```
; This is a proof of the Paris-Harrington Ramsey theorem,
; and some related results.
; The proof is described in our paper, "A Ramsey Theorem
; in Boyer-Moore Logic".

; The following takes [ 79.1 1561.9 92.8 ]  on a DECstation 5000/125
```

EVENT: Start with the initial **nqthm** theory.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                         BASIC ARITHMETIC                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; (expt m n ) = m^n, as in Lisp
```

DEFINITION:
expt (m, n)
$=$ **if** $n \simeq 0$ **then** 1
else $m * \text{expt}(m, n - 1)$ **endif**

THEOREM: distributivity
$((u * y) + (a * y)) = ((u + a) * y)$

THEOREM: associativity-of-times
$(m * u * y) = ((m * u) * y)$

THEOREM: addition-of-exponents
$\text{expt}\,(m,\ a + b) = (\text{expt}\,(m,\ a) * \text{expt}\,(m,\ b))$

```
; consider the exponenial b^e -- it's monotonic as a function
; of b and of e separately


; as a function of b
```

THEOREM: monoton-of-exp-1
$(b1 \le b2) \rightarrow (\text{expt}\,(b1,\ e) \le \text{expt}\,(b2,\ e))$

THEOREM: plus-difference
$((e1 \le e2) \wedge (e1 \in \mathbf{N}) \wedge (e2 \in \mathbf{N})) \rightarrow ((e1 + (e2 - e1)) = e2)$

```
; as a function of e:
```

THEOREM: monoton-of-exp-aux1
$((e1 \le e2) \wedge (e1 \in \mathbf{N}) \wedge (e2 \in \mathbf{N}))$
$\rightarrow \quad (\text{expt}\,(b,\ e2) = (\text{expt}\,(b,\ e1) * \text{expt}\,(b,\ e2 - e1)))$

THEOREM: expt-is-positive
$(b \not\simeq 0) \rightarrow (0 < \text{expt}\,(b,\ e))$

THEOREM: monoton-of-exp-aux2
$(b \not\simeq 0) \rightarrow (x \le (x * \text{expt}\,(b,\ e)))$

THEOREM: monoton-of-exp-2
$((e1 \le e2) \wedge (e1 \in \mathbf{N}) \wedge (e2 \in \mathbf{N}) \wedge (b \not\simeq 0))$
$\rightarrow \quad (\text{expt}\,(b,\ e1) \le \text{expt}\,(b,\ e2))$

```
; 0 is an exception because 0^0 = 1 but 0^1 = 0
```

EVENT: Disable monoton-of-exp-aux1.

EVENT: Disable monoton-of-exp-aux2.

```
; (magic s) is the function g(s) which "magically" makes the
; Ramsey proof work out.  Here we just prove some basic arithmetic properties
; of it
```

Definition:
$\text{magic}(s) = (((3 * s) + 3) * \text{expt}(s + 2, s + 2))$

Theorem: magic-is-a-number
$\text{magic}(n) \in \mathbf{N}$

```
; now, we have to prove that magic is monotonic
```

Theorem: magic-is-monotonic-aux1
$(m < n) \rightarrow (((3 * m) + 3) < ((3 * n) + 3))$

Theorem: magic-is-monotonic-aux2
$(m < n) \rightarrow (\text{expt}(m + 2, m + 2) \leq \text{expt}(n + 2, n + 2))$

Theorem: magic-is-monotonic-aux3
$0 < \text{expt}(s + 2, s + 2)$

Theorem: magic-is-monotonic-aux4
$((x < y) \wedge (a \leq b) \wedge (0 < a) \wedge (0 < b))$
$\rightarrow \quad ((x * a) < (y * b))$

Theorem: magic-is-monotonic
$(m < n) \rightarrow (\text{magic}(m) < \text{magic}(n))$

Event: Disable magic-is-monotonic-aux1.

Event: Disable magic-is-monotonic-aux2.

Event: Disable magic-is-monotonic-aux3.

Event: Disable magic-is-monotonic-aux4.

Event: Disable magic.

```
; we need that (magic n) >= n ; this should follow just by monotonicity
```

Theorem: magic-is-bigger-aux1
$((n \neq 0) \wedge (\text{magic}(n) = 0)) \rightarrow (n \notin \mathbf{N})$

Theorem: magic-is-bigger-aux2
$((n \in \mathbf{N}) \wedge (\text{magic}(n - 1) \not< (n - 1))) \rightarrow (\text{magic}(n) \not< n)$

THEOREM: magic-is-bigger
$(n \in \mathbf{N}) \rightarrow (\text{magic}\,(n) \not< n)$

EVENT: Disable magic-is-bigger-aux1.

EVENT: Disable magic-is-bigger-aux2.

```
; the above is sufficient for all the basic theory of {alpha}(n)
; and alpha-large sets.
; BUT, we need some further arithmetic properties of magic which
; get used in the Ramsey theorem

; first, let's prove that * is monotonic; + will be handled
; by built-in linear arithmetic
```

THEOREM: times-is-monotonic
$((x \leq y) \land (a \leq b)) \rightarrow ((x * a) \leq (y * b))$

THEOREM: times-right-ident
$(x \in \mathbf{N}) \rightarrow ((x * 1) = x)$

THEOREM: times-left-ident
$(x \in \mathbf{N}) \rightarrow ((1 * x) = x)$

```
; now, just a chain of uninteresting inequalities culminating
; in exactly what we need about magic(s)
; call these magic-1, magic-2, ...
; they won't be rewrite rules, so we don't need to disable them later
```

EVENT: Enable magic.

```
; we need its defn for some other properties
```

THEOREM: magic-1
$((2 * s) + 3) \leq \text{magic}\,(s)$

THEOREM: magic-2
$(s \in \mathbf{N}) \rightarrow (\text{expt}\,(s,\, s) \leq \text{expt}\,(s + 2,\, s + 2))$

THEOREM: magic-3
$(r \in \mathbf{N}) \rightarrow (((3 * r) * \text{expt}\,(r,\, r)) \leq \text{magic}\,(r))$

THEOREM: magic-4
$(s \in \mathbf{N}) \rightarrow (\text{expt}\,(2,\ s) \leq \text{expt}\,(s + 2,\ s + 2))$

THEOREM: magic-5
$(s \in \mathbf{N}) \rightarrow (\text{expt}\,(2,\ s) \leq \text{magic}\,(s))$

THEOREM: magic-6
$(s \in \mathbf{N}) \rightarrow (\text{expt}\,(s,\ \text{expt}\,(2,\ s)) \leq \text{expt}\,(\text{magic}\,(s),\ \text{magic}\,(s)))$

THEOREM: magic-7
$(s \in \mathbf{N})$
$\rightarrow \quad (((3 * \text{magic}\,(s)) * \text{expt}\,(s,\ \text{expt}\,(2,\ s)))$
$\qquad \leq \quad ((3 * \text{magic}\,(s)) * \text{expt}\,(\text{magic}\,(s),\ \text{magic}\,(s))))$

THEOREM: magic-8
$(s \in \mathbf{N}) \rightarrow (((3 * \text{magic}\,(s)) * \text{expt}\,(s,\ \text{expt}\,(2,\ s))) \leq \text{magic}\,(\text{magic}\,(s)))$

THEOREM: magic-9
$((2 * s) + 3 + (2 * \text{magic}\,(s))) \leq (3 * \text{magic}\,(s))$

THEOREM: magic-10
$(s \in \mathbf{N})$
$\rightarrow \quad ((((2 * s) + 3 + (2 * \text{magic}\,(s))) * \text{expt}\,(s,\ \text{expt}\,(2,\ s)))$
$\qquad \leq \quad \text{magic}\,(\text{magic}\,(s)))$

```
; now, let's throw in some more parameters.  We have
; a = the norm of some ordinal;
; d = a + c ; s = z + d, q >= magic(z + d), c > 0
; we want to conclude
;(leq
; (times
; (plus (times 2 a) (times 2 (magic z)) 3 c)
; (expt c (expt 2 z)) )
; (magic q) )
; 1. show (leq   (plus (times 2 a) (times 2 (magic z)) 3 c)
;                (plus (times 2 s) 3 (times 2 (magic s))))
; 2. show (leq (expt c (expt 2 z)) (expt s (expt 2 s)))
; then apply magic-10

; step 1
```

THEOREM: magic-11
$((d = (a + c)) \wedge (s = (z + d)) \wedge (c \not\simeq 0))$
$\rightarrow \quad (((2 * a) + (2 * \text{magic}\,(z)) + 3 + c)$
$\qquad \leq \quad ((2 * s) + 3 + (2 * \text{magic}\,(s))))$

```
; step 2
```

THEOREM: magic-12
$((d = (a + c)) \land (s = (z + d))) \rightarrow (\mathrm{expt}\,(2,\, z) \leq \mathrm{expt}\,(2,\, s))$

THEOREM: magic-13
$((d = (a + c)) \land (s = (z + d)))$
$\rightarrow \quad (\mathrm{expt}\,(c,\, \mathrm{expt}\,(2,\, z)) \leq \mathrm{expt}\,(s,\, \mathrm{expt}\,(2,\, s)))$

```
; putting the two steps together:
```

THEOREM: magic-14
$((d = (a + c)) \land (s = (z + d)) \land (c \not\simeq 0))$
$\rightarrow \quad ((((2 * a) + (2 * \mathrm{magic}\,(z)) + 3 + c)$
$\quad * \quad \mathrm{expt}\,(c,\, \mathrm{expt}\,(2,\, z)))$
$\quad \leq \quad (((2 * s) + 3 + (2 * \mathrm{magic}\,(s)))$
$\qquad * \quad \mathrm{expt}\,(s,\, \mathrm{expt}\,(2,\, s))))$

```
; now, adding magic-10
```

THEOREM: magic-15
$((d = (a + c)) \land (s = (z + d)) \land (c \not\simeq 0))$
$\rightarrow \quad ((((2 * a) + (2 * \mathrm{magic}\,(z)) + 3 + c)$
$\quad * \quad \mathrm{expt}\,(c,\, \mathrm{expt}\,(2,\, z)))$
$\quad \leq \quad \mathrm{magic}\,(\mathrm{magic}\,(s)))$

```
; now, the final result doesn't mention s at all -- rather,
; it's in terms of q >= magic(z + d)  (where s is z+d)
```

```
; first, isolate the monotnicity of magic
```

THEOREM: magic-16-aux1
$(v \notin \mathbf{N}) \rightarrow ((w * v) = 0)$

THEOREM: magic-16
$(v \notin \mathbf{N}) \rightarrow (\mathrm{magic}\,(v) = 12)$

EVENT: Disable magic-16-aux1.

THEOREM: magic-17
$(v \leq w) \rightarrow (\mathrm{magic}\,(v) \leq \mathrm{magic}\,(w))$

THEOREM: magic-18
$((d = (a + c)) \land (\mathrm{magic}\,(z + d) \leq q) \land (c \not\simeq 0))$
$\rightarrow \quad ((((2 * a) + (2 * \mathrm{magic}\,(z)) + 3 + c)$
$\quad * \quad \mathrm{expt}\,(c,\, \mathrm{expt}\,(2,\, z)))$
$\quad \leq \quad \mathrm{magic}\,(q))$

```
; this concludes all the arithmetic facts about magic that we need
```

EVENT: Disable magic.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                        PROPERTIES OF LISTS                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


; append is build-in, but the following simple lemmas aren't
```

THEOREM: append-works-left
$(x \in \mathit{lst1}) \rightarrow (x \in \text{append}\,(\mathit{lst1},\, \mathit{lst2}))$

THEOREM: append-works-right
$(y \in \mathit{lst2}) \rightarrow (y \in \text{append}\,(\mathit{lst1},\, \mathit{lst2}))$

DEFINITION:
length (set)
$=$ **if** listp (set) **then** $1 +$ length (cdr (set))
else 0 **endif**

THEOREM: length-of-append
length (append $(\mathit{lst1},\, \mathit{lst2})$) $=$ (length $(\mathit{lst1})$ $+$ length $(\mathit{lst2})$)

```
;;;;;;;;;;  product of two lists
```

DEFINITION:
cons-all $(\mathit{item},\, \mathit{lst})$
$=$ **if** listp (lst) **then** cons (cons $(\mathit{item}$, car $(\mathit{lst}))$, cons-all $(\mathit{item}$, cdr $(\mathit{lst})))$
else nil endif

DEFINITION:
product $(\mathit{lst1},\, \mathit{lst2})$
$=$ **if** $\mathit{lst1} \simeq$ **nil then nil**
else append (cons-all (car $(\mathit{lst1})$, $\mathit{lst2}$), product (cdr $(\mathit{lst1})$, $\mathit{lst2}$)) **endif**

```
; (product '(1 2) '(3 4 5) ) -->
; ((1 . 3) (1 . 4) (1 . 5) (2 . 3) (2 . 4) (2 . 5))

; we need to prove this contains all pairs
; first, for cons-all
```

THEOREM: all-pairs-in-cons-all
$(y \in lst) \rightarrow (\text{cons}(item, y) \in \text{cons-all}(item, lst))$

THEOREM: all-pairs
$((x \in lst1) \wedge (y \in lst2)) \rightarrow (\text{cons}(x, y) \in \text{product}(lst1, lst2))$

```
; We never need the other direction -- that the product
; contains only such pairs
```

EVENT: Disable product.

```
; there are two notions of "sublist" for lists
; 1: (subsetp s1 s2) means that every member of s1 is a member of s2
; 2: (sublistp lst1 lst2) means that lst1 is a consecutive
;     sublist of lst2 -- so, a list of length n always
;     has exactly 2^n sublists.
; For our intended representation of sets as increasing
; lists of natural numbers, (1) and (2) are equivalent.
; In general, (2) implies (1).
; in both cases, for non-nil-terminated lists -- e.g.
; (A1 A2 A3 . B), we ignore the B
```

DEFINITION:
subsetp $(s1, s2)$
= **if** listp $(s1)$ **then** $(\text{car}(s1) \in s2) \wedge \text{subsetp}(\text{cdr}(s1), s2)$
else t endif

```
; there are two fundamental properties of subsetp
; 1. If (subsetp s1 s2) and (member x s1) then (member x s2)
; 2. If (not (subsetp s1 s2)), then there is an x such
;    that (member x s1) and (not (member x s2))
;    Call this object (memb-of-dif s1 s2)
```

THEOREM: subsetp-works-1
$(\text{subsetp}(s1, s2) \wedge \textit{and} \wedge (x \in s1)) \rightarrow (x \in s2)$

DEFINITION:
memb-of-dif $(s1, s2)$
= **if** listp $(s1)$
then if $\text{car}(s1) \in s2$ **then** memb-of-dif $(\text{cdr}(s1), s2)$
else car $(s1)$ **endif**
else nil endif

THEOREM: subsetp-works-2
$(\neg\ \text{subsetp}\,(s1,\ s2))$
$\rightarrow\quad ((\text{memb-of-dif}\,(s1,\ s2) \in s1) \wedge (\text{memb-of-dif}\,(s1,\ s2) \notin s2))$

EVENT: Disable memb-of-dif.

```
; It's really just a Skolem function for (not (subsetp s1 s2))
; We should never need its definition

; the following facts may also be useful:
```

THEOREM: subset-of-empty-set
$\text{listp}\,(s) \rightarrow (\neg\ \text{subsetp}\,(s,\ \mathbf{nil}))$

THEOREM: transitivity-of-subset
$(\text{subsetp}\,(s1,\ s2) \wedge \text{subsetp}\,(s2,\ s3)) \rightarrow \text{subsetp}\,(s1,\ s3)$

THEOREM: cdr-is-subset-aux1
$(x \in \text{cdr}\,(s)) \rightarrow (x \in s)$

THEOREM: cdr-is-subset
$\text{subsetp}\,(\text{cdr}\,(s),\ s)$

EVENT: Disable cdr-is-subset-aux1.

THEOREM: cdr-of-subset
$(\text{subsetp}\,(s1,\ s2) \wedge \text{listp}\,(s1)) \rightarrow \text{subsetp}\,(\text{cdr}\,(s1),\ s2)$

THEOREM: nil-is-subset
$\text{subsetp}\,(\mathbf{nil},\ s)$

```
; the following might useful in proving consequences
; of (subsetp s1 s2) by induction -- reducing to
; (subsetp s1 (cdr s2))
```

THEOREM: subset-of-cdr
$(\text{subsetp}\,(s1,\ s2) \wedge (\text{car}\,(s2) \notin s1)) \rightarrow \text{subsetp}\,(s1,\ \text{cdr}\,(s2))$

THEOREM: non-empty-subset
$(\text{subsetp}\,(lst1,\ lst2) \wedge \text{listp}\,(lst1)) \rightarrow \text{listp}\,(lst2)$

THEOREM: car-of-subset
$(\text{subsetp}\,(lst1,\ lst2) \wedge \text{listp}\,(lst1)) \rightarrow (\text{car}\,(lst1) \in lst2)$

```
; now, consider sub-lists
```

DEFINITION:
sublistp ($lst1$, $lst2$)
$=$ **if** listp ($lst2$)
then sublistp ($lst1$, cdr ($lst2$))
$\vee$ (listp ($lst1$)
$\wedge$ (car ($lst1$) $=$ car ($lst2$))
$\wedge$ sublistp (cdr ($lst1$), cdr ($lst2$)))
else $\neg$ listp ($lst1$) **endif**

THEOREM: sublist-implies-subset
sublistp ($lst1$, $lst2$) $\rightarrow$ subsetp ($lst1$, $lst2$)

THEOREM: transitivity-of-sublist
(sublistp ($s1$, $s2$) $\wedge$ sublistp ($s2$, $s3$)) $\rightarrow$ sublistp ($s1$, $s3$)

```
; (power-set S) will actually be the set of all
; (nil-terminated)
; sublists of S -- but this will be all subsets
; in our intended "standard" representation of sets
```

DEFINITION:
power-set (set)
$=$ **if** listp (set)
then append (cons-all (car (set), power-set (cdr (set))),
power-set (cdr (set)))
else list (**nil**) **endif**

```
; this has 2^(length elements)
```

THEOREM: size-of-cons-all
length (cons-all ($item$, lst)) $=$ length (lst)

THEOREM: size-of-power-set
length (power-set (set)) $=$ expt (2, length (set))

```
; now, we prove that the power-set contains all sublists,
; and only sublists -- of course, the power set is only
; collecting proper lists
```

Definition:
properp (lst)
$=$ **if** listp (lst) **then** properp (cdr (lst))
else lst = **nil** **endif**

Theorem: cons-preserves-proper
properp (lst) $\rightarrow$ properp (cons ($item$, lst))

Definition:
members-all-properp ($biglst$)
$=$ **if** listp ($biglst$)
then properp (car ($biglst$)) $\wedge$ members-all-properp (cdr ($biglst$))
else **t** **endif**

Theorem: append-preserves-members-all-properp
(members-all-properp ($biglst1$) $\wedge$ members-all-properp ($biglst2$))
$\rightarrow$ members-all-properp (append ($biglst1$, $biglst2$))

Theorem: cons-all-preserves-members-all-properp
members-all-properp ($biglst$) $\rightarrow$ members-all-properp (cons-all ($item$, $biglst$))

```
; of coure, we chould check the defining property:
```

Theorem: members-all-properp-works
(($lst \in biglst$) $\wedge$ members-all-properp ($biglst$)) $\rightarrow$ properp (lst)

Theorem: only-proper-aux1
members-all-properp (power-set (lst))

Theorem: only-proper
($lst \in$ power-set (set)) $\rightarrow$ properp (lst)

Event: Disable only-proper-aux1.

Definition:
members-all-sublistp ($biglst$, set)
$=$ **if** listp ($biglst$)
then sublistp (car ($biglst$), set)
$\wedge$ members-all-sublistp (cdr ($biglst$), set)
else **t** **endif**

Theorem: append-preserves-members-all-sublistp
(members-all-sublistp ($biglst1$, set) $\wedge$ members-all-sublistp ($biglst2$, set))
$\rightarrow$ members-all-sublistp (append ($biglst1$, $biglst2$), set)

THEOREM: cons-all-preserves-members-all-sublistp
members-all-sublistp ($biglst$, set)
$\rightarrow$ members-all-sublistp (cons-all ($item$, $biglst$), cons ($item$, set))

```
; of coure, we chould check the defining property:
```

THEOREM: members-all-sublistp-works
(($lst \in biglst$) $\wedge$ members-all-sublistp ($biglst$, set))
$\rightarrow$ sublistp (lst, set)

THEOREM: only-sublists-aux1
members-all-sublistp (power-set (set), set)

THEOREM: only-sublists
($lst \in$ power-set (set)) $\rightarrow$ sublistp (lst, set)

```
; also, we get all sublists
```

THEOREM: all-sublists
(sublistp (lst, set) $\wedge$ properp (lst)) $\rightarrow$ ($lst \in$ power-set (set))

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                              STANDARD SETS                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; These standard lists of numbers in strictly increasing order.
```

DEFINITION:
setp (s)
= **if** listp (s)
then (car (s) $\in$ **N**)
$\wedge$ setp (cdr (s))
$\wedge$ ((cdr (s) = **nil**) $\vee$ (car (s) < cadr (s)))
else s = **nil endif**

```
;;;;; large, in Paris-Harrington sense:
```

DEFINITION:
largep (set) = (setp (set) $\wedge$ listp (set) $\wedge$ (length (set) $\not<$ car (set)))

THEOREM: empty-is-nil
(setp (s) $\wedge$ ($\neg$ listp (s))) $\rightarrow$ ((s = **nil**) = **t**)

```
; so it can be a rewrite rule -- can't rewrite a var
```

THEOREM: increasing
$(\text{setp}(s) \land (x \in \text{cdr}(s))) \rightarrow (\text{car}(s) < x)$

```
; Since the order is increasing, the members don't occur twice
```

THEOREM: not-again
$\text{setp}(s) \rightarrow (\text{car}(s) \notin \text{cdr}(s))$

```
; the min of a set is its first element
```

THEOREM: min-is-first
$(\text{setp}(s) \land (x < \text{car}(s))) \rightarrow (x \notin s)$

```
; the max of a set is its last element
```

DEFINITION:
last (*set*)
= **if** listp (*set*)
then if listp (cdr (*set*)) **then** last (cdr (*set*))
else car (*set*) **endif**
else 0 **endif**

```
; the tail of a set is a set
```

THEOREM: tail-of-a-set
$(\text{setp}(s) \land \text{listp}(s)) \rightarrow \text{setp}(\text{cdr}(s))$

```
; this is useful in inductive proofs about sets
```

THEOREM: first-before-last
$(\text{setp}(s) \land \text{listp}(\text{cdr}(s))) \rightarrow (\text{car}(s) < \text{last}(s))$

THEOREM: max-is-last
$(\text{setp}(s) \land (\text{last}(s) < x)) \rightarrow (x \notin s)$

```
; sets are proper lists:
```

THEOREM: sets-are-proper
$\text{setp}(s) \rightarrow \text{properp}(s)$

```
;; now, a few lemmas about the first elements of subsets
```

THEOREM: compare-first-elts
$(\text{setp}(s1) \land \text{setp}(s2) \land \text{listp}(s1) \land \text{subsetp}(s1, s2))$
$\rightarrow \quad (\text{car}(s1) \not< \text{car}(s2))$

```
; now, if (subsetp s1 s2) -- there are two cases
; the cars are the same, or (car s2) is smaller
```

THEOREM: smaller-cars-in-subset
$$
\begin{aligned}
&(\text{setp}(s1) \\
&\ \land\ \ \text{setp}(s2) \\
&\ \land\ \ \text{listp}(s1) \\
&\ \land\ \ \text{subsetp}(s1, s2) \\
&\ \land\ \ (\text{car}(s2) < \text{car}(s1))) \\
&\rightarrow\ \ \text{subsetp}(s1, \text{cdr}(s2))
\end{aligned}
$$

THEOREM: same-cars-in-subset-aux1
$(\text{setp}(s1) \land \text{listp}(s1) \land \text{listp}(\text{cdr}(s1))) \rightarrow (\text{car}(s1) < \text{cadr}(s1))$

THEOREM: same-cars-in-subset-aux2
$$
\begin{aligned}
&(\text{setp}(s1) \\
&\ \land\ \ \text{setp}(s2) \\
&\ \land\ \ \text{listp}(s1) \\
&\ \land\ \ \text{subsetp}(s1, s2) \\
&\ \land\ \ \text{listp}(\text{cdr}(s1)) \\
&\ \land\ \ (\text{car}(s2) = \text{car}(s1))) \\
&\rightarrow\ \ \text{subsetp}(\text{cdr}(s1), \text{cdr}(s2))
\end{aligned}
$$

THEOREM: same-cars-in-subset
$$
\begin{aligned}
&(\text{setp}(s1) \\
&\ \land\ \ \text{setp}(s2) \\
&\ \land\ \ \text{listp}(s1) \\
&\ \land\ \ \text{subsetp}(s1, s2) \\
&\ \land\ \ (\text{car}(s2) = \text{car}(s1))) \\
&\rightarrow\ \ \text{subsetp}(\text{cdr}(s1), \text{cdr}(s2))
\end{aligned}
$$

EVENT: Disable same-cars-in-subset-aux1.

EVENT: Disable same-cars-in-subset-aux2.

```
; now, we want to prove that subset implies sublist for sets
```

DEFINITION:

bad-for-subset-implies-sublist $(set1, set2)$
$= \quad (\text{subsetp}(set1, set2)$
$\quad\quad \wedge \quad \text{setp}(set1)$
$\quad\quad \wedge \quad \text{setp}(set2)$
$\quad\quad \wedge \quad (\neg\, \text{sublistp}(set1, set2)))$

THEOREM: subset-implies-sublist-aux1
$(\neg\, \text{listp}(set1)) \rightarrow (\neg\, \text{bad-for-subset-implies-sublist}(set1, set2))$

THEOREM: subset-implies-sublist-aux2
$(\neg\, \text{listp}(set2)) \rightarrow (\neg\, \text{bad-for-subset-implies-sublist}(set1, set2))$

```
; now, we do induction-- if (subsetp set1 set2) and the cars are the
; same, and the cdrs are subsets
```

THEOREM: subset-implies-sublist-aux3
$(\text{listp}(set1)$
$\wedge \quad \text{listp}(set2)$
$\wedge \quad (\text{car}(set2) = \text{car}(set1))$
$\wedge \quad \text{sublistp}(\text{cdr}(set1), \text{cdr}(set2)))$
$\rightarrow \quad \text{sublistp}(set1, set2)$

THEOREM: subset-implies-sublist-aux4
$(\text{listp}(set1)$
$\wedge \quad \text{listp}(set2)$
$\wedge \quad (\text{car}(set2) = \text{car}(set1))$
$\wedge \quad \text{bad-for-subset-implies-sublist}(set1, set2))$
$\rightarrow \quad \text{bad-for-subset-implies-sublist}(\text{cdr}(set1), \text{cdr}(set2))$

THEOREM: subset-implies-sublist-aux5
$(\text{listp}(set1)$
$\wedge \quad \text{listp}(set2)$
$\wedge \quad (\text{car}(set2) < \text{car}(set1))$
$\wedge \quad \text{bad-for-subset-implies-sublist}(set1, set2))$
$\rightarrow \quad \text{bad-for-subset-implies-sublist}(set1, \text{cdr}(set2))$

```
; Now, use the fact that (car set2) <= (car set1)
```

THEOREM: subset-implies-sublist-aux6
$(\text{listp}(set1) \wedge \text{listp}(set2) \wedge \text{bad-for-subset-implies-sublist}(set1, set2))$
$\rightarrow \quad ((\text{car}(set2) < \text{car}(set1)) \vee (\text{car}(set2) = \text{car}(set1)))$

THEOREM: subset-implies-sublist-aux7
$(\text{listp}(set1) \wedge \text{listp}(set2) \wedge \text{bad-for-subset-implies-sublist}(set1, set2))$
$\rightarrow \quad (\text{bad-for-subset-implies-sublist}(set1, \text{cdr}(set2))$
$\quad\quad \vee \quad \text{bad-for-subset-implies-sublist}(\text{cdr}(set1), \text{cdr}(set2)))$

```
; so there are no bad pairs, by induction -- we have to force the
; correct induction
```

DEFINITION:
subset-implies-sublist-kludge ($set1$, $set2$)
= **if** listp ($set1$)
  then if listp ($set2$)
    then subset-implies-sublist-kludge (cdr ($set1$), cdr ($set2$))
      + subset-implies-sublist-kludge ($set1$, cdr ($set2$))
    else 0 **endif**
  else 0 **endif**

THEOREM: subset-implies-sublist-aux8
$\neg$ bad-for-subset-implies-sublist ($set1$, $set2$)

EVENT: Disable subset-implies-sublist-kludge.

EVENT: Disable subset-implies-sublist-aux1.

EVENT: Disable subset-implies-sublist-aux2.

EVENT: Disable subset-implies-sublist-aux3.

EVENT: Disable subset-implies-sublist-aux4.

EVENT: Disable subset-implies-sublist-aux5.

EVENT: Disable subset-implies-sublist-aux6.

EVENT: Disable subset-implies-sublist-aux7.

THEOREM: subset-implies-sublist
(subsetp ($set1$, $set2$) $\wedge$ setp ($set1$) $\wedge$ setp ($set2$)) $\rightarrow$ sublistp ($set1$, $set2$)

EVENT: Disable subset-implies-sublist-aux8.

EVENT: Disable bad-for-subset-implies-sublist.

```
;;;;;;;;;;
```

```
; now,, we prove that every subset of set is in the power-set
```

THEOREM: all-subsets
$(\text{setp}\,(\textit{set}) \land \text{setp}\,(x) \land \text{subsetp}\,(x,\ \textit{set})) \rightarrow (x \in \text{power-set}\,(\textit{set}))$

```
;;;;;;;;;;;;;;;;;;;  extensionality
; Two standard sets, s1, s2, are equal if they have the same members:
;  (implies
;      (and (setp s1) (setp s2) (subsetp s1 s2) (subsetp s2 s1))
;      (equal s1 s2) )
;  The proof is "by induction", but it requires a bunch of auxilliaries,
;  which we intruduce, and then disable after the proof
```

DEFINITION:
bad-for-ext $(\textit{s1},\ \textit{s2})$
$$
\begin{aligned}
= \quad & (\text{setp}\,(\textit{s1}) \\
& \land \quad \text{setp}\,(\textit{s2}) \\
& \land \quad \text{subsetp}\,(\textit{s1},\ \textit{s2}) \\
& \land \quad \text{subsetp}\,(\textit{s2},\ \textit{s1}) \\
& \land \quad (\textit{s1} \neq \textit{s2}))
\end{aligned}
$$

```
; so, we want to show, by induction, that this can't happen.
```

THEOREM: extensionality-aux1
$(\neg\ \text{listp}\,(\textit{s1})) \rightarrow (\neg\ \text{bad-for-ext}\,(\textit{s1},\ \textit{s2}))$

THEOREM: extensionality-aux2
$(\neg\ \text{listp}\,(\textit{s2})) \rightarrow (\neg\ \text{bad-for-ext}\,(\textit{s1},\ \textit{s2}))$

```
; now, (bad-for-ext s1 s2) implies that s1, s2 aren't empty, and we can
; look at their first element.  By (subsetp s1 s2) (subsetp s2 s1)
; the first elements are the same.
```

THEOREM: extensionality-aux3
$$
\begin{aligned}
& (\text{setp}\,(\textit{s1}) \\
& \ \land \quad \text{setp}\,(\textit{s2}) \\
& \ \land \quad \text{listp}\,(\textit{s1}) \\
& \ \land \quad \text{listp}\,(\textit{s2}) \\
& \ \land \quad \text{subsetp}\,(\textit{s1},\ \textit{s2}) \\
& \ \land \quad \text{subsetp}\,(\textit{s2},\ \textit{s1})) \\
& \rightarrow \quad (\text{car}\,(\textit{s1}) = \text{car}\,(\textit{s2}))
\end{aligned}
$$

THEOREM: extensionality-aux4
$(\text{listp}(s1) \land \text{listp}(s2) \land \text{bad-for-ext}(s1, s2)) \rightarrow (\text{car}(s1) = \text{car}(s2))$

EVENT: Disable extensionality-aux3.

```
; now, we should get that the cdrs are bad
```

THEOREM: extensionality-aux5
$(\text{listp}(s1) \land \text{listp}(s2) \land \text{bad-for-ext}(s1, s2))$
$\rightarrow \quad \text{bad-for-ext}(\text{cdr}(s1), \text{cdr}(s2))$

EVENT: Disable extensionality-aux4.

DEFINITION:
extensionality-kludge $(s1, s2)$
= **if** $\neg$ listp $(s1)$ **then** `0`
elseif $\neg$ listp $(s2)$ **then** `0`
else extensionality-kludge (cdr $(s1)$, cdr $(s2)$) **endif**

THEOREM: extensionality-aux6
$\neg \text{bad-for-ext}(s1, s2)$

EVENT: Disable extensionality-aux1.

EVENT: Disable extensionality-aux2.

EVENT: Disable extensionality-aux5.

EVENT: Disable extensionality-aux6.

THEOREM: extensionality
$(\text{setp}(s1) \land \text{setp}(s2) \land \text{subsetp}(s1, s2) \land \text{subsetp}(s2, s1))$
$\rightarrow \quad (s1 = s2)$

EVENT: Disable extensionality-kludge.

```
;;;;;;;;;;  segments

; (segment 3 6) is (3 4 5 6)
; These are special kinds of sets.  More on them will appear
; below; here we just put the defn and a few basic facts.
```

DEFINITION:
segment (m, n)
$=$ **if** $(m \leq n) \wedge (m \in \mathbf{N}) \wedge (n \in \mathbf{N})$ **then** cons $(m, \text{segment}\,(1 + m, n))$
else nil endif

THEOREM: size-of-segment
$((m \leq n) \wedge (m \in \mathbf{N}) \wedge (n \in \mathbf{N}))$
$\rightarrow$ $(\text{length}\,(\text{segment}\,(m, n)) = ((1 + n) - m))$

THEOREM: members-of-segment
$((m \leq n) \wedge (m \in \mathbf{N}) \wedge (n \in \mathbf{N}))$
$\rightarrow$ $((x \in \text{segment}\,(m, n)) = ((x \in \mathbf{N}) \wedge (x \leq n) \wedge (m \leq x)))$

EVENT: Disable segment.

