apply (rule Ord_ordertype [OF WO_R])
apply (rule ordertype_eqpoll [THEN eqpoll_imp_lepoll, THEN lepoll_infinite])

```

```

  apply (rule WO_R)
apply (rule Infinite)
done

```

```

lemma (in AC16) ex_subset_eqpoll_n: "n ∈ nat ==> ∃z. z ⊆ y & n ≈ z"
apply (erule nat_lepoll_imp_ex_eqpoll_n)
apply (rule lepoll_trans [OF nat_lepoll_ordertype])
apply (rule ordertype_eqpoll [THEN eqpoll_sym, THEN eqpoll_imp_lepoll])

```

```

apply (rule WO_R)
done

```

```

lemma (in AC16) exists_proper_in_s: "u ∈ x ==> ∃v ∈ s(u). succ(k) ≲
v Int y"
apply (rule ccontr)
apply (subgoal_tac "∀v ∈ s (u) . k ≈ v Int y")
prefer 2 apply (simp add: s_def, blast intro: succ_not_lepoll_imp_eqpoll)
apply (unfold k_def)
apply (insert all_ex "includes" lnat)
apply (rule ex_subset_eqpoll_n [THEN exE], assumption)
apply (rule noLepoll [THEN notE])
apply (blast intro: lepoll_trans [OF y_lepoll_subset_s subset_s_lepoll_w])
done

```

```

lemma (in AC16) exists_in_MM: "u ∈ x ==> ∃w ∈ MM. u ∈ w"
apply (erule exists_proper_in_s [THEN bexE])
apply (unfold MM_def s_def, fast)
done

```

```

lemma (in AC16) exists_in_LL: "u ∈ x ==> ∃w ∈ LL. u ∈ GG'w"
apply (rule exists_in_MM [THEN bexE], assumption)
apply (rule bexI)
apply (erule_tac [2] Int_in_LL)
apply (unfold GG_def)
apply (simp add: Int_in_LL)
apply (subst unique_superset_in_MM [THEN the_equality2])
apply (fast elim!: Int_in_LL)+
done

```

```

lemma (in AC16) OUN_eq_x: "well_ord(LL,S) ==>
(⋃ b<ordertype(LL,S). GG ' (converse(ordermap(LL,S)) ' b)) = x"
apply (rule equalityI)
apply (rule subsetI)
apply (erule OUN_E)
apply (rule GG_subset [THEN subsetD])

```

```

prefer 2 apply assumption
apply (blast intro: ltD ordermap_bij [THEN bij_converse_bij, THEN bij_is_fun,
                                     THEN apply_type])

apply (rule subsetI)
apply (erule exists_in_LL [THEN bexE])
apply (force intro: ltI [OF _ Ord_ordertype]
       ordermap_type [THEN apply_type]
       simp add: ordermap_bij [THEN bij_is_inj, THEN left_inverse])
done

```

```

lemma (in AC16) in_MM_eqpoll_n: "w ∈ MM ==> w ≈ succ(k #+ m)"
apply (unfold MM_def)
apply (fast dest: "includes" [THEN subsetD])
done

```

```

lemma (in AC16) in_LL_eqpoll_n: "w ∈ LL ==> succ(k) ≲ w"
by (unfold LL_def MM_def, fast)

```

```

lemma (in AC16) in_LL: "w ∈ LL ==> w ⊆ (THE x. x ∈ MM ∧ w ⊆ x)"
by (erule subset_trans [OF in_LL_eq_Int [THEN equalityD1] Int_lower1])

```

```

lemma (in AC16) all_in_lepoll_m:
  "well_ord(LL,S) ==>
   ∀ b<ordertype(LL,S). GG ‘ (converse(ordermap(LL,S)) ‘ b) ≲ m"
apply (unfold GG_def)
apply (rule oallI)
apply (simp add: ltD ordermap_bij [THEN bij_converse_bij, THEN bij_is_fun,
                                  THEN apply_type])
apply (insert "includes")
apply (rule eqpoll_sum_imp_Diff_lepoll)
apply (blast del: subsetI
        dest!: ltD
        intro!: eqpoll_sum_imp_Diff_lepoll in_LL_eqpoll_n
        intro: in_LL unique_superset1 [THEN in_MM_eqpoll_n]
        ordermap_bij [THEN bij_converse_bij, THEN bij_is_fun,
                     THEN apply_type])+)
done

```

```

lemma (in AC16) conclusion:
  "∃ a f. Ord(a) & domain(f) = a & (⋃ b<a. f ‘ b) = x & (∀ b<a. f ‘
b ≲ m)"
apply (rule well_ord_LL [THEN exE])
apply (rename_tac S)
apply (rule_tac x = "ordertype (LL,S)" in exI)

```

```

apply (rule_tac x = "λb ∈ ordertype(LL,S).
      GG ' (converse (ordermap (LL,S)) ' b)" in exI)
apply (simp add: ltD)
apply (blast intro: lam_funtype [THEN domain_of_fun]
      Ord_ordertype OUN_eq_x all_in_lepoll_m [THEN ospec])
done

```

```

theorem AC16_W04:
  "[| AC16(k #+ m, k); 0 < k; 0 < m; k ∈ nat; m ∈ nat |] ==> W04(m)"
apply (unfold AC16_def W04_def)
apply (rule allI)
apply (case_tac "Finite (A)")
apply (rule lemma1, assumption+)
apply (cut_tac lemma2)
apply (elim exE conjE)
apply (erule_tac x = "A Un y" in allE)
apply (frule infinite_Un, drule mp, assumption)
apply (erule zero_lt_natE, assumption, clarify)
apply (blast intro: AC16.conclusion)
done

```

end

theory AC17_AC1 imports HH begin

```

lemma AC0_AC1_lemma: "[| f:(Π X ∈ A. X); D ⊆ A |] ==> ∃g. g:(Π X ∈
D. X)"
by (fast intro!: lam_type apply_type)

```

```

lemma AC0_AC1: "AC0 ==> AC1"
apply (unfold AC0_def AC1_def)
apply (blast intro: AC0_AC1_lemma)
done

```

```

lemma AC1_AC0: "AC1 ==> AC0"
by (unfold AC0_def AC1_def, blast)

```

```

lemma AC1_AC17_lemma: "f ∈ (Π X ∈ Pow(A) - {0}. X) ==> f ∈ (Pow(A)
- {0} -> A)"
apply (rule Pi_type, assumption)
apply (drule apply_type, assumption, fast)
done

```

```

lemma AC1_AC17: "AC1 ==> AC17"
apply (unfold AC1_def AC17_def)
apply (rule allI)
apply (rule ballI)
apply (erule_tac x = "Pow (A) -{0}" in allE)
apply (erule impE, fast)
apply (erule exE)
apply (rule bexI)
apply (erule_tac [2] AC1_AC17_lemma)
apply (rule apply_type, assumption)
apply (fast dest!: AC1_AC17_lemma elim!: apply_type)
done

```

```

lemma UN_eq_imp_well_ord:
  "[| x - (∪ j ∈ LEAST i. HH(λX ∈ Pow(x)-{0}. {f'X}, x, i) = {x}.
    HH(λX ∈ Pow(x)-{0}. {f'X}, x, j)) = 0;
   f ∈ Pow(x)-{0} -> x |]
  ==> ∃r. well_ord(x,r)"
apply (rule exI)
apply (erule well_ord_rvimage
  [OF bij_Least_HH_x [THEN bij_converse_bij, THEN bij_is_inj]
   Ord_Least [THEN well_ord_Memrel]], assumption)
done

```

```

lemma not_AC1_imp_ex:
  "~AC1 ==> ∃A. ∀f ∈ Pow(A)-{0} -> A. ∃u ∈ Pow(A)-{0}. f'u ∉ u"
apply (unfold AC1_def)
apply (erule swap)
apply (rule allI)
apply (erule swap)

```

```

apply (rule_tac x = "Union (A)" in exI)
apply (blast intro: lam_type)
done

lemma AC17_AC1_aux1:
  "[|  $\forall f \in \text{Pow}(x) - \{0\} \rightarrow x. \exists u \in \text{Pow}(x) - \{0\}. f'u \notin u;$ 
     $\exists f \in \text{Pow}(x) - \{0\} \rightarrow x.$ 
     $x - (\bigcup a \in (\text{LEAST } i. \text{HH}(\lambda X \in \text{Pow}(x) - \{0\}. \{f'X\}, x, i) = \{x\})).$ 

     $\text{HH}(\lambda X \in \text{Pow}(x) - \{0\}. \{f'X\}, x, a) = 0$  |]
  ==> P"
apply (erule bexE)
apply (erule UN_eq_imp_well_ord [THEN exE], assumption)
apply (erule ex_choice_fun_Pow [THEN exE])
apply (erule ballE)
apply (fast intro: apply_type del: DiffE)
apply (erule notE)
apply (rule Pi_type, assumption)
apply (blast dest: apply_type)
done

lemma AC17_AC1_aux2:
  "~ ( $\exists f \in \text{Pow}(x) - \{0\} \rightarrow x. x - F(f) = 0$ )
  ==> ( $\lambda f \in \text{Pow}(x) - \{0\} \rightarrow x. x - F(f)$ )
   $\in (\text{Pow}(x) - \{0\} \rightarrow x) \rightarrow \text{Pow}(x) - \{0\}$ "
by (fast intro!: lam_type dest!: Diff_eq_0_iff [THEN iffD1])

lemma AC17_AC1_aux3:
  "[|  $f'Z \in Z; Z \in \text{Pow}(x) - \{0\}$  |]
  ==> ( $\lambda X \in \text{Pow}(x) - \{0\}. \{f'X\}'Z \in \text{Pow}(Z) - \{0\}$ )"
by auto

lemma AC17_AC1_aux4:
  " $\exists f \in F. f'((\lambda f \in F. Q(f))'f) \in (\lambda f \in F. Q(f))'f$ 
  ==>  $\exists f \in F. f'Q(f) \in Q(f)$ "
by simp

lemma AC17_AC1: "AC17 ==> AC1"
apply (unfold AC17_def)
apply (rule classical)
apply (erule not_AC1_imp_ex [THEN exE])
apply (case_tac
  " $\exists f \in \text{Pow}(x) - \{0\} \rightarrow x.$ 
   $x - (\bigcup a \in (\text{LEAST } i. \text{HH}(\lambda X \in \text{Pow}(x) - \{0\}. \{f'X\}, x, i) = \{x\})$ 
   $. \text{HH}(\lambda X \in \text{Pow}(x) - \{0\}. \{f'X\}, x, a) = 0$ ")
apply (erule AC17_AC1_aux1, assumption)
apply (drule AC17_AC1_aux2)
apply (erule allE)
apply (drule bspec, assumption)

```

```

apply (drule AC17_AC1_aux4)
apply (erule bexE)
apply (drule apply_type, assumption)
apply (simp add: HH_Least_eq_x del: Diff_iff )
apply (drule AC17_AC1_aux3, assumption)
apply (fast dest!: subst_elem [OF _ HH_Least_eq_x [symmetric]]
      f_subset_imp_HH_subset elim!: mem_irrefl)
done

```

```

lemma AC1_AC2_aux1:
  "[| f:( $\Pi$  X  $\in$  A. X); B  $\in$  A; 0 $\notin$ A |] ==> {f'B}  $\subseteq$  B Int {f'C. C
 $\in$  A}"
by (fast elim!: apply_type)

```

```

lemma AC1_AC2_aux2:
  "[| pairwise_disjoint(A); B  $\in$  A; C  $\in$  A; D  $\in$  B; D  $\in$  C |] ==>
f'B = f'C"
by (unfold pairwise_disjoint_def, fast)

```