```
; for now
```

```
; some lemmas about length:
```

THEOREM: length-of-sublist
$\text{sublistp}\,(lst1, lst2) \rightarrow (\text{length}\,(lst2) \not< \text{length}\,(lst1))$

THEOREM: length-of-subset
$(\text{setp}\,(x) \wedge \text{setp}\,(y) \wedge \text{subsetp}\,(x, y)) \rightarrow (\text{length}\,(y) \not< \text{length}\,(x))$

```
; a few more set facts
```

THEOREM: empty-subset
$(\neg\, \text{listp}\,(s1)) \rightarrow \text{subsetp}\,(s1, s2)$

THEOREM: car-before-cadr
$(\text{listp}\,(\text{cdr}\,(s)) \wedge \text{setp}\,(s)) \rightarrow (\text{car}\,(s) < \text{cadr}\,(s))$

```
; now, we prove that (and (setp s1) (setp s2) (listp s1) (subsetp s1 s2))
; implies (subsetp (cdr s1) (cdr s2))
; this is by examination of cases:
```

THEOREM: cdr-cdr-subset-aux1
$(\text{setp}\,(s1) \wedge \text{setp}\,(s2) \wedge \text{listp}\,(s1) \wedge \text{subsetp}\,(s1, s2))$
$\rightarrow$ $((\text{car}\,(s2) = \text{car}\,(s1)) \vee (\text{car}\,(s2) < \text{car}\,(s1)))$

THEOREM: cdr-cdr-subset
$(\text{setp}(s1) \land \text{setp}(s2) \land \text{listp}(s1) \land \text{subsetp}(s1, s2))$
$\rightarrow \quad \text{subsetp}(\text{cdr}(s1), \text{cdr}(s2))$

EVENT: Disable cdr-cdr-subset-aux1.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                ORDINALS                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


; ordinalp (the property of representing an ordinal < epsilon_0)
; and ord-lessp (the < on the ordinals) are built-in
; we define all the basic function so that they do the right thing
; on legal notations for ordinals; we don't care what they do on non-ordinals

; Ordinals are represented as:
; The natural number n represents itself.
; (alpha beta gamma . n) represents omega^alpha + omega^beta + omega^gamma + n
; For this to be legal, alpha >= beta >= gamma -- this ensures
;  that each ordinal has a unique representation


; temporarily disable some set stuff
```

EVENT: Disable subsetp.

EVENT: Disable setp.

EVENT: Disable max-is-last.

EVENT: Disable tail-of-a-set.

EVENT: Disable increasing.

EVENT: Disable compare-first-elts.

```
;;;;;;;  some basic properties of order on the ordinals
```

THEOREM: irreflex
$\neg\, \text{ord-lessp}(x, x)$

THEOREM: nocycle
$\neg\,(\text{ord-lessp}\,(x,\ y) \wedge \text{ord-lessp}\,(y,\ x))$

THEOREM: no-cycle-alt
$\text{ord-lessp}\,(y,\ x) \rightarrow (\neg\ \text{ord-lessp}\,(x,\ y))$

THEOREM: trichotomy
$(\text{ordinalp}\,(sigma) \wedge \text{ordinalp}\,(tau))$
$\rightarrow \quad ((sigma = tau) \vee \text{ord-lessp}\,(sigma,\ tau) \vee \text{ord-lessp}\,(tau,\ sigma))$

THEOREM: transitivity
$(\text{ord-lessp}\,(alpha,\ beta) \wedge \text{ord-lessp}\,(beta,\ gamma))$
$\rightarrow \quad \text{ord-lessp}\,(alpha,\ gamma)$

THEOREM: transitivity-alt
$(\text{ordinalp}\,(alpha)$
$\wedge \quad \text{ordinalp}\,(beta)$
$\wedge \quad \text{ordinalp}\,(gamma)$
$\wedge \quad (\neg\ \text{ord-lessp}\,(beta,\ alpha))$
$\wedge \quad \text{ord-lessp}\,(beta,\ gamma))$
$\rightarrow \quad \text{ord-lessp}\,(alpha,\ gamma)$

THEOREM: ords-below-2
$(\text{ordinalp}\,(alpha) \wedge (\neg\ \text{ord-lessp}\,(\mathtt{1},\ alpha)))$
$\rightarrow \quad ((alpha = \mathtt{0}) \vee (alpha = \mathtt{1}))$

THEOREM: postive-ord-lessp
$(\text{ordinalp}\,(gamma) \wedge \text{ordinalp}\,(delta) \wedge \text{ord-lessp}\,(gamma,\ delta))$
$\rightarrow \quad \text{ord-lessp}\,(\mathtt{0},\ delta)$

```
;;;;;;;  successors and limit ordinals
```

DEFINITION:
successorp $(alpha)$
$=$ **if** listp $(alpha)$ **then** successorp (cdr $(alpha)$)
else $\mathtt{0} < alpha$ **endif**

DEFINITION:
$\text{limitp}\,(alpha) = (\text{listp}\,(alpha) \wedge (\neg\ \text{successorp}\,(alpha)))$

THEOREM: successors-are-positive
$(\text{ordinalp}\,(alpha) \wedge \text{successorp}\,(alpha)) \rightarrow \text{ord-lessp}\,(\mathtt{0},\ alpha)$

THEOREM: three-kinds
$\text{ordinalp}\,(alpha)$
$\rightarrow \quad (\text{limitp}\,(alpha) \vee \text{successorp}\,(alpha) \vee (alpha = \mathtt{0}))$

DEFINITION:
successor $(alpha)$
$=$ **if** listp $(alpha)$ **then** cons (car $(alpha)$, successor (cdr $(alpha)$))
else $1 + alpha$ **endif**

```
; the successor of 8 is 9
; the successor of (alpha beta . 8) is (alpha beta . 9)
```

THEOREM: successor-is-an-ordinal
ordinalp $(alpha) \rightarrow$ ordinalp (successor $(alpha)$)

THEOREM: successor-is-a-successor
successorp (successor $(alpha)$)

THEOREM: successor-is-bigger
ord-lessp $(alpha$, successor $(alpha))$

THEOREM: succ-preserves-order
ord-lessp $(alpha, beta) \rightarrow$ ord-lessp (successor $(alpha)$, successor $(beta)$)

THEOREM: nothing-between
$\neg$ (ord-lessp $(alpha, beta) \land$ ord-lessp $(beta$, successor $(alpha)))$

DEFINITION:
predecessor $(alpha)$
$=$ **if** listp $(alpha)$ **then** cons (car $(alpha)$, predecessor (cdr $(alpha)$))
else $alpha - 1$ **endif**

```
; these are inverses of each other
```

THEOREM: predecessor-suc
ordinalp $(alpha) \rightarrow$ (predecessor (successor $(alpha)$) $= alpha$)

THEOREM: suc-predecessor
(ordinalp $(alpha) \land$ successorp $(alpha)$)
$\rightarrow$ (successor (predecessor $(alpha)$) $= alpha$)

THEOREM: predecessor-is-an-ordinal
ordinalp $(alpha) \rightarrow$ ordinalp (predecessor $(alpha)$)

THEOREM: predecessor-is-smaller
(ordinalp $(alpha) \land$ successorp $(alpha)$)
$\rightarrow$ ord-lessp (predecessor $(alpha)$, $alpha$)

THEOREM: predecessor-of-0
predecessor $(\texttt{0}) = \texttt{0}$

THEOREM: predecessor-of-limit
$(\text{ordinalp}\,(alpha) \land \text{limitp}\,(alpha)) \rightarrow (\text{predecessor}\,(alpha) = alpha)$

```
;;;;;;;;;;  ord-leq :  a <= on ordinals
```

DEFINITION: $\text{ord-leq}\,(alpha,\ beta) = (\neg\ \text{ord-lessp}\,(beta,\ alpha))$

THEOREM: leq-as-an-or
$(\text{ordinalp}\,(rho) \land \text{ordinalp}\,(sigma) \land \text{ord-leq}\,(rho,\ sigma))$
$\rightarrow \quad (\text{ord-lessp}\,(rho,\ sigma) \lor (rho = sigma))$

THEOREM: leq-0
$(\text{ordinalp}\,(delta) \land \text{ord-leq}\,(delta,\ \texttt{0})) \rightarrow ((delta = \texttt{0}) = \mathbf{t})$

THEOREM: cars-go-down
$\text{ordinalp}\,(beta) \rightarrow \text{ord-leq}\,(\text{cadr}\,(beta),\ \text{car}\,(beta))$

THEOREM: ord-leq-is-transitive
$(\text{ordinalp}\,(alpha)$
$\land \quad \text{ordinalp}\,(beta)$
$\land \quad \text{ordinalp}\,(gamma)$
$\land \quad \text{ord-leq}\,(alpha,\ beta)$
$\land \quad \text{ord-leq}\,(beta,\ gamma))$
$\rightarrow \quad \text{ord-leq}\,(alpha,\ gamma)$

THEOREM: ord-leq-zero
$(\text{ordinalp}\,(delta) \land \text{ord-leq}\,(delta,\ \texttt{0})) \rightarrow ((delta = \texttt{0}) = \mathbf{t})$

THEOREM: cars-are-ordinals
$\text{ordinalp}\,(alpha) \rightarrow \text{ordinalp}\,(\text{car}\,(alpha))$

THEOREM: cdrs-are-ordinals
$\text{ordinalp}\,(alpha) \rightarrow \text{ordinalp}\,(\text{cdr}\,(alpha))$

```
; Now, we want to prove that (cdr alpha) < alpha
; this should be trivial but seems to require help
```

DEFINITION:
$\text{bad-for-cdrs-are-smaller}\,(alpha)$
$= \quad (\text{ordinalp}\,(alpha)$
$\quad \land \quad \text{listp}\,(alpha)$
$\quad \land \quad (\neg\ \text{ord-lessp}\,(\text{cdr}\,(alpha),\ alpha)))$

THEOREM: cdrs-are-smaller-aux1
$(alpha \simeq \textbf{nil}) \rightarrow (\neg \text{ bad-for-cdrs-are-smaller}(alpha))$

THEOREM: cdrs-are-smaller-aux2
$(\text{cdr}(alpha) \simeq \textbf{nil}) \rightarrow (\neg \text{ bad-for-cdrs-are-smaller}(alpha))$

THEOREM: cdrs-are-smaller-aux3
$$\begin{array}{ll} (\text{ordinalp}(alpha) & \\ \ \wedge & \text{listp}(\text{cdr}(alpha)) \\ \ \wedge & (\text{car}(alpha) = \text{cadr}(alpha)) \\ \ \wedge & \text{ord-lessp}(\text{cddr}(alpha), \text{cdr}(alpha))) \\ \rightarrow & \text{ord-lessp}(\text{cdr}(alpha), alpha) \end{array}$$

THEOREM: cdrs-are-smaller-aux4
$(\text{ordinalp}(alpha) \wedge \text{listp}(\text{cdr}(alpha)) \wedge \text{ord-lessp}(\text{car}(alpha), \text{cadr}(alpha)))$
$\rightarrow \quad \text{ord-lessp}(\text{cdr}(alpha), alpha)$

THEOREM: cdrs-are-smaller-aux5
$(\text{ordinalp}(alpha) \wedge \text{listp}(\text{cdr}(alpha)))$
$\rightarrow \quad ((\text{car}(alpha) = \text{cadr}(alpha)) \vee \text{ord-lessp}(\text{cadr}(alpha), \text{car}(alpha)))$

EVENT: Disable cdrs-are-smaller-aux3.

EVENT: Disable cdrs-are-smaller-aux4.

EVENT: Disable cdrs-are-smaller-aux5.

THEOREM: cdrs-are-smaller-aux6
$(\text{ordinalp}(alpha) \wedge \text{listp}(\text{cdr}(alpha)) \wedge \text{ord-lessp}(\text{cddr}(alpha), \text{cdr}(alpha)))$
$\rightarrow \quad \text{ord-lessp}(\text{cdr}(alpha), alpha)$

THEOREM: cdrs-are-smaller-aux7
$(\text{listp}(\text{cdr}(alpha)) \wedge \text{bad-for-cdrs-are-smaller}(alpha))$
$\rightarrow \quad \text{bad-for-cdrs-are-smaller}(\text{cdr}(alpha))$

EVENT: Disable cdrs-are-smaller-aux2.

THEOREM: cdrs-are-smaller-aux8
$\text{bad-for-cdrs-are-smaller}(alpha) \rightarrow \text{bad-for-cdrs-are-smaller}(\text{cdr}(alpha))$

THEOREM: cdrs-are-smaller-aux9
$\neg \text{ bad-for-cdrs-are-smaller}(alpha)$

EVENT: Disable cdrs-are-smaller-aux1.

EVENT: Disable cdrs-are-smaller-aux6.

EVENT: Disable cdrs-are-smaller-aux7.

EVENT: Disable cdrs-are-smaller-aux8.

EVENT: Disable cdrs-are-smaller-aux9.

THEOREM: cdrs-are-smaller
$(\text{ordinalp}\,(alpha) \land \text{listp}\,(alpha)) \rightarrow \text{ord-lessp}\,(\text{cdr}\,(alpha),\ alpha)$

EVENT: Disable bad-for-cdrs-are-smaller.

THEOREM: cars-of-smaller-ordinals
$(\text{ordinalp}\,(alpha) \land \text{ordinalp}\,(beta) \land \text{ord-lessp}\,(alpha,\ beta))$
$\rightarrow \quad (\text{ord-lessp}\,(\text{car}\,(alpha),\ \text{car}\,(beta)) \lor (\text{car}\,(alpha) = \text{car}\,(beta)))$

THEOREM: limit-not-succ
$(\text{ordinalp}\,(alpha) \land \text{ordinalp}\,(delta) \land \text{limitp}\,(alpha))$
$\rightarrow \quad (\text{successor}\,(delta) \neq alpha)$

THEOREM: succ-below-limit
$(\text{ordinalp}\,(alpha)$
$\land \quad \text{ordinalp}\,(delta)$
$\land \quad \text{ord-lessp}\,(delta,\ alpha)$
$\land \quad \text{limitp}\,(alpha))$
$\rightarrow \quad \text{ord-lessp}\,(\text{successor}\,(delta),\ alpha)$

THEOREM: irreflex-of-ord-leq
$(\text{ordinalp}\,(sigma)$
$\land \quad \text{ordinalp}\,(tau)$
$\land \quad \text{ord-leq}\,(sigma,\ tau)$
$\land \quad \text{ord-leq}\,(tau,\ sigma))$
$\rightarrow \quad ((sigma = tau) = \mathbf{t})$

THEOREM: transitivity-alt2
$(\text{ordinalp}\,(alpha)$
$\land \quad \text{ordinalp}\,(beta)$
$\land \quad \text{ordinalp}\,(gamma)$
$\land \quad \text{ord-lessp}\,(alpha,\ beta)$

$\wedge$ ord-leq (*beta*, *gamma*))
$\rightarrow$ ord-lessp (*alpha*, *gamma*)

```
;;;;;;;;;;;;;;;;;;;;  multiplicities
; (mult delta alpha) is the coefficient of omega^delta in the
; Cantor normal form of alpha.  We want to define this, and
; prove that alpha is determined by its multiplicities
; This is slightly messy, since delta = 0 is a special case
; in the Boyer-Moore notation -- let's do that separately
; (mult 0 alpha) will be (number-part alpha)
```

DEFINITION:
number-part (*alpha*)
= **if** *alpha* $\simeq$ **nil** **then** *alpha*
else number-part (cdr (*alpha*)) **endif**

THEOREM: number-part-is-a-number
ordinalp (*alpha*) $\rightarrow$ (number-part (*alpha*) $\in \mathbf{N}$)

```
; for delta > 0, (mult delta alpha) = (pos-mult delta alpha)
```

DEFINITION:
pos-mult (*delta*, *alpha*)
= **if** *alpha* $\simeq$ **nil** **then** 0
elseif *delta* = car (*alpha*) **then** 1 + pos-mult (*delta*, cdr (*alpha*))
else pos-mult (*delta*, cdr (*alpha*)) **endif**

THEOREM: pos-mult-is-a-number
pos-mult (*delta*, *alpha*) $\in \mathbf{N}$

DEFINITION:
mult (*delta*, *alpha*)
= **if** *delta* = 0 **then** number-part (*alpha*)
else pos-mult (*delta*, *alpha*) **endif**

THEOREM: mult-is-a-number
ordinalp (*alpha*) $\rightarrow$ (mult (*delta*, *alpha*) $\in \mathbf{N}$)

THEOREM: mult-in-a-number
((*alpha* $\in \mathbf{N}$) $\wedge$ (*delta* $\neq$ 0)) $\rightarrow$ (mult (*delta*, *alpha*) = 0)

```
; now, we prove if delta > (car alpha), (mult delta alpha) = 0
; this is true for numbers alpha because the car of a non-list = 0
; Prover needs help to find the correct induction
```

DEFINITION:
bad-for-bound-on-mult (*delta*, *alpha*)
$=$ (ordinalp (*alpha*)
$\wedge$ ordinalp (*delta*)
$\wedge$ ord-lessp (car (*alpha*), *delta*)
$\wedge$ (mult (*delta*, *alpha*) $\neq$ `0`))

```
; we want to show this can't happen
```

THEOREM: bound-on-mult-aux1
(*alpha* $\simeq$ **nil**) $\rightarrow$ ($\neg$ bad-for-bound-on-mult (*delta*, *alpha*))

THEOREM: bound-on-mult-aux2
(listp (*alpha*) $\wedge$ ordinalp (*alpha*))
$\rightarrow$ ($\neg$ ord-lessp (car (*alpha*), cadr (*alpha*)))

THEOREM: bound-on-mult-aux3
ordinalp (*alpha*) $\rightarrow$ ordinalp (car (*alpha*))

THEOREM: bound-on-mult-aux4
ordinalp (*alpha*) $\rightarrow$ ordinalp (cadr (*alpha*))

THEOREM: bound-on-mult-aux5
(listp (*alpha*)
$\wedge$ ordinalp (*alpha*)
$\wedge$ ordinalp (*delta*)
$\wedge$ ord-lessp (car (*alpha*), *delta*))
$\rightarrow$ ord-lessp (cadr (*alpha*), *delta*)

THEOREM: bound-on-mult-aux6
(listp (*alpha*) $\wedge$ bad-for-bound-on-mult (*delta*, *alpha*))
$\rightarrow$ bad-for-bound-on-mult (*delta*, cdr (*alpha*))

DEFINITION:
bound-on-mult-kludge (*alpha*)
$=$ **if** *alpha* $\simeq$ **nil** **then** `0`
else bound-on-mult-kludge (cdr (*alpha*)) **endif**

THEOREM: bound-on-mult-aux7
$\neg$ bad-for-bound-on-mult (*delta*, *alpha*)

THEOREM: bound-on-mult
(ordinalp (*alpha*) $\wedge$ ordinalp (*delta*) $\wedge$ ord-lessp (car (*alpha*), *delta*))
$\rightarrow$ (mult (*delta*, *alpha*) $=$ `0`)

EVENT: Disable bound-on-mult-aux1.

EVENT: Disable bound-on-mult-aux2.

EVENT: Disable bound-on-mult-aux3.

EVENT: Disable bound-on-mult-aux4.

EVENT: Disable bound-on-mult-aux5.

EVENT: Disable bound-on-mult-aux6.

EVENT: Disable bound-on-mult-aux7.

EVENT: Disable bad-for-bound-on-mult.

EVENT: Disable bound-on-mult-kludge.

```
; now, (different-mult alpha beta) returns a delta
; such that (mult delta alpha) != (mult delta beta)
```

DEFINITION:
different-mult (*alpha*, *beta*)
= **if** car (*alpha*) = car (*beta*)
  then if listp (*alpha*) **then** different-mult (cdr (*alpha*), cdr (*beta*))
    else 0 **endif**
  elseif ord-lessp (car (*alpha*), car (*beta*)) **then** car (*beta*)
  else car (*alpha*) **endif**

```
; summarize the features of the definition
```

THEOREM: different-mult-is-ord
(ordinalp (*alpha*) $\land$ ordinalp (*beta*))
$\rightarrow$ ordinalp (different-mult (*alpha*, *beta*))

THEOREM: different-mult-for-numbers
$((alpha \in \mathbf{N}) \land (beta \in \mathbf{N})) \rightarrow (\text{different-mult}\,(alpha,\ beta) = 0)$

```
;;;;;;;;;;  The following is causing us a lot of trouble:
```

EVENT: Disable transitivity-alt.

EVENT: Disable irreflex-of-ord-leq.

THEOREM: different-mult-number-list
$((alpha \in \mathbf{N}) \land \text{ordinalp}\,(beta) \land \text{listp}\,(beta))$
$\rightarrow \quad (\text{different-mult}\,(alpha,\ beta) = \text{car}\,(beta))$

THEOREM: different-mult-list-number
$((beta \in \mathbf{N}) \land \text{ordinalp}\,(alpha) \land \text{listp}\,(alpha))$
$\rightarrow \quad (\text{different-mult}\,(alpha,\ beta) = \text{car}\,(alpha))$

THEOREM: different-mult-for-lists-same-car
$(\text{ordinalp}\,(alpha)$
$\land \quad \text{ordinalp}\,(beta)$
$\land \quad \text{listp}\,(alpha)$
$\land \quad \text{listp}\,(beta)$
$\land \quad (\text{car}\,(alpha) = \text{car}\,(beta)))$
$\rightarrow \quad (\text{different-mult}\,(alpha,\ beta) = \text{different-mult}\,(\text{cdr}\,(alpha),\ \text{cdr}\,(beta)))$

THEOREM: different-mult-for-smaller-car
$\text{ord-lessp}\,(\text{car}\,(alpha),\ \text{car}\,(beta))$
$\rightarrow \quad (\text{different-mult}\,(alpha,\ beta) = \text{car}\,(beta))$

THEOREM: different-mult-for-larger-car
$\text{ord-lessp}\,(\text{car}\,(beta),\ \text{car}\,(alpha))$
$\rightarrow \quad (\text{different-mult}\,(alpha,\ beta) = \text{car}\,(alpha))$

EVENT: Disable different-mult.

```
; now, we have to prove that it works
; this needs some help
```

DEFINITION:
bad-for-different-mult-works $(alpha,\ beta)$
$= \quad (\text{ordinalp}\,(alpha)$
$\quad \land \quad \text{ordinalp}\,(beta)$
$\quad \land \quad (alpha \neq beta)$
$\quad \land \quad (\text{mult}\,(\text{different-mult}\,(alpha,\ beta),\ alpha)$
$\qquad = \quad \text{mult}\,(\text{different-mult}\,(alpha,\ beta),\ beta)))$

```
; if they have different cars, this should follow
; by "bound-on-mult"
```

THEOREM: different-mult-works-aux1
$(\text{ordinalp}\,(\textit{sigma}) \land (\text{car}\,(\textit{sigma}) \neq \mathtt{0}))$
$\rightarrow \quad (\text{mult}\,(\text{car}\,(\textit{sigma}),\ \textit{sigma}) \neq \mathtt{0})$

```
; the following is causing us a lot of trouble
```

EVENT: Disable no-cycle-alt.

EVENT: Disable ord-leq-is-transitive.

THEOREM: different-mult-works-aux2
$\text{ord-lessp}\,(\text{car}\,(\textit{alpha}),\ \text{car}\,(\textit{beta}))$
$\rightarrow \quad (\neg\ \text{bad-for-different-mult-works}\,(\textit{alpha},\ \textit{beta}))$

THEOREM: different-mult-works-aux3
$\text{ord-lessp}\,(\text{car}\,(\textit{beta}),\ \text{car}\,(\textit{alpha}))$
$\rightarrow \quad (\neg\ \text{bad-for-different-mult-works}\,(\textit{alpha},\ \textit{beta}))$

THEOREM: different-mult-works-aux4
$((\textit{alpha} \in \mathbf{N}) \land (\textit{beta} \in \mathbf{N}))$
$\rightarrow \quad (\neg\ \text{bad-for-different-mult-works}\,(\textit{alpha},\ \textit{beta}))$

THEOREM: different-mult-works-aux5
$((\textit{alpha} \in \mathbf{N}) \land \text{ordinalp}\,(\textit{beta}) \land \text{listp}\,(\textit{beta}))$
$\rightarrow \quad \text{ord-lessp}\,(\text{car}\,(\textit{alpha}),\ \text{car}\,(\textit{beta}))$

THEOREM: different-mult-works-aux6
$((\textit{alpha} \simeq \mathbf{nil}) \land \text{listp}\,(\textit{beta}))$
$\rightarrow \quad (\neg\ \text{bad-for-different-mult-works}\,(\textit{alpha},\ \textit{beta}))$

THEOREM: different-mult-works-aux7
$(\text{listp}\,(\textit{alpha}) \land (\textit{beta} \simeq \mathbf{nil}))$
$\rightarrow \quad (\neg\ \text{bad-for-different-mult-works}\,(\textit{alpha},\ \textit{beta}))$

THEOREM: different-mult-works-aux8
$((\textit{alpha} \simeq \mathbf{nil}) \land (\textit{beta} \simeq \mathbf{nil}))$
$\rightarrow \quad (\neg\ \text{bad-for-different-mult-works}\,(\textit{alpha},\ \textit{beta}))$

EVENT: Disable different-mult-works-aux4.

EVENT: Disable different-mult-works-aux5.

EVENT: Disable different-mult-works-aux6.

EVENT: Disable different-mult-works-aux7.

EVENT: Disable different-mult-works-aux8.

THEOREM: different-mult-works-aux9
$((\mathit{alpha} \simeq \mathbf{nil}) \lor (\mathit{beta} \simeq \mathbf{nil}))$
$\rightarrow \quad (\neg\ \text{bad-for-different-mult-works}\,(\mathit{alpha},\ \mathit{beta}))$

```
; so, they are both compound
```

EVENT: Disable different-mult-works-aux1.

EVENT: Disable different-mult-works-aux2.

EVENT: Disable different-mult-works-aux3.

THEOREM: different-mult-works-aux10
$(\text{car}\,(\mathit{beta}) \neq \text{car}\,(\mathit{alpha}))$
$\rightarrow \quad (\neg\ \text{bad-for-different-mult-works}\,(\mathit{alpha},\ \mathit{beta}))$

```
; so, if they are bad,  they are both compound and their cars
; are the same
```

THEOREM: same-mult-for-cdr
$(\text{listp}\,(\mathit{alpha})$
$\land \quad \text{listp}\,(\mathit{beta})$
$\land \quad (\text{car}\,(\mathit{alpha}) = \text{car}\,(\mathit{beta}))$
$\land \quad (\text{mult}\,(\mathit{delta},\ \mathit{alpha}) = \text{mult}\,(\mathit{delta},\ \mathit{beta})))$
$\rightarrow \quad (\text{mult}\,(\mathit{delta},\ \text{cdr}\,(\mathit{alpha})) = \text{mult}\,(\mathit{delta},\ \text{cdr}\,(\mathit{beta})))$

THEOREM: different-mult-works-aux11
$(\text{listp}\,(\mathit{alpha})$
$\land \quad \text{listp}\,(\mathit{beta})$
$\land \quad (\text{car}\,(\mathit{alpha}) = \text{car}\,(\mathit{beta}))$
$\land \quad \text{bad-for-different-mult-works}\,(\mathit{alpha},\ \mathit{beta}))$
$\rightarrow \quad \text{bad-for-different-mult-works}\,(\text{cdr}\,(\mathit{alpha}),\ \text{cdr}\,(\mathit{beta}))$

EVENT: Disable different-mult-works-aux10.

THEOREM: different-mult-works-aux12
$(\text{listp}\,(\mathit{alpha}) \land \text{listp}\,(\mathit{beta}) \land \text{bad-for-different-mult-works}\,(\mathit{alpha},\ \mathit{beta}))$
$\rightarrow \quad \text{bad-for-different-mult-works}\,(\text{cdr}\,(\mathit{alpha}),\ \text{cdr}\,(\mathit{beta}))$

DEFINITION:
different-mult-works-kludge (*alpha*, *beta*)
= **if** listp (*alpha*) $\wedge$ listp (*beta*)
then different-mult-works-kludge (cdr (*alpha*), cdr (*beta*))
else 0 **endif**

THEOREM: different-mult-works-aux13
$\neg$ bad-for-different-mult-works (*alpha*, *beta*)

EVENT: Disable different-mult-works-kludge.

EVENT: Disable different-mult-works-aux9.

EVENT: Disable different-mult-works-aux11.

EVENT: Disable different-mult-works-aux12.

EVENT: Disable different-mult-works-aux13.

THEOREM: different-mult-works
(ordinalp (*alpha*) $\wedge$ ordinalp (*beta*) $\wedge$ (*alpha* $\neq$ *beta*))
$\rightarrow$ (mult (different-mult (*alpha*, *beta*), *alpha*)
$\neq$ mult (different-mult (*alpha*, *beta*), *beta*))

EVENT: Disable bad-for-different-mult-works.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                           NATURAL SUM                                   ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;  alpha # lambda
; this is defined by merging the Cantor Normal Forms
; as with the above, there are two cases because
; of the special treatment of numbers

; we must verify 5 things:
; 0. this is an ordinal
; 1. bound-below:  alpha # lambda > lambda  if alpha != 0
; 2. bound above:  car(alpha # lambda) is the larger of the two cars
; 3. monotonic:  alpha < beta --> alpha # lambda < beta # lambda
; 4. mult-is-additive:  mult(delta, alpha # lambda) =
```

```
;     mult(delta, alpha) + mult(delta, lambda)
;   this will easily imply that # is assoc and commutative

; we do this for lambda in omega and lambda = (sigma . 0)
;  and then put them together

; first alpha # n when n is a number
```

DEFINITION:
num-sharp ($alpha$, n)
$=$ **if** $alpha \simeq$ **nil** **then** $alpha + n$
else cons (car ($alpha$), num-sharp (cdr ($alpha$), n)) **endif**

THEOREM: num-sharp-is-an-ordinal
(ordinalp ($alpha$) $\wedge$ ($n \in \mathbf{N}$)) $\rightarrow$ ordinalp (num-sharp ($alpha$, n))

THEOREM: num-sharp-and-zero
ordinalp ($alpha$) $\rightarrow$ (num-sharp ($alpha$, 0) $=$ $alpha$)

THEOREM: num-sharp-and-successor
num-sharp ($alpha$, 1) $=$ successor ($alpha$)

THEOREM: num-sharp-gets-bigger
(ordinalp ($alpha$) $\wedge$ ($n \in \mathbf{N}$) $\wedge$ ($n \neq 0$))
$\rightarrow$ ord-lessp ($alpha$, num-sharp ($alpha$, n))

THEOREM: car-of-num-sharp
($n \in \mathbf{N}$) $\rightarrow$ (car (num-sharp ($alpha$, n)) $=$ car ($alpha$))

THEOREM: num-sharp-is-monotonic
(ordinalp ($alpha$) $\wedge$ ordinalp ($beta$) $\wedge$ ($n \in \mathbf{N}$) $\wedge$ ord-lessp ($alpha$, $beta$))
$\rightarrow$ ord-lessp (num-sharp ($alpha$, n), num-sharp ($beta$, n))

```
; check out multiplicities
```

THEOREM: number-part-of-num-sharp
number-part (num-sharp ($alpha$, n)) $=$ (number-part ($alpha$) $+$ n)

THEOREM: pos-mult-of-num-sharp
pos-mult ($delta$, num-sharp ($alpha$, n)) $=$ pos-mult ($delta$, $alpha$)

THEOREM: mult-of-num-sharp
(($n \in \mathbf{N}$) $\wedge$ ordinalp ($delta$) $\wedge$ ordinalp ($alpha$))
$\rightarrow$ (mult ($delta$, num-sharp ($alpha$, n))
$=$ (mult ($delta$, $alpha$) $+$ mult ($delta$, n)))

```
; now, alpha # omega^delta where delta > 0
; this is (insert delta alpha) -- we insert delta in the CNF of alpha
```

Definition:
insert ($delta$, $alpha$)
$=$ **if** $alpha \simeq$ **nil** **then** cons ($delta$, $alpha$)
elseif ord-lessp ($delta$, car ($alpha$))
then cons (car ($alpha$), insert ($delta$, cdr ($alpha$)))
else cons ($delta$, $alpha$) **endif**

```
; the car is the larger of the two cars
```

Theorem: car-of-insert-a
(ord-lessp ($delta$, car ($alpha$)) $\wedge$ ordinalp ($alpha$))
$\rightarrow$ (car (insert ($delta$, $alpha$)) $=$ car ($alpha$))

Theorem: car-of-insert-b
($\neg$ ord-lessp ($delta$, car ($alpha$))) $\rightarrow$ (car (insert ($delta$, $alpha$)) $=$ $delta$)

Theorem: insert-is-an-ordinal
(ordinalp ($alpha$) $\wedge$ ordinalp ($delta$) $\wedge$ ($delta \neq$ 0))
$\rightarrow$ ordinalp (insert ($delta$, $alpha$))

```
; now, we want to show that alpha < (insert delta alpha)
; this need some help
```

Theorem: insert-is-cons
(ordinalp ($alpha$)
$\wedge$ ordinalp ($delta$)
$\wedge$ ($delta \neq$ 0)
$\wedge$ ($\neg$ ord-lessp ($delta$, car ($alpha$))))
$\rightarrow$ (insert ($delta$, $alpha$) $=$ cons ($delta$, $alpha$))

Theorem: insert-is-bigger-aux1
(ordinalp ($alpha$)
$\wedge$ ordinalp ($delta$)
$\wedge$ ($delta \neq$ 0)
$\wedge$ listp ($alpha$)
$\wedge$ ($\neg$ ord-lessp ($delta$, car ($alpha$))))
$\rightarrow$ ord-lessp ($alpha$, insert ($delta$, $alpha$))

DEFINITION:
bad-for-insert-is-bigger $(alpha,\ delta)$
$=\quad$ (ordinalp $(alpha)$
$\quad\wedge\quad$ ordinalp $(delta)$
$\quad\wedge\quad$ $(delta \neq \mathtt{0})$
$\quad\wedge\quad$ $(\neg$ ord-lessp $(alpha,$ insert $(delta,\ alpha))))$

THEOREM: insert-is-bigger-aux2
$(alpha \simeq \mathbf{nil}) \rightarrow (\neg$ bad-for-insert-is-bigger $(alpha,\ delta))$

THEOREM: insert-is-bigger-aux3
(listp $(alpha) \wedge (\neg$ ord-lessp $(delta,$ car $(alpha))))$
$\rightarrow\quad(\neg$ bad-for-insert-is-bigger $(alpha,\ delta))$

THEOREM: insert-small-ordinal
(ordinalp $(alpha) \wedge$ listp $(alpha) \wedge$ ord-lessp $(delta,$ car $(alpha)))$
$\rightarrow\quad$ (insert $(delta,\ alpha) =$ cons (car $(alpha),$ insert $(delta,$ cdr $(alpha))))$

THEOREM: insert-is-bigger-aux4
(ordinalp $(alpha)$
$\wedge\quad$ listp $(alpha)$
$\wedge\quad$ ord-lessp $(delta,$ car $(alpha))$
$\wedge\quad$ ord-lessp (cdr $(alpha),$ insert $(delta,$ cdr $(alpha))))$
$\rightarrow\quad$ ord-lessp $(alpha,$ insert $(delta,\ alpha))$

THEOREM: insert-is-bigger-aux5
(listp $(alpha)$
$\wedge\quad$ ord-lessp $(delta,$ car $(alpha))$
$\wedge\quad$ bad-for-insert-is-bigger $(alpha,\ delta))$
$\rightarrow\quad$ bad-for-insert-is-bigger (cdr $(alpha),\ delta)$

THEOREM: insert-is-bigger-aux6
(listp $(alpha) \wedge$ bad-for-insert-is-bigger $(alpha,\ delta))$
$\rightarrow\quad$ bad-for-insert-is-bigger (cdr $(alpha),\ delta)$

THEOREM: insert-is-bigger-aux7
$\neg$ bad-for-insert-is-bigger $(alpha,\ delta)$

THEOREM: insert-is-bigger
(ordinalp $(alpha) \wedge$ ordinalp $(delta) \wedge (delta \neq \mathtt{0}))$
$\rightarrow\quad$ ord-lessp $(alpha,$ insert $(delta,\ alpha))$

EVENT: Disable insert-is-bigger-aux1.

EVENT: Disable insert-is-bigger-aux2.

EVENT: Disable insert-is-bigger-aux3.

EVENT: Disable insert-is-bigger-aux4.

EVENT: Disable insert-is-bigger-aux5.

EVENT: Disable insert-is-bigger-aux6.

EVENT: Disable insert-is-bigger-aux7.