```

lemma AC1_AC2: "AC1 ==> AC2"
apply (unfold AC1_def AC2_def)
apply (rule allI)
apply (rule impI)
apply (elim asm_rl conjE allE exE impE, assumption)
apply (intro exI ballI equalityI)
prefer 2 apply (rule AC1_AC2_aux1, assumption+)
apply (fast elim!: AC1_AC2_aux2 elim: apply_type)
done

```

```

lemma AC2_AC1_aux1: "0 $\notin$ A ==> 0  $\notin$  {B*{B}. B  $\in$  A}"
by (fast dest!: sym [THEN Sigma_empty_iff [THEN iffD1]])

```

```

lemma AC2_AC1_aux2: "[| X*{X} Int C = {y}; X  $\in$  A |]
  ==> (THE y. X*{X} Int C = {y}): X*A"
apply (rule subst_elem [of y])
apply (blast elim!: equalityE)
apply (auto simp add: singleton_eq_iff)

```

done

lemma AC2_AC1_aux3:

" $\forall D \in \{E^*\{E\}. E \in A\}. \exists y. D \text{ Int } C = \{y\}$
 $\implies (\lambda x \in A. \text{fst}(\text{THE } z. (x^*\{x\} \text{ Int } C = \{z\}))) \in (\Pi X \in A. X)$ "

apply (rule lam_type)

apply (drule bspec, blast)

apply (blast intro: AC2_AC1_aux2 fst_type)

done

lemma AC2_AC1: "AC2 \implies AC1"

apply (unfold AC1_def AC2_def pairwise_disjoint_def)

apply (intro allI impI)

apply (elim allE impE)

prefer 2 apply (fast elim!: AC2_AC1_aux3)

apply (blast intro!: AC2_AC1_aux1)

done

lemma empty_notin_images: " $0 \notin \{R^*\{x\}. x \in \text{domain}(R)\}$ "

by blast

lemma AC1_AC4: "AC1 \implies AC4"

apply (unfold AC1_def AC4_def)

apply (intro allI impI)

apply (drule spec, drule mp [OF _ empty_notin_images])

apply (best intro!: lam_type elim!: apply_type)

done

lemma AC4_AC3_aux1: " $f \in A \rightarrow B \implies (\bigcup z \in A. \{z\} * f^{\prime} z) \subseteq A * \text{Union}(B)$ "

by (fast dest!: apply_type)

lemma AC4_AC3_aux2: " $\text{domain}(\bigcup z \in A. \{z\} * f(z)) = \{a \in A. f(a) \neq 0\}$ "

by blast

lemma AC4_AC3_aux3: " $x \in A \implies (\bigcup z \in A. \{z\} * f(z))^{\prime} \{x\} = f(x)$ "

by fast

lemma AC4_AC3: "AC4 \implies AC3"

apply (unfold AC3_def AC4_def)

```

apply (intro allI ballI)
apply (elim allE impE)
apply (erule AC4_AC3_aux1)
apply (simp add: AC4_AC3_aux2 AC4_AC3_aux3 cong add: Pi_cong)
done

```

```

lemma AC3_AC1_lemma:
  "b ∉ A ==> (∏ x ∈ {a ∈ A. id(A) 'a ≠ b}. id(A) 'x) = (∏ x ∈ A. x)"
apply (simp add: id_def cong add: Pi_cong)
apply (rule_tac b = A in subst_context, fast)
done

```

```

lemma AC3_AC1: "AC3 ==> AC1"
apply (unfold AC1_def AC3_def)
apply (fast intro!: id_type elim: AC3_AC1_lemma [THEN subst])
done

```

```

lemma AC4_AC5: "AC4 ==> AC5"
apply (unfold range_def AC4_def AC5_def)
apply (intro allI ballI)
apply (elim allE impE)
apply (erule fun_is_rel [THEN converse_type])
apply (erule exE)
apply (rename_tac g)
apply (rule_tac x=g in bexI)
apply (blast dest: apply_equality range_type)
apply (blast intro: Pi_type dest: apply_type fun_is_rel)
done

```

```

lemma AC5_AC4_aux1: "R ⊆ A*B ==> (λx ∈ R. fst(x)) ∈ R -> A"
by (fast intro!: lam_type fst_type)

```

```

lemma AC5_AC4_aux2: "R ⊆ A*B ==> range(λx ∈ R. fst(x)) = domain(R)"
by (unfold lam_def, force)

```

```

lemma AC5_AC4_aux3: "[| ∃ f ∈ A->C. P(f, domain(f)); A=B |] ==> ∃ f ∈

```

```

B->C. P(f,B)"
apply (erule bexE)
apply (frule domain_of_fun, fast)
done

lemma AC5_AC4_aux4: "[| R ⊆ A*B; g ∈ C->R; ∀x ∈ C. (λz ∈ R. fst(z))'
(g'x) = x |]
    ==> (λx ∈ C. snd(g'x)): (Π x ∈ C. R' '{x})"
apply (rule lam_type)
apply (force dest: apply_type)
done

lemma AC5_AC4: "AC5 ==> AC4"
apply (unfold AC4_def AC5_def, clarify)
apply (elim allE ballE)
apply (drule AC5_AC4_aux3 [OF _ AC5_AC4_aux2], assumption)
apply (fast elim!: AC5_AC4_aux4)
apply (blast intro: AC5_AC4_aux1)
done

lemma AC1_iff_AC6: "AC1 <-> AC6"
by (unfold AC1_def AC6_def, blast)

end

theory AC18_AC19 imports AC_Equiv begin

constdefs
  uu      :: "i => i"
  "uu(a) == {c Un {0}. c ∈ a}"

lemma PROD_subsets:
  "[| f ∈ (Π b ∈ {P(a). a ∈ A}. b); ∀a ∈ A. P(a)≤Q(a) |]
    ==> (λa ∈ A. f'P(a)) ∈ (Π a ∈ A. Q(a))"
by (rule lam_type, drule apply_type, auto)

lemma lemma_AC18:

```

```

"[/  $\forall A. 0 \notin A \rightarrow (\exists f. f \in (\prod X \in A. X)); A \neq 0$  ]]
==>  $(\bigcap a \in A. \bigcup b \in B(a). X(a, b)) \subseteq$ 
 $(\bigcup f \in \prod a \in A. B(a). \bigcap a \in A. X(a, f'a))$ "
apply (rule subsetI)
apply (erule_tac x = "{b \in B (a) . x \in X (a,b) }. a \in A}" in allE)
apply (erule impE, fast)
apply (erule exE)
apply (rule UN_I)
  apply (fast elim!: PROD_subsets)
apply (simp, fast elim!: not_emptyE dest: apply_type [OF _ RepFunI])
done

```