```
;;;; now --  check out multiplicities
```

THEOREM: number-part-of-insert
number-part (insert ($delta$, $alpha$)) = number-part ($alpha$)

THEOREM: pos-mult-same
pos-mult ($delta$, insert ($delta$, $alpha$)) = (1 + pos-mult ($delta$, $alpha$))

THEOREM: pos-mult-different
($x \neq delta$) $\rightarrow$ (pos-mult (x, insert ($delta$, $alpha$)) = pos-mult (x, $alpha$))

THEOREM: mult-of-insert-a
(ordinalp ($alpha$) $\wedge$ ($sigma \neq delta$))
$\rightarrow$ (mult ($sigma$, insert ($delta$, $alpha$)) = mult ($sigma$, $alpha$))

THEOREM: mult-of-insert-b
(ordinalp ($alpha$) $\wedge$ ($delta \neq \mathtt{0}$))
$\rightarrow$ (mult ($delta$, insert ($delta$, $alpha$)) = (1 + mult ($delta$, $alpha$)))

```
; we still have to check monotonicity --
; but first, define #  and prove it's assoc and commut
; just to make sure we've got it right.
```

DEFINITION:
sharp ($alpha$, $beta$)
= **if** $beta \simeq$ **nil then** num-sharp ($alpha$, $beta$)
else insert (car ($beta$), sharp ($alpha$, cdr ($beta$))) **endif**

THEOREM: sharp-is-an-ordinal
(ordinalp ($alpha$) $\wedge$ ordinalp ($beta$)) $\rightarrow$ ordinalp (sharp ($alpha$, $beta$))

THEOREM: sharp-with-0
$\text{ordinalp}\,(\mathit{alpha}) \rightarrow (\text{sharp}\,(\mathit{alpha},\ \mathtt{0}) = \mathit{alpha})$

THEOREM: sharp-gets-bigger
$(\text{ordinalp}\,(\mathit{alpha}) \wedge \text{ordinalp}\,(\mathit{beta}) \wedge (\mathit{beta} \neq \mathtt{0}))$
$\rightarrow \quad \text{ord-lessp}\,(\mathit{alpha},\ \text{sharp}\,(\mathit{alpha},\ \mathit{beta}))$

```
; now we prove that (car (sharp alpha beta)) =  (car alpha)
; in the case that (car alpha) >= (car beta) -- the other way
; around will follow once we prove # is commutative
```

THEOREM: car-of-insert-aux1
$((\mathit{delta} = \text{car}\,(\mathit{alpha})) \wedge \text{ordinalp}\,(\mathit{alpha}) \wedge \text{ordinalp}\,(\mathit{delta}))$
$\rightarrow \quad (\text{car}\,(\text{insert}\,(\mathit{delta},\ \mathit{alpha})) = \text{car}\,(\mathit{alpha}))$

THEOREM: car-of-insert-c
$(\text{ord-leq}\,(\mathit{delta},\ \text{car}\,(\mathit{alpha})) \wedge \text{ordinalp}\,(\mathit{alpha}) \wedge \text{ordinalp}\,(\mathit{delta}))$
$\rightarrow \quad (\text{car}\,(\text{insert}\,(\mathit{delta},\ \mathit{alpha})) = \text{car}\,(\mathit{alpha}))$

THEOREM: car-of-sharp-aux1
$((\neg\ \text{listp}\,(\mathit{beta})) \wedge \text{ordinalp}\,(\mathit{alpha}) \wedge \text{ordinalp}\,(\mathit{beta}))$
$\rightarrow \quad (\text{car}\,(\text{sharp}\,(\mathit{alpha},\ \mathit{beta})) = \text{car}\,(\mathit{alpha}))$

THEOREM: car-of-sharp-aux2
$(\text{ord-leq}\,(\text{car}\,(\mathit{beta}),\ \text{car}\,(\mathit{alpha})) \wedge \text{ordinalp}\,(\mathit{alpha}) \wedge \text{ordinalp}\,(\mathit{beta}))$
$\rightarrow \quad \text{ord-leq}\,(\text{cadr}\,(\mathit{beta}),\ \text{car}\,(\mathit{alpha}))$

THEOREM: car-of-sharp-a
$(\text{ord-leq}\,(\text{car}\,(\mathit{beta}),\ \text{car}\,(\mathit{alpha})) \wedge \text{ordinalp}\,(\mathit{alpha}) \wedge \text{ordinalp}\,(\mathit{beta}))$
$\rightarrow \quad (\text{car}\,(\text{sharp}\,(\mathit{alpha},\ \mathit{beta})) = \text{car}\,(\mathit{alpha}))$

EVENT: Disable car-of-sharp-aux1.

EVENT: Disable car-of-sharp-aux2.

```
; now, check out that multiplicities are additive
```

DEFINITION:
$\text{bad-for-mult-of-sharp}\,(\mathit{sigma},\ \mathit{alpha},\ \mathit{beta})$
$= \quad (\text{ordinalp}\,(\mathit{sigma})$
$\quad\quad \wedge \quad \text{ordinalp}\,(\mathit{alpha})$
$\quad\quad \wedge \quad \text{ordinalp}\,(\mathit{beta})$
$\quad\quad \wedge \quad (\text{mult}\,(\mathit{sigma},\ \text{sharp}\,(\mathit{alpha},\ \mathit{beta}))$
$\quad\quad\quad\quad \neq \quad (\text{mult}\,(\mathit{sigma},\ \mathit{alpha}) + \text{mult}\,(\mathit{sigma},\ \mathit{beta}))))$

THEOREM: mult-of-sharp-aux1
$(beta \simeq \mathbf{nil}) \rightarrow (\neg\ \text{bad-for-mult-of-sharp}\,(sigma,\ alpha,\ beta))$

THEOREM: mult-of-sharp-aux2
$(\text{listp}\,(beta) \land \text{ordinalp}\,(beta)) \rightarrow (\text{car}\,(beta) \neq \mathtt{0})$

THEOREM: mult-of-sharp-aux3
$$
\begin{array}{ll}
(\text{listp}\,(beta) & \\
\ \land & (sigma \neq \text{car}\,(beta)) \\
\ \land & \text{ordinalp}\,(sigma) \\
\ \land & \text{ordinalp}\,(alpha) \\
\ \land & \text{ordinalp}\,(beta)) \\
\rightarrow & (\text{mult}\,(sigma,\ \text{sharp}\,(alpha,\ \text{cdr}\,(beta))) = \text{mult}\,(sigma,\ \text{sharp}\,(alpha,\ beta)))
\end{array}
$$

THEOREM: mult-of-sharp-aux4
$$
\begin{array}{ll}
(\text{listp}\,(beta) \land (sigma \neq \text{car}\,(beta))) & \\
\rightarrow & (\text{mult}\,(sigma,\ \text{cdr}\,(beta)) = \text{mult}\,(sigma,\ beta))
\end{array}
$$

THEOREM: mult-of-sharp-aux5
$$
\begin{array}{ll}
(\text{listp}\,(beta) & \\
\ \land & \text{bad-for-mult-of-sharp}\,(sigma,\ alpha,\ beta) \\
\ \land & (sigma \neq \text{car}\,(beta))) \\
\rightarrow & \text{bad-for-mult-of-sharp}\,(sigma,\ alpha,\ \text{cdr}\,(beta))
\end{array}
$$

THEOREM: mult-of-sharp-aux6
$$
\begin{array}{ll}
(\text{listp}\,(beta) \land \text{ordinalp}\,(beta)) & \\
\rightarrow & (\text{mult}\,(\text{car}\,(beta),\ beta) = (1 + \text{mult}\,(\text{car}\,(beta),\ \text{cdr}\,(beta))))
\end{array}
$$

THEOREM: mult-of-sharp-aux7
$$
\begin{array}{ll}
(\text{listp}\,(beta) \land \text{bad-for-mult-of-sharp}\,(\text{car}\,(beta),\ alpha,\ beta)) & \\
\rightarrow & \text{bad-for-mult-of-sharp}\,(\text{car}\,(beta),\ alpha,\ \text{cdr}\,(beta))
\end{array}
$$

THEOREM: mult-of-sharp-aux8
$$
\begin{array}{ll}
(\text{listp}\,(beta) \land \text{bad-for-mult-of-sharp}\,(sigma,\ alpha,\ beta)) & \\
\rightarrow & \text{bad-for-mult-of-sharp}\,(sigma,\ alpha,\ \text{cdr}\,(beta))
\end{array}
$$

THEOREM: mult-of-sharp-aux9
$\neg\ \text{bad-for-mult-of-sharp}\,(sigma,\ alpha,\ beta)$

THEOREM: mult-of-sharp
$$
\begin{array}{lll}
(\text{ordinalp}\,(sigma) \land \text{ordinalp}\,(alpha) \land \text{ordinalp}\,(beta)) & & \\
\rightarrow & (\text{mult}\,(sigma,\ \text{sharp}\,(alpha,\ beta)) & \\
 & \quad = & (\text{mult}\,(sigma,\ alpha) + \text{mult}\,(sigma,\ beta)))
\end{array}
$$

EVENT: Disable bad-for-mult-of-sharp.

EVENT: Disable mult-of-sharp-aux1.

EVENT: Disable mult-of-sharp-aux2.

EVENT: Disable mult-of-sharp-aux3.

EVENT: Disable mult-of-sharp-aux4.

EVENT: Disable mult-of-sharp-aux5.

EVENT: Disable mult-of-sharp-aux6.

EVENT: Disable mult-of-sharp-aux7.

EVENT: Disable mult-of-sharp-aux8.

EVENT: Disable mult-of-sharp-aux9.

```
; now, we should show that sharp is commutative and associative
```

THEOREM: commut-of-sharp
(ordinalp (*alpha*) $\land$ ordinalp (*beta*))
$\rightarrow$ (sharp (*alpha*, *beta*) = sharp (*beta*, *alpha*))

THEOREM: assoc-of-sharp
(ordinalp (*alpha*) $\land$ ordinalp (*beta*) $\land$ ordinalp (*gamma*))
$\rightarrow$ (sharp (sharp (*alpha*, *beta*), *gamma*) = sharp (*alpha*, sharp (*beta*, *gamma*)))

```
;;; now, we can finish the discussion of the car of a #
```

THEOREM: car-of-sharp-b
(ord-leq (car (*alpha*), car (*beta*)) $\land$ ordinalp (*alpha*) $\land$ ordinalp (*beta*))
$\rightarrow$ (car (sharp (*alpha*, *beta*)) = car (*beta*))

```
; now, turn to monotonicity:  alpha < beta --> alpha # lambda < beta # lambda
; this should follow by induction if we can do monotonicty of insert

; first, do it in the case that delta >= (car beta)
```

EVENT: Disable car-of-insert-aux1.

THEOREM: monotonicity-of-insert-aux1
(ordinalp ($alpha$)
 $\wedge$ ordinalp ($beta$)
 $\wedge$ ordinalp ($delta$)
 $\wedge$ ($\neg$ ord-lessp ($delta$, car ($beta$)))
 $\wedge$ ($delta \neq$ 0)
 $\wedge$ ord-lessp ($alpha$, $beta$))
$\rightarrow$ ord-lessp (insert ($delta$, $alpha$), insert ($delta$, $beta$))

```
; now, consider case that delta < (car beta)

; first, suppose also (car alpha) < (car beta)
; Then (car (insert delta alpha)) < (car beta)
; so (insert delta alpha) < beta < (insert delta beta)
```

THEOREM: monotonicity-of-insert-aux2
(ordinalp ($alpha$)
 $\wedge$ ordinalp ($sigma$)
 $\wedge$ ordinalp ($delta$)
 $\wedge$ ord-lessp ($delta$, $sigma$)
 $\wedge$ ord-lessp (car ($alpha$), $sigma$))
$\rightarrow$ ord-lessp (car (insert ($delta$, $alpha$)), $sigma$)

THEOREM: monotonicity-of-insert-aux3
(ordinalp ($alpha$)
 $\wedge$ ordinalp ($beta$)
 $\wedge$ ordinalp ($delta$)
 $\wedge$ ord-lessp ($delta$, car ($beta$))
 $\wedge$ ($delta \neq$ 0)
 $\wedge$ ord-lessp (car ($alpha$), car ($beta$)))
$\rightarrow$ ord-lessp (insert ($delta$, $alpha$), $beta$)

THEOREM: monotonicity-of-insert-aux4
(ordinalp ($alpha$)
 $\wedge$ ordinalp ($beta$)
 $\wedge$ ordinalp ($delta$)
 $\wedge$ ord-lessp ($delta$, car ($beta$))
 $\wedge$ ($delta \neq$ 0)
 $\wedge$ ord-lessp (car ($alpha$), car ($beta$)))
$\rightarrow$ ord-lessp (insert ($delta$, $alpha$), insert ($delta$, $beta$))

```
; this concludes the case that delta < (car beta) and (car alpha) < (car beta)
; the remaining case is that delta < (car beta) and (car alpha) = (car beta)
; this should proceed by induction
```

DEFINITION:
bad-for-monotonicity-of-insert $(alpha,\ beta,\ delta)$
$= \ \ $ (ordinalp $(alpha)$
$\quad\wedge\ $ ordinalp $(beta)$
$\quad\wedge\ $ ordinalp $(delta)$
$\quad\wedge\ $ $(delta \neq 0)$
$\quad\wedge\ $ ord-lessp $(alpha,\ beta)$
$\quad\wedge\ $ $(\neg$ ord-lessp (insert $(delta,\ alpha)$, insert $(delta,\ beta)$)))

THEOREM: monotonicity-of-insert-aux5
bad-for-monotonicity-of-insert $(alpha,\ beta,\ delta)$
$\rightarrow\ \ $ ord-lessp $(delta$, car $(beta))$

THEOREM: monotonicity-of-insert-aux6
bad-for-monotonicity-of-insert $(alpha,\ beta,\ delta)$
$\rightarrow\ \ $ $(\neg$ ord-lessp (car $(alpha)$, car $(beta)$))

THEOREM: monotonicity-of-insert-aux7
bad-for-monotonicity-of-insert $(alpha,\ beta,\ delta)$
$\rightarrow\ \ $ (car $(alpha)$ = car $(beta)$)

```
; now we know that any counter-example satisfies
; delta  car alpha = car beta
; so, the insert's start off with delta, and we can use induction

; check that alpha and beta are not numbers
```

THEOREM: monotonicity-of-insert-aux8
bad-for-monotonicity-of-insert $(alpha,\ beta,\ delta)$
$\rightarrow\ \ $ (listp $(alpha)$ $\wedge$ listp $(beta)$)

THEOREM: monotonicity-of-insert-aux9
bad-for-monotonicity-of-insert $(alpha,\ beta,\ delta)$
$\rightarrow\ \ $ bad-for-monotonicity-of-insert (cdr $(alpha)$, cdr $(beta)$, $delta$)

DEFINITION:
monotonicity-of-insert-kludge $(alpha,\ beta)$
$= \ \ $ **if** listp $(alpha)$ $\wedge$ listp $(beta)$
then monotonicity-of-insert-kludge (cdr $(alpha)$, cdr $(beta)$)
else 0 **endif**

THEOREM: monotonicity-of-insert-aux10
$\neg$ bad-for-monotonicity-of-insert ($alpha$, $beta$, $delta$)

THEOREM: monotonicity-of-insert
(ordinalp ($alpha$)
 $\wedge$ ordinalp ($beta$)
 $\wedge$ ordinalp ($delta$)
 $\wedge$ ($delta \neq 0$)
 $\wedge$ ord-lessp ($alpha$, $beta$))
$\rightarrow$ ord-lessp (insert ($delta$, $alpha$), insert ($delta$, $beta$))

EVENT: Disable monotonicity-of-insert-aux1.

EVENT: Disable monotonicity-of-insert-aux2.

EVENT: Disable monotonicity-of-insert-aux3.

EVENT: Disable monotonicity-of-insert-aux4.

EVENT: Disable monotonicity-of-insert-aux5.

EVENT: Disable monotonicity-of-insert-aux6.

EVENT: Disable monotonicity-of-insert-aux7.

EVENT: Disable monotonicity-of-insert-aux8.

EVENT: Disable monotonicity-of-insert-aux9.

EVENT: Disable monotonicity-of-insert-aux10.

EVENT: Disable monotonicity-of-insert-kludge.

EVENT: Disable bad-for-monotonicity-of-insert.

```
; now, we can prove monotonicity of sharp
```

THEOREM: monotonicity-of-sharp
(ordinalp ($alpha$)
$\land$ ordinalp ($beta$)
$\land$ ordinalp ($lambda$)
$\land$ ord-lessp ($alpha$, $beta$))
$\rightarrow$ ord-lessp (sharp ($alpha$, $lambda$), sharp ($beta$, $lambda$))

```
;;;;;;;  disable a bunch of stuff --  these are causing
;;;; a lot of trouble by firing when we don't want them to:
```

EVENT: Disable ord-lessp.

EVENT: Disable ord-leq.

EVENT: Disable leq-0.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                          NORMS AND PRED                                ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;   norms = count ;
```

DEFINITION: norm (x) = count (x)

```
; This defn is just so that we can disable norm without disabling count
```

THEOREM: norm-is-a-number
norm (x) $\in \mathbf{N}$

THEOREM: norm-of-a-number
($n \in \mathbf{N}$) $\rightarrow$ (norm (n) = n)

THEOREM: norm-of-successor
ordinalp ($alpha$) $\rightarrow$ (norm (successor ($alpha$)) = (1 + norm ($alpha$)))

THEOREM: norm-of-predecessor
(ordinalp ($alpha$) $\land$ successorp ($alpha$))
$\rightarrow$ (norm (predecessor ($alpha$)) = (norm ($alpha$) $-$ 1))

THEOREM: norm-of-cons
listp ($alpha$)
$\rightarrow$ (norm ($alpha$) = (1 + (norm (car ($alpha$)) + norm (cdr ($alpha$)))))

THEOREM: norm-non-zero
$\text{listp}\,(alpha) \rightarrow (\text{norm}\,(alpha) \neq \mathtt{0})$

EVENT: Disable norm.

```
; now, (norm-up-to n) will contain all ordinals with
; norm n or below, plus a lot of other stuff
```

DEFINITION:
norm-up-to (n)
$=$ **if** $n \simeq \mathtt{0}$ **then** list $(\mathtt{0})$
else append (segment $(\mathtt{0},\ n)$,
product (norm-up-to $(n - 1)$, norm-up-to $(n - 1)$)) **endif**

THEOREM: norm-up-to-big-enuf-aux1
$(\text{ordinalp}\,(alpha) \wedge (\text{norm}\,(alpha) \leq n) \wedge (alpha \simeq \mathbf{nil}))$
$\rightarrow \quad (alpha \in \text{norm-up-to}\,(n))$

```
; now, we just consider (listp alpha)
```

THEOREM: norm-up-to-big-enuf-aux2
$(\text{listp}\,(alpha) \wedge (\text{norm}\,(alpha) \leq n)) \rightarrow (n \not\simeq \mathtt{0})$

THEOREM: norm-up-to-big-enuf-aux3
$(\text{ordinalp}\,(alpha) \wedge (\text{norm}\,(alpha) \leq n) \wedge (n \simeq \mathtt{0}))$
$\rightarrow \quad (alpha \in \text{norm-up-to}\,(n))$

THEOREM: norm-up-to-big-enuf-aux4
$((x \in \text{norm-up-to}\,(n)) \wedge (y \in \text{norm-up-to}\,(n)))$
$\rightarrow \quad (\text{cons}\,(x,\ y) \in \text{norm-up-to}\,(1 + n))$

EVENT: Disable norm-up-to.

EVENT: Disable norm-of-cons.

```
; needed for next defn to work
```

DEFINITION:
norm-up-to-big-enuf-kludge $(alpha,\ n)$
$=$ **if** listp $(alpha) \wedge (n \not\simeq \mathtt{0})$
then norm-up-to-big-enuf-kludge (car $(alpha)$, $n - 1$)
$+$ norm-up-to-big-enuf-kludge (cdr $(alpha)$, $n - 1$)
else $\mathtt{0}$ **endif**

EVENT: Enable norm-of-cons.

DEFINITION:
bad-for-norm-up-to-big-enuf $(alpha, n)$
$=$ (ordinalp $(alpha)$
$\wedge$ (norm $(alpha) \leq n$)
$\wedge$ $(alpha \notin$ norm-up-to $(n)))$

THEOREM: norm-up-to-big-enuf-aux5
bad-for-norm-up-to-big-enuf $(alpha, n) \rightarrow$ (listp $(alpha) \wedge (n \not\simeq 0)$)

THEOREM: norm-up-to-big-enuf-aux6
(listp $(alpha)$
$\wedge$ $(n \not\simeq 0)$
$\wedge$ $(\neg$ bad-for-norm-up-to-big-enuf (car $(alpha)$, $n - 1$))
$\wedge$ $(\neg$ bad-for-norm-up-to-big-enuf (cdr $(alpha)$, $n - 1$)))
$\rightarrow$ $(\neg$ bad-for-norm-up-to-big-enuf $(alpha, n))$

THEOREM: norm-up-to-big-enuf-aux7
$\neg$ bad-for-norm-up-to-big-enuf $(alpha, n)$

THEOREM: norm-up-to-big-enuf
(ordinalp $(alpha) \wedge$ (norm $(alpha) \leq n$)) $\rightarrow$ $(alpha \in$ norm-up-to $(n))$

EVENT: Disable norm-up-to-big-enuf-aux1.

EVENT: Disable norm-up-to-big-enuf-aux2.

EVENT: Disable norm-up-to-big-enuf-aux3.

EVENT: Disable norm-up-to-big-enuf-aux4.

EVENT: Disable norm-up-to-big-enuf-aux5.

EVENT: Disable norm-up-to-big-enuf-aux6.

EVENT: Disable norm-up-to-big-enuf-aux7.

EVENT: Disable norm-up-to-big-enuf-kludge.

EVENT: Disable bad-for-norm-up-to-big-enuf.

```
;;;; norm of sharp
; prove that the norm of alpha # beta is the sum of the norms
```

THEOREM: norm-of-num-sharp
(ordinalp ($alpha$) $\wedge$ ($n \in \mathbf{N}$))
$\rightarrow$ (norm (num-sharp ($alpha$, n)) = (norm ($alpha$) + norm (n)))

THEOREM: norm-of-insert
ordinalp ($alpha$)
$\rightarrow$ (norm (insert ($delta$, $alpha$)) = (1 + (norm ($delta$) + norm ($alpha$))))

THEOREM: norm-of-sharp
(ordinalp ($alpha$) $\wedge$ ordinalp ($beta$))
$\rightarrow$ (norm (sharp ($alpha$, $beta$)) = (norm ($alpha$) + norm ($beta$)))

```
;;;;;;;;;;;;;;;;;;  (pred alpha n)  = {alpha}(n)
; this is 0 if alpha = 0   ; otherwise, it's the largest ordinal
;   beta < alpha such that (norm beta) <= (norm alpha) + magic(n)

; first, define (max-below alpha lst) to be the largest ordinal in lst
; of norm <= n
; which is less than alpha.  Define it to be 0 in the default cases.
```

DEFINITION:
max-below ($alpha$, lst, n)
= **if** listp (lst)
  then if ordinalp (car (lst))
    $\wedge$ (norm (car (lst)) $\leq n$)
    $\wedge$ ord-lessp (max-below ($alpha$, cdr (lst), n), car (lst))
    $\wedge$ ord-lessp (car (lst), $alpha$) **then** car (lst)
   else max-below ($alpha$, cdr (lst), n) **endif**
  else 0 **endif**

```
; let's prove the basic properties of this and then disable it
```

THEOREM: max-below-is-an-ordinal
ordinalp (max-below ($alpha$, lst, n))

THEOREM: max-below-is-below
ord-lessp (0, $alpha$) $\rightarrow$ ord-lessp (max-below ($alpha$, lst, n), $alpha$)

THEOREM: max-below-is-max
$$(\text{ordinalp}(\mathit{beta}) \land (\text{norm}(\mathit{beta}) \leq n) \land (\mathit{beta} \in \mathit{lst}) \land \text{ord-lessp}(\mathit{beta}, \mathit{alpha})) \rightarrow (\neg\, \text{ord-lessp}(\text{max-below}(\mathit{alpha}, \mathit{lst}, n), \mathit{beta}))$$

THEOREM: norm-of-max-below
$\text{norm}(\text{max-below}(\mathit{alpha}, \mathit{lst}, n)) \leq n$

EVENT: Enable ord-lessp.

THEOREM: max-below-of-0
$\text{max-below}(\texttt{0}, \mathit{lst}, n) = \texttt{0}$

EVENT: Disable ord-lessp.

EVENT: Disable max-below.

```
; we just need its properties

; now, we want to define {alpha}(n) = the largest
; ordinal < alpha with norm <= norm(alpha) + magic(n)
; The basic properties of magic have been derived in the
; beginning arithmetic section
```

DEFINITION:
$$\text{pred}(\mathit{alpha}, n) = \text{max-below}(\mathit{alpha}, \text{norm-up-to}(\text{norm}(\mathit{alpha}) + \text{magic}(n)), \text{norm}(\mathit{alpha}) + \text{magic}(n))$$

```
; let's prove the basic properties of this and then disable it
; that is -- (pred 0 n) = 0
; if alpha > 0, then (pred alpha n) < alpha and is the largest
; ordinal < alpha whose norm <= (norm alpha) + magic(n)
```

THEOREM: pred-of-0
$\text{pred}(\texttt{0}, n) = \texttt{0}$

THEOREM: pred-is-below
$\text{ord-lessp}(\texttt{0}, \mathit{alpha}) \rightarrow \text{ord-lessp}(\text{pred}(\mathit{alpha}, n), \mathit{alpha})$

Theorem: pred-is-an-ordinal
ordinalp (pred ($alpha$, n))

Theorem: upper-bound-on-norm-of-pred
norm (pred ($alpha$, n)) $\leq$ (norm ($alpha$) + magic (n))

Theorem: pred-is-largest
(ordinalp ($alpha$)
$\land$ ordinalp ($beta$)
$\land$ (norm ($beta$) $\leq$ (norm ($alpha$) + magic (n)))
$\land$ ord-lessp ($beta$, $alpha$))
$\rightarrow$ ord-leq ($beta$, pred ($alpha$, n))

Event: Disable pred.

```
; let's compute the norm of (pred alpha n) when alpha is a limit.
```

Event: Enable ord-lessp.

Theorem: pred-below-limit
(ordinalp ($alpha$) $\land$ limitp ($alpha$)) $\rightarrow$ ord-lessp (pred ($alpha$, n), $alpha$)

Event: Disable ord-lessp.

Theorem: norm-of-pred-of-limit-aux1
(ordinalp ($alpha$) $\land$ limitp ($alpha$))
$\rightarrow$ ord-lessp (successor (pred ($alpha$, n)), $alpha$)

Theorem: norm-of-pred-of-limit-aux2
(ordinalp ($alpha$) $\land$ limitp ($alpha$))
$\rightarrow$ ((norm ($alpha$) + magic (n)) $<$ norm (successor (pred ($alpha$, n))))

Theorem: norm-of-pred-of-limit-aux3
(ordinalp ($alpha$) $\land$ limitp ($alpha$))
$\rightarrow$ ((norm ($alpha$) + magic (n)) $<$ (1 + norm (pred ($alpha$, n))))

Theorem: norm-of-pred-of-limit
(ordinalp ($alpha$) $\land$ limitp ($alpha$))
$\rightarrow$ (norm (pred ($alpha$, n)) = (norm ($alpha$) + magic (n)))

Event: Disable norm-of-pred-of-limit-aux1.

Event: Disable norm-of-pred-of-limit-aux2.

EVENT: Disable norm-of-pred-of-limit-aux3.

```
; now, consider (pred alpha n) for successor alpha
```

THEOREM: pred-of-succ-aux1
$(\text{ordinalp}\,(alpha) \land \text{successorp}\,(alpha)) \rightarrow \text{ord-lessp}\,(\text{pred}\,(alpha,\ n),\ alpha)$

THEOREM: pred-of-succ-aux2
$(\text{ordinalp}\,(alpha) \land \text{successorp}\,(alpha))$
$\rightarrow \quad \text{ord-leq}\,(\text{pred}\,(alpha,\ n),\ \text{predecessor}\,(alpha))$

THEOREM: pred-of-succ-aux3
$(\text{ordinalp}\,(alpha) \land \text{successorp}\,(alpha))$
$\rightarrow \quad (\text{norm}\,(\text{predecessor}\,(alpha)) \leq (\text{norm}\,(alpha) + \text{magic}\,(n)))$

THEOREM: pred-of-succ-aux4
$(\text{ordinalp}\,(alpha) \land \text{successorp}\,(alpha))$
$\rightarrow \quad \text{ord-leq}\,(\text{predecessor}\,(alpha),\ \text{pred}\,(alpha,\ n))$

THEOREM: pred-of-succ
$(\text{ordinalp}\,(alpha) \land \text{successorp}\,(alpha))$
$\rightarrow \quad (\text{pred}\,(alpha,\ n) = \text{predecessor}\,(alpha))$

EVENT: Disable pred-of-succ-aux1.

EVENT: Disable pred-of-succ-aux2.

EVENT: Disable pred-of-succ-aux3.

EVENT: Disable pred-of-succ-aux4.

THEOREM: norm-of-pred-of-succ
$(\text{ordinalp}\,(alpha) \land \text{successorp}\,(alpha))$
$\rightarrow \quad (\text{norm}\,(\text{pred}\,(alpha,\ n)) = (\text{norm}\,(alpha) - 1))$

```
; now, by cases {alpha}(n) has norm at least ||alpha||-1 for all alpha
```

THEOREM: lower-bound-on-norm-of-pred
$\text{ordinalp}\,(alpha) \rightarrow (\text{norm}\,(\text{pred}\,(alpha,\ n)) \not< (\text{norm}\,(alpha) - 1))$

```
;;;;;;;;; pred of a sharp

; if beta > 0,
; we show:  alpha # {beta}(n) <= {alpha # beta}(n)
; the proof is
; 1.  alpha # {beta}(n) < alpha # beta by monotonicity of #
; 2.  norm(alpha # {beta}(n)) <= norm(alpha # beta) + magic(n)
; 3. Result follows by lemma "pred-is-largest"

; step 1
```

THEOREM: pred-sharp-below
(ordinalp ($alpha$) $\land$ ordinalp ($beta$) $\land$ ord-lessp (`0`, $beta$))
$\rightarrow$ ord-lessp (sharp ($alpha$, pred ($beta$, n)), sharp ($alpha$, $beta$))

```
; step 2
```

THEOREM: norm-of-pred-sharp
(ordinalp ($alpha$) $\land$ ordinalp ($beta$) $\land$ ($n \in \mathbf{N}$))
$\rightarrow$ (norm (sharp ($alpha$, pred ($beta$, n)))
$\leq$ (norm (sharp ($alpha$, $beta$)) + magic (n)))

```
; step 3
```

THEOREM: pred-sharp
(ordinalp ($alpha$) $\land$ ordinalp ($beta$) $\land$ ($n \in \mathbf{N}$) $\land$ ord-lessp (`0`, $beta$))
$\rightarrow$ ord-leq (sharp ($alpha$, pred ($beta$, n)), pred (sharp ($alpha$, $beta$), n))

```
;;;;;;;;  monotonicity of pred

; {alpha}(n) is a monotonic function of n

; first, a trivial fact about ord-leq
```

EVENT: Enable ord-leq.