```

lemma AC1_AC18: "AC1 ==> PROP AC18"
apply (unfold AC1_def)
apply (rule AC18.intro)
apply (fast elim!: lemma_AC18 apply_type intro!: equalityI INT_I UN_I)
done

```

```

theorem (in AC18) AC19
apply (unfold AC19_def)
apply (intro allI impI)
apply (rule AC18 [of _ "%x. x", THEN mp], blast)
done

```

```

lemma RepRep_conj:
"[/  $A \neq 0; 0 \notin A$  ]] ==>  $\{uu(a). a \in A\} \neq 0 \ \& \ 0 \notin \{uu(a). a$ 
 $\in A\}$ "
apply (unfold uu_def, auto)
apply (blast dest!: sym [THEN RepFun_eq_0_iff [THEN iffD1]])
done

```

```

lemma lemma1_1: "[/  $c \in a; x = c \text{ Un } \{0\}; x \notin a$  ]] ==>  $x - \{0\} \in a$ "
apply clarify
apply (rule subst_elem, assumption)
apply (fast elim: notE subst_elem)
done

```

```

lemma lemma1_2:
"[/  $f'(uu(a)) \notin a; f \in (\prod B \in \{uu(a). a \in A\}. B); a \in A$  ]]
==>  $f'(uu(a)) - \{0\} \in a$ "

```

```

apply (unfold uu_def, fast elim!: lemma1_1 dest!: apply_type)
done

lemma lemma1: " $\exists f. f \in (\prod B \in \{uu(a). a \in A\}. B) \implies \exists f. f \in (\prod B \in A. B)$ "
apply (erule exE)
apply (rule_tac x = " $\lambda a \in A. \text{if } (f' (uu(a)) \in a, f' (uu(a)), f' (uu(a)) - \{0\})$ "
      in exI)
apply (rule lam_type)
apply (simp add: lemma1_2)
done

lemma lemma2_1: " $a \neq 0 \implies 0 \in (\bigcup b \in uu(a). b)$ "
by (unfold uu_def, auto)

lemma lemma2: " $[| A \neq 0; 0 \notin A |] \implies (\bigcap x \in \{uu(a). a \in A\}. \bigcup b \in x. b) \neq 0$ "
apply (erule not_emptyE)
apply (rule_tac a = 0 in not_emptyI)
apply (fast intro!: lemma2_1)
done

lemma AC19_AC1: "AC19  $\implies$  AC1"
apply (unfold AC19_def AC1_def, clarify)
apply (case_tac "A=0", force)
apply (erule_tac x = "{uu (a) . a \in A}" in allE)
apply (erule impE)
apply (erule RepRep_conj, assumption)
apply (rule lemma1)
apply (drule lemma2, assumption, auto)
done

end

theory DC imports AC_Equiv Hartog Cardinal_aux begin

lemma RepFun_lepoll: " $Ord(a) \implies \{P(b). b \in a\} \lesssim a$ "
apply (unfold lepoll_def)
apply (rule_tac x = " $\lambda z \in RepFun (a,P) . LEAST i. z=P (i)$ " in exI)
apply (rule_tac d="%z. P (z)" in lam_injective)
  apply (fast intro!: Least_in_Ord)
apply (rule sym)
apply (fast intro: LeastI Ord_in_Ord)
done

```

Trivial in the presence of AC, but here we need a wellordering of X

```

lemma image_Ord_lepoll: "[| f ∈ X->Y; Ord(X) |] ==> f`X ≲ X"
apply (unfold lepoll_def)
apply (rule_tac x = "λx ∈ f`X. LEAST y. f`y = x" in exI)
apply (rule_tac d = "%z. f`z" in lam_injective)
apply (fast intro!: Least_in_Ord apply_equality, clarify)
apply (rule LeastI)
  apply (erule apply_equality, assumption+)
apply (blast intro: Ord_in_Ord)
done

lemma range_subset_domain:
  "[| R ⊆ X*X;  !!g. g ∈ X ==> ∃u. <g,u> ∈ R |]
   ==> range(R) ⊆ domain(R)"
by blast

lemma cons_fun_type: "g ∈ n->X ==> cons(<n,x>, g) ∈ succ(n) -> cons(x,
X)"
apply (unfold succ_def)
apply (fast intro!: fun_extend elim!: mem_irrefl)
done

lemma cons_fun_type2:
  "[| g ∈ n->X; x ∈ X |] ==> cons(<n,x>, g) ∈ succ(n) -> X"
by (erule cons_absorb [THEN subst], erule cons_fun_type)

lemma cons_image_n: "n ∈ nat ==> cons(<n,x>, g)`n = g`n"
by (fast elim!: mem_irrefl)

lemma cons_val_n: "g ∈ n->X ==> cons(<n,x>, g)`n = x"
by (fast intro!: apply_equality elim!: cons_fun_type)

lemma cons_image_k: "k ∈ n ==> cons(<n,x>, g)`k = g`k"
by (fast elim: mem_asym)

lemma cons_val_k: "[| k ∈ n; g ∈ n->X |] ==> cons(<n,x>, g)`k = g`k"
by (fast intro!: apply_equality consI2 elim!: cons_fun_type apply_Pair)

lemma domain_cons_eq_succ: "domain(f)=x ==> domain(cons(<x,y>, f)) =
succ(x)"
by (simp add: domain_cons succ_def)

lemma restrict_cons_eq: "g ∈ n->X ==> restrict(cons(<n,x>, g), n) =
g"
apply (simp add: restrict_def Pi_iff)
apply (blast intro: elim: mem_irrefl)
done

lemma succ_in_succ: "[| Ord(k); i ∈ k |] ==> succ(i) ∈ succ(k)"
apply (rule Ord_linear [of "succ(i)" "succ(k)", THEN disjE])

```