THEOREM: ord-leq-is-idempotent
ord-leq (x, x)

EVENT: Disable ord-leq.

THEOREM: monot-of-pred-aux1
$(\text{ord-lessp}\,(\texttt{0},\ \textit{alpha})$
$\land\quad \text{ordinalp}\,(\textit{alpha})$
$\land\quad (m \in \mathbf{N})$
$\land\quad (n \in \mathbf{N})$
$\land\quad (m \leq n))$
$\rightarrow\quad \text{ord-leq}\,(\text{pred}\,(\textit{alpha},\ m),\ \text{pred}\,(\textit{alpha},\ n))$

THEOREM: monot-of-pred-aux2
$((\neg\ \text{ord-lessp}\,(\texttt{0},\ \textit{alpha}))$
$\land\quad \text{ordinalp}\,(\textit{alpha})$
$\land\quad (m \in \mathbf{N})$
$\land\quad (n \in \mathbf{N})$
$\land\quad (m \leq n))$
$\rightarrow\quad \text{ord-leq}\,(\text{pred}\,(\textit{alpha},\ m),\ \text{pred}\,(\textit{alpha},\ n))$

THEOREM: monot-of-pred
$(\text{ordinalp}\,(\textit{alpha}) \land (m \in \mathbf{N}) \land (n \in \mathbf{N}) \land (m \leq n))$
$\rightarrow\quad \text{ord-leq}\,(\text{pred}\,(\textit{alpha},\ m),\ \text{pred}\,(\textit{alpha},\ n))$

EVENT: Disable monot-of-pred-aux1.

EVENT: Disable monot-of-pred-aux2.

```
; the norm of pred is also monotonic
; we can prove this by cases -- successor, limit, 0
```

THEOREM: monot-of-norm-of-pred-aux1
$((\textit{alpha} = \texttt{0}) \land \text{ordinalp}\,(\textit{alpha}) \land (m \in \mathbf{N}) \land (n \in \mathbf{N}) \land (m \leq n))$
$\rightarrow\quad (\text{norm}\,(\text{pred}\,(\textit{alpha},\ n)) \not< \text{norm}\,(\text{pred}\,(\textit{alpha},\ m)))$

THEOREM: monot-of-norm-of-pred-aux2
$(\text{limitp}\,(\textit{alpha}) \land \text{ordinalp}\,(\textit{alpha}) \land (m \in \mathbf{N}) \land (n \in \mathbf{N}) \land (m \leq n))$
$\rightarrow\quad (\text{norm}\,(\text{pred}\,(\textit{alpha},\ n)) \not< \text{norm}\,(\text{pred}\,(\textit{alpha},\ m)))$

THEOREM: monot-of-norm-of-pred-aux3
$(\text{successorp}\,(\textit{alpha}) \land \text{ordinalp}\,(\textit{alpha}) \land (m \in \mathbf{N}) \land (n \in \mathbf{N}) \land (m \leq n))$
$\rightarrow\quad (\text{norm}\,(\text{pred}\,(\textit{alpha},\ n)) \not< \text{norm}\,(\text{pred}\,(\textit{alpha},\ m)))$

THEOREM: monot-of-norm-of-pred
$(\text{ordinalp}\,(\textit{alpha}) \land (m \in \mathbf{N}) \land (n \in \mathbf{N}) \land (m \leq n))$
$\rightarrow\quad (\text{norm}\,(\text{pred}\,(\textit{alpha},\ n)) \not< \text{norm}\,(\text{pred}\,(\textit{alpha},\ m)))$

EVENT: Disable monot-of-norm-of-pred-aux1.

EVENT: Disable monot-of-norm-of-pred-aux2.

EVENT: Disable monot-of-norm-of-pred-aux3.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                           O-LARGE SETS                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; We define the notion :  X is alpha-large
; It is intended that X is a set in standard form, so the car is the min
; Largep was the Paris-Harrington defn of large;
; o-largep is ordinal version
```

DEFINITION:
o-largep (set, $alpha$)
$=$ **if** $alpha = 0$ **then t**
elseif $set \simeq$ **nil then f**
else o-largep (cdr (set), pred ($alpha$, car (set))) **endif**

```
; prove the basic properties and then disable the defn

; base cases
```

THEOREM: all-zero-large
o-largep (set, 0)

THEOREM: empty-not-large
(($alpha \neq 0$) $\wedge$ ($set \simeq$ **nil**)) $\rightarrow$ ($\neg$ o-largep (set, $alpha$))

THEOREM: recursive-case-for-large
(listp (set) $\wedge$ ($alpha \neq 0$))
$\rightarrow$ (o-largep (set, $alpha$) $=$ o-largep (cdr (set), pred ($alpha$, car (set))))

THEOREM: positive-large
(ordinalp ($alpha$) $\wedge$ ($\neg$ o-largep (set, $alpha$))) $\rightarrow$ ord-lessp (0, $alpha$)

EVENT: Disable o-largep.

```
; lookat alpha-large when alpha is a successor ordinal
```

THEOREM: successor-large-non-empty
$(\text{ordinalp}\,(\mathit{alpha}) \land \text{successorp}\,(\mathit{alpha}) \land (\neg\ \text{listp}\,(\mathit{set})))$
$\rightarrow \quad (\neg\ \text{o-largep}\,(\mathit{set},\ \mathit{alpha}))$

THEOREM: successor-large
$(\text{ordinalp}\,(\mathit{alpha}) \land \text{successorp}\,(\mathit{alpha}))$
$\rightarrow \quad (\text{o-largep}\,(\mathit{set},\ \mathit{alpha})$
$\quad\quad = \quad (\text{listp}\,(\mathit{set}) \land \text{o-largep}\,(\text{cdr}\,(\mathit{set}),\ \text{predecessor}\,(\mathit{alpha}))))$

```
; lookat n-large for numbers n
```

THEOREM: number-large
$(n \in \mathbf{N}) \rightarrow (\text{o-largep}\,(\mathit{set},\ n) = (n \leq \text{length}\,(\mathit{set})))$

```
;;;;;;;;;;;;  omega
; more on omega = '(1 . 0)
; we want to discuss  omega-large -- but first
; we need to discuss {omega}(n) = magic(n) + 2
```

THEOREM: omega-is-a-limit
limitp (`'(1 . 0)`)

EVENT: Enable ord-lessp.

THEOREM: numbers-below-omega
(ordinalp (alpha) $\land$ ord-lessp (alpha, `'(1 . 0)`)) $\rightarrow$ ($\mathit{alpha} \in \mathbf{N}$)

EVENT: Disable ord-lessp.

THEOREM: pred-of-omega-aux1
pred (`'(1 . 0)`, n) $\in \mathbf{N}$

THEOREM: pred-of-omega-aux2
($n \in \mathbf{N}$) $\rightarrow$ (norm (pred (`'(1 . 0)`, n)) = (magic (n) + 2))

THEOREM: pred-of-omega
($n \in \mathbf{N}$) $\rightarrow$ (pred (`'(1 . 0)`, n) = (magic (n) + 2))

EVENT: Disable pred-of-omega-aux1.

EVENT: Disable pred-of-omega-aux2.

```
; now, consider omega-large -- here
; we use (setp set) -- so we know that (car set) is a number
```

EVENT: Enable setp.

THEOREM: car-of-a-set
$\text{setp}(\text{cons}(x,\ \mathit{tail})) \rightarrow (x \in \mathbf{N})$

EVENT: Disable setp.

THEOREM: omega-large
$(\text{setp}(\mathit{set}) \land \text{o-largep}(\mathit{set},\ \texttt{'(1 . 0)}))$
$\rightarrow \quad (\text{listp}(\mathit{set}) \land ((\text{magic}(\text{car}(\mathit{set})) + \texttt{3}) \leq \text{length}(\mathit{set})))$

```
; in particular, omega-large implies large in the Paris-Harrington sense:
```

THEOREM: omega-large-implies-large
$(\text{setp}(\mathit{set}) \land \text{o-largep}(\mathit{set},\ \texttt{'(1 . 0)})) \rightarrow \text{largep}(\mathit{set})$

```
;;;;;;;;; let's see what it means for a singleton to be alpha-large
; this can only happen for alpha = 0 or 1
; First, note that {alpha}(n) = 0 implies alpha is 0 or 1
```

THEOREM: norm-above-1-aux1
$((\mathit{alpha} \in \mathbf{N}) \land \text{ord-lessp}(\texttt{1},\ \mathit{alpha}) \land \text{ordinalp}(\mathit{alpha}))$
$\rightarrow \quad (\texttt{1} < \text{norm}(\mathit{alpha}))$

EVENT: Enable norm.

EVENT: Enable ord-lessp.

THEOREM: norm-above-1-aux2
$((\mathit{alpha} \notin \mathbf{N}) \land \text{ord-lessp}(\texttt{1},\ \mathit{alpha}) \land \text{ordinalp}(\mathit{alpha}))$
$\rightarrow \quad (\texttt{1} < \text{norm}(\mathit{alpha}))$

EVENT: Disable norm.

EVENT: Disable ord-lessp.

THEOREM: norm-above-1
$(\text{ord-lessp}(\texttt{1},\ \mathit{alpha}) \land \text{ordinalp}(\mathit{alpha})) \rightarrow (\texttt{1} < \text{norm}(\mathit{alpha}))$

EVENT: Disable norm-above-1-aux1.

EVENT: Disable norm-above-1-aux2.

THEOREM: pred-not-0
(ord-lessp (1, *alpha*) ∧ ordinalp (*alpha*)) → ord-leq (1, pred (*alpha*, *n*))

THEOREM: singletons-not-large
(o-largep (*set*, *alpha*) ∧ ordinalp (*alpha*) ∧ ord-lessp (1, *alpha*))
→ listp (cdr (*set*))

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                       O-LARGE AND SUBSETS                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; The main fact here is:  Let x = min(X) = car(X).
; If X is beta-large, alpha <= beta, X is a subset of Y
; and norm(alpha) <= norm(beta) + magic(x),
; then Y is alpha-large.
; This is true even if X is empty, since then beta = alpha = 0
; (and x = car(nil) = 0)
```

DEFINITION:
bad-for-large-superset-ord (*alpha*, *beta*, *x*, *y*)
= (ordinalp (*alpha*)
  ∧ ordinalp (*beta*)
  ∧ setp (*x*)
  ∧ setp (*y*)
  ∧ subsetp (*x*, *y*)
  ∧ o-largep (*x*, *beta*)
  ∧ ord-leq (*alpha*, *beta*)
  ∧ (norm (*alpha*) $\leq$ (norm (*beta*) + magic (car (*x*))))
  ∧ (¬ o-largep (*y*, *alpha*)))

```
; we want to show this can't happen.
; first, some base cases
```

THEOREM: large-superset-ord-aux1
bad-for-large-superset-ord (*alpha*, *beta*, *x*, *y*) → ord-lessp (0, *alpha*)

THEOREM: large-superset-ord-aux2
(ordinalp (*alpha*)

$\wedge$ ordinalp (*beta*)
$\wedge$ ord-lessp (0, *alpha*)
$\wedge$ ord-leq (*alpha*, *beta*))
$\rightarrow$ ord-lessp (0, *beta*)

THEOREM: large-superset-ord-aux3
bad-for-large-superset-ord (*alpha*, *beta*, *x*, *y*) $\rightarrow$ ord-lessp (0, *beta*)

```
; so, alpha and beta are larger than 0
; next, check that can't have alpha = beta = 1
```

THEOREM: large-superset-ord-aux4
$\neg$ bad-for-large-superset-ord (1, 1, *x*, *y*)

```
; now,  show X can't be a singleton, and alpha > 1
```

THEOREM: large-superset-ord-aux5
bad-for-large-superset-ord (*alpha*, *beta*, *x*, *y*)
$\rightarrow$ (ordinalp (*alpha*) $\wedge$ ordinalp (*beta*))

THEOREM: large-superset-ord-aux6
bad-for-large-superset-ord (*alpha*, *beta*, *x*, *y*)
$\rightarrow$ (ord-lessp (1, *alpha*) $\vee$ ord-lessp (1, *beta*))

THEOREM: large-superset-ord-aux7
bad-for-large-superset-ord (*alpha*, *beta*, *x*, *y*) $\rightarrow$ ord-leq (*alpha*, *beta*)

THEOREM: large-superset-ord-aux8
bad-for-large-superset-ord (*alpha*, *beta*, *x*, *y*) $\rightarrow$ ord-lessp (1, *beta*)

THEOREM: large-superset-ord-aux9
bad-for-large-superset-ord (*alpha*, *beta*, *x*, *y*) $\rightarrow$ listp (cdr (*x*))

```
; now, we want to show that
; (bad-for-large-superset-ord alpha beta X Y) is always false
; we know it implies alpha > 0, beta > 1, and X, Y are lists (i.e., not empty)
; the induction splits into two cases:

; Case 1:  alpha = beta
; we show (bad-for-large-superset-ord
;        (pred alpha (car Y)) (pred alpha (car X)) (cdr X)  (cdr Y) )
; Case 2: alpha < beta
; we show (bad-for-large-superset-ord
;        alpha (pred beta (car X)) (cdr X)   Y )
```

```
; BEGIN Case 1  alpha = beta
; we check the various clauses in bad for
;        (pred alpha (car Y))  (pred alpha (car X)) (cdr X)  (cdr Y)
; as much as possible, we keep alpha, beta separate
; so the work can be used in Case 2
```

THEOREM: large-superset-ord-aux10
bad-for-large-superset-ord $(alpha, beta, x, y)$
$\rightarrow$ (ordinalp (pred $(alpha$, car $(y)))$ $\wedge$ ordinalp (pred $(beta$, car $(x))))$

THEOREM: large-superset-ord-aux11
bad-for-large-superset-ord $(alpha, beta, x, y)$ $\rightarrow$ (listp (x) $\wedge$ listp $(y))$

THEOREM: large-superset-ord-aux12
bad-for-large-superset-ord $(alpha, beta, x, y)$
$\rightarrow$ (setp (cdr (x)) $\wedge$ setp (cdr (y)))

THEOREM: large-superset-ord-aux13
bad-for-large-superset-ord $(alpha, beta, x, y)$ $\rightarrow$ subsetp (cdr (x), cdr (y))

THEOREM: large-superset-ord-aux14
bad-for-large-superset-ord $(alpha, beta, x, y)$
$\rightarrow$ o-largep (cdr (x), pred $(beta$, car $(x)))$

THEOREM: large-superset-ord-aux15
bad-for-large-superset-ord $(alpha, alpha, x, y)$
$\rightarrow$ ord-leq (pred $(alpha$, car $(y))$, pred $(alpha$, car $(x)))$

THEOREM: large-superset-ord-aux16
bad-for-large-superset-ord $(alpha, alpha, x, y)$
$\rightarrow$ (norm (pred $(alpha$, car $(y))$)
$\quad\leq$ (norm (pred $(alpha$, car $(x))$) $+$ magic (car (cdr (x)))))

THEOREM: large-superset-ord-aux17
bad-for-large-superset-ord $(alpha, alpha, x, y)$
$\rightarrow$ ($\neg$ o-largep (cdr (y), pred $(alpha$, car $(y))))$

THEOREM: large-superset-ord-aux18
bad-for-large-superset-ord $(alpha, alpha, x, y)$
$\rightarrow$ bad-for-large-superset-ord (pred $(alpha$, car $(y))$,
pred $(alpha$, car $(x))$,
cdr (x),
cdr (y))

```
; END Case 1

; BEGIN  Case 2: alpha < beta
; we check the various requrements for
;  (bad-for-large-superset-ord alpha (pred beta (car X)) (cdr X)   Y)
; alpha and (pred beta (car X)) are ordinals by  the defn of bad and aux10
; (cdr X) and Y are sets by the defn of bad and aux12
; (o-largep (cdr X) (pred beta (car X)))  by aux14
; (not (o-largep Y alpha))  by defn of bad
; next
```

THEOREM: large-superset-ord-aux19
(bad-for-large-superset-ord ($alpha$, $beta$, x, y) $\land$ ord-lessp ($alpha$, $beta$))
$\rightarrow$ ord-leq ($alpha$, pred ($beta$, car (x)))

```
; finally, we have to use (car X) < (cadr X) with the
; following simple ordinal fact:
```

THEOREM: large-superset-ord-aux20
(ordinalp ($beta$) $\land$ ($v \in \mathbf{N}$) $\land$ ($z \in \mathbf{N}$) $\land$ ($v < z$))
$\rightarrow$ ((norm ($beta$) + magic (v)) $\leq$ (norm (pred ($beta$, z)) + magic (z)))

```
; we use this with v = (car X) and z = (cadr X)
```

THEOREM: large-superset-ord-aux21
bad-for-large-superset-ord ($alpha$, $beta$, x, y) $\rightarrow$ (car (x) $<$ cadr (x))

THEOREM: large-superset-ord-aux22
(bad-for-large-superset-ord ($alpha$, $beta$, x, y) $\land$ ord-lessp ($alpha$, $beta$))
$\rightarrow$ (norm ($alpha$) $\leq$ (norm (pred ($beta$, car (x))) + magic (car (cdr (x)))))

THEOREM: large-superset-ord-aux23
bad-for-large-superset-ord ($alpha$, $beta$, x, y) $\rightarrow$ subsetp (cdr (x), y)

THEOREM: large-superset-ord-aux24
(bad-for-large-superset-ord ($alpha$, $beta$, x, y) $\land$ ord-lessp ($alpha$, $beta$))
$\rightarrow$ bad-for-large-superset-ord ($alpha$, pred ($beta$, car (x)), cdr (x), y)

```
; END Case 2

; now, we should show that case 1 or case 2 holds
```

THEOREM: large-superset-ord-aux25
bad-for-large-superset-ord $(alpha, beta, x, y)$
$\rightarrow$ (ord-lessp $(alpha, beta) \lor (alpha = beta))$

EVENT: Disable large-superset-ord-aux1.

EVENT: Disable large-superset-ord-aux2.

EVENT: Disable large-superset-ord-aux3.

EVENT: Disable large-superset-ord-aux4.

EVENT: Disable large-superset-ord-aux5.

EVENT: Disable large-superset-ord-aux6.

EVENT: Disable large-superset-ord-aux7.

EVENT: Disable large-superset-ord-aux8.

EVENT: Disable large-superset-ord-aux9.

EVENT: Disable large-superset-ord-aux10.

EVENT: Disable large-superset-ord-aux11.

EVENT: Disable large-superset-ord-aux12.

EVENT: Disable large-superset-ord-aux13.

EVENT: Disable large-superset-ord-aux14.

EVENT: Disable large-superset-ord-aux15.

EVENT: Disable large-superset-ord-aux16.

EVENT: Disable large-superset-ord-aux17.

EVENT: Disable large-superset-ord-aux18.

EVENT: Disable large-superset-ord-aux19.

EVENT: Disable large-superset-ord-aux21.

EVENT: Disable large-superset-ord-aux22.

EVENT: Disable large-superset-ord-aux23.

EVENT: Disable large-superset-ord-aux24.

EVENT: Disable large-superset-ord-aux25.

```
; summarizing
```

THEOREM: large-superset-ord-aux26
bad-for-large-superset-ord ($alpha$, $beta$, x, y)
$\rightarrow$ (bad-for-large-superset-ord ($alpha$, pred ($beta$, car (x)), cdr (x), y)
$\vee$ bad-for-large-superset-ord (pred ($alpha$, car (y)),
pred ($alpha$, car (x)),
cdr (x),
cdr (y)))

THEOREM: large-superset-ord-aux27
bad-for-large-superset-ord ($alpha$, $beta$, x, y) $\rightarrow$ listp (x)

DEFINITION:
large-superset-ord-kludge ($alpha$, $beta$, x, y)
= **if** $x \simeq$ **nil** **then** 0
else large-superset-ord-kludge ($alpha$, pred ($beta$, car (x)), cdr (x), y)
+ large-superset-ord-kludge (pred ($alpha$, car (y)),
pred ($alpha$, car (x)),
cdr (x),
cdr (y)) **endif**

THEOREM: large-superset-ord-aux28
(($\neg$ bad-for-large-superset-ord ($alpha$, pred ($beta$, car (x)), cdr (x), y))

$$\begin{array}{ll}\land & (\neg \text{ bad-for-large-superset-ord}\,(\text{pred}\,(alpha,\ \text{car}\,(y)), \\ & \qquad\qquad\qquad\qquad\qquad\qquad \text{pred}\,(alpha,\ \text{car}\,(x)), \\ & \qquad\qquad\qquad\qquad\qquad\qquad \text{cdr}\,(x), \\ & \qquad\qquad\qquad\qquad\qquad\qquad \text{cdr}\,(y)))) \\ \to & (\neg \text{ bad-for-large-superset-ord}\,(alpha,\ beta,\ x,\ y))\end{array}$$

THEOREM: large-superset-ord-aux29
$\neg$ bad-for-large-superset-ord ($alpha$, $beta$, x, y)

THEOREM: large-superset-ord

$$\begin{array}{ll}\multicolumn{2}{l}{(\text{ordinalp}\,(alpha)} \\ \land & \text{ordinalp}\,(beta) \\ \land & \text{setp}\,(x) \\ \land & \text{setp}\,(y) \\ \land & \text{subsetp}\,(x,\ y) \\ \land & \text{o-largep}\,(x,\ beta) \\ \land & \text{ord-leq}\,(alpha,\ beta) \\ \land & (\text{norm}\,(alpha) \leq (\text{norm}\,(beta) + \text{magic}\,(\text{car}\,(x))))) \\ \to & \text{o-largep}\,(y,\ alpha)\end{array}$$

EVENT: Disable large-superset-ord-aux26.

EVENT: Disable large-superset-ord-aux27.

EVENT: Disable large-superset-ord-aux28.

EVENT: Disable large-superset-ord-aux29.

EVENT: Disable large-superset-ord-kludge.

```
;;;;;;;; two important special cases

; Special case 1 -- alpha = beta
```

THEOREM: large-goes-up

$$\begin{array}{ll}\multicolumn{2}{l}{(\text{ordinalp}\,(alpha)} \\ \land & \text{setp}\,(x) \\ \land & \text{setp}\,(y) \\ \land & \text{subsetp}\,(x,\ y) \\ \land & \text{o-largep}\,(x,\ alpha)) \\ \to & \text{o-largep}\,(y,\ alpha)\end{array}$$

```
; Special case 2 -- X = Y
```

THEOREM: subsetp-is-idempotent
subsetp$(s,\ s)$

THEOREM: large-with-smaller-ord
$$
\begin{array}{ll}
 & (\text{ordinalp}\,(alpha) \\
\land & \text{ordinalp}\,(beta) \\
\land & \text{setp}\,(x) \\
\land & \text{o-largep}\,(x,\ beta) \\
\land & \text{ord-leq}\,(alpha,\ beta) \\
\land & (\text{norm}\,(alpha) \leq (\text{norm}\,(beta) + \text{magic}\,(\text{car}\,(x))))) \\
\rightarrow & \text{o-largep}\,(x,\ alpha)
\end{array}
$$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                               PIGEON-2                                ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; this is the two-set pigeon-hole principle:
; if X is a subset of A union B, and X is alpha # beta-large
; then A is alpha-large or B is beta-large

;;;;;; covers
; Rather than define union, we just define what it means
; for A union B to cover X
```

DEFINITION:
covers$(a,\ b,\ x)$
$$
\begin{array}{ll}
= & \textbf{if } x \simeq \textbf{nil} \textbf{ then t} \\
 & \textbf{else } \text{covers}\,(a,\ b,\ \text{cdr}\,(x)) \\
 & \quad\quad \land \quad ((\text{car}\,(x) \in a) \lor (\text{car}\,(x) \in b)) \textbf{ endif}
\end{array}
$$

```
; some basic lemmas expressing the recursion
```

THEOREM: covers-empty
$(x \simeq \textbf{nil}) \rightarrow \text{covers}\,(a,\ b,\ x)$

THEOREM: covers-negative
$(\text{listp}\,(x) \land (\text{car}\,(x) \notin a) \land (\text{car}\,(x) \notin b)) \rightarrow (\neg\ \text{covers}\,(a,\ b,\ x))$

THEOREM: covers-positive-a
$(\text{listp}\,(x) \land \text{covers}\,(a,\ b,\ \text{cdr}\,(x)) \land (\text{car}\,(x) \in a)) \rightarrow \text{covers}\,(a,\ b,\ x)$

THEOREM: covers-positive-b
$(\text{listp}\,(x) \land \text{covers}\,(a,\ b,\ \text{cdr}\,(x)) \land (\text{car}\,(x) \in b)) \rightarrow \text{covers}\,(a,\ b,\ x)$

THEOREM: member-of-covered-set
$((u \in x) \land \text{covers}(a,\ b,\ x)) \rightarrow ((u \in a) \lor (u \in b))$

```
; if A,B don't cover X, we can compute an element of X which is not covered
```

DEFINITION:
uncovered $(a,\ b,\ x)$
$=$ **if** $x \simeq$ **nil then nil**
elseif covers $(a,\ b,\ \text{cdr}(x))$ **then** car (x)
else uncovered $(a,\ b,\ \text{cdr}(x))$ **endif**

THEOREM: uncovered-isnt-covered
$(\neg\ \text{covers}(a,\ b,\ x))$
$\rightarrow\ ((\text{uncovered}(a,\ b,\ x) \in x)$
$\land\ (\text{uncovered}(a,\ b,\ x) \notin a)$
$\land\ (\text{uncovered}(a,\ b,\ x) \notin b))$

THEOREM: cdr-is-covered
$\text{covers}(a,\ b,\ x) \rightarrow \text{covers}(a,\ b,\ \text{cdr}(x))$

```
; we probably don't need the defn of covers any more
```

EVENT: Disable covers.

THEOREM: covers-is-symmetric
$\text{covers}(a,\ b,\ x) \rightarrow \text{covers}(b,\ a,\ x)$

THEOREM: discard-the-car-a
$(\text{covers}(a,\ b,\ x) \land \text{listp}(a) \land (\text{car}(a) \notin x)) \rightarrow \text{covers}(\text{cdr}(a),\ b,\ x)$

THEOREM: discard-the-car-b
$(\text{covers}(a,\ b,\ x) \land \text{listp}(b) \land (\text{car}(b) \notin x)) \rightarrow \text{covers}(a,\ \text{cdr}(b),\ x)$

```
;;;;  for sets
; special facts, for standard sets, using min-is-first
```

THEOREM: discard-the-car-set-a
$(\text{setp}(a) \land \text{listp}(a) \land \text{setp}(x) \land \text{covers}(a,\ b,\ x) \land (\text{car}(a) < \text{car}(x)))$
$\rightarrow\ \text{covers}(\text{cdr}(a),\ b,\ x)$

THEOREM: discard-the-car-set-b
$(\text{setp}(b) \land \text{listp}(b) \land \text{setp}(x) \land \text{covers}(a,\ b,\ x) \land (\text{car}(b) < \text{car}(x)))$
$\rightarrow\ \text{covers}(a,\ \text{cdr}(b),\ x)$

```
; now if X is non-empty and covered by A union B --
; there are two cases
; 1.  A is non-empty and its min is <= min X
; 2.  B is non-empty and its min is <= min X
; in case 1, (cdr X) is covered by (cdr A) union B
; likewise in case 2

; the two cases
```

THEOREM: smaller-car-when-covered

$$(\text{setp}(a) \land \text{setp}(b) \land \text{setp}(x) \land \text{listp}(x) \land \text{covers}(a,\ b,\ x))$$
$$\rightarrow \quad ((\text{listp}(a) \land (\text{car}(a) \leq \text{car}(x))) \quad \lor \quad (\text{listp}(b) \land (\text{car}(b) \leq \text{car}(x))))$$

```
; now, consider case 1
; as auxiliaries, we consider sub-cases depending on whether
; (cdr X) is empty or not

; first, if (cdr X) is non-empty
```

THEOREM: cdr-is-covered-set-a-aux1

$$\begin{aligned}
&(\text{listp}(\text{cdr}(x)) \\
&\ \land \quad \text{setp}(a) \\
&\ \land \quad \text{setp}(b) \\
&\ \land \quad \text{setp}(x) \\
&\ \land \quad \text{listp}(x) \\
&\ \land \quad \text{covers}(a,\ b,\ x) \\
&\ \land \quad \text{listp}(a) \\
&\ \land \quad (\text{car}(a) \leq \text{car}(x))) \\
&\rightarrow \quad (\text{car}(a) < \text{cadr}(x))
\end{aligned}$$

THEOREM: cdr-is-covered-set-a-aux2

$$\begin{aligned}
&(\text{listp}(\text{cdr}(x)) \\
&\ \land \quad \text{setp}(a) \\
&\ \land \quad \text{setp}(b) \\
&\ \land \quad \text{setp}(x) \\
&\ \land \quad \text{listp}(x) \\
&\ \land \quad \text{covers}(a,\ b,\ x) \\
&\ \land \quad \text{listp}(a) \\
&\ \land \quad (\text{car}(a) \leq \text{car}(x))) \\
&\rightarrow \quad \text{covers}(\text{cdr}(a),\ b,\ \text{cdr}(x))
\end{aligned}$$

```
; now, if cdr X is empty, it's trivial, so
```

THEOREM: cdr-is-covered-set-a

$$
\begin{array}{ll}
 & (\mathrm{setp}\,(a) \\
\wedge & \mathrm{setp}\,(b) \\
\wedge & \mathrm{setp}\,(x) \\
\wedge & \mathrm{listp}\,(x) \\
\wedge & \mathrm{covers}\,(a,\ b,\ x) \\
\wedge & \mathrm{listp}\,(a) \\
\wedge & (\mathrm{car}\,(a) \leq \mathrm{car}\,(x))) \\
\rightarrow & \mathrm{covers}\,(\mathrm{cdr}\,(a),\ b,\ \mathrm{cdr}\,(x))
\end{array}
$$

EVENT: Disable cdr-is-covered-set-a-aux1.

EVENT: Disable cdr-is-covered-set-a-aux2.

```
; now, by symmetry:
```

THEOREM: cdr-is-covered-set-b

$$
\begin{array}{ll}
 & (\mathrm{setp}\,(a) \\
\wedge & \mathrm{setp}\,(b) \\
\wedge & \mathrm{setp}\,(x) \\
\wedge & \mathrm{listp}\,(x) \\
\wedge & \mathrm{covers}\,(a,\ b,\ x) \\
\wedge & \mathrm{listp}\,(b) \\
\wedge & (\mathrm{car}\,(b) \leq \mathrm{car}\,(x))) \\
\rightarrow & \mathrm{covers}\,(a,\ \mathrm{cdr}\,(b),\ \mathrm{cdr}\,(x))
\end{array}
$$

```
;;;;;;;;;; sharp-and-cdr
; before going to pigeon-hole, we need a lemma that
; if X is alpha # beta - large, beta > 0, and m <= (car X) then
; (cdr X) is alpha # {beta}(m) -- large
; Also, by symmetry, the same holds for alpha/beta reversed
; X must be non-empty, since (cdr nil) = 0 -- not a set

; first observe that (cdr X) is {alpha # beta}(x) -- large
```

EVENT: Enable o-largep.

THEOREM: sharp-and-cdr-aux1

$$
\begin{array}{ll}
 & (\mathrm{ordinalp}\,(alpha) \\
\wedge & \mathrm{ordinalp}\,(beta) \\
\wedge & \mathrm{listp}\,(x)
\end{array}
$$

$\wedge$ o-largep $(x$, sharp $(alpha,\ beta)))$
$\rightarrow$ o-largep (cdr (x), pred (sharp $(alpha,\ beta)$, car $(x)))$

```
; now note that if (cdr X) is empty, {alpha # beta}(x) = 0
; this should be the "trivial" case
```

THEOREM: sharp-and-cdr-aux2
((cdr $(x) \simeq$ **nil**)
$\wedge$ ordinalp $(alpha)$
$\wedge$ ordinalp $(beta)$
$\wedge$ listp (x)
$\wedge$ o-largep $(x$, sharp $(alpha,\ beta)))$
$\rightarrow$ (pred (sharp $(alpha,\ beta)$, car $(x)) = \mathtt{0})$

EVENT: Disable o-largep.