```

apply (fast elim: Ord_in_Ord mem_irrefl mem_asym)+
done

lemma restrict_eq_imp_val_eq:
  "[|restrict(f, domain(g)) = g; x ∈ domain(g)|]
  ==> f`x = g`x"
by (erule subst, simp add: restrict)

lemma domain_eq_imp_fun_type: "[| domain(f)=A; f ∈ B->C |] ==> f ∈ A->C"
by (frule domain_of_fun, fast)

lemma ex_in_domain: "[| R ⊆ A * B; R ≠ 0 |] ==> ∃x. x ∈ domain(R)"
by (fast elim!: not_emptyE)

constdefs

DC  :: "i => o"
     "DC(a) == ∀X R. R ⊆ Pow(X)*X &
              (∀Y ∈ Pow(X). Y < a --> (∃x ∈ X. <Y,x> ∈ R))
              --> (∃f ∈ a->X. ∀b<a. <f`b,f`b> ∈ R)"

DCO :: o
     "DCO == ∀A B R. R ⊆ A*B & R≠0 & range(R) ⊆ domain(R)
              --> (∃f ∈ nat->domain(R). ∀n ∈ nat. <f`n,f`succ(n)>:R)"

ff  :: "[i, i, i, i] => i"
     "ff(b, X, Q, R) ==
      transrec(b, %c r. THE x. first(x, {x ∈ X. <r`c, x> ∈ R},
Q))"

locale (open) DCO_imp =
  fixes XX and RR and X and R

  assumes all_ex: "∀Y ∈ Pow(X). Y < nat --> (∃x ∈ X. <Y, x> ∈ R)"

  defines XX_def: "XX == (⋃n ∈ nat. {f ∈ n->X. ∀k ∈ n. <f`k, f`k>
∈ R})"
    and RR_def: "RR == {<z1,z2>:XX*XX. domain(z2)=succ(domain(z1))
& restrict(z2, domain(z1)) = z1}"

```

```
lemma (in DCO_imp) lemma1_1: "RR  $\subseteq$  XX*XX"
by (unfold RR_def, fast)
```

```
lemma (in DCO_imp) lemma1_2: "RR  $\neq$  0"
apply (unfold RR_def XX_def)
apply (rule all_ex [THEN ballE])
apply (erule_tac [2] notE [OF _ empty_subsetI [THEN PowI]])
apply (erule_tac impE [OF _ nat_0I [THEN n_lesspoll_nat]])
apply (erule bexE)
apply (rule_tac a = "<0, {<0, x>}>" in not_emptyI)
apply (rule CollectI)
apply (rule SigmaI)
apply (rule nat_0I [THEN UN_I])
apply (simp (no_asm_simp) add: nat_0I [THEN UN_I])
apply (rule nat_1I [THEN UN_I])
apply (force intro!: singleton_fun [THEN Pi_type]
      simp add: singleton_0 [symmetric])
apply (simp add: singleton_0)
done
```

```
lemma (in DCO_imp) lemma1_3: "range(RR)  $\subseteq$  domain(RR)"
apply (unfold RR_def XX_def)
apply (rule range_subset_domain, blast, clarify)
apply (frule fun_is_rel [THEN image_subset, THEN PowI,
      THEN all_ex [THEN bspec]])
apply (erule impE[OF _ lesspoll_trans1[OF image_Ord_lepoll
      [OF _ nat_into_Ord] n_lesspoll_nat]],
      assumption+)
apply (erule bexE)
```

```

apply (rule_tac x = "cons (<n,x>, g) " in exI)
apply (rule CollectI)
apply (force elim!: cons_fun_type2
      simp add: cons_image_n cons_val_n cons_image_k cons_val_k)
apply (simp add: domain_of_fun succ_def restrict_cons_eq)
done

lemma (in DCO_imp) lemma2:
  "[|  $\forall n \in \text{nat}. \langle f'n, f'succ(n) \rangle \in \text{RR}; f \in \text{nat} \rightarrow \text{XX}; n \in \text{nat} \mid]$ 
  ==>  $\exists k \in \text{nat}. f'succ(n) \in k \rightarrow X \ \& \ n \in k$ 
      &  $\langle f'succ(n)'n, f'succ(n)'n \rangle \in R$ "
apply (induct_tac "n")
apply (drule apply_type [OF _ nat_1I])
apply (drule bspec [OF _ nat_0I])
apply (simp add: XX_def, safe)
apply (rule rev_bexI, assumption)
apply (subgoal_tac "0  $\in$  y", force)
apply (force simp add: RR_def
      intro: ltD elim!: nat_0_le [THEN leE])

apply (drule bspec [OF _ nat_succI], assumption)
apply (subgoal_tac "f ' succ (succ (x))  $\in$  succ (k)  $\rightarrow$ X")
apply (drule apply_type [OF _ nat_succI [THEN nat_succI]], assumption)
apply (simp (no_asm_use) add: XX_def RR_def)
apply safe
apply (frule_tac a="succ(k)" in domain_of_fun [symmetric, THEN trans],
      assumption)
apply (frule_tac a=y in domain_of_fun [symmetric, THEN trans],
      assumption)
apply (fast elim!: nat_into_Ord [THEN succ_in_succ]
      dest!: bspec [OF _ nat_into_Ord [THEN succ_in_succ]])
apply (drule domain_of_fun)
apply (simp add: XX_def RR_def, clarify)
apply (blast dest: domain_of_fun [symmetric, THEN trans] )
done

lemma (in DCO_imp) lemma3_1:
  "[|  $\forall n \in \text{nat}. \langle f'n, f'succ(n) \rangle \in \text{RR}; f \in \text{nat} \rightarrow \text{XX}; m \in \text{nat} \mid]$ 
  ==>  $\{f'succ(x)'x. x \in m\} = \{f'succ(m)'x. x \in m\}$ "
apply (subgoal_tac " $\forall x \in m. f'succ (m) 'x = f'succ (x) 'x$ ")
apply simp
apply (induct_tac "m", blast)
apply (rule ballI)
apply (erule succE)
apply (rule restrict_eq_imp_val_eq)
apply (drule bspec [OF _ nat_succI], assumption)

```