```
; now, if m <= (car X) then alpha # {beta}(m) <= {alpha # beta}(car X)
; proof:  alpha # {beta}(m) <= {alpha # beta}(m) <= {alpha # beta}(car X)
; the first <= is by pred-sharp, the second by monoton of pred
```

THEOREM: sharp-and-cdr-aux3
(ordinalp $(alpha)$
$\wedge$ ordinalp $(beta)$
$\wedge$ ord-lessp $(\mathtt{0},\ beta)$
$\wedge$ $(m \in \mathbf{N})$
$\wedge$ $(n \in \mathbf{N})$
$\wedge$ $(m \leq n))$
$\rightarrow$ ord-leq (sharp $(alpha$, pred $(beta,\ m))$, pred (sharp $(alpha,\ beta),\ n))$

THEOREM: sharp-and-cdr-aux4
(ordinalp $(alpha)$
$\wedge$ ordinalp $(beta)$
$\wedge$ ord-lessp $(\mathtt{0},\ beta)$
$\wedge$ setp (x)
$\wedge$ $(m \in \mathbf{N})$
$\wedge$ $(m \leq$ car $(x)))$
$\rightarrow$ ord-leq (sharp $(alpha$, pred $(beta,\ m))$, pred (sharp $(alpha,\ beta)$, car $(x)))$

```
; now, if (cdr X) is empty, alpha # {beta}(m) = 0
```

THEOREM: sharp-and-cdr-aux5
(listp (x)

$\wedge \quad (\neg\ \text{listp}\,(\text{cdr}\,(x)))$
$\wedge \quad \text{o-largep}\,(x,\ \text{sharp}\,(\mathit{alpha},\ \mathit{beta}))$
$\wedge \quad \text{ordinalp}\,(\mathit{alpha})$
$\wedge \quad \text{ordinalp}\,(\mathit{beta})$
$\wedge \quad \text{ord-lessp}\,(\mathtt{0},\ \mathit{beta})$
$\wedge \quad \text{setp}\,(x)$
$\wedge \quad (m \in \mathbf{N})$
$\wedge \quad (m \leq \text{car}\,(x)))$
$\rightarrow \quad (\text{sharp}\,(\mathit{alpha},\ \text{pred}\,(\mathit{beta},\ m)) = \mathtt{0})$

```
; in particular, our result is trivial if (cdr X) is empty
```

THEOREM: sharp-and-cdr-aux6
$((\neg\ \text{listp}\,(\text{cdr}\,(x)))$
$\wedge \quad \text{ordinalp}\,(\mathit{alpha})$
$\wedge \quad \text{ordinalp}\,(\mathit{beta})$
$\wedge \quad \text{ord-lessp}\,(\mathtt{0},\ \mathit{beta})$
$\wedge \quad \text{setp}\,(x)$
$\wedge \quad \text{listp}\,(x)$
$\wedge \quad (m \in \mathbf{N})$
$\wedge \quad (m \leq \text{car}\,(x))$
$\wedge \quad \text{o-largep}\,(x,\ \text{sharp}\,(\mathit{alpha},\ \mathit{beta})))$
$\rightarrow \quad \text{o-largep}\,(\text{cdr}\,(x),\ \text{sharp}\,(\mathit{alpha},\ \text{pred}\,(\mathit{beta},\ m)))$

```
; now, consider the case where (cdr X) is non-empty
```

THEOREM: sharp-and-cdr-aux7
$(\text{setp}\,(x) \wedge \text{listp}\,(\text{cdr}\,(x)) \wedge (m \in \mathbf{N}) \wedge (m \leq \text{car}\,(x)))$
$\rightarrow \quad (m < \text{cadr}\,(x))$

```
; we have, from the above:
; aux1 : (o-largep (cdr X) (pred (sharp alpha beta) (car X))  )
; aux4
; (ord-leq (sharp alpha (pred beta m)) (pred (sharp alpha beta) (car X)) )
; aux7 :  m < (cadr X)
; we want to apply large-with-smaller-ordinal to (cdr X)
; so we need that
;  (norm (sharp alpha (pred beta m))) <=
;    (plus (norm (pred (sharp alpha beta) (car X))) (magic (cadr X)))
; prove this separately, as a lemma:
```

THEOREM: sharp-and-cdr-aux8

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{ordinalp}\,(\mathit{alpha})} \\
\land & \text{ordinalp}\,(\mathit{beta}) \\
\land & \text{ord-lessp}\,(\texttt{0},\ \mathit{beta}) \\
\land & (m \in \mathbf{N}) \\
\land & (\mathit{x0} \in \mathbf{N}) \\
\land & (m < \mathit{x1})) \\
\rightarrow & (\text{norm}\,(\text{sharp}\,(\mathit{alpha},\ \text{pred}\,(\mathit{beta},\ m))) \\
 & \quad \leq \quad (\text{norm}\,(\text{pred}\,(\text{sharp}\,(\mathit{alpha},\ \mathit{beta}),\ \mathit{x0})) + \text{magic}\,(\mathit{x1})))
\end{array}
$$

```
; intended :  x0 is (car X) ; x1 is (cadr x)
```

THEOREM: sharp-and-cdr-aux9

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(\text{cdr}\,(x))} \\
\land & \text{ordinalp}\,(\mathit{alpha}) \\
\land & \text{ordinalp}\,(\mathit{beta}) \\
\land & \text{ord-lessp}\,(\texttt{0},\ \mathit{beta}) \\
\land & \text{setp}\,(x) \\
\land & \text{listp}\,(x) \\
\land & (m \in \mathbf{N}) \\
\land & (m \leq \text{car}\,(x))) \\
\rightarrow & (\text{norm}\,(\text{sharp}\,(\mathit{alpha},\ \text{pred}\,(\mathit{beta},\ m))) \\
 & \quad \leq \quad (\text{norm}\,(\text{pred}\,(\text{sharp}\,(\mathit{alpha},\ \mathit{beta}),\ \text{car}\,(x))) + \text{magic}\,(\text{cadr}\,(x))))
\end{array}
$$

THEOREM: sharp-and-cdr-aux10

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(\text{cdr}\,(x))} \\
\land & \text{ordinalp}\,(\mathit{alpha}) \\
\land & \text{ordinalp}\,(\mathit{beta}) \\
\land & \text{ord-lessp}\,(\texttt{0},\ \mathit{beta}) \\
\land & \text{setp}\,(x) \\
\land & \text{listp}\,(x) \\
\land & (m \in \mathbf{N}) \\
\land & (m \leq \text{car}\,(x)) \\
\land & \text{o-largep}\,(x,\ \text{sharp}\,(\mathit{alpha},\ \mathit{beta}))) \\
\rightarrow & \text{o-largep}\,(\text{cdr}\,(x),\ \text{sharp}\,(\mathit{alpha},\ \text{pred}\,(\mathit{beta},\ m)))
\end{array}
$$

THEOREM: sharp-and-cdr-b

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{ordinalp}\,(\mathit{alpha})} \\
\land & \text{ordinalp}\,(\mathit{beta}) \\
\land & \text{ord-lessp}\,(\texttt{0},\ \mathit{beta}) \\
\land & \text{setp}\,(x) \\
\land & \text{listp}\,(x) \\
\land & (m \in \mathbf{N}) \\
\land & (m \leq \text{car}\,(x)) \\
\land & \text{o-largep}\,(x,\ \text{sharp}\,(\mathit{alpha},\ \mathit{beta}))) \\
\rightarrow & \text{o-largep}\,(\text{cdr}\,(x),\ \text{sharp}\,(\mathit{alpha},\ \text{pred}\,(\mathit{beta},\ m)))
\end{array}
$$

EVENT: Disable sharp-and-cdr-aux1.

EVENT: Disable sharp-and-cdr-aux2.

EVENT: Disable sharp-and-cdr-aux3.

EVENT: Disable sharp-and-cdr-aux4.

EVENT: Disable sharp-and-cdr-aux5.

EVENT: Disable sharp-and-cdr-aux6.

EVENT: Disable sharp-and-cdr-aux7.

EVENT: Disable sharp-and-cdr-aux8.

EVENT: Disable sharp-and-cdr-aux9.

EVENT: Disable sharp-and-cdr-aux10.

```
; the same thing with alpha replacing beta follows automatically by symmetry
```

THEOREM: sharp-and-cdr-a

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{ordinalp}\,(\mathit{alpha})} \\
\land & \text{ordinalp}\,(\mathit{beta}) \\
\land & \text{ord-lessp}\,(\mathtt{0},\ \mathit{alpha}) \\
\land & \text{setp}\,(x) \\
\land & \text{listp}\,(x) \\
\land & (m \in \mathbf{N}) \\
\land & (m \leq \text{car}\,(x)) \\
\land & \text{o-largep}\,(x,\, \text{sharp}\,(\mathit{alpha},\ \mathit{beta}))) \\
\rightarrow & \text{o-largep}\,(\text{cdr}\,(x),\, \text{sharp}\,(\text{pred}\,(\mathit{alpha},\ m),\ \mathit{beta}))
\end{array}
$$

```
;;;;;;;;;;  Proof of pigeon-2
; This is the version with 2 holes:
; if (covers A B X) and X is alpha # beta -- large
; then A is alpha -- large     or      B is beta -- large
```

```
; we prove this by induction, showing the following can't happen
```

Definition:
bad-for-pigeon-2 $(a,\ b,\ x,\ alpha,\ beta)$
$=$ (ordinalp $(alpha)$
$\wedge$ ordinalp $(beta)$
$\wedge$ setp (a)
$\wedge$ setp (b)
$\wedge$ setp (x)
$\wedge$ covers $(a,\ b,\ x)$
$\wedge$ o-largep $(x,$ sharp $(alpha,\ beta))$
$\wedge$ $(\neg$ o-largep $(a,\ alpha))$
$\wedge$ $(\neg$ o-largep $(b,\ beta)))$

```
; first, kill off some base cases:

; alpha and beta are > 0
```

Theorem: pigeon-2-aux1
bad-for-pigeon-2 $(a,\ b,\ x,\ alpha,\ beta) \rightarrow$ ord-lessp $(\mathtt{0},\ alpha)$

Theorem: pigeon-2-aux2
bad-for-pigeon-2 $(a,\ b,\ x,\ alpha,\ beta) \rightarrow$ ord-lessp $(\mathtt{0},\ beta)$

Theorem: pigeon-2-aux3
bad-for-pigeon-2 $(a,\ b,\ x,\ alpha,\ beta) \rightarrow$ (ordinalp $(alpha) \wedge$ ordinalp $(beta))$

Theorem: pigeon-2-aux4
bad-for-pigeon-2 $(a,\ b,\ x,\ alpha,\ beta) \rightarrow$ ord-lessp $(\mathtt{0},$ sharp $(alpha,\ beta))$

Theorem: pigeon-2-aux5
bad-for-pigeon-2 $(a,\ b,\ x,\ alpha,\ beta) \rightarrow$ listp (x)

Theorem: pigeon-2-aux6
bad-for-pigeon-2 $(a,\ b,\ x,\ alpha,\ beta)$
$\rightarrow$ (setp $(a) \wedge$ setp $(b) \wedge$ setp $(x) \wedge$ listp $(x) \wedge$ covers $(a,\ b,\ x))$

Theorem: pigeon-2-aux7
bad-for-pigeon-2 $(a,\ b,\ x,\ alpha,\ beta)$
$\rightarrow$ ((listp $(a) \wedge$ (car $(a) \leq$ car $(x)))$
$\vee$ (listp $(b) \wedge$ (car $(b) \leq$ car $(x))))$

```
; now, consider the first case:
; we have (and (listp A) (leq (car A) (car X)))
; we want to show that
; (bad-for-pigeon-2 (cdr A) B (cdr X)  (pred alpha (car A)) beta)

; we check the five lines  in the definition

; line1
```

THEOREM: pigeon-2-aux8
bad-for-pigeon-2 $(a,\ b,\ x,\ alpha,\ beta)$
$\rightarrow$ (ordinalp (pred $(alpha$, car $(a)))$ $\wedge$ ordinalp $(beta))$

```
; line2
```

THEOREM: pigeon-2-aux9
(bad-for-pigeon-2 $(a,\ b,\ x,\ alpha,\ beta)$ $\wedge$ listp (a) $\wedge$ (car $(a) \leq$ car $(x)))$
$\rightarrow$ (setp (cdr (a))
$\wedge$ setp (b)
$\wedge$ setp (cdr (x))
$\wedge$ covers (cdr (a), b, cdr (x)))

```
; line3
```

THEOREM: pigeon-2-aux10
(bad-for-pigeon-2 $(a,\ b,\ x,\ alpha,\ beta)$ $\wedge$ (car $(a) \leq$ car $(x)))$
$\rightarrow$ o-largep (cdr (x), sharp (pred $(alpha$, car $(a))$, $beta))$

```
; line4
```

THEOREM: pigeon-2-aux11
(bad-for-pigeon-2 $(a,\ b,\ x,\ alpha,\ beta)$ $\wedge$ listp $(a))$
$\rightarrow$ ($\neg$ o-largep (cdr (a), pred $(alpha$, car $(a))))$

```
; line5
```

THEOREM: pigeon-2-aux12
bad-for-pigeon-2 $(a,\ b,\ x,\ alpha,\ beta)$ $\rightarrow$ ($\neg$ o-largep $(b,\ beta))$

```
; summarize what we need for
; (bad-for-pigeon-2 (cdr A) B (cdr X) (pred alpha (car A)) beta)
```

THEOREM: pigeon-2-aux13
(ordinalp (pred $(alpha$, car $(a)))$
$\wedge$ ordinalp $(beta)$

$\wedge$ setp (cdr (a))
$\wedge$ setp (b)
$\wedge$ setp (cdr (x))
$\wedge$ covers (cdr (a), b, cdr (x))
$\wedge$ o-largep (cdr (x), sharp (pred ($alpha$, car (a)), $beta$))
$\wedge$ ($\neg$ o-largep (cdr (a), pred ($alpha$, car (a))))
$\wedge$ ($\neg$ o-largep (b, $beta$)))
$\rightarrow$ bad-for-pigeon-2 (cdr (a), b, cdr (x), pred ($alpha$, car (a)), $beta$)

```
; putting them together in the first case
```

THEOREM: pigeon-2-aux14
(bad-for-pigeon-2 (a, b, x, $alpha$, $beta$) $\wedge$ listp (a) $\wedge$ (car (a) $\leq$ car (x)))
$\rightarrow$ bad-for-pigeon-2 (cdr (a), b, cdr (x), pred ($alpha$, car (a)), $beta$)

```
; this ends the first case
; now, consider the second case
; we have (and (listp B) (leq (car B) (car X)))
; we want to show that
; (bad-for-pigeon-2  A (cdr B) (cdr X)  alpha (pred beta (car B)))

; again we check the five lines  in the definition

; line1
```

THEOREM: pigeon-2-aux15
bad-for-pigeon-2 (a, b, x, $alpha$, $beta$)
$\rightarrow$ (ordinalp ($alpha$) $\wedge$ ordinalp (pred ($beta$, car (b))))

```
; line2
```

THEOREM: pigeon-2-aux16
(bad-for-pigeon-2 (a, b, x, $alpha$, $beta$) $\wedge$ listp (b) $\wedge$ (car (b) $\leq$ car (x)))
$\rightarrow$ (setp (a)
  $\wedge$ setp (cdr (b))
  $\wedge$ setp (cdr (x))
  $\wedge$ covers (a, cdr (b), cdr (x)))

```
; line3
```

THEOREM: pigeon-2-aux17
(bad-for-pigeon-2 (a, b, x, $alpha$, $beta$) $\wedge$ (car (b) $\leq$ car (x)))
$\rightarrow$ o-largep (cdr (x), sharp ($alpha$, pred ($beta$, car (b))))

```
; line5
```

THEOREM: pigeon-2-aux18
(bad-for-pigeon-2 $(a,\ b,\ x,\ \textit{alpha},\ \textit{beta}) \land$ listp (b))
$\rightarrow$ ($\neg$ o-largep (cdr (b), pred ($\textit{beta}$, car (b))))

```
; line4
```

THEOREM: pigeon-2-aux19
bad-for-pigeon-2 $(a,\ b,\ x,\ \textit{alpha},\ \textit{beta}) \rightarrow$ ($\neg$ o-largep $(a,\ \textit{alpha})$)

```
; summarize what we need for
; (bad-for-pigeon-2  (cdr B) (cdr X) alpha (pred beta (car B)))
```

THEOREM: pigeon-2-aux20
(ordinalp ($\textit{alpha}$)
$\land$ ordinalp (pred ($\textit{beta}$, car (b)))
$\land$ setp (a)
$\land$ setp (cdr (b))
$\land$ setp (cdr (x))
$\land$ covers (a, cdr (b), cdr (x))
$\land$ o-largep (cdr (x), sharp ($\textit{alpha}$, pred ($\textit{beta}$, car (b))))
$\land$ ($\neg$ o-largep $(a,\ \textit{alpha})$)
$\land$ ($\neg$ o-largep (cdr (b), pred ($\textit{beta}$, car (b)))))
$\rightarrow$ bad-for-pigeon-2 (a, cdr (b), cdr (x), $\textit{alpha}$, pred ($\textit{beta}$, car (b)))

```
; putting them together in the second case
```

THEOREM: pigeon-2-aux21
(bad-for-pigeon-2 $(a,\ b,\ x,\ \textit{alpha},\ \textit{beta}) \land$ listp $(b) \land$ (car $(b) \leq$ car (x)))
$\rightarrow$ bad-for-pigeon-2 (a, cdr (b), cdr (x), $\textit{alpha}$, pred ($\textit{beta}$, car (b)))

```
; now the point is:  if (bad-for-pigeon-2 A B X alpha beta)
; then (listp X) ( by aux5 ) and (cdr X) messes
; up with something -- so we can induct on X
```

THEOREM: pigeon-2-aux22
bad-for-pigeon-2 $(a,\ b,\ x,\ \textit{alpha},\ \textit{beta})$
$\rightarrow$ (bad-for-pigeon-2 (cdr (a), b, cdr (x), pred ($\textit{alpha}$, car (a)), $\textit{beta}$)
$\lor$ bad-for-pigeon-2 (a, cdr (b), cdr (x), $\textit{alpha}$, pred ($\textit{beta}$, car (b))))

DEFINITION:
pigeon-2-kludge $(a,\ b,\ x,\ \textit{alpha},\ \textit{beta})$
= **if** $x \simeq$ **nil** **then** `0`
else pigeon-2-kludge (cdr (a), b, cdr (x), pred ($\textit{alpha}$, car (a)), $\textit{beta}$)
+ pigeon-2-kludge (a,

cdr (b),
cdr (x),
alpha,
pred (*beta*, car (b))) **endif**

THEOREM: pigeon-2-aux23
$\neg$ bad-for-pigeon-2 (a, b, x, *alpha*, *beta*)

`; finally, the GOAL:`

THEOREM: pigeon-2
(ordinalp (*alpha*)
$\land$ ordinalp (*beta*)
$\land$ setp (a)
$\land$ setp (b)
$\land$ setp (x)
$\land$ covers (a, b, x)
$\land$ o-largep (x, sharp (*alpha*, *beta*)))
$\rightarrow$ (o-largep (a, *alpha*) $\lor$ o-largep (b, *beta*))

EVENT: Disable bad-for-pigeon-2.

EVENT: Disable pigeon-2-kludge.

EVENT: Disable pigeon-2-aux1.

EVENT: Disable pigeon-2-aux2.

EVENT: Disable pigeon-2-aux3.

EVENT: Disable pigeon-2-aux4.

EVENT: Disable pigeon-2-aux5.

EVENT: Disable pigeon-2-aux6.

EVENT: Disable pigeon-2-aux7.

EVENT: Disable pigeon-2-aux8.

EVENT: Disable pigeon-2-aux9.

EVENT: Disable pigeon-2-aux10.

EVENT: Disable pigeon-2-aux11.

EVENT: Disable pigeon-2-aux12.

EVENT: Disable pigeon-2-aux13.

EVENT: Disable pigeon-2-aux14.

EVENT: Disable pigeon-2-aux15.

EVENT: Disable pigeon-2-aux16.

EVENT: Disable pigeon-2-aux17.

EVENT: Disable pigeon-2-aux18.

EVENT: Disable pigeon-2-aux19.

EVENT: Disable pigeon-2-aux20.

EVENT: Disable pigeon-2-aux21.

EVENT: Disable pigeon-2-aux22.

EVENT: Disable pigeon-2-aux23.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                  FUNCTIONS AND HOMOGENEOUS SETS                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; view every S-expression as an association list, and hence a function
```

DEFINITION: $\text{funcall}(g,\ x) = \text{cadr}(\text{assoc}(x,\ g))$

```
; since (cadr nil) = 0, we always have 0 in the range:
```

DEFINITION:
$\text{range}(g)$
$= \ \ \textbf{if}\ g \simeq \textbf{nil}\ \ \textbf{then}\ \text{list}(0)$
$\quad\ \ \textbf{else}\ \text{cons}(\text{cadar}(g),\ \text{range}(\text{cdr}(g)))\ \textbf{endif}$

THEOREM: range-is-big-enuf
$\text{funcall}(g,\ x) \in \text{range}(g)$

EVENT: Disable range.

```
; we don't care exactly what it is
```

EVENT: Disable funcall.

```
; for now -- later, we'll need its definition to show that we
; can build up functions with prescribed definitions
```

DEFINITION:
$\text{mapsto}(g,\ set1,\ set2)$
$= \ \ \textbf{if}\ set1 \simeq \textbf{nil}\ \ \textbf{then}\ \textbf{t}$
$\quad\ \ \textbf{else}\ \text{mapsto}(g,\ \text{cdr}(set1),\ set2)$
$\qquad\quad \wedge \ \ (\text{funcall}(g,\ \text{car}(set1)) \in set2)\ \textbf{endif}$

THEOREM: mapsto-works
$(\text{mapsto}(g,\ set1,\ set2) \wedge (x \in set1)) \rightarrow (\text{funcall}(g,\ x) \in set2)$

EVENT: Disable mapsto.

```
;;;;;;;;;;;;;;;;;;;;;;;;
; let's disable some ordinal stuff which seems to slow down the set proofs
```

EVENT: Disable numbers-below-omega.

EVENT: Disable cdrs-are-ordinals.

EVENT: Disable ord-leq-zero.

EVENT: Disable ord-leq-is-transitive.

```
;;;;;;;;;;;;  notion of homogeneous

; (homp set g n) means that  set is homogeneous for g as a partition
; of n-tuples:  that is,
; whenever s1, s2 are subsets of set of size <= n, and
; |s1| =|s2| and g(s1) = g(s2) < c, then g(s1) = g(s2)

; we first define what a counter-example would be
; search through the list of all possible counter-examples
```

DEFINITION:
hom-bad-pairp $(x,\ y,\ set,\ g,\ n)$
$=$ (sublistp $(x,\ set)$
$\wedge$ sublistp $(y,\ set)$
$\wedge$ setp (x)
$\wedge$ setp (y)
$\wedge$ (length $(x) = n$)
$\wedge$ (length $(y) = n$)
$\wedge$ (funcall $(g,\ x) \neq$ funcall $(g,\ y)$))

```
; use sublistp rather than subsetp because power-set was defined
; to work with sublistp.  These are equivalent anyway when
; (setp set), which is what we're interested in.

; pairs is a list of pairs which are possible counter-examples
; counter-hom looks for a pair (x y) which refutes homogeneous
; if it fails, it returns 0

; disable bad-pair temporarily
```

EVENT: Disable hom-bad-pairp.

DEFINITION:
counter-hom $(set,\ g,\ n,\ pairs)$
$=$ **if** *pairs* $\simeq$ **nil then** 0
elseif listp (counter-hom $(set,\ g,\ n,$ cdr $(pairs)))$
then counter-hom $(set,\ g,\ n,$ cdr $(pairs))$
elseif hom-bad-pairp (caar $(pairs)$, cdar $(pairs)$, $set,\ g,\ n)$
then cons (caar $(pairs)$, cdar $(pairs)$)
else 0 **endif**

THEOREM: bad-pair-is-bad
listp (counter-hom $(set,\ g,\ n,\ pairs)$)

$\rightarrow$ hom-bad-pairp (car (counter-hom (set, g, n, $pairs$)),
cdr (counter-hom (set, g, n, $pairs$)),
set,
g,
n)

THEOREM: counter-hom-finds-one
((cons (x, y) $\in$ $pairs$) $\wedge$ hom-bad-pairp (x, y, set, g, n))
$\rightarrow$ listp (counter-hom (set, g, n, $pairs$))

EVENT: Disable counter-hom.

```
; now -- homp means we search thru all pairs from the
; power-set and don't find one
```

DEFINITION:
homp (set, g, n)
= (counter-hom (set, g, n, product (power-set (set), power-set (set))) $\simeq$ **nil**)

```
; now, we prove that homp is necessary and sufficient
; this needs the defn of hom-bad-pairp
```

EVENT: Enable hom-bad-pairp.

THEOREM: homp-is-sufficient-a
(homp (set, g, n)
$\wedge$ sublistp (x, set)
$\wedge$ sublistp (y, set)
$\wedge$ setp (x)
$\wedge$ setp (y)
$\wedge$ (length (x) = n)
$\wedge$ (length (y) = n))
$\rightarrow$ (funcall (g, x) = funcall (g, y))

```
;;; in our paper, it would be less confusing to have a version
; to quote which uses subsetp instead of sublistp
```

THEOREM: homp-is-sufficient
(homp (set, g, n)
$\wedge$ subsetp (x, set)
$\wedge$ subsetp (y, set)
$\wedge$ setp (set)

$\wedge$ setp (x)
$\wedge$ setp (y)
$\wedge$ (length $(x) = n$)
$\wedge$ (length $(y) = n$))
$\rightarrow$ (funcall $(g,\ x)$ = funcall $(g,\ y)$)

```
; to prove necessary, it would be nice to have an abbreviation for
; the x and y coordinate of the counter-example
```

DEFINITION:
counter-hom-x $(set,\ g,\ n)$
= car (counter-hom $(set,\ g,\ n,$ product (power-set (set), power-set (set))))

DEFINITION:
counter-hom-y $(set,\ g,\ n)$
= cdr (counter-hom $(set,\ g,\ n,$ product (power-set (set), power-set (set))))

THEOREM: homp-is-necessary
($\neg$ homp $(set,\ g,\ n)$)
$\rightarrow$ (sublistp (counter-hom-x $(set,\ g,\ n)$, set)
$\wedge$ sublistp (counter-hom-y $(set,\ g,\ n)$, set)
$\wedge$ setp (counter-hom-x $(set,\ g,\ n)$)
$\wedge$ setp (counter-hom-y $(set,\ g,\ n)$)
$\wedge$ (length (counter-hom-x $(set,\ g,\ n)$) $= n$)
$\wedge$ (length (counter-hom-y $(set,\ g,\ n)$) $= n$)
$\wedge$ (funcall (g, counter-hom-x $(set,\ g,\ n)$)
$\neq$ funcall (g, counter-hom-y $(set,\ g,\ n)$))))

EVENT: Disable homp.

EVENT: Disable hom-bad-pairp.

EVENT: Disable counter-hom-x.

EVENT: Disable counter-hom-y.

```
;;;;;;;;;;;;;;;  add-on-end
; before discussing pre-homogeneous sets, we have to talk
; about adding one element at the end of a set -- maybe
; it's simpler to define this outright, rather than using "append"
```

DEFINITION:
add-on-end $(item,\ set)$
$=$ **if** $set \simeq$ **nil then** list $(item)$
else cons (car (set), add-on-end $(item,\ cdr\ (set))$) **endif**

```
; for sets, this is set Union {item}
```

THEOREM: add-in-end-is-last
last (add-on-end $(item,\ set)$) $= item$

THEOREM: add-in-end-and-length
length (add-on-end $(item,\ set)$) $= (1 +$ length $(set))$

```
; we should probably define all-but-last  -- everything
; except the last element
```

THEOREM: all-but-last-aux1
listp (cdr (lst)) $\rightarrow$ listp (lst)

```
; needed for the recursion
```

DEFINITION:
all-but-last (lst)
$=$ **if** cdr $(lst) \simeq$ **nil then nil**
else cons (car (lst), all-but-last (cdr (lst))) **endif**

EVENT: Disable all-but-last-aux1.

THEOREM: all-but-last-and-length
listp (cdr (lst)) $\rightarrow$ (length (all-but-last (lst)) $=$ (length $(lst) - 1$))

THEOREM: all-but-last-of-add-on-end
properp $(lst) \rightarrow$ (all-but-last (add-on-end $(item,\ lst)$) $= lst$)

EVENT: Enable setp.

THEOREM: add-on-end-makes-sets
(setp $(set) \wedge (n \in \mathbf{N}) \wedge$ (last $(set) < n$)) $\rightarrow$ setp (add-on-end $(n,\ set)$)

EVENT: Disable setp.

THEOREM: all-but-last-is-sublist
sublistp (all-but-last (lst), lst)

THEOREM: all-but-last-is-sublist-a
sublistp ($s1$, set) $\rightarrow$ sublistp (all-but-last ($s1$), set)

THEOREM: all-but-last-is-non-empty
($2 \leq$ length (lst)) $\rightarrow$ listp (all-but-last (lst))

EVENT: Enable setp.

THEOREM: all-but-last-is-a-set
setp (set) $\rightarrow$ setp (all-but-last (set))

EVENT: Disable setp.

THEOREM: re-assemble-at-end
(properp (lst) $\wedge$ listp (lst))
$\rightarrow$ (add-on-end (last (lst), all-but-last (lst)) $=$ lst)

THEOREM: length-of-all-but-last
listp (x) $\rightarrow$ (length (all-but-last (x)) $=$ (length (x) $-$ 1))

```
;;;;;;;;;;;;  notion of pre-homogeneous

; (pre-homp set g) means that  set is pre-homogeneous for g as a partition
; that is:  whenever s1 is a non-empty subset of set, y,z are members of
; set and are larger than max(s1) :
; g(s1 U {y}) = g(s1 U {z})

; we use the same "bad" method
```

DEFINITION:
pre-hom-bad-triplep (y, z, $s1$, set, g)
$=$ (sublistp ($s1$, set)
$\wedge$ listp ($s1$)
$\wedge$ setp ($s1$)
$\wedge$ ($y \in set$)
$\wedge$ ($z \in set$)
$\wedge$ (last ($s1$) $< y$)
$\wedge$ (last ($s1$) $< z$)
$\wedge$ (funcall (g, add-on-end (y, $s1$)) $\neq$ funcall (g, add-on-end (z, $s1$))))

```
; disable bad-triple temporarily
```

EVENT: Disable pre-hom-bad-triplep.

```
; triples will be just ( (a .  b) .  c) -- so we get a,b,c as the
; caar, cdar, cdr
; then all triples are getten as (product (product set set) (power-set set))
```

DEFINITION:
counter-pre-hom (*set*, *g*, *triples*)
= **if** *triples* $\simeq$ **nil** **then** 0
 elseif listp (counter-pre-hom (*set*, *g*, cdr (*triples*)))
 then counter-pre-hom (*set*, *g*, cdr (*triples*))
 elseif pre-hom-bad-triplep (caaar (*triples*),
 cdaar (*triples*),
 cdar (*triples*),
 set,
 g)
 then cons (cons (caaar (*triples*), cdaar (*triples*)), cdar (*triples*))
 else 0 **endif**

THEOREM: bad-triple-is-bad
listp (counter-pre-hom (*set*, *g*, *triples*))
$\rightarrow$ pre-hom-bad-triplep (caar (counter-pre-hom (*set*, *g*, *triples*)),
 cdar (counter-pre-hom (*set*, *g*, *triples*)),
 cdr (counter-pre-hom (*set*, *g*, *triples*)),
 set,
 g)

THEOREM: counter-pre-hom-finds-one
((cons (cons (*y*, *z*), *s1*) $\in$ *triples*) $\land$ pre-hom-bad-triplep (*y*, *z*, *s1*, *set*, *g*))
$\rightarrow$ listp (counter-pre-hom (*set*, *g*, *triples*))

EVENT: Disable counter-pre-hom.

DEFINITION:
pre-homp (*set*, *g*)
= (counter-pre-hom (*set*, *g*, product (product (*set*, *set*), power-set (*set*))) $\simeq$ **nil**)

```
; now, we prove that pre-homp is necessary and sufficient
; this needs the defn of pre-hom-bad-triplep
```

EVENT: Enable pre-hom-bad-triplep.

THEOREM: pre-homp-is-sufficient
(pre-homp (*set*, *g*)
 $\land$ sublistp (*s1*, *set*)

$\wedge$ listp (*s1*)
$\wedge$ setp (*s1*)
$\wedge$ ($y \in set$)
$\wedge$ ($z \in set$)
$\wedge$ (last (*s1*) $< y$)
$\wedge$ (last (*s1*) $< z$))
$\rightarrow$ (funcall (g, add-on-end (y, *s1*)) = funcall (g, add-on-end (z, *s1*)))

```
;  to prove necessary, we use an abbreviation for
; the y,z, s1 of the counter-example
```

DEFINITION:
counter-pre-hom-y (*set*, *g*)
= caar (counter-pre-hom (*set*, *g*, product (product (*set*, *set*), power-set (*set*))))

DEFINITION:
counter-pre-hom-z (*set*, *g*)
= cdar (counter-pre-hom (*set*, *g*, product (product (*set*, *set*), power-set (*set*))))

DEFINITION:
counter-pre-hom-s1 (*set*, *g*)
= cdr (counter-pre-hom (*set*, *g*, product (product (*set*, *set*), power-set (*set*))))

THEOREM: pre-homp-is-necessary
($\neg$ pre-homp (*set*, *g*))
$\rightarrow$ (sublistp (counter-pre-hom-s1 (*set*, *g*), *set*)
  $\wedge$ listp (counter-pre-hom-s1 (*set*, *g*))
  $\wedge$ setp (counter-pre-hom-s1 (*set*, *g*))
  $\wedge$ (counter-pre-hom-y (*set*, *g*) $\in$ *set*)
  $\wedge$ (counter-pre-hom-z (*set*, *g*) $\in$ *set*)
  $\wedge$ (last (counter-pre-hom-s1 (*set*, *g*)) $<$ counter-pre-hom-y (*set*, *g*))
  $\wedge$ (last (counter-pre-hom-s1 (*set*, *g*)) $<$ counter-pre-hom-z (*set*, *g*))
  $\wedge$ (funcall (g,
        add-on-end (counter-pre-hom-y (*set*, *g*),
              counter-pre-hom-s1 (*set*, *g*)))
    $\neq$ funcall (g,
        add-on-end (counter-pre-hom-z (*set*, *g*),
              counter-pre-hom-s1 (*set*, *g*)))))

EVENT: Disable pre-homp.

EVENT: Disable pre-hom-bad-triplep.

EVENT: Disable counter-pre-hom-y.

EVENT: Disable counter-pre-hom-z.