```

    apply (simp add: RR_def)
    apply (drule lemma2, assumption+)
    apply (fast dest!: domain_of_fun)
  apply (drule_tac x = xa in bspec, assumption)
  apply (erule sym [THEN trans, symmetric])
  apply (rule restrict_eq_imp_val_eq [symmetric])
    apply (drule bspec [OF _ nat_succI], assumption)
    apply (simp add: RR_def)
  apply (drule lemma2, assumption+)
  apply (blast dest!: domain_of_fun
    intro: nat_into_Ord OrdmemD [THEN subsetD])
done

lemma (in DCO_imp) lemma3:
  "[|  $\forall n \in \text{nat}. \langle f'n, f'succ(n) \rangle \in \text{RR}; f \in \text{nat} \rightarrow \text{XX}; m \in \text{nat} \mid]$ "

  ==> ( $\lambda x \in \text{nat}. f'succ(x)'x$ ) ' $m = f'succ(m)'m$ '
  apply (erule natE, simp)
  apply (subst image_lam)
    apply (fast elim!: OrdmemD [OF nat_succI Ord_nat])
  apply (subst lemma3_1, assumption+)
    apply fast
  apply (fast dest!: lemma2
    elim!: image_fun [symmetric, OF _ OrdmemD [OF _ nat_into_Ord]])
done

theorem DCO_imp_DC_nat: "DC0 ==> DC(nat)"
  apply (unfold DC_def DCO_def, clarify)
  apply (elim allE)
  apply (erule impE)

  apply (blast intro!: DCO_imp.lemma1_1 DCO_imp.lemma1_2 DCO_imp.lemma1_3)
  apply (erule bexE)
  apply (rule_tac x = " $\lambda n \in \text{nat}. f'succ (n) 'n$ " in rev_bexI)
    apply (rule lam_type, blast dest!: DCO_imp.lemma2 intro: fun_weaken_type)
  apply (rule oallI)
  apply (frule DCO_imp.lemma2, assumption)
    apply (blast intro: fun_weaken_type)
  apply (erule ltD)

  apply (subst DCO_imp.lemma3, assumption+)
    apply (fast elim!: fun_weaken_type)
  apply (erule ltD)
  apply (force simp add: lt_def)
done

```

```

lemma singleton_in_funs:
  "x ∈ X ==> {<0,x>} ∈
    (⋃ n ∈ nat. {f ∈ succ(n)->X. ∀ k ∈ n. <f'k, f'succ(k)> ∈
R})"
apply (rule nat_0I [THEN UN_I])
apply (force simp add: singleton_0 [symmetric]
      intro!: singleton_fun [THEN Pi_type])
done

locale (open) imp_DCO =
  fixes XX and RR and x and R and f and allRR
  defines XX_def: "XX == (⋃ n ∈ nat.
    {f ∈ succ(n)->domain(R). ∀ k ∈ n. <f'k, f'succ(k)>
∈ R})"
  and RR_def:
    "RR == {<z1,z2>:Fin(XX)*XX.
      (domain(z2)=succ(⋃ f ∈ z1. domain(f))
      & (∀ f ∈ z1. restrict(z2, domain(f)) = f))
      | (~ (∃ g ∈ XX. domain(g)=succ(⋃ f ∈ z1. domain(f))
      & (∀ f ∈ z1. restrict(g, domain(f)) = f)) & z2={<0,x>})}"
  and allRR_def:
    "allRR == ∀ b<nat.
      <f' b, f' b> ∈
      {<z1,z2>∈Fin(XX)*XX. (domain(z2)=succ(⋃ f ∈ z1. domain(f))
      & (⋃ f ∈ z1. domain(f)) = b
      & (∀ f ∈ z1. restrict(z2,domain(f))
= f))}"

lemma (in imp_DCO) lemma4:
  "[| range(R) ⊆ domain(R); x ∈ domain(R) |]
  ==> RR ⊆ Pow(XX)*XX &
    (∀ Y ∈ Pow(XX). Y < nat --> (∃ x ∈ XX. <Y,x>:RR))"
apply (rule conjI)
apply (force dest!: FinD [THEN PowI] simp add: RR_def)
apply (rule impI [THEN ballI])
apply (drule Finite_Fin [OF lesspoll_nat_is_Finite PowD], assumption)
apply (case_tac
  "∃ g ∈ XX. domain (g) =
    succ(⋃ f ∈ Y. domain(f)) & (∀ f∈Y. restrict(g, domain(f))
= f)")
apply (simp add: RR_def, blast)
apply (safe del: domainE)
apply (unfold XX_def RR_def)
apply (rule rev_bexI, erule singleton_in_funs)
apply (simp add: nat_0I [THEN rev_bexI] cons_fun_type2)
done

```