EVENT: Disable counter-pre-hom-s1.

```
;;;;;;;;;;;;;;;;;;;;   derived partitions
; This is used in the induction step in Ramsey's Theorem --
;  passing from n-1 - tuples to n - tuples
; Think of g as a partition of n-tuples (n >= 2)
; If set is pre-hom for g, there is a derived partition h, on n-1 - tuples
; such that whenever hset is homogeneous for h, hset is homogeneous for g

; to compute (h s1) :  apply g to s1 U {x}, where x =
; (find-larger-element s1 set) -- this returns an element x of set
; which is a number and larger than (last s1)
; it returns nil if there is no such number
; so -- by testing for numberp, we know if we have found one.
```

DEFINITION:
find-larger-element (*s1*, *set*)
= **if** *set* $\simeq$ **nil then nil**
elseif (car (*set*) $\in$ **N**) $\land$ (last (*s1*) $<$ car (*set*)) **then** car (*set*)
else find-larger-element (*s1*, cdr (*set*)) **endif**

THEOREM: find-larger-element-works-a
(find-larger-element (*s1*, *set*) $\in$ **N**)
$\rightarrow$ ((find-larger-element (*s1*, *set*) $\in$ *set*)
$\land$ (find-larger-element (*s1*, *set*) $\in$ **N**)
$\land$ (last (*s1*) $<$ find-larger-element (*s1*, *set*)))

THEOREM: find-larger-element-works-b
(($x \in$ *set*) $\land$ ($x \in$ **N**) $\land$ (last (*s1*) $< x$))
$\rightarrow$ (find-larger-element (*s1*, *set*) $\in$ **N**)

EVENT: Disable find-larger-element.

```
; we don't care which one we found

; if s1 is a subset of set and s1 has length at least 2,
; then (find-larger-element (all-but-last) set )  really
; finds one -- since (last s1) is a candidate.

; first, we have to show it really is a candidate
```

```
; the following is probably needed also for properties
; of pre-homogeneous sets
```

THEOREM: last-is-a-member
$\text{listp}\,(lst) \rightarrow (\text{last}\,(lst) \in lst)$

THEOREM: last-is-a-candidate-1
$(\text{sublistp}\,(s1,\ set) \wedge (2 \leq \text{length}\,(s1))) \rightarrow (\text{last}\,(s1) \in set)$

EVENT: Enable setp.

THEOREM: members-are-numbers
$(\text{setp}\,(set) \wedge (x \in set)) \rightarrow (x \in \mathbf{N})$

EVENT: Disable setp.

THEOREM: last-is-a-candidate-2
$(\text{setp}\,(s1) \wedge (2 \leq \text{length}\,(s1))) \rightarrow (\text{last}\,(s1) \in \mathbf{N})$

EVENT: Enable setp.

THEOREM: last-is-a-candidate-3
$(\text{setp}\,(s1) \wedge (2 \leq \text{length}\,(s1)))$
$\rightarrow \quad (\text{last}\,(\text{all-but-last}\,(s1)) < \text{last}\,(s1))$

EVENT: Disable setp.

THEOREM: larger-elt-is-found
$(\text{setp}\,(s1) \wedge \text{sublistp}\,(s1,\ set) \wedge (2 \leq \text{length}\,(s1)))$
$\rightarrow \quad (\text{find-larger-element}\,(\text{all-but-last}\,(s1),\ set) \in \mathbf{N})$

```
; it follows that if g is pre-homogeneous on set, then (g s1) =
; (g  (add-on-end
;         (find-larger-element (all-but-last s1) set)
;         (all-but-last s1) ) )
; that is, (g s1) is determined as the derived partition
;  applied to (all-but-last s1)
```

THEOREM: pre-hom-and-all-but-last
$(\text{pre-homp}\,(set,\ g) \wedge \text{setp}\,(s1) \wedge \text{sublistp}\,(s1,\ set) \wedge (2 \leq \text{length}\,(s1)))$
$\rightarrow \quad (\text{funcall}\,(g,\ s1)$
$\quad = \quad \text{funcall}\,(g,$
$\qquad \text{add-on-end}\,(\text{find-larger-element}\,(\text{all-but-last}\,(s1),\ set),$
$\qquad\qquad \text{all-but-last}\,(s1))))$

```
; now we simply define the derived partition h = (derived g set)  so that
;(h s) =
;(funcall g
;    (add-on-end
;        (find-larger-element s set)
;        s ))
; but to define a function h like this, we have to run down
; a list of subsets -- i.e., (power-set set)
; and construct h as a list of pairs
```

DEFINITION:
derived-aux (g, set, lst)
$=$ **if** $lst \simeq$ **nil then nil**
else cons (list (car (lst),
funcall (g,
add-on-end (find-larger-element (car (lst),
set),
car (lst)))),
derived-aux (g, set, cdr (lst))) **endif**

THEOREM: derived-aux1
($s \in lst$)
$\rightarrow$ (assoc (s, derived-aux (g, set, lst))
$=$ list (s, funcall (g, add-on-end (find-larger-element (s, set), s))))

EVENT: Enable funcall.

THEOREM: derived-aux2
($s \in lst$)
$\rightarrow$ (funcall (derived-aux (g, set, lst), s)
$=$ funcall (g, add-on-end (find-larger-element (s, set), s)))

EVENT: Disable funcall.

EVENT: Disable derived-aux.

EVENT: Disable derived-aux1.

DEFINITION: derived (g, set) $=$ derived-aux (g, set, power-set (set))

THEOREM: derived-works
(sublistp (s, set) $\wedge$ setp (s))

$\rightarrow$ (funcall (derived (g, set), s)
$= \text{funcall}(g, \text{add-on-end}(\text{find-larger-element}(s, set), s)))$

EVENT: Disable derived.

EVENT: Disable derived-aux2.

```
; now, if g is pre-homogeneous, then
; (g s1) is computed from ((derived g set) s), where
; s = (all-but-last s1)
```

THEOREM: derived-and-prehom
$(\text{pre-homp}(set, g) \wedge \text{setp}(s1) \wedge \text{sublistp}(s1, set) \wedge (2 \leq \text{length}(s1)))$
$\rightarrow (\text{funcall}(g, s1) = \text{funcall}(\text{derived}(g, set), \text{all-but-last}(s1)))$

```
; now suppose n>=2 and  g is prehom on set
; then say we find small-set is a subset of set which
; is homogeneous for (derivde g set) on the n-1 -- tuples
; then small-set is homogeneous for g on the n -- tuples

; to prove this, we consider any two subsets x, y of small-set
; of size n and prove that g(x) = h(all-but-last x) = h(all-but-last y)
; = g(y) , where h is the derived partition
```

THEOREM: derived-partition-lemma-aux1
$(\text{length}(x) \not< 2) \rightarrow (\text{length}(\text{all-but-last}(x)) = (\text{length}(x) - 1))$

THEOREM: derived-partition-lemma-aux2
$((2 \leq n)$
$\wedge$ pre-homp (set, g)
$\wedge$ sublistp ($small\text{-}set$, set)
$\wedge$ homp ($small\text{-}set$, derived (g, set), $n - 1$)
$\wedge$ sublistp (x, $small\text{-}set$)
$\wedge$ sublistp (y, $small\text{-}set$)
$\wedge$ setp (x)
$\wedge$ setp (y)
$\wedge$ $(\text{length}(x) = n)$
$\wedge$ $(\text{length}(y) = n))$
$\rightarrow (\text{funcall}(g, x) = \text{funcall}(g, y))$

```
; now, if (homp small-set g n) fails, lookat a counter-example
```

THEOREM: derived-partition-lemma
$((2 \le n)$
$\land$ pre-homp (set, g)
$\land$ sublistp ($small\text{-}set$, set)
$\land$ homp ($small\text{-}set$, derived (g, set), $n - 1$))
$\rightarrow$ homp ($small\text{-}set$, g, n)

EVENT: Disable derived-partition-lemma-aux1.

EVENT: Disable derived-partition-lemma-aux2.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                           THE TAIL LEMMA                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Suppose ZZ is a set which is omega^alpha + omega + k  -- large.
; let z = min ZZ.
; If we take cdr's magic(z+d) times, we get a set QQ such that
; 1. QQ is a set and omega^alpha -- large
; 2. QQ is a subset of ZZ
; 3. min(QQ) >= min(ZZ) + magic(z+k).

; To prove this, we analyze ordinals delta of the forms:
;   type A: delta = omega^alpha + omega + k (k > 0)   (alpha 1 . k) k > 0
;   type B: delta = omega^alpha +  k        (k > 0)   (alpha . k)  k > 0
;   type C: delta = omega^alpha + omega               (alpha 1 . 0)

; As a first step, consider the {delta}(n) for these ordinals.

; type A  : (cons alpha (cons 1 k)) : k > 0
```

EVENT: Enable ord-lessp.

THEOREM: type-a-is-ordinal
(ordinalp ($alpha$) $\land$ ord-leq (1, $alpha$) $\land$ $(0 < k)$)
$\rightarrow$ ordinalp (cons ($alpha$, cons (1, k)))

EVENT: Disable ord-lessp.

THEOREM: type-a-is-a-successor
$(0 < k) \rightarrow$ successorp (cons ($alpha$, cons (1, k)))

THEOREM: predecessor-of-type-a
$(0 < k)$
$\rightarrow$ (predecessor (cons (*alpha*, cons (1, *k*))) = cons (*alpha*, cons (1, $k - 1$)))

THEOREM: pred-type-a
(ordinalp (*alpha*) $\wedge$ ord-leq (1, *alpha*) $\wedge$ $(0 < k)$)
$\rightarrow$ (pred (cons (*alpha*, cons (1, *k*)), *n*) = cons (*alpha*, cons (1, $k - 1$)))

```
; type B  : (cons alpha k) : k > 0
```

EVENT: Enable ord-lessp.

THEOREM: type-b-is-ordinal
(ordinalp (*alpha*) $\wedge$ ord-leq (1, *alpha*) $\wedge$ $(0 < k)$)
$\rightarrow$ ordinalp (cons (*alpha*, *k*))

EVENT: Disable ord-lessp.

THEOREM: type-b-is-a-successor
$(0 < k)$ $\rightarrow$ successorp (cons (*alpha*, *k*))

THEOREM: predecessor-of-type-b
$(0 < k)$ $\rightarrow$ (predecessor (cons (*alpha*, *k*)) = cons (*alpha*, $k - 1$))

THEOREM: pred-type-b
(ordinalp (*alpha*) $\wedge$ ord-leq (1, *alpha*) $\wedge$ $(0 < k)$)
$\rightarrow$ (pred (cons (*alpha*, *k*), *n*) = cons (*alpha*, $k - 1$))

```
; type C  : (cons alpha (cons 1 0))
```

EVENT: Enable ord-lessp.

THEOREM: type-c-is-ordinal
(ordinalp (*alpha*) $\wedge$ ord-leq (1, *alpha*)) $\rightarrow$ ordinalp (cons (*alpha*, cons (1, 0)))

EVENT: Disable ord-lessp.

THEOREM: type-c-is-a-limit
limitp (cons (*alpha*, cons (1, 0)))

```
; Now we want:
; (equal
;   (pred (cons alpha (cons 1 0)) n)
```

```
;   (cons alpha (plus (magic n) 2)) )

; first, compute the norms
```

THEOREM: norm-of-type-c
$\text{norm}\,(\text{cons}\,(\mathit{alpha},\ \text{cons}\,(\mathtt{1},\ \mathtt{0}))) = (\text{norm}\,(\mathit{alpha}) + \mathtt{3})$

THEOREM: norm-of-pred-of-type-c
$(\text{ordinalp}\,(\mathit{alpha}) \land \text{ord-leq}\,(\mathtt{1},\ \mathit{alpha}) \land (n \in \mathbf{N}))$
$\to\ \ (\text{norm}\,(\text{pred}\,(\text{cons}\,(\mathit{alpha},\ \text{cons}\,(\mathtt{1},\ \mathtt{0})),\ n))$
$\quad = \ (\text{norm}\,(\mathit{alpha}) + \mathtt{3} + \text{magic}\,(n)))$

THEOREM: norm-of-right-answer-c
$(n \in \mathbf{N})$
$\to\ \ (\text{norm}\,(\text{cons}\,(\mathit{alpha},\ \text{magic}\,(n) + \mathtt{2}))$
$\quad = \ (\text{norm}\,(\mathit{alpha}) + \mathtt{3} + \text{magic}\,(n)))$

```
; now, we prove that the right answer,  omega^alpha + (magic n) + 2
; is less than omega^alpha + omega  -- then we need
; only show that anything strictly betwwen
; omega^alpha + (magic n) + 2 and omega^alpha + omega  has larger norm
```

THEOREM: pred-type-c-aux1
$\text{ord-lessp}\,(\text{cons}\,(\mathit{alpha},\ \text{magic}\,(n) + \mathtt{2}),\ \text{cons}\,(\mathit{alpha},\ \text{cons}\,(\mathtt{1},\ \mathtt{0})))$

THEOREM: pred-type-c-aux2
$(\text{ordinalp}\,(\mathit{alpha}) \land \text{ord-leq}\,(\mathtt{1},\ \mathit{alpha}) \land (n \in \mathbf{N}))$
$\to\ \ \text{ord-leq}\,(\text{cons}\,(\mathit{alpha},\ \text{magic}\,(n) + \mathtt{2}),\ \text{pred}\,(\text{cons}\,(\mathit{alpha},\ \text{cons}\,(\mathtt{1},\ \mathtt{0})),\ n))$

```
; so, the only remaining possibility is that
; (pred (cons alpha (cons 1 0)) n)   is strictly between
; (cons alpha (plus (magic n) 2))   and (cons alpha (cons 1 0))
; but then its norm would be too large

; any ordinal strictly between must be of the form
;  (cons alpha j)  where j > (plus (magic n) 2)
```

THEOREM: pred-type-c-aux3
$(\text{ordinalp}\,(\mathit{alpha})$
$\ \land\ \ \text{ord-leq}\,(\mathtt{1},\ \mathit{alpha})$
$\ \land\ \ \text{ordinalp}\,(\mathit{sigma})$

$\wedge \quad (r \in \mathbf{N})$
$\wedge \quad \text{ord-lessp}\,(\text{cons}\,(\mathit{alpha},\ r),\ \mathit{sigma})$
$\wedge \quad \text{ord-lessp}\,(\mathit{sigma},\ \text{cons}\,(\mathit{alpha},\ \text{cons}\,(\mathtt{1},\ \mathtt{0}))))$
$\rightarrow \quad ((\text{car}\,(\mathit{sigma}) = \mathit{alpha}) \wedge (\text{cdr}\,(\mathit{sigma}) \in \mathbf{N}) \wedge (r < \text{cdr}\,(\mathit{sigma})))$

THEOREM: pred-type-c-aux4
$(\text{ordinalp}\,(\mathit{alpha})$
$\wedge \quad \text{ord-leq}\,(\mathtt{1},\ \mathit{alpha})$
$\wedge \quad \text{ordinalp}\,(\mathit{sigma})$
$\wedge \quad (n \in \mathbf{N})$
$\wedge \quad \text{ord-lessp}\,(\text{cons}\,(\mathit{alpha},\ \text{magic}\,(n) + \mathtt{2}),\ \mathit{sigma})$
$\wedge \quad \text{ord-lessp}\,(\mathit{sigma},\ \text{cons}\,(\mathit{alpha},\ \text{cons}\,(\mathtt{1},\ \mathtt{0}))))$
$\rightarrow \quad ((\text{norm}\,(\mathit{alpha}) + \mathtt{3} + \text{magic}\,(n)) < \text{norm}\,(\mathit{sigma}))$

THEOREM: pred-type-c-aux5
$(\text{ordinalp}\,(\mathit{alpha}) \wedge \text{ord-leq}\,(\mathtt{1},\ \mathit{alpha}) \wedge (n \in \mathbf{N}))$
$\rightarrow \quad (\neg\ \text{ord-lessp}\,(\text{cons}\,(\mathit{alpha},\ \text{magic}\,(n) + \mathtt{2}),$
$\qquad\qquad \text{pred}\,(\text{cons}\,(\mathit{alpha},\ \text{cons}\,(\mathtt{1},\ \mathtt{0})),\ n)))$

THEOREM: pred-type-c
$(\text{ordinalp}\,(\mathit{alpha}) \wedge \text{ord-leq}\,(\mathtt{1},\ \mathit{alpha}) \wedge (n \in \mathbf{N}))$
$\rightarrow \quad (\text{pred}\,(\text{cons}\,(\mathit{alpha},\ \text{cons}\,(\mathtt{1},\ \mathtt{0})),\ n) = \text{cons}\,(\mathit{alpha},\ \text{magic}\,(n) + \mathtt{2}))$

EVENT: Disable pred-type-c-aux1.

EVENT: Disable pred-type-c-aux2.

EVENT: Disable pred-type-c-aux3.

EVENT: Disable pred-type-c-aux4.

EVENT: Disable pred-type-c-aux5.

```
;; OK -- that finishes the {delta}(n) discussion

;;;;;;;; nthcdr
; this is like the common lisp function
```

DEFINITION:
$\text{nthcdr}\,(n,\ \mathit{lst})$
$= \quad \textbf{if } n \simeq \mathtt{0} \textbf{ then } \mathit{lst}$
$\qquad \textbf{else } \text{cdr}\,(\text{nthcdr}\,(n - 1,\ \mathit{lst}))\ \textbf{endif}$

```
; some trivail properties:
```

THEOREM: nthcdr-0
$(n \simeq \mathtt{0}) \rightarrow (\mathrm{nthcdr}\,(n,\ lst) = lst)$

THEOREM: nthcdr-1
$\mathrm{nthcdr}\,(\mathtt{1},\ lst) = \mathrm{cdr}\,(lst)$

THEOREM: nthcdr-add
$\mathrm{nthcdr}\,(m,\ \mathrm{nthcdr}\,(n,\ lst)) = \mathrm{nthcdr}\,(m + n,\ lst)$

```
; since the cdr of a non-list is 0:
```

THEOREM: length-of-nthcdr
$(\mathrm{nthcdr}\,(n,\ lst) \neq \mathtt{0}) \rightarrow (\mathrm{length}\,(\mathrm{nthcdr}\,(n,\ lst)) = (\mathrm{length}\,(lst) - n))$

```
; since non-lists are subsets of everying
```

EVENT: Enable subsetp.

THEOREM: nthcdr-is-subset
$\mathrm{subsetp}\,(\mathrm{nthcdr}\,(n,\ lst),\ lst)$

EVENT: Disable subsetp.

```
; now, special facts about sets --
; we want to know that (nthcdr n set) is a subset of set
; whose car is at least  (car set) + n
; (assuming  the nthdr isn't empty)
```

THEOREM: cdr-is-a-set
$(\mathrm{setp}\,(set) \wedge (\mathrm{cdr}\,(set) \neq \mathtt{0})) \rightarrow \mathrm{setp}\,(\mathrm{cdr}\,(set))$

THEOREM: nthcdr-is-a-set
$(\mathrm{setp}\,(set) \wedge (\mathrm{nthcdr}\,(n,\ set) \neq \mathtt{0})) \rightarrow \mathrm{setp}\,(\mathrm{nthcdr}\,(n,\ set))$

THEOREM: cdr-is-above
$(\mathrm{setp}\,(set) \wedge \mathrm{listp}\,(\mathrm{cdr}\,(set))) \rightarrow (\mathrm{car}\,(set) < \mathrm{car}\,(\mathrm{cdr}\,(set)))$

THEOREM: cdr-is-above-by-1
$(\mathrm{setp}\,(set) \wedge \mathrm{listp}\,(\mathrm{cdr}\,(set))) \rightarrow (\mathrm{car}\,(\mathrm{cdr}\,(set)) \not< (1 + \mathrm{car}\,(set)))$

Definition:
bad-for-nthcdr-is-above-by-n $(n,\ set)$
$=$ (setp (set)
$\wedge$ listp (nthcdr $(n,\ set)$)
$\wedge$ (car (nthcdr $(n,\ set)$) $<$ (n + car (set))))

Theorem: nthcdr-is-above-by-n-aux1
$(n \simeq 0) \rightarrow (\neg$ bad-for-nthcdr-is-above-by-n $(n,\ set))$

Theorem: nthcdr-is-above-by-n-aux2
(setp (set) $\wedge$ ($\neg$ setp (nthcdr $(n,\ set)$)))
$\rightarrow$ ($\neg$ listp (cdr (nthcdr $(n,\ set)$)))

Theorem: nthcdr-is-above-by-n-aux3
bad-for-nthcdr-is-above-by-n $(1 +\ n,\ set)$
$\rightarrow$ bad-for-nthcdr-is-above-by-n $(n,\ set)$

Theorem: nthcdr-is-above-by-n-aux4
$\neg$ bad-for-nthcdr-is-above-by-n $(n,\ set)$

Theorem: nthcdr-is-above-by-n
(setp (set) $\wedge$ listp (nthcdr $(n,\ set)$))
$\rightarrow$ (car (nthcdr $(n,\ set)$) $\not<$ (n + car (set)))

```
;;;;;;;;;;;
; now, suppose Z is omega^alpha + omega + d large (say, alpha > 0),
; and z = min(Z).   Let Q = (nthcdr (magic(z +d )  Z))
; then Q is omega^alpha large (in particular, non-empty) , and
; then, by the above, Q is a subset of Z and min(Q)  >= magic(z + d)

; we consider the three types of ordinals in sequence.

; one more simple fact -- needed in the following inductive proofs
```

Theorem: nthcdr-of-cdr
$(k \not\simeq 0) \rightarrow$ (nthcdr $(k - 1$, cdr $(set))$ = nthcdr $(k,\ set)$)

```
; type A  : (cons alpha (cons 1 k)) : k > 0  : omega^alpha + omega + k
; we iterate k times to get to Z' which is omega^alpha + omega -- large
```

Theorem: cdr-large-a
(setp (set)
$\wedge$ ordinalp $(alpha)$
$\wedge$ ord-leq $(1,\ alpha)$
$\wedge$ $(0 < k)$
$\wedge$ o-largep $(set$, cons $(alpha$, cons $(1,\ k))))$
$\rightarrow$ o-largep (cdr (set), cons $(alpha$, cons $(1,\ k - 1)))$

DEFINITION:
bad-for-nthcdr-large-a $(set,\ alpha,\ k)$
$=$ (setp (set)
$\land$ ordinalp $(alpha)$
$\land$ ord-leq $(\texttt{1},\ alpha)$
$\land$ $(k \in \mathbf{N})$
$\land$ o-largep $(set,$ cons $(alpha,$ cons $(\texttt{1},\ k)))$
$\land$ $((\neg$ setp (nthcdr $(k,\ set)))$
$\lor$ $(\neg$ o-largep (nthcdr $(k,\ set),$ cons $(alpha,$ cons $(\texttt{1},\ \texttt{0}))))))$

THEOREM: nthcdr-large-a-aux1
$(k \simeq \texttt{0}) \rightarrow (\neg$ bad-for-nthcdr-large-a $(set,\ alpha,\ k))$

THEOREM: nthcdr-large-a-aux2
setp (cons $(z,\ x)) \rightarrow$ setp (x)

THEOREM: nthcdr-large-a-aux3
o-largep $(v,$ cons $(alpha,$ cons $(\texttt{1},\ k))) \rightarrow$ listp (v)

THEOREM: nthcdr-large-a-aux4
$((k \not\simeq \texttt{0}) \land$ bad-for-nthcdr-large-a $(set,\ alpha,\ k))$
$\rightarrow$ bad-for-nthcdr-large-a (cdr $(set),\ alpha,\ k - 1)$

```
; force correct induction
```

DEFINITION:
nthcdr-large-a-kludge $(set,\ alpha,\ k)$
$=$ **if** $\texttt{0} < k$ **then** nthcdr-large-a-kludge (cdr $(set),\ alpha,\ k - 1)$
else $\texttt{0}$ **endif**

THEOREM: nthcdr-large-a-aux5
$\neg$ bad-for-nthcdr-large-a $(set,\ alpha,\ k)$

THEOREM: nthcdr-large-a
(setp (set)
$\land$ ordinalp $(alpha)$
$\land$ ord-leq $(\texttt{1},\ alpha)$
$\land$ $(k \in \mathbf{N})$
$\land$ o-largep $(set,$ cons $(alpha,$ cons $(\texttt{1},\ k))))$
$\rightarrow$ (setp (nthcdr $(k,\ set))$
$\land$ o-largep (nthcdr $(k,\ set),$ cons $(alpha,$ cons $(\texttt{1},\ \texttt{0}))))$

EVENT: Disable nthcdr-large-a-aux1.

EVENT: Disable nthcdr-large-a-aux2.

EVENT: Disable nthcdr-large-a-aux3.

EVENT: Disable nthcdr-large-a-aux4.

EVENT: Disable nthcdr-large-a-aux5.

EVENT: Disable bad-for-nthcdr-large-a.

EVENT: Disable nthcdr-large-a-kludge.

```
; repeat for
; type B  : (cons alpha  k) : k > 0  : omega^alpha +  k
; start with Z whih is omega^alpha +  k -- large
; we iterate k times to get to Z' which is omega^alpha  -- large
```

THEOREM: cdr-large-b
$(\text{setp}\,(set)$
$\land\quad \text{ordinalp}\,(alpha)$
$\land\quad \text{ord-leq}\,(\mathtt{1},\ alpha)$
$\land\quad (\mathtt{0} < k)$
$\land\quad \text{o-largep}\,(set,\ \text{cons}\,(alpha,\ k)))$
$\to\quad \text{o-largep}\,(\text{cdr}\,(set),\ \text{cons}\,(alpha,\ k - 1))$

DEFINITION:
$\text{bad-for-nthcdr-large-b}\,(set,\ alpha,\ k)$
$=\quad (\text{setp}\,(set)$
$\land\quad \text{ordinalp}\,(alpha)$
$\land\quad \text{ord-leq}\,(\mathtt{1},\ alpha)$
$\land\quad (k \in \mathbf{N})$
$\land\quad \text{o-largep}\,(set,\ \text{cons}\,(alpha,\ k))$
$\land\quad ((\neg\ \text{setp}\,(\text{nthcdr}\,(k,\ set)))$
$\lor\quad (\neg\ \text{o-largep}\,(\text{nthcdr}\,(k,\ set),\ \text{cons}\,(alpha,\ \mathtt{0})))))$

THEOREM: nthcdr-large-b-aux1
$(k \simeq \mathtt{0}) \to (\neg\ \text{bad-for-nthcdr-large-b}\,(set,\ alpha,\ k))$

THEOREM: nthcdr-large-b-aux2
$\text{setp}\,(\text{cons}\,(z,\ x)) \to \text{setp}\,(x)$

THEOREM: nthcdr-large-b-aux4
$((k \not\simeq \mathtt{0}) \land \text{bad-for-nthcdr-large-b}\,(set,\ alpha,\ k))$
$\to\quad \text{bad-for-nthcdr-large-b}\,(\text{cdr}\,(set),\ alpha,\ k - 1)$

```
; force correct induction
```

DEFINITION:
nthcdr-large-b-kludge $(set,\ alpha,\ k)$
$=$ **if** $0 < k$ **then** nthcdr-large-b-kludge $(\text{cdr}\,(set),\ alpha,\ k - 1)$
else 0 **endif**

THEOREM: nthcdr-large-b-aux5
$\neg$ bad-for-nthcdr-large-b $(set,\ alpha,\ k)$

THEOREM: nthcdr-large-b
(setp (set)
$\land$ ordinalp $(alpha)$
$\land$ ord-leq $(1,\ alpha)$
$\land$ $(k \in \mathbf{N})$
$\land$ o-largep $(set,\ \text{cons}\,(alpha,\ k)))$
$\rightarrow$ (setp (nthcdr $(k,\ set)$) $\land$ o-largep (nthcdr $(k,\ set)$, cons $(alpha,\ 0)$))

EVENT: Disable nthcdr-large-b-aux1.

EVENT: Disable nthcdr-large-b-aux2.

EVENT: Disable nthcdr-large-b-aux4.

EVENT: Disable nthcdr-large-b-aux5.

EVENT: Disable bad-for-nthcdr-large-b.

EVENT: Disable nthcdr-large-b-kludge.

```
; type C  : (cons alpha (cons 1 0))  : omega^alpha + omega
; start with Z which is omega^alpha + omega -- large
; then its cdr is a set and is omega^alpha + (magic z) + 2 -- large
```

THEOREM: nthcdr-c-1
(setp (set)
$\land$ ordinalp $(alpha)$
$\land$ ord-leq $(1,\ alpha)$
$\land$ o-largep $(set,\ \text{cons}\,(alpha,\ \text{cons}\,(1,\ 0))))$
$\rightarrow$ setp (cdr (set)))

Theorem: nthcdr-c-2
(setp (*set*)
 ∧ ordinalp (*alpha*)
 ∧ ord-leq (1, *alpha*)
 ∧ o-largep (*set*, cons (*alpha*, cons (1, 0))))
→ o-largep (cdr (*set*), cons (*alpha*, magic (car (*set*)) + 2))

```
; now; return to the tail lemma:
; Suppose ZZ is a set which is omega^alpha + omega + d  -- large.
; let z = min ZZ.
; If we take cdr's magic(z+d) times, we get a set QQ such that
; 1. QQ is a set and omega^alpha -- large
; 2. QQ is a subset of ZZ
; 3. min(QQ) >= min(ZZ) + magic(z+d).
; call these parts tail-lemma-1, tail-lemma-2, tail-lemma-3


; First, apply the reductions for types A, C, B in sequence,
; to get an omega^alpha -- large set
```

Theorem: tail-lemma-1-aux1
(setp (*zz*)
 ∧ ordinalp (*alpha*)
 ∧ ord-leq (1, *alpha*)
 ∧ ($d \in \mathbf{N}$)
 ∧ o-largep (*zz*, cons (*alpha*, cons (1, *d*)))
 ∧ (*z1* = nthcdr (*d*, *zz*))
 ∧ (*z2* = cdr (*z1*))
 ∧ (*z3* = nthcdr (magic (car (*z1*)) + 2, *z2*)))
→ (setp (*z3*) ∧ o-largep (*z3*, cons (*alpha*, 0)))

Theorem: tail-lemma-1-aux2
(setp (*zz*)
 ∧ ordinalp (*alpha*)
 ∧ ord-leq (1, *alpha*)
 ∧ ($d \in \mathbf{N}$)
 ∧ o-largep (*zz*, cons (*alpha*, cons (1, *d*)))
 ∧ (*z1* = nthcdr (*d*, *zz*))
 ∧ (*z2* = cdr (*z1*))
 ∧ (*z3* = nthcdr (magic (car (nthcdr (*d*, *zz*))) + 2, cdr (nthcdr (*d*, *zz*)))))
→ (setp (*z3*) ∧ o-largep (*z3*, cons (*alpha*, 0)))

Theorem: tail-lemma-1-aux3
nthcdr (*i*, cdr (*set*)) = nthcdr (1 + *i*, *set*)

THEOREM: tail-lemma-1-aux4
$\text{nthcdr}(j + 2, \text{cdr}(\text{nthcdr}(d, zz))) = \text{nthcdr}(1 + (j + 2), \text{nthcdr}(d, zz))$

THEOREM: tail-lemma-1-aux5
$\text{nthcdr}(j + 2, \text{cdr}(\text{nthcdr}(d, zz))) = \text{nthcdr}(j + 3 + d, zz)$

THEOREM: tail-lemma-1-aux6
$\text{nthcdr}(\text{magic}(\text{car}(\text{nthcdr}(d, zz))) + 2, \text{cdr}(\text{nthcdr}(d, zz)))$
$= \quad \text{nthcdr}(\text{magic}(\text{car}(\text{nthcdr}(d, zz))) + 3 + d, zz)$

THEOREM: tail-lemma-1-aux7
$(\text{setp}(zz)$
$\wedge \quad \text{ordinalp}(alpha)$
$\wedge \quad \text{ord-leq}(1, alpha)$
$\wedge \quad (d \in \mathbf{N})$
$\wedge \quad \text{o-largep}(zz, \text{cons}(alpha, \text{cons}(1, d)))$
$\wedge \quad (z3 = \text{nthcdr}(\text{magic}(\text{car}(\text{nthcdr}(d, zz))) + 3 + d, zz)))$
$\rightarrow \quad (\text{setp}(z3) \wedge \text{o-largep}(z3, \text{cons}(alpha, 0)))$

```
; now, what we want is not Z3, but
; QQ = (nthcdr (magic (plus (car ZZ) d))  ZZ)
; so, we need that QQ is a subset of Z3, which involves
; showing that
;  (magic (plus (car ZZ) d)) <= (plus (magic (car (nthcdr d ZZ))) 3 d)
; magic is monotonic, so we need only:
; (plus (car ZZ) d) <=  (car (nthcdr d ZZ))
```

THEOREM: tail-lemma-1-aux8
$(\text{setp}(zz)$
$\wedge \quad \text{ordinalp}(alpha)$
$\wedge \quad \text{ord-leq}(1, alpha)$
$\wedge \quad (d \in \mathbf{N})$
$\wedge \quad \text{o-largep}(zz, \text{cons}(alpha, \text{cons}(1, d))))$
$\rightarrow \quad \text{listp}(\text{nthcdr}(d, zz))$

THEOREM: tail-lemma-1-aux9
$(\text{setp}(zz)$
$\wedge \quad \text{ordinalp}(alpha)$
$\wedge \quad \text{ord-leq}(1, alpha)$
$\wedge \quad (d \in \mathbf{N})$
$\wedge \quad \text{o-largep}(zz, \text{cons}(alpha, \text{cons}(1, d))))$
$\rightarrow \quad (\text{car}(\text{nthcdr}(d, zz)) \not< (\text{car}(zz) + d))$

THEOREM: tail-lemma-1-aux10

$$(\mathrm{setp}\,(zz) \\ \land\ \mathrm{ordinalp}\,(alpha) \\ \land\ \text{ord-leq}\,(\mathtt{1},\ alpha) \\ \land\ (d \in \mathbf{N}) \\ \land\ \text{o-largep}\,(zz,\ \mathrm{cons}\,(alpha,\ \mathrm{cons}\,(\mathtt{1},\ d)))) \\ \rightarrow\ (\mathrm{magic}\,(\mathrm{car}\,(\mathrm{nthcdr}\,(d,\ zz))) \not< \mathrm{magic}\,(\mathrm{car}\,(zz) + d))$$

THEOREM: tail-lemma-1-aux11

$$(\mathrm{setp}\,(zz) \\ \land\ \mathrm{ordinalp}\,(alpha) \\ \land\ \text{ord-leq}\,(\mathtt{1},\ alpha) \\ \land\ (d \in \mathbf{N}) \\ \land\ \text{o-largep}\,(zz,\ \mathrm{cons}\,(alpha,\ \mathrm{cons}\,(\mathtt{1},\ d)))) \\ \rightarrow\ (\mathrm{magic}\,(\mathrm{car}\,(zz) + d) < (\mathrm{magic}\,(\mathrm{car}\,(\mathrm{nthcdr}\,(d,\ zz))) + \mathtt{3} + d))$$

THEOREM: tail-lemma-1-aux12

$$(i < j) \rightarrow (((j - i) + i) = j)$$

THEOREM: tail-lemma-1-aux13

$$(i < j) \rightarrow \mathrm{subsetp}\,(\mathrm{nthcdr}\,(j,\ zz),\ \mathrm{nthcdr}\,(i,\ zz))$$

THEOREM: tail-lemma-1-aux14

$$(\mathrm{setp}\,(zz) \\ \land\ \mathrm{ordinalp}\,(alpha) \\ \land\ \text{ord-leq}\,(\mathtt{1},\ alpha) \\ \land\ (d \in \mathbf{N}) \\ \land\ \text{o-largep}\,(zz,\ \mathrm{cons}\,(alpha,\ \mathrm{cons}\,(\mathtt{1},\ d))) \\ \land\ (qq = \mathrm{nthcdr}\,(\mathrm{magic}\,(\mathrm{car}\,(zz) + d),\ zz)) \\ \land\ (z3 = \mathrm{nthcdr}\,(\mathrm{magic}\,(\mathrm{car}\,(\mathrm{nthcdr}\,(d,\ zz))) + \mathtt{3} + d,\ zz))) \\ \rightarrow\ \mathrm{subsetp}\,(z3,\ qq)$$

```
; now, QQ will be a set if it's not 0
```

THEOREM: tail-lemma-1-aux15

$$(\mathrm{setp}\,(zz) \\ \land\ \mathrm{ordinalp}\,(alpha) \\ \land\ \text{ord-leq}\,(\mathtt{1},\ alpha) \\ \land\ (d \in \mathbf{N}) \\ \land\ \text{o-largep}\,(zz,\ \mathrm{cons}\,(alpha,\ \mathrm{cons}\,(\mathtt{1},\ d))) \\ \land\ (z3 = \mathrm{nthcdr}\,(\mathrm{magic}\,(\mathrm{car}\,(\mathrm{nthcdr}\,(d,\ zz))) + \mathtt{3} + d,\ zz))) \\ \rightarrow\ \mathrm{listp}\,(z3)$$

```
; all these aux lemmas are getting in our way now -- let's
; just disable them and just quote the ones we need
```

EVENT: Disable tail-lemma-1-aux1.

EVENT: Disable tail-lemma-1-aux2.

EVENT: Disable tail-lemma-1-aux3.

EVENT: Disable tail-lemma-1-aux4.

EVENT: Disable tail-lemma-1-aux5.

EVENT: Disable tail-lemma-1-aux6.

EVENT: Disable tail-lemma-1-aux7.

EVENT: Disable tail-lemma-1-aux8.

EVENT: Disable tail-lemma-1-aux9.

EVENT: Disable tail-lemma-1-aux10.

EVENT: Disable tail-lemma-1-aux11.

EVENT: Disable tail-lemma-1-aux12.

EVENT: Disable tail-lemma-1-aux13.

EVENT: Disable tail-lemma-1-aux14.

EVENT: Disable tail-lemma-1-aux15.

THEOREM: tail-lemma-1-aux16
(setp (zz)
$\land$ ordinalp $(alpha)$
$\land$ ord-leq (1, $alpha$)
$\land$ $(d \in \mathbf{N})$
$\land$ o-largep $(zz$, cons $(alpha$, cons $(1, d)))$

$\wedge$ $(qq = \text{nthcdr}(\text{magic}(\text{car}(zz) + d), zz))$
$\wedge$ $(z3 = \text{nthcdr}(\text{magic}(\text{car}(\text{nthcdr}(d, zz))) + 3 + d, zz)))$
$\rightarrow$ $\text{listp}(qq)$

EVENT: Disable tail-lemma-1-aux16.

THEOREM: tail-lemma-1-aux17
$(\text{setp}(zz)$
$\wedge$ $\text{ordinalp}(alpha)$
$\wedge$ $\text{ord-leq}(1, alpha)$
$\wedge$ $(d \in \mathbf{N})$
$\wedge$ $\text{o-largep}(zz, \text{cons}(alpha, \text{cons}(1, d)))$
$\wedge$ $(qq = \text{nthcdr}(\text{magic}(\text{car}(zz) + d), zz))$
$\wedge$ $(z3 = \text{nthcdr}(\text{magic}(\text{car}(\text{nthcdr}(d, zz))) + 3 + d, zz)))$
$\rightarrow$ $(\text{setp}(qq) \wedge \text{o-largep}(qq, \text{cons}(alpha, 0)))$

THEOREM: tail-lemma-1
$(\text{setp}(zz)$
$\wedge$ $\text{ordinalp}(alpha)$
$\wedge$ $\text{ord-leq}(1, alpha)$
$\wedge$ $(d \in \mathbf{N})$
$\wedge$ $\text{o-largep}(zz, \text{cons}(alpha, \text{cons}(1, d)))$
$\wedge$ $(qq = \text{nthcdr}(\text{magic}(\text{car}(zz) + d), zz)))$
$\rightarrow$ $(\text{setp}(qq) \wedge \text{o-largep}(qq, \text{cons}(alpha, 0)))$

EVENT: Disable tail-lemma-1-aux17.

```
; now return to parts 2 and 3 of the tail-lemma:
; Suppose ZZ is a set which is omega^alpha + omega + d  -- large.
; let z = min ZZ.
; If we take cdr's magic(z+d) times, we get a set QQ such that
; 1. QQ is a set and omega^alpha -- large
; 2. QQ is a subset of ZZ
; 3. min(QQ) >= min(ZZ) + magic(z+d).
; call these parts tail-lemma-1, tail-lemma-2, tail-lemma-3
```

THEOREM: tail-lemma-2
$(\text{setp}(zz)$
$\wedge$ $\text{ordinalp}(alpha)$
$\wedge$ $\text{ord-leq}(1, alpha)$
$\wedge$ $(d \in \mathbf{N})$
$\wedge$ $\text{o-largep}(zz, \text{cons}(alpha, \text{cons}(1, d)))$

$\land$ (qq = nthcdr (magic (car (zz) + d), zz)))
$\rightarrow$ subsetp (qq, zz)

THEOREM: tail-lemma-aux1
(setp (zz)
$\land$ ordinalp (alpha)
$\land$ ord-leq (1, alpha)
$\land$ ($d \in \mathbf{N}$)
$\land$ o-largep (zz, cons (alpha, cons (1, d)))
$\land$ (qq = nthcdr (magic (car (zz) + d), zz)))
$\rightarrow$ listp (qq)

THEOREM: tail-lemma-3
(setp (zz)
$\land$ ordinalp (alpha)
$\land$ ord-leq (1, alpha)
$\land$ ($d \in \mathbf{N}$)
$\land$ o-largep (zz, cons (alpha, cons (1, d)))
$\land$ (qq = nthcdr (magic (car (zz) + d), zz)))
$\rightarrow$ (car (qq) $\not<$ (car (zz) + magic (car (zz) + d)))

EVENT: Disable tail-lemma-aux1.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                      PROPERTIES OF PHI AND STAR                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; phi(alpha, c) = omega^alpha + omega + norm(alpha) + c
```

DEFINITION:
phi (alpha, c)
= **if** ordinalp (alpha) $\land$ ord-leq (1, alpha) $\land$ ($c \in \mathbf{N}$)
then cons (alpha, cons (1, norm (alpha) + c))
else norm (alpha) + norm (alpha) + c + 3 **endif**

THEOREM: phi-is-an-ord
ordinalp (phi (alpha, c))

THEOREM: norm-of-phi
($c \in \mathbf{N}$)
$\rightarrow$ (norm (phi (alpha, c)) = (norm (alpha) + norm (alpha) + c + 3))

DEFINITION:
star (alpha, n)
= **if** $n \simeq 0$ **then** 0
else sharp (alpha, star (alpha, n − 1)) **endif**

THEOREM: star-is-an-ord
$\text{ordinalp}(\mathit{alpha}) \rightarrow \text{ordinalp}(\text{star}(\mathit{alpha},\ n))$

THEOREM: star-with-0
$(n \simeq \texttt{0}) \rightarrow (\text{star}(\mathit{alpha},\ n) = \texttt{0})$

THEOREM: star-with-1
$\text{ordinalp}(\mathit{alpha}) \rightarrow (\text{star}(\mathit{alpha},\ \texttt{1}) = \mathit{alpha})$

THEOREM: norm-of-star
$(\text{ordinalp}(\mathit{alpha}) \land (n \in \mathbf{N}))$
$\rightarrow \quad (\text{norm}(\text{star}(\mathit{alpha},\ n)) = (n * \text{norm}(\mathit{alpha})))$

THEOREM: sharp-with-0-reversed
$\text{ordinalp}(\mathit{alpha}) \rightarrow (\text{sharp}(\mathit{alpha},\ \texttt{0}) = \mathit{alpha})$

THEOREM: car-of-star
$(\text{ordinalp}(\mathit{alpha}) \land (\texttt{0} < n)) \rightarrow (\text{car}(\text{star}(\mathit{alpha},\ n)) = \text{car}(\mathit{alpha}))$

```
;; now we show: "bound-on-phi":
; if alpha < beta , then phi(alpha,c) * n < omega^beta  (for any n)
```

THEOREM: bound-on-phi-aux1
$\text{ordinalp}(\mathit{alpha}) \rightarrow ((\neg\ \text{ord-leq}(\texttt{1},\ \mathit{alpha})) = (\mathit{alpha} = \texttt{0}))$

THEOREM: bound-on-phi-aux2
$\text{ord-lessp}(\text{car}(\mathit{sigma}),\ \text{car}(\mathit{tau})) \rightarrow \text{ord-lessp}(\mathit{sigma},\ \mathit{tau})$

```
; now, we just need to look at the cars:
```

THEOREM: car-of-phi
$(\text{ordinalp}(\mathit{alpha}) \land \text{ord-leq}(\texttt{1},\ \mathit{alpha}) \land (c \in \mathbf{N}))$
$\rightarrow \quad (\text{car}(\text{phi}(\mathit{alpha},\ c)) = \mathit{alpha})$

THEOREM: bound-on-phi-aux3
$(\text{ordinalp}(\mathit{alpha}) \land \text{ord-leq}(\texttt{1},\ \mathit{alpha}) \land (c \in \mathbf{N}) \land (n \not\simeq \texttt{0}))$
$\rightarrow \quad (\text{car}(\text{star}(\text{phi}(\mathit{alpha},\ c),\ n)) = \mathit{alpha})$

THEOREM: bound-on-phi-aux4
$(\text{ordinalp}(\mathit{alpha}) \land \text{ord-leq}(\texttt{1},\ \mathit{alpha}) \land (c \in \mathbf{N}) \land (n \simeq \texttt{0}))$
$\rightarrow \quad (\text{car}(\text{star}(\text{phi}(\mathit{alpha},\ c),\ n)) = \texttt{0})$

THEOREM: bound-on-phi-aux5
$(\text{ordinalp}(\mathit{alpha}) \land (\neg\ \text{ord-leq}(\texttt{1},\ \mathit{alpha})))$
$\rightarrow \quad (\text{car}(\text{star}(\text{phi}(\mathit{alpha},\ c),\ n)) = \texttt{0})$

THEOREM: bound-on-phi
(ordinalp $(alpha)$ $\wedge$ ordinalp $(beta)$ $\wedge$ ord-lessp $(alpha,\ beta)$ $\wedge$ $(c \in \mathbf{N})$)
$\rightarrow$ ord-lessp (star (phi $(alpha,\ c)$, n), cons $(beta,\ 0)$)

EVENT: Disable bound-on-phi-aux1.

EVENT: Disable bound-on-phi-aux2.

EVENT: Disable bound-on-phi-aux3.

EVENT: Disable bound-on-phi-aux4.

EVENT: Disable bound-on-phi-aux5.

```
;;; we plan to apply norm-star-phi with n = c^{2^z}
```

THEOREM: norm-star-phi-aux1
(ordinalp $(alpha)$ $\wedge$ $(n \in \mathbf{N})$ $\wedge$ $(c \in \mathbf{N})$)
$\rightarrow$ (norm (star (phi $(alpha,\ c)$, n))
= $(n *$ (norm $(alpha)$ + norm $(alpha)$ + c + 3)))

```
; revise the format to be consistent with lemma magic-18 in Basic arithmetic
; We need some trivial arithmetic to do this.
```

THEOREM: times-with-zero-1
$(x \simeq 0) \rightarrow ((x * y) = 0)$

THEOREM: times-with-zero-2
$(y \simeq 0) \rightarrow ((x * y) = 0)$

THEOREM: commut-of-times-aux1
$(z \in \mathbf{N}) \rightarrow ((y * (1 + z)) = (y + (z * y)))$

THEOREM: commut-of-times
$(x * y) = (y * x)$

THEOREM: norm-star-phi-aux2
$((x \in \mathbf{N}) \wedge (c \in \mathbf{N}))$
$\rightarrow$ $(((x + x) + (1 + (2 + c))) = (x + x + c + 3))$

THEOREM: norm-star-phi-aux3
$((x \in \mathbf{N}) \wedge (n \in \mathbf{N}) \wedge (c \in \mathbf{N}))$
$\rightarrow$ $((n * (x + x + c + 3)) = (((2 * x) + 3 + c) * n))$

THEOREM: norm-star-phi
$(\text{ordinalp}\,(alpha) \land (n \in \mathbf{N}) \land (c \in \mathbf{N}))$
$\rightarrow \quad (\text{norm}\,(\text{star}\,(\text{phi}\,(alpha,\ c),\ n))$
$\quad = \quad (((2 * \text{norm}\,(alpha)) + 3 + c) * n))$

EVENT: Disable norm-star-phi-aux1.

EVENT: Disable norm-star-phi-aux2.

EVENT: Disable norm-star-phi-aux3.

EVENT: Disable phi.

```
; probably we don't need its defn again
```

```
;;;;; lemma "norm-star-exp-phi"
;; One more arithmetic fact about phi -- to be used
; in the "cdr-lemma" below.

; suppose  alpha-hat = {alpha}(z),  c > 0
; d = norm(alpha) + c, and q >= magic(z+d).
; Then:    norm(phi(alpha-hat, c) * c^(2^z)) <= magic(q).
; reason   norm(alpha-hat) <= norm(alpha) + magic(z)
; so norm(phi(alpha-hat, c) * c^(2^z)) <=
;     2 norm (alpha) + 2 magic(z) + 3 + c  <=
;     magic(q)    (by lemma magic-18)

; first, all we use about alpha-hat is its norm
```

THEOREM: norm-star-exp-phi-aux1
$(alpha\text{-}hat = \text{pred}\,(alpha,\ z))$
$\rightarrow \quad (\text{norm}\,(alpha\text{-}hat) \leq (\text{norm}\,(alpha) + \text{magic}\,(z)))$

```
; second, plug alpha-hat into norm-star-phi
```

THEOREM: norm-star-exp-phi-aux2
$(\text{ordinalp}\,(alpha) \land (n \in \mathbf{N}) \land (c \in \mathbf{N}) \land (alpha\text{-}hat = \text{pred}\,(alpha,\ z)))$
$\rightarrow \quad (\text{norm}\,(\text{star}\,(\text{phi}\,(alpha\text{-}hat,\ c),\ n))$
$\quad = \quad (((2 * \text{norm}\,(alpha\text{-}hat)) + 3 + c) * n))$

```
; now, we apply aux1 to get an inequality
; since this involves a product with n, we need some preliminary
; arithmetic.  Think of
; a = norm(alpha)
; a-hat = norm(alpha-hat)
; nsp = (norm (star (phi alpha-hat c) n ))
; mz = (magic z)
```

THEOREM: norm-star-exp-phi-aux3
$(a\text{-}hat \leq (a + mz))$
$\rightarrow \quad (((2 * a\text{-}hat) + 3 + c)$
$\quad \leq \quad ((2 * a) + (2 * mz) + 3 + c))$

THEOREM: norm-star-exp-phi-aux4
$(a\text{-}hat \leq (a + mz))$
$\rightarrow \quad ((((2 * a\text{-}hat) + 3 + c) * n)$
$\quad \leq \quad (((2 * a) + (2 * mz) + 3 + c) * n))$

THEOREM: norm-star-exp-phi-aux5
$((nsp = (((2 * a\text{-}hat) + 3 + c) * n)) \wedge (a\text{-}hat \leq (a + mz)))$
$\rightarrow \quad (nsp \leq (((2 * a) + (2 * mz) + 3 + c) * n))$

THEOREM: norm-star-exp-phi-aux6
$(\text{ordinalp}\,(alpha) \wedge (n \in \mathbf{N}) \wedge (c \in \mathbf{N}) \wedge (alpha\text{-}hat = \text{pred}\,(alpha,\ z)))$
$\rightarrow \quad (\text{norm}\,(\text{star}\,(\text{phi}\,(alpha\text{-}hat,\ c),\ n))$
$\quad \leq \quad (((2 * \text{norm}\,(alpha)) + (2 * \text{magic}\,(z)) + 3 + c)$
$\quad\quad * \quad n))$

THEOREM: norm-star-exp-phi
$(\text{ordinalp}\,(alpha)$
$\wedge \quad (alpha\text{-}hat = \text{pred}\,(alpha,\ z))$
$\wedge \quad (c \not\simeq 0)$
$\wedge \quad (d = (\text{norm}\,(alpha) + c))$
$\wedge \quad (\text{magic}\,(z + d) \leq q))$
$\rightarrow \quad (\text{norm}\,(\text{star}\,(\text{phi}\,(alpha\text{-}hat,\ c),\ \text{expt}\,(c,\ \text{expt}\,(2,\ z)))) \leq \text{magic}\,(q))$

EVENT: Disable norm-star-exp-phi-aux1.

EVENT: Disable norm-star-exp-phi-aux2.

EVENT: Disable norm-star-exp-phi-aux3.

EVENT: Disable norm-star-exp-phi-aux4.

EVENT: Disable norm-star-exp-phi-aux5.

EVENT: Disable norm-star-exp-phi-aux6.

EVENT: Disable norm-star-phi.

```
; this is firing when we don't want it to
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                              THE CDR LEMMA                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Suppose ZZ is a set which is phi(alpha,c) -- large ; alpha >= 1 ; c >= 1
; let z = min ZZ, and alpha-hat = {alpha}(z)
; then
; 1. (cdr ZZ) is a set
; 2. (cdr ZZ) is phi(alpha-hat,c) * c^(2^z)  --  large
; call these cdr-lemma-1 and cdr-lemma-2

; cdr-lemma-1 should be a triviality -- it just says that ZZ is non-empty
```

THEOREM: cdr-lemma-1
(setp (*zz*) $\land$ o-largep (*zz*, phi (*alpha*, *c*))) $\rightarrow$ setp (cdr (*zz*))

```
; for cdr-lemma-2, let
;       d = norm(alpha) + c
;       QQ = (nthcdr (magic (plus (car ZZ) d)) ZZ )
;        q = (car QQ)
; by defn of phi, ZZ is omega^alpha + omega + d -- large
; then
; QQ is a set and is omega^alpha -- large (by tail-lemma-1)
; q >= z + magic(z+d)  (by tail-lemma 3)
; phi(alpha-hat,c)* c^(2^z) < omega^alpha (by bound-on-phi)
; norm( phi(alpha-hat,c)* c^(2^z)) <= magic(q) (by norm-star-exp-phi)
; QQ is phi(alpha-hat,c) * c^(2^z)  --  large  (by large-with-smaller-ord)
; QQ is a subset of (cdr Z)  (since (magic (plus (car ZZ) d)) >= 1)
; (cdr ZZ) is phi(alpha-hat,c) * c^(2^z)  --  large  (by large-goes-up)

; unwind defn of phi
```

THEOREM: cdr-lemma-aux1
(ordinalp (*alpha*)
 $\land$ ord-leq (1, *alpha*)
 $\land$ ($c \in \mathbf{N}$)
 $\land$ setp (*zz*)
 $\land$ o-largep (*zz*, phi (*alpha*, *c*))
 $\land$ (d = (norm (*alpha*) + c)))
$\rightarrow$ o-largep (*zz*, cons (*alpha*, cons (1, *d*)))

THEOREM: cdr-lemma-aux2
(ordinalp (*alpha*)

$\wedge$ ord-leq (1, *alpha*)
$\wedge$ $(c \in \mathbf{N})$
$\wedge$ setp (*zz*)
$\wedge$ o-largep (*zz*, phi (*alpha*, *c*))
$\wedge$ $(d = (\text{norm}(alpha) + c))$
$\wedge$ $(qq = \text{nthcdr}(\text{magic}(\text{car}(zz) + d), zz)))$
$\rightarrow$ (setp (*qq*) $\wedge$ o-largep (*qq*, cons (*alpha*, 0)))

THEOREM: cdr-lemma-aux3
(ordinalp (*alpha*)
$\wedge$ ord-leq (1, *alpha*)
$\wedge$ $(c \in \mathbf{N})$
$\wedge$ setp (*zz*)
$\wedge$ o-largep (*zz*, phi (*alpha*, *c*))
$\wedge$ $(d = (\text{norm}(alpha) + c))$
$\wedge$ $(qq = \text{nthcdr}(\text{magic}(\text{car}(zz) + d), zz)))$
$\rightarrow$ $((\text{car}(zz) + \text{magic}(\text{car}(zz) + d)) \le \text{car}(qq))$

THEOREM: cdr-lemma-aux4
(ordinalp (*alpha*) $\wedge$ ord-leq (1, *alpha*)) $\rightarrow$ ord-lessp (0, *alpha*)

THEOREM: cdr-lemma-aux5
(ordinalp (*alpha*) $\wedge$ ord-leq (1, *alpha*)) $\rightarrow$ ord-lessp (pred (*alpha*, *z*), *alpha*)

THEOREM: cdr-lemma-aux6
(ordinalp (*alpha*)
$\wedge$ ord-leq (1, *alpha*)
$\wedge$ $(c \in \mathbf{N})$
$\wedge$ (*alpha-hat* = pred (*alpha*, *z*)))
$\rightarrow$ ord-lessp (star (phi (*alpha-hat*, *c*), *n*), cons (*alpha*, 0))

THEOREM: cdr-lemma-aux7
(ordinalp (*alpha*)
$\wedge$ ord-leq (1, *alpha*)
$\wedge$ $(c \not\simeq 0)$
$\wedge$ setp (*zz*)
$\wedge$ (*alpha-hat* = pred (*alpha*, car (*zz*)))
$\wedge$ o-largep (*zz*, phi (*alpha*, *c*))
$\wedge$ $(d = (\text{norm}(alpha) + c))$
$\wedge$ $(qq = \text{nthcdr}(\text{magic}(\text{car}(zz) + d), zz)))$
$\rightarrow$ (norm (star (phi (*alpha-hat*, *c*), expt (*c*, expt (2, car (*zz*)))))
$\le$ magic (car (*qq*)))

```
; the form of large-with-smaller-ord we need:
```

THEOREM: cdr-lemma-aux8
(ordinalp (*sigma*)
 $\wedge$ ordinalp (*tau*)
 $\wedge$ setp (*qq*)
 $\wedge$ o-largep (*qq*, *tau*)
 $\wedge$ ord-lessp (*sigma*, *tau*)
 $\wedge$ (norm (*sigma*) $\leq$ magic (car (*qq*))))
$\rightarrow$ o-largep (*qq*, *sigma*)

THEOREM: cdr-lemma-aux9
(ordinalp (*alpha*)
 $\wedge$ ord-leq (1, *alpha*)
 $\wedge$ ($c \not\simeq 0$)
 $\wedge$ (*alpha-hat* = pred (*alpha*, car (*zz*)))
 $\wedge$ setp (*zz*)
 $\wedge$ o-largep (*zz*, phi (*alpha*, *c*))
 $\wedge$ (d = (norm (*alpha*) + c))
 $\wedge$ (*qq* = nthcdr (magic (car (*zz*) + d), *zz*)))
$\rightarrow$ o-largep (*qq*, star (phi (*alpha-hat*, *c*), expt (*c*, expt (2, car (*zz*)))))

```
; we know that (cdr ZZ) is a set (cdr-lemma-1) and
; that QQ is a set (aux2)
; now we just need that QQ is a subset of (cdr ZZ)
```

THEOREM: cdr-lemma-aux10
(setp (*zz*) $\wedge$ o-largep (*zz*, phi (*alpha*, *c*))) $\rightarrow$ listp (*zz*)

THEOREM: cdr-lemma-aux11
$2 <$ magic (s)

THEOREM: cdr-lemma-aux12
($x \not\simeq 0$) $\rightarrow$ (nthcdr (x, *zz*) = nthcdr ($x - 1$, cdr (*zz*)))

THEOREM: cdr-lemma-aux13
($x \not\simeq 0$) $\rightarrow$ subsetp (nthcdr (x, *zz*), cdr (*zz*))

THEOREM: cdr-lemma-aux14
(setp (*zz*) $\wedge$ (*qq* = nthcdr (magic (u), *zz*))) $\rightarrow$ subsetp (*qq*, cdr (*zz*))

THEOREM: cdr-lemma-aux15
(ordinalp (*alpha*)
 $\wedge$ ord-leq (1, *alpha*)
 $\wedge$ ($c \in \mathbf{N}$)
 $\wedge$ setp (*zz*)
 $\wedge$ o-largep (*zz*, phi (*alpha*, *c*)))
$\rightarrow$ setp (nthcdr (magic (car (*zz*) + (norm (*alpha*) + c)), *zz*))

```
; disable all the aux's now -- the prover doesn't get them right
```

EVENT: Disable cdr-lemma-aux1.

EVENT: Disable cdr-lemma-aux2.

EVENT: Disable cdr-lemma-aux3.

EVENT: Disable cdr-lemma-aux4.

EVENT: Disable cdr-lemma-aux5.

EVENT: Disable cdr-lemma-aux6.

EVENT: Disable cdr-lemma-aux7.

EVENT: Disable cdr-lemma-aux8.

EVENT: Disable cdr-lemma-aux9.

EVENT: Disable cdr-lemma-aux10.

EVENT: Disable cdr-lemma-aux11.

EVENT: Disable cdr-lemma-aux12.

EVENT: Disable cdr-lemma-aux13.

EVENT: Disable cdr-lemma-aux14.

EVENT: Disable cdr-lemma-aux15.

THEOREM: cdr-lemma-2
(ordinalp ($alpha$)
$\land$ ord-leq (1, $alpha$)
$\land$ ($c \not\simeq 0$)

$$\begin{array}{ll} \wedge & (\textit{alpha-hat} = \mathrm{pred}\,(\textit{alpha},\ \mathrm{car}\,(\textit{zz}))) \\ \wedge & \mathrm{setp}\,(\textit{zz}) \\ \wedge & \text{o-largep}\,(\textit{zz},\ \mathrm{phi}\,(\textit{alpha},\ c))) \\ \rightarrow & \text{o-largep}\,(\mathrm{cdr}\,(\textit{zz}),\ \mathrm{star}\,(\mathrm{phi}\,(\textit{alpha-hat},\ c),\ \mathrm{expt}\,(c,\ \mathrm{expt}\,(2,\ \mathrm{car}\,(\textit{zz}))))) \end{array}$$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                       THE PIGEON-HOLE PRINCIPLE                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; We need a pigeon lemma which says:  If c>0,
;  (mapsto g XX lst),   XX is alpha * c -- large, and c >= |lst|
; then there is a  set YY such that YY is a subset of XX,
; YY is alpha -- large and  g is constant on YY.
; YY will be computed as: (extract-pigeon g XX lst alpha c)
; then the lemma has 4 parts:
; "pigeon-A" says that YY is a subset of XX
; "pigeon-B" says that YY is a set
; "pigeon-C" says that g is constant on YY
; "pigeon-D" says that YY is alpha-large


;;;; first, some facts about mapsto and constant

; define "g is constant on lst"
```

DEFINITION:
constantp $(g,\ \mathit{lst})$

$$\begin{array}{ll} = & \textbf{if } \mathit{lst} \simeq \textbf{nil} \textbf{ then t} \\ & \textbf{elseif } \mathrm{cdr}\,(\mathit{lst}) \simeq \textbf{nil} \textbf{ then t} \\ & \textbf{else } (\mathrm{funcall}\,(g,\ \mathrm{car}\,(\mathit{lst})) = \mathrm{funcall}\,(g,\ \mathrm{cadr}\,(\mathit{lst}))) \\ & \qquad \wedge \quad \mathrm{constantp}\,(g,\ \mathrm{cdr}\,(\mathit{lst}))\ \textbf{endif} \end{array}$$

THEOREM: constant-same-value

$$\begin{array}{ll} & (\mathrm{constantp}\,(g,\ \mathit{lst}) \wedge (x \in \mathit{lst}) \wedge (y \in \mathit{lst})) \\ \rightarrow & (\mathrm{funcall}\,(g,\ x) = \mathrm{funcall}\,(g,\ y)) \end{array}$$

```
; constantp is trivially true for lists of length 0 and 1
```

THEOREM: constant-0
$(\mathit{lst} \simeq \textbf{nil}) \rightarrow \mathrm{constantp}\,(g,\ \mathit{lst})$

THEOREM: constant-1
$(\mathrm{cdr}\,(\mathit{lst}) \simeq \textbf{nil}) \rightarrow \mathrm{constantp}\,(g,\ \mathit{lst})$

```
; if not (constantp g lst),  there is a pair of members
; dif1(g,lst) and dif2(g,lst) such that
; g(dif1(g,lst)) != g(dif2(g,lst))
```

Definition:
dif-pair $(g,\ lst)$
$=$ **if** $lst \simeq$ **nil then nil**
elseif cdr $(lst) \simeq$ **nil then nil**
elseif funcall $(g,$ car $(lst)) \neq$ funcall $(g,$ cadr $(lst))$
then cons (car (lst), cadr (lst))
else dif-pair $(g,$ cdr $(lst))$ **endif**

Definition: dif1 $(g,\ lst)$ = car (dif-pair $(g,\ lst)$)

Definition: dif2 $(g,\ lst)$ = cdr (dif-pair $(g,\ lst)$)

Theorem: non-constant-different-values
$(\neg$ constantp $(g,\ lst))$
$\rightarrow$ ((dif1 $(g,\ lst) \in lst$)
$\wedge$ (dif2 $(g,\ lst) \in lst$)
$\wedge$ (funcall $(g,$ dif1 $(g,\ lst)) \neq$ funcall $(g,$ dif2 $(g,\ lst)$))))

Event: Disable constantp.

Event: Disable dif-pair.

Event: Disable dif1.

Event: Disable dif2.

```
; if (mapgst g XX lst) and (length lst) <= 1 then g is constant on XX:
```

Theorem: constant-range-1-aux1
((length $(lst) \leq 1) \wedge (u \in lst) \wedge (v \in lst)) \rightarrow (u = v)$

Theorem: constant-range-1
(mapsto $(g,\ xx,\ lst) \wedge$ (length $(lst) \leq 1)) \rightarrow$ constantp $(g,\ xx)$

Event: Disable constant-range-1-aux1.