```

lemma (in imp_DCO) UN_image_succ_eq:
  "[| f ∈ nat->X; n ∈ nat |]
  ==> (⋃ x ∈ f'`succ(n). P(x)) = P(f'n) Un (⋃ x ∈ f'`n. P(x))"
by (simp add: image_fun OrdmemD)

lemma (in imp_DCO) UN_image_succ_eq_succ:
  "[| (⋃ x ∈ f'`n. P(x)) = y; P(f'n) = succ(y);
  f ∈ nat -> X; n ∈ nat |] ==> (⋃ x ∈ f'`succ(n). P(x)) = succ(y)"
by (simp add: UN_image_succ_eq, blast)

lemma (in imp_DCO) apply_domain_type:
  "[| h ∈ succ(n) -> D; n ∈ nat; domain(h)=succ(y) |] ==> h'y ∈ D"
by (fast elim: apply_type dest!: trans [OF sym domain_of_fun])

lemma (in imp_DCO) image_fun_succ:
  "[| h ∈ nat -> X; n ∈ nat |] ==> h'`succ(n) = cons(h'n, h'`n)"
by (simp add: image_fun OrdmemD)

lemma (in imp_DCO) f_n_type:
  "[| domain(f'n) = succ(k); f ∈ nat -> XX; n ∈ nat |]
  ==> f'n ∈ succ(k) -> domain(R)"
apply (unfold XX_def)
apply (drule apply_type, assumption)
apply (fast elim: domain_eq_imp_fun_type)
done

lemma (in imp_DCO) f_n_pairs_in_R [rule_format]:
  "[| h ∈ nat -> XX; domain(h'n) = succ(k); n ∈ nat |]
  ==> ∀ i ∈ k. <h'n'i, h'n'succ(i)> ∈ R"
apply (unfold XX_def)
apply (drule apply_type, assumption)
apply (elim UN_E CollectE)
apply (drule domain_of_fun [symmetric, THEN trans], assumption, simp)
done

lemma (in imp_DCO) restrict_cons_eq_restrict:
  "[| restrict(h, domain(u))=u; h ∈ n->X; domain(u) ⊆ n |]
  ==> restrict(cons(<n, y>, h), domain(u)) = u"
apply (unfold restrict_def)
apply (simp add: restrict_def Pi_iff)
apply (erule sym [THEN trans, symmetric])
apply (blast elim: mem_irrefl)
done

lemma (in imp_DCO) all_in_image_restrict_eq:
  "[| ∀ x ∈ f'`n. restrict(f'n, domain(x))=x;
  f ∈ nat -> XX;
  n ∈ nat; domain(f'n) = succ(n);"

```

```

      ( $\bigcup x \in f' 'n. \text{domain}(x) \subseteq n$  |]
    ==>  $\forall x \in f' 'succ(n). \text{restrict}(\text{cons}(\langle succ(n), y \rangle, f' 'n), \text{domain}(x))$ 
    = x"
  apply (rule ballI)
  apply (simp add: image_fun_succ)
  apply (drule f_n_type, assumption+)
  apply (erule disjE)
    apply (simp add: domain_of_fun restrict_cons_eq)
  apply (blast intro!: restrict_cons_eq_restrict)
  done

lemma (in imp_DCO) simplify_recursion:
  "[|  $\forall b \langle nat. \langle f' 'b, f' 'b \rangle \in RR;$ 
     $f \in nat \rightarrow XX; \text{range}(R) \subseteq \text{domain}(R); x \in \text{domain}(R)$  |]
  ==> allRR"
  apply (unfold RR_def allRR_def)
  apply (rule oallI, drule ltD)
  apply (erule nat_induct)
  apply (drule_tac x=0 in ospec, blast intro: Limit_has_0)
  apply (force simp add: singleton_fun [THEN domain_of_fun] singleton_in_funs)

  apply (simp only: separation split)
  apply (drule_tac x="succ(xa)" in ospec, blast intro: ltI)
  apply (elim conjE disjE)
  apply (force elim!: trans subst_context
    intro!: UN_image_succ_eq_succ)
  apply (erule notE)
  apply (simp add: XX_def UN_image_succ_eq_succ)
  apply (elim conjE bexE)
  apply (drule apply_domain_type, assumption+)
  apply (erule domainE)+
  apply (frule f_n_type)
  apply (simp add: XX_def, assumption+)
  apply (rule rev_bexI, erule nat_succI)
  apply (rename_tac m i j y z)
  apply (rule_tac x = "cons(\langle succ(m), z \rangle, f' 'm)" in bexI)
  prefer 2 apply (blast intro: cons_fun_type2)
  apply (rule conjI)
  prefer 2 apply (fast del: ballI subsetI
    elim: trans [OF _ subst_context, THEN domain_cons_eq_succ]
    subst_context
    all_in_image_restrict_eq [simplified XX_def]
    trans equalityD1)

  apply (rule ballI)
  apply (erule succE)

```

```

    apply (simp add: cons_val_n cons_val_k)

apply (drule f_n_pairs_in_R [simplified XX_def, OF _ domain_of_fun],
      assumption, assumption, assumption)
apply (simp add: nat_into_Ord [THEN succ_in_succ] succI2 cons_val_k)
done

lemma (in imp_DCO) lemma2:
  "[| allRR; f ∈ nat->XX; range(R) ⊆ domain(R); x ∈ domain(R); n
  ∈ nat |]
  ==> f'n ∈ succ(n) -> domain(R) & (∀i ∈ n. <f'n'i, f'n'succ(i)>:R)"
  apply (unfold allRR_def)
  apply (drule ospec)
  apply (erule ltI [OF _ Ord_nat])
  apply (erule CollectE, simp)
  apply (rule conjI)
  prefer 2 apply (fast elim!: f_n_pairs_in_R trans subst_context)
  apply (unfold XX_def)
  apply (fast elim!: trans [THEN domain_eq_imp_fun_type] subst_context)
  done

lemma (in imp_DCO) lemma3:
  "[| allRR; f ∈ nat->XX; n∈nat; range(R) ⊆ domain(R); x ∈ domain(R)
  |]
  ==> f'n'n = f'succ(n)'n"
  apply (frule lemma2 [THEN conjunct1, THEN domain_of_fun], assumption+)
  apply (unfold allRR_def)
  apply (drule ospec)
  apply (drule ltI [OF nat_succI Ord_nat], assumption, simp)
  apply (elim conjE ballE)
  apply (erule restrict_eq_imp_val_eq [symmetric], force)
  apply (simp add: image_fun OrdmemD)
  done

theorem DC_nat_imp_DCO: "DC(nat) ==> DCO"
  apply (unfold DC_def DCO_def)
  apply (intro allI impI)
  apply (erule asm_rl conjE ex_in_domain [THEN exE] allE)+
  apply (erule impE [OF _ imp_DCO.lemma4], assumption+)
  apply (erule bexE)
  apply (drule imp_DCO.simplify_recursion, assumption+)
  apply (rule_tac x = "λn ∈ nat. f'n'n" in bexI)
  apply (rule_tac [2] lam_type)
  apply (erule_tac [2] apply_type [OF imp_DCO.lemma2 [THEN conjunct1] succI1])
  apply (rule ballI)
  apply (frule_tac n="succ(n)" in imp_DCO.lemma2,
        (assumption/erule nat_succI)+)

```