```
;;;;; we prove the pigeon-hole principle as follows:

; Suppose (mapsto g XX lst) , and (listp lst)
; there are two subsets of XX:
; AA = (first-part g XX lst) = set of elements of XX which
;   map to the element (car lst)
; BB = (rest-part g XX lst) = set of elements of XX which
;   map into (cdr lst)
; then (covers AA BB XX), g is constant on AA, and maps BB to (cdr lst)
; (star alpha n) is (sharp alpha (star alpha (sub1 n)))
; so we will have that if XX is (star alpha n) -- large
; then either AA is alpha-large or BB is
;       (sharp alpha (star alpha (sub1 n))) -- large

;  AA:
```

DEFINITION:
first-part $(g,\ xx,\ lst)$
$=$ **if** $xx \simeq$ **nil then nil**
elseif funcall $(g,\ \text{car}\,(xx)) = \text{car}\,(lst)$
then cons $(\text{car}\,(xx),\ \text{first-part}\,(g,\ \text{cdr}\,(xx),\ lst))$
else first-part $(g,\ \text{cdr}\,(xx),\ lst)$ **endif**

THEOREM: value-on-first-part
$(x \in \text{first-part}\,(g,\ xx,\ lst)) \rightarrow (\text{funcall}\,(g,\ x) = \text{car}\,(lst))$

THEOREM: constantp-on-first-part
constantp $(g,\ \text{first-part}\,(g,\ xx,\ lst))$

```
; BB:
```

DEFINITION:
rest-part $(g,\ xx,\ lst)$
$=$ **if** $xx \simeq$ **nil then nil**
elseif funcall $(g,\ \text{car}\,(xx)) \in \text{cdr}\,(lst)$
then cons $(\text{car}\,(xx),\ \text{rest-part}\,(g,\ \text{cdr}\,(xx),\ lst))$
else rest-part $(g,\ \text{cdr}\,(xx),\ lst)$ **endif**

THEOREM: values-on-rest-part
$(x \in \text{rest-part}\,(g,\ xx,\ lst)) \rightarrow (\text{funcall}\,(g,\ x) \in \text{cdr}\,(lst))$

THEOREM: mapsto-rest-part
mapsto $(g,\ \text{rest-part}\,(g,\ xx,\ lst),\ \text{cdr}\,(lst))$

```
;; covering properties
```

THEOREM: members-of-first-part
$(\text{mapsto}(g,\ xx,\ lst) \land (\text{funcall}(g,\ x) = \text{car}(lst)) \land (x \in xx))$
$\rightarrow \quad (x \in \text{first-part}(g,\ xx,\ lst))$

THEOREM: members-of-rest-part
$(\text{mapsto}(g,\ xx,\ lst) \land (\text{funcall}(g,\ x) \in \text{cdr}(lst)) \land (x \in xx))$
$\rightarrow \quad (x \in \text{rest-part}(g,\ xx,\ lst))$

THEOREM: first-rest-member
$(\text{mapsto}(g,\ xx,\ lst) \land (x \in xx))$
$\rightarrow \quad ((x \in \text{first-part}(g,\ xx,\ lst)) \lor (x \in \text{rest-part}(g,\ xx,\ lst)))$

THEOREM: first-rest-cover
$\text{mapsto}(g,\ xx,\ lst) \rightarrow \text{covers}(\text{first-part}(g,\ xx,\ lst),\ \text{rest-part}(g,\ xx,\ lst),\ xx)$

```
; we also need that the first-part and the rest-part are subsets
```

THEOREM: first-part-is-sublist
$\text{sublistp}(\text{first-part}(g,\ xx,\ lst),\ xx)$

THEOREM: rest-part-is-sublist
$\text{sublistp}(\text{rest-part}(g,\ xx,\ lst),\ xx)$

THEOREM: first-part-is-subset
$\text{subsetp}(\text{first-part}(g,\ xx,\ lst),\ xx)$

THEOREM: rest-part-is-subset
$\text{subsetp}(\text{rest-part}(g,\ xx,\ lst),\ xx)$

```
; now, prove they are sets
; this is a little inelegant -- we just repeat the same
; argument for first-part and for rest-part

; first-part
```

THEOREM: first-part-is-set-aux1
$(u \in \text{first-part}(g,\ xx,\ lst)) \rightarrow (u \in xx)$

THEOREM: first-part-is-set-aux2
$(u \in \text{first-part}(g,\ \text{cdr}(xx),\ lst)) \rightarrow (u \in \text{cdr}(xx))$

THEOREM: first-part-is-set-aux3
$(\text{setp}(xx) \land (u \in \text{first-part}(g,\ \text{cdr}(xx),\ lst))) \rightarrow (\text{car}(xx) < u)$

THEOREM: first-part-is-set-aux4
$(\text{setp}(xx) \land \text{listp}(\text{first-part}(g,\ \text{cdr}(xx),\ lst)))$
$\rightarrow \quad (\text{car}(xx) < \text{car}(\text{first-part}(g,\ \text{cdr}(xx),\ lst)))$

```
; we need the following trivial set fact:
```

EVENT: Enable setp.

THEOREM: set-builder
$(\text{setp}(s) \land (x \in \mathbf{N}) \land (x < \text{car}(s))) \rightarrow \text{setp}(\text{cons}(x, s))$

EVENT: Disable setp.

THEOREM: first-part-is-set-aux5
$$\begin{aligned}
&(\text{setp}(xx) \\
&\ \land \quad \text{listp}(\text{first-part}(g, \text{cdr}(xx), lst)) \\
&\ \land \quad \text{setp}(\text{first-part}(g, \text{cdr}(xx), lst))) \\
&\rightarrow \quad \text{setp}(\text{first-part}(g, xx, lst))
\end{aligned}$$

THEOREM: first-part-is-set-aux6
$(\text{first-part}(g, xx, lst) \simeq \mathbf{nil}) \rightarrow (\text{first-part}(g, xx, lst) = \mathbf{nil})$

THEOREM: first-part-is-set-aux7
$$\begin{aligned}
&(\text{setp}(xx) \land (\text{first-part}(g, \text{cdr}(xx), lst) \simeq \mathbf{nil})) \\
&\rightarrow \quad \text{setp}(\text{first-part}(g, xx, lst))
\end{aligned}$$

THEOREM: first-part-is-set-aux8
$(\neg\, \text{listp}(xx)) \rightarrow \text{setp}(\text{first-part}(g, xx, lst))$

THEOREM: first-part-is-set-aux9
$$\begin{aligned}
&(\text{setp}(xx) \land \text{setp}(\text{first-part}(g, \text{cdr}(xx), lst))) \\
&\rightarrow \quad \text{setp}(\text{first-part}(g, xx, lst))
\end{aligned}$$

THEOREM: first-part-is-set
$\text{setp}(xx) \rightarrow \text{setp}(\text{first-part}(g, xx, lst))$

EVENT: Disable first-part-is-set-aux1.

EVENT: Disable first-part-is-set-aux2.

EVENT: Disable first-part-is-set-aux3.

EVENT: Disable first-part-is-set-aux4.

EVENT: Disable first-part-is-set-aux5.

EVENT: Disable first-part-is-set-aux6.

EVENT: Disable first-part-is-set-aux7.

EVENT: Disable first-part-is-set-aux8.

EVENT: Disable first-part-is-set-aux9.

```
;;; rest-part
```

THEOREM: rest-part-is-set-aux1
$(u \in \text{rest-part}(g,\ xx,\ lst)) \rightarrow (u \in xx)$

THEOREM: rest-part-is-set-aux2
$(u \in \text{rest-part}(g,\ \text{cdr}(xx),\ lst)) \rightarrow (u \in \text{cdr}(xx))$

THEOREM: rest-part-is-set-aux3
$(\text{setp}(xx) \wedge (u \in \text{rest-part}(g,\ \text{cdr}(xx),\ lst))) \rightarrow (\text{car}(xx) < u)$

THEOREM: rest-part-is-set-aux4
$(\text{setp}(xx) \wedge \text{listp}(\text{rest-part}(g,\ \text{cdr}(xx),\ lst)))$
$\rightarrow \quad (\text{car}(xx) < \text{car}(\text{rest-part}(g,\ \text{cdr}(xx),\ lst)))$

THEOREM: rest-part-is-set-aux5
$(\text{setp}(xx)$
$\wedge \quad \text{listp}(\text{rest-part}(g,\ \text{cdr}(xx),\ lst))$
$\wedge \quad \text{setp}(\text{rest-part}(g,\ \text{cdr}(xx),\ lst)))$
$\rightarrow \quad \text{setp}(\text{rest-part}(g,\ xx,\ lst))$

THEOREM: rest-part-is-set-aux6
$(\text{rest-part}(g,\ xx,\ lst) \simeq \textbf{nil}) \rightarrow (\text{rest-part}(g,\ xx,\ lst) = \textbf{nil})$

```
; a trivial set fact:
```

THEOREM: singleton-set
$(m \in \mathbf{N}) \rightarrow \text{setp}(\text{list}(m))$

THEOREM: rest-part-is-set-aux7
$(\text{setp}(xx) \wedge (\text{rest-part}(g,\ \text{cdr}(xx),\ lst) \simeq \textbf{nil})) \rightarrow \text{setp}(\text{rest-part}(g,\ xx,\ lst))$

THEOREM: rest-part-is-set-aux8
$(\neg\ \text{listp}(xx)) \rightarrow \text{setp}(\text{rest-part}(g,\ xx,\ lst))$

THEOREM: rest-part-is-set-aux9
$(\text{setp}(xx) \wedge \text{setp}(\text{rest-part}(g,\ \text{cdr}(xx),\ lst))) \rightarrow \text{setp}(\text{rest-part}(g,\ xx,\ lst))$

THEOREM: rest-part-is-set
setp $(xx) \rightarrow$ setp (rest-part $(g,\ xx,\ lst)$)

EVENT: Disable rest-part-is-set-aux1.

EVENT: Disable rest-part-is-set-aux2.

EVENT: Disable rest-part-is-set-aux3.

EVENT: Disable rest-part-is-set-aux4.

EVENT: Disable rest-part-is-set-aux5.

EVENT: Disable rest-part-is-set-aux6.

EVENT: Disable rest-part-is-set-aux7.

EVENT: Disable rest-part-is-set-aux8.

EVENT: Disable rest-part-is-set-aux9.

```
;;;;;; now, back to the pigeon-hole principle:
; if (mapsto g XX lst) and XX is alpha * c -- large, and c >= |lst|
; and c > 0,
; then there is a YY subset XX such that YY is alpha -- large
; and g is constant on YY.

; YY will be (extract-pigeon g XX lst alpha c)
```

DEFINITION:
extract-pigeon $(g,\ xx,\ lst,\ alpha,\ c)$
= **if** $c \leq 1$ **then** xx
elseif o-largep (first-part $(g,\ xx,\ lst)$, $alpha$) **then** first-part $(g,\ xx,\ lst)$
else extract-pigeon (g, rest-part $(g,\ xx,\ lst)$, cdr (lst), $alpha$, $c - 1$) **endif**

```
; "pigeon-A" says that YY is a subset of XX
; "pigeon-B" says that YY is a set
; "pigeon-C" says that g is constant on YY
; "pigeon-D" says that YY is alpha-large
```

THEOREM: pigeon-a
subsetp (extract-pigeon (g, xx, lst, $alpha$, c), xx)

THEOREM: pigeon-b
setp (xx) $\rightarrow$ setp (extract-pigeon (g, xx, lst, $alpha$, c))

THEOREM: pigeon-c-aux1
(mapsto (g, xx, lst) $\land$ (length (lst) $\leq$ c) $\land$ ($c \leq$ 1))
$\rightarrow$ constantp (g, extract-pigeon (g, xx, lst, $alpha$, c))

THEOREM: pigeon-c-aux2
((1 $<$ c) $\land$ ($\neg$ o-largep (first-part (g, xx, lst), $alpha$)))
$\rightarrow$ (extract-pigeon (g, xx, lst, $alpha$, c)
= extract-pigeon (g, rest-part (g, xx, lst), cdr (lst), $alpha$, $c - 1$))

THEOREM: pigeon-c
(mapsto (g, xx, lst) $\land$ (length (lst) $\leq$ c))
$\rightarrow$ constantp (g, extract-pigeon (g, xx, lst, $alpha$, c))

EVENT: Disable pigeon-c-aux1.

EVENT: Disable pigeon-c-aux2.

```
; now, for D, we have to induct on c
```

DEFINITION:
bad-for-pigeon-d (g, xx, lst, $alpha$, c)
= (ordinalp ($alpha$)
$\land$ setp (xx)
$\land$ o-largep (xx, star ($alpha$, c))
$\land$ mapsto (g, xx, lst)
$\land$ (length (lst) $\leq$ c)
$\land$ ($c \not\simeq 0$)
$\land$ ($\neg$ o-largep (extract-pigeon (g, xx, lst, $alpha$, c), $alpha$)))

```
; basis:
```

THEOREM: pigeon-d-aux1
($c \simeq 0$) $\rightarrow$ ($\neg$ bad-for-pigeon-d (g, xx, lst, $alpha$, c))

THEOREM: pigeon-d-aux2
$\neg$ bad-for-pigeon-d (g, xx, lst, $alpha$, 1)

```
; now we have to work on the induction step, where c > 1:
```

THEOREM: pigeon-d-aux3
$((1 < c) \land \text{bad-for-pigeon-d}\,(g,\ xx,\ lst,\ alpha,\ c))$
$\rightarrow \quad (\neg\ \text{o-largep}\,(\text{first-part}\,(g,\ xx,\ lst),\ alpha))$

THEOREM: pigeon-d-aux4
$((1 < c) \land \text{bad-for-pigeon-d}\,(g,\ xx,\ lst,\ alpha,\ c))$
$\rightarrow \quad (\text{extract-pigeon}\,(g,\ xx,\ lst,\ alpha,\ c)$
$\quad = \quad \text{extract-pigeon}\,(g,\ \text{rest-part}\,(g,\ xx,\ lst),\ \text{cdr}\,(lst),\ alpha,\ c - 1))$

THEOREM: pigeon-d-aux5
$(\text{ordinalp}\,(alpha)$
$\land \quad (1 < c)$
$\land \quad \text{setp}\,(a)$
$\land \quad \text{setp}\,(b)$
$\land \quad \text{setp}\,(x)$
$\land \quad \text{covers}\,(a,\ b,\ x)$
$\land \quad \text{o-largep}\,(x,\ \text{star}\,(alpha,\ c))$
$\land \quad (\neg\ \text{o-largep}\,(a,\ alpha)))$
$\rightarrow \quad \text{o-largep}\,(b,\ \text{star}\,(alpha,\ c - 1))$

THEOREM: pigeon-d-aux6
$(\text{ordinalp}\,(alpha)$
$\land \quad \text{mapsto}\,(g,\ xx,\ lst)$
$\land \quad (1 < c)$
$\land \quad \text{setp}\,(xx)$
$\land \quad \text{o-largep}\,(xx,\ \text{star}\,(alpha,\ c))$
$\land \quad (\neg\ \text{o-largep}\,(\text{first-part}\,(g,\ xx,\ lst),\ alpha)))$
$\rightarrow \quad \text{o-largep}\,(\text{rest-part}\,(g,\ xx,\ lst),\ \text{star}\,(alpha,\ c - 1))$

```
; induction step:
```

THEOREM: pigeon-d-aux7
$((1 < c) \land \text{listp}\,(lst) \land \text{bad-for-pigeon-d}\,(g,\ xx,\ lst,\ alpha,\ c))$
$\rightarrow \quad \text{bad-for-pigeon-d}\,(g,\ \text{rest-part}\,(g,\ xx,\ lst),\ \text{cdr}\,(lst),\ alpha,\ c - 1)$

```
; now, what if lst is empty
```

THEOREM: pigeon-d-aux8
$(\text{mapsto}\,(g,\ xx,\ lst) \land (\neg\ \text{listp}\,(lst))) \rightarrow (\neg\ \text{listp}\,(xx))$

THEOREM: pigeon-d-aux9
$(\text{ordinalp}\,(alpha) \land (alpha \neq 0) \land (c \not\simeq 0)) \rightarrow (\text{star}\,(alpha,\ c) \neq 0)$

THEOREM: pigeon-d-aux10
$(\text{ordinalp}\,(alpha) \land (c \not\simeq 0) \land \text{o-largep}\,(xx,\ \text{star}\,(alpha,\ c)) \land (xx \simeq \textbf{nil}))$
$\rightarrow \quad (alpha = 0)$

THEOREM: pigeon-d-aux11
$((c \not\simeq 0) \land (lst \simeq \mathbf{nil})) \rightarrow (\neg \text{ bad-for-pigeon-d}\,(g,\ xx,\ lst,\ alpha,\ c))$

```
; revised induction step:
```

THEOREM: pigeon-d-aux12
$((1 < c) \land \text{bad-for-pigeon-d}\,(g,\ xx,\ lst,\ alpha,\ c))$
$\rightarrow \quad \text{bad-for-pigeon-d}\,(g,\ \text{rest-part}\,(g,\ xx,\ lst),\ \text{cdr}\,(lst),\ alpha,\ c - 1)$

```
; rephrase the  basis
```

THEOREM: pigeon-d-aux13
$(1 \not< c) \rightarrow (\neg \text{ bad-for-pigeon-d}\,(g,\ xx,\ lst,\ alpha,\ c))$

THEOREM: pigeon-d-aux14
$\neg \text{ bad-for-pigeon-d}\,(g,\ xx,\ lst,\ alpha,\ c)$

THEOREM: pigeon-d
$(\text{ordinalp}\,(alpha)$
$\land \quad \text{setp}\,(xx)$
$\land \quad \text{o-largep}\,(xx,\ \text{star}\,(alpha,\ c))$
$\land \quad \text{mapsto}\,(g,\ xx,\ lst)$
$\land \quad (\text{length}\,(lst) \leq c)$
$\land \quad (c \not\simeq 0))$
$\rightarrow \quad \text{o-largep}\,(\text{extract-pigeon}\,(g,\ xx,\ lst,\ alpha,\ c),\ alpha)$

EVENT: Disable bad-for-pigeon-d.

EVENT: Disable pigeon-d-aux1.

EVENT: Disable pigeon-d-aux2.

EVENT: Disable pigeon-d-aux3.

EVENT: Disable pigeon-d-aux4.

EVENT: Disable pigeon-d-aux5.

EVENT: Disable pigeon-d-aux6.

EVENT: Disable pigeon-d-aux7.

EVENT: Disable pigeon-d-aux8.

EVENT: Disable pigeon-d-aux9.

EVENT: Disable pigeon-d-aux10.

EVENT: Disable pigeon-d-aux11.

EVENT: Disable pigeon-d-aux12.

EVENT: Disable pigeon-d-aux13.

EVENT: Disable pigeon-d-aux14.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                 RANGES                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; We follow the usual set-theoretic convention that a partition into
; c pieces is just a map into c = {0 ... c-1}.
; when we work with partitions g on tuples from a set XX --
; it will be simpler to just assume that g of anything is < c
; Since our definition has (funcall g x) = 0 whenever x is
; outside the natural range of g, this will not cause any problem

; first, let's define the intended domain of g
```

DEFINITION:
domain (g)
$=$ **if** $g \simeq$ **nil then nil**
elseif listp (car (g)) **then** cons (caar (g), domain (cdr (g)))
else domain (cdr (g)) **endif**

```
; even for 0, if 0 is not in (domain g), then (assoc 0 g) is an
; atom, so its cdr is 0
```

THEOREM: assoc-outside-domain
$(x \notin \text{domain}\,(g)) \rightarrow (\text{cdr}\,(\text{assoc}\,(x,\ g)) = 0)$

EVENT: Enable funcall.

THEOREM: funcall-outside-domain
$(x \notin \text{domain}(g)) \rightarrow (\text{funcall}(g,\ x) = \mathtt{0})$

EVENT: Disable funcall.

DEFINITION:
rangep $(g,\ c)$
$= \quad$ **if** $g \simeq$ **nil** **then** $\mathtt{0} < c$
else rangep $(\text{cdr}(g),\ c) \land (\text{cadar}(g) \in \mathbf{N}) \land (\text{cadar}(g) < c)$ **endif**

THEOREM: rangep-is-positive
$\text{rangep}(g,\ c) \rightarrow ((\mathtt{0} < c) = \mathbf{t})$

THEOREM: rangep-bounds
$\text{rangep}(g,\ c) \rightarrow (\text{funcall}(g,\ x) < c)$

THEOREM: rangep-numbers
$\text{rangep}(g,\ c) \rightarrow (\text{funcall}(g,\ x) \in \mathbf{N})$

EVENT: Disable rangep.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                              SEGMENTS                                ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;; we need some more on segments now, since (segment 0 c) will
;; be used as the range of a function
```

EVENT: Enable segment.

THEOREM: segments-non-empty
$((m \leq n) \land (m \in \mathbf{N}) \land (n \in \mathbf{N})) \rightarrow \text{listp}(\text{segment}(m,\ n))$

THEOREM: car-of-segment
$\text{listp}(\text{segment}(m,\ n)) \rightarrow (\text{car}(\text{segment}(m,\ n)) = m)$

THEOREM: cdr-of-segment
$\text{listp}(\text{segment}(m,\ n)) \rightarrow (\text{cdr}(\text{segment}(m,\ n)) = \text{segment}(1 + m,\ n))$

THEOREM: empty-segment
$((m \notin \mathbf{N}) \lor (n \notin \mathbf{N}) \lor (n < m)) \rightarrow (\text{segment}(m,\ n) = \mathbf{nil})$

THEOREM: non-list-segment
$(\neg\ \text{listp}(\text{segment}(m,\ n))) \rightarrow (\text{segment}(m,\ n) = \mathbf{nil})$

THEOREM: recursive-case-for-segment
$((m \leq n) \land (m \in \mathbf{N}) \land (n \in \mathbf{N}))$
$\rightarrow \quad (\text{segment}\,(m,\ n) = \text{cons}\,(m, \text{segment}\,(1 +\ m,\ n)))$

EVENT: Disable segment.

THEOREM: singleton-segment
$(m \in \mathbf{N}) \rightarrow (\text{segment}\,(m,\ m) = \text{list}\,(m))$

THEOREM: cadr-of-segment
$\text{listp}\,(\text{cdr}\,(\text{segment}\,(m,\ n))) \rightarrow (\text{cadr}\,(\text{segment}\,(m,\ n)) = (1 +\ m))$

```
; let's prove segments are sets -- this is a bit of a pain
```

DEFINITION:
$\text{bad-for-segments-are-sets}\,(m,\ n) = (\neg\ \text{setp}\,(\text{segment}\,(m,\ n)))$

THEOREM: segments-are-sets-aux1
$\text{bad-for-segments-are-sets}\,(m,\ n) \rightarrow ((m \in \mathbf{N}) \land (n \in \mathbf{N}) \land (m \leq n))$

THEOREM: segments-are-sets-aux2
$(m \in \mathbf{N}) \rightarrow (\neg\ \text{bad-for-segments-are-sets}\,(m,\ m))$

THEOREM: segments-are-sets-aux3
$(\neg\ ((m \in \mathbf{N}) \land (n \in \mathbf{N}) \land (m < n)))$
$\rightarrow \quad (\neg\ \text{bad-for-segments-are-sets}\,(m,\ n))$

THEOREM: segments-are-sets
$\text{setp}\,(\text{segment}\,(m,\ n))$

EVENT: Disable segments-are-sets-aux1.

EVENT: Disable segments-are-sets-aux2.

EVENT: Disable segments-are-sets-aux3.

EVENT: Disable bad-for-segments-are-sets.

```
; another trivial segment fact:
```

THEOREM: first-of-segment
$((m \in \mathbf{N}) \land (n \in \mathbf{N}) \land (m \leq n)) \rightarrow (\text{car}\,(\text{segment}\,(m,\ n)) = m)$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                       BASIS OF RAMSEY THEOREM                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; This will be "ramsey-basis" -- the Ramsey theorem for 1-tuples.
; Suppose (rangep g c) (so c > 0), XX is a set which is alpha * c -- large.
; Then there is a  set YY such that YY is a subset of XX,
; YY is alpha -- large,  g is constant on the singletons of YY --
; that is:  (homp YY g 1)
; the proof is to define an auxilliary function h -- so that
; (funcall h x) = (funcall g (list x)),
; and then apply the pigeon-hole principle

; call this h "(sing-fn g)"
; this will be (sing-fn-aux g (domain g))
; if h = (sing-fn-aux g lst)  and (list x) is in lst, then
; (funcall h x) = (funcall g (list x))

; then YY will be (extract-ramsey-basis g XX alpha c) =
; (extract-pigeon (sing-fn g) XX (segment 0 c) alpha c)
```

Definition:
sing-fn-aux $(g,\ lst)$
$=$ **if** $lst \simeq$ **nil then nil**
elseif car $(lst) =$ list (caar (lst))
then cons (list (caar (lst), funcall $(g,$ car $(lst))$), sing-fn-aux $(g,$ cdr $(lst))$)
else sing-fn-aux $(g,$ cdr $(lst))$ **endif**

Theorem: sing-fn-aux-1
(list $(x) \in lst$)
$\rightarrow$ (assoc $(x,$ sing-fn-aux $(g,\ lst)) =$ list $(x,$ funcall $(g,$ list $(x))))$

Theorem: sing-fn-aux-2
(list $(x) \notin lst) \rightarrow (\neg$ listp (assoc $(x,$ sing-fn-aux $(g,\ lst))))$

Event: Enable funcall.

Theorem: sing-fn-aux-3
(list $(x) \in lst$)
$\rightarrow$ (funcall (sing-fn-aux $(g,\ lst),\ x) =$ funcall $(g,$ list $(x)))$

Theorem: sing-fn-aux-4
(list $(x) \notin lst) \rightarrow$ (funcall (sing-fn-aux $(g,\ lst),\ x) = 0)$

EVENT: Disable funcall.

DEFINITION: $\text{sing-fn}\,(g) = \text{sing-fn-aux}\,(g,\ \text{domain}\,(g))$

THEOREM: sing-fn-on-sings
$\text{funcall}\,(\text{sing-fn}\,(g),\ x) = \text{funcall}\,(g,\ \text{list}\,(x))$

EVENT: Disable sing-fn-aux.

EVENT: Disable sing-fn-aux-1.

EVENT: Disable sing-fn-aux-2.

EVENT: Disable sing-fn-aux-3.

EVENT: Disable sing-fn-aux-4.

EVENT: Disable sing-fn.

```
; all we need is its properties

; now, if (rangep g c) (so c > 0) , then
; g takes anything to (segment 0 (sub1 c)) = {0 ... c-1} = c
; this is a list of length c, so the pigeon hole principle
; should apply.

; as preliminaries
```

THEOREM: size-of-initial-segment
$(0 < c) \rightarrow (\text{length}\,(\text{segment}\,(0,\ c - 1)) = c)$

THEOREM: members-of-initial-segment
$(0 < c) \rightarrow ((x \in \text{segment}\,(0,\ c - 1)) = ((x \in \mathbf{N}) \wedge (x < c)))$

THEOREM: bound-values-of-sing-fn
$\text{rangep}\,(g,\ c) \rightarrow (\text{funcall}\,(\text{sing-fn}\,(g),\ x) < c)$

THEOREM: number-values-of-sing-fn
$\text{rangep}\,(g,\ c) \rightarrow (\text{funcall}\,(\text{sing-fn}\,(g),\ x) \in \mathbf{N})$

THEOREM: sing-fn-mapsto-aux1
$\text{rangep}\,(g,\ c) \rightarrow (\text{funcall}\,(\text{sing-fn}\,(g),\ x) \in \text{segment}\,(0,\ c - 1))$

THEOREM: sing-fn-mapsto
rangep$(g,\ c) \rightarrow$ mapsto (sing-fn (g), xx, segment $(0,\ c-1))$

EVENT: Disable sing-fn-mapsto-aux1.

```
;; now, we are set up for a proof of the basis of Ramsey's theorem

; the YY that works is given by
```

DEFINITION:
extract-ramsey-basis $(g,\ xx,\ alpha,\ c)$
$=$ extract-pigeon (sing-fn (g), xx, segment $(0,\ c-1)$, $alpha$, $c)$

```
; We're assuming
; (rangep g c) (so c > 0), XX is a set which is alpha * c -- large.

; "ramsey-basis-A" says that YY is a subset of XX
; "ramsey-basis-B" says that YY is a set
; "ramsey-basis-C" says that (homp YY g 1)
; "ramsey-basis-D" says that YY is alpha-large
```

THEOREM: ramsey-basis-a
subsetp (extract-ramsey-basis $(g,\ xx,\ alpha,\ c)$, $xx)$

THEOREM: ramsey-basis-b
setp $(xx) \rightarrow$ setp (extract-ramsey-basis $(g,\ xx,\ alpha,\ c))$

```
; now to prove (homp YY g 1) --
; we need to prove that
; (constantp (sing-fn g) YY) implies (homp YY g 1)
; this is true because we've proved if (not ((homp YY g n)))
; we have an explicit counter-example
; first, let's recast this counter-example in the case n = 1
```

THEOREM: ramsey-basis-c-aux1
sublistp (list (x), $set) \rightarrow (x \in set)$

DEFINITION:
counter-hom-aux-x $(set,\ g) =$ car (counter-hom-x $(set,\ g,\ 1))$

DEFINITION:
counter-hom-aux-y $(set,\ g) =$ car (counter-hom-y $(set,\ g,\ 1))$

THEOREM: ramsey-basis-c-aux2
$(\text{setp}(\mathit{lst}) \land (\text{length}(\mathit{lst}) = \texttt{1})) \rightarrow (\text{list}(\text{car}(\mathit{lst})) = \mathit{lst})$

THEOREM: ramsey-basis-c-aux3
$(\neg\ \text{homp}(\mathit{set},\ g,\ \texttt{1}))$
$\rightarrow\quad (\text{counter-hom-x}(\mathit{set},\ g,\ \texttt{1}) = \text{list}(\text{counter-hom-aux-x}(\mathit{set},\ g)))$

THEOREM: ramsey-basis-c-aux4
$(\neg\ \text{homp}(\mathit{set},\ g,\ \texttt{1}))$
$\rightarrow\quad (\text{counter-hom-y}(\mathit{set},\ g,\ \texttt{1}) = \text{list}(\text{counter-hom-aux-y}(\mathit{set},\ g)))$

THEOREM: ramsey-basis-c-aux5
$(\neg\ \text{homp}(\mathit{set},\ g,\ \texttt{1}))$
$\rightarrow\quad ((\text{counter-hom-aux-x}(\mathit{set},\ g) \in \mathit{set})$
$\quad\land\quad (\text{counter-hom-aux-y}(\mathit{set},\ g) \in \mathit{set})$
$\quad\land\quad (\text{funcall}(\text{sing-fn}(g),\ \text{counter-hom-aux-x}(\mathit{set},\ g))$
$\quad\quad\neq\quad \text{funcall}(\text{sing-fn}(g),\ \text{counter-hom-aux-y}(\mathit{set},\ g))))$

THEOREM: ramsey-basis-c-aux6
$\text{constantp}(\text{sing-fn}(g),\ \mathit{set}) \rightarrow \text{homp}(\mathit{set},\ g,\ \texttt{1})$

THEOREM: ramsey-basis-c
$\text{rangep}(g,\ c) \rightarrow \text{homp}(\text{extract-ramsey-basis}(g,\ \mathit{xx},\ \mathit{alpha},\ c),\ g,\ \texttt{1})$

EVENT: Disable counter-hom-aux-x.

EVENT: Disable counter-hom-aux-y.

EVENT: Disable ramsey-basis-c-aux1.

EVENT: Disable ramsey-basis-c-aux2.

EVENT: Disable ramsey-basis-c-aux3.

EVENT: Disable ramsey-basis-c-aux4.

EVENT: Disable ramsey-basis-c-aux5.

EVENT: Disable ramsey-basis-c-aux6.

THEOREM: ramsey-basis-d
$(\text{ordinalp}(\mathit{alpha}) \land \text{setp}(\mathit{xx}) \land \text{o-largep}(\mathit{xx},\ \text{star}(\mathit{alpha},\ c)) \land \text{rangep}(g,\ c))$
$\rightarrow\quad \text{o-largep}(\text{extract-ramsey-basis}(g,\ \mathit{xx},\ \mathit{alpha},\ c),\ \mathit{alpha})$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                      MISCELLANEOUS CONSTRUCTIONS                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;  vn

; first, to make our notation more compact:
; Look a number z as a von Neumann ordinal --
; the set of all smaller numbers.
; to make this a standard set, we list them in increasing order
```

DEFINITION:
vn (z)
$=$ **if** $z \simeq 0$ **then nil**
else add-on-end ($z - 1$, vn ($z - 1$)) **endif**

THEOREM: size-of-vn
($z \in \mathbf{N}$) $\rightarrow$ (length (vn (z)) $=$ z)

THEOREM: last-of-vn
($z \not\simeq 0$) $\rightarrow$ (last (vn (z)) $=$ ($z - 1$))

THEOREM: vn-is-a-set-aux1
(($1 < z$) $\wedge$ setp (vn ($z - 1$))) $\rightarrow$ setp (vn (z))

THEOREM: vn-is-a-set-aux2
($z \simeq 0$) $\rightarrow$ setp (vn (z))

THEOREM: vn-is-a-set-aux3
setp (vn (1))

THEOREM: vn-is-a-set
setp (vn (z))

EVENT: Disable vn-is-a-set-aux1.

EVENT: Disable vn-is-a-set-aux2.

EVENT: Disable vn-is-a-set-aux3.

```
; to prove that the members of (vn z) are the numbers below z
; we have to say what the members of add-on-end are
```

THEOREM: members-of-add-on-end
$(x \in \text{add-on-end}(item, lst)) = ((x \in lst) \lor (x = item))$

THEOREM: members-of-vn-aux1
$(z \simeq 0) \rightarrow (x \notin \text{vn}(z))$

THEOREM: members-of-vn-aux2
$(z \not\simeq 0) \rightarrow ((x \in \text{vn}(z)) = ((x \in \text{vn}(z - 1)) \lor (x = (z - 1))))$

EVENT: Disable vn.

```
; probably, we will never need its defn
```

THEOREM: members-of-vn
$(x \in \text{vn}(z)) = ((x \in \mathbf{N}) \land (x < z))$

EVENT: Disable members-of-vn-aux1.

EVENT: Disable members-of-vn-aux2.

```
;;;;;;;;;;;;;   all-maps
;;; let c>0

; (standard-fn g lst c) says that g is a function in standard
;  form from lst into c.   If lst = (x_1 ... x_n . foo), then
; g should be ( (x_1 v_1) ... (x_n v_n))
; we will build these in the induction step for Ramsey's theorem.
; now, we just want to prove that there are c^(length lst) of these

; we define (all-maps lst c), show that it contains all standard
; functions, and show that its size is c^(length lst)
```

DEFINITION:
standard-function (g, lst, c)
$=$ **if** $lst \simeq$ **nil then** $g =$ **nil**
else standard-function (cdr (g), cdr (lst), c)
$\land$ listp (car (g))
$\land$ (car (g) = list (car (lst), cadar (g)))
$\land$ ((cadar $(g) \in \mathbf{N}) \land$ (cadar $(g) < c$)) **endif**

```
; in an r-loop
; *(standard-function '((a 3) (b 1)) '(a b) 4 )
```

```
;  T
; *(standard-function '((a 3) (b 1)) '(a b) 3 )
;  F
; *(standard-function '((a 3) (b 1)) '(a b c) 5 )
;  F
```

THEOREM: tail-of-standard-function
standard-function $(g,\ \text{cons}\,(x,\ lst),\ c) \rightarrow$ standard-function $(\text{cdr}\,(g),\ lst,\ c)$

```
; to build up (all-maps lst c):
; suppose g is a standard fn from (cons x lst) to c
; and (cdr g) is a member of a family FF
; then g is of the form (cons (list x i) h) for some i < c
; there are exactly c possibilites for (cons x i)

; we define (extend-family FF x c) to be all such g --
; then this will have length = c * |