```

apply (drule imp_DC0.lemma3, auto)
done

lemma fun_Ord_inj:
  "[| f ∈ a->X; Ord(a);
    !!b c. [| b<c; c ∈ a |] ==> f`b≠f`c |]
  ==> f ∈ inj(a, X)"
apply (unfold inj_def, simp)
apply (intro ballI impI)
apply (rule_tac j=x in Ord_in_Ord [THEN Ord_linear_lt], assumption+)
apply (blast intro: Ord_in_Ord, auto)
apply (atomize, blast dest: not_sym)
done

lemma value_in_image: "[| f ∈ X->Y; A ⊆ X; a ∈ A |] ==> f`a ∈ f``A"
by (fast elim!: image_fun [THEN ssubst])

theorem DC_W03: "(∀K. Card(K) --> DC(K)) ==> W03"
apply (unfold DC_def W03_def)
apply (rule allI)
apply (case_tac "A < Hartog (A)")
apply (fast dest!: lesspoll_imp_ex_lt_eqpoll
  intro!: Ord_Hartog leI [THEN le_imp_subset])
apply (erule allE impE)+
apply (rule Card_Hartog)
apply (erule_tac x = A in allE)
apply (erule_tac x = "{<z1,z2> ∈ Pow (A) *A . z1 < Hartog (A) & z2 ∉
z1}"
  in allE)
apply simp
apply (erule impE, fast elim: lesspoll_lemma [THEN not_emptyE])
apply (erule bexE)
apply (rule Hartog_lepoll_selfE)
apply (rule lepoll_def [THEN def_imp_iff, THEN iffD2])
apply (rule exI, rule fun_Ord_inj, assumption, rule Ord_Hartog)
apply (drule value_in_image)
apply (drule OrdmemD, rule Ord_Hartog, assumption+, erule ltD)
apply (drule ospec)
apply (blast intro: ltI Ord_Hartog, force)
done

```

```

lemma images_eq:
  "[|  $\forall x \in A. f'x=g'x; f \in Df \rightarrow Cf; g \in Dg \rightarrow Cg; A \subseteq Df; A \subseteq Dg$  |]
    ==> f'A = g'A"
apply (simp (no_asm_simp) add: image_fun)
done

lemma lam_images_eq:
  "[| Ord(a); b  $\in$  a |] ==> ( $\lambda x \in a. h(x)$ )'b = ( $\lambda x \in b. h(x)$ )'b"
apply (rule images_eq)
  apply (rule ballI)
  apply (drule OrdmemD [THEN subsetD], assumption+)
  apply simp
  apply (fast elim!: RepFunI OrdmemD intro!: lam_type)+
done

lemma lam_type_RepFun: "( $\lambda b \in a. h(b)$ )  $\in$  a  $\rightarrow$  {h(b). b  $\in$  a}"
by (fast intro!: lam_type RepFunI)

lemma lemmaX:
  "[|  $\forall Y \in Pow(X). Y < K \rightarrow (\exists x \in X. <Y, x> \in R);$ 
    b  $\in$  K; Z  $\in$  Pow(X); Z < K |]
    ==> {x  $\in$  X. <Z,x>  $\in$  R}  $\neq$  0"
by blast

lemma W01_DC_lemma:
  "[| Card(K); well_ord(X,Q);
     $\forall Y \in Pow(X). Y < K \rightarrow (\exists x \in X. <Y, x> \in R); b \in K$  |]
    ==> ff(b, X, Q, R)  $\in$  {x  $\in$  X. <( $\lambda c \in b. ff(c, X, Q, R)$ )'b, x>
 $\in$  R}"
  apply (rule_tac P = "b  $\in$  K" in impE, (erule_tac [2] asm_rl)+)
  apply (rule_tac i=b in Card_is_Ord [THEN Ord_in_Ord, THEN trans_induct],
    assumption+)
  apply (rule impI)
  apply (rule ff_def [THEN def_transrec, THEN ssubst])
  apply (erule the_first_in, fast)
  apply (simp add: image_fun [OF lam_type_RepFun subset_refl])
  apply (erule lemmaX, assumption)
  apply (blast intro: Card_is_Ord OrdmemD [THEN subsetD])
  apply (blast intro: lesspoll_trans1 in_Card_imp_lesspoll RepFun_lepoll)
done

theorem W01_DC_Card: "W01 ==>  $\forall K. Card(K) \rightarrow DC(K)"$ 
  apply (unfold DC_def W01_def)
  apply (rule allI impI)+
  apply (erule allE exE conjE)+
  apply (rule_tac x = " $\lambda b \in K. ff(b, X, Ra, R)$ " in bexI)

```

```
apply (simp add: lam_images_eq [OF Card_is_Ord ltD])
apply (fast elim!: ltE W01_DC_lemma [THEN CollectD2])
apply (rule_tac lam_type)
apply (rule W01_DC_lemma [THEN CollectD1], assumption+)
done

end
```

References

- [1] Lawrence C. Paulson and Krzysztof Grąbczewski. Mechanizing set theory: Cardinal arithmetic and the axiom of choice. *Journal of Automated Reasoning*, 17(3):291–323, December 1996.
- [2] Herman Rubin and Jean E. Rubin. *Equivalents of the Axiom of Choice, II*. North-Holland, 1985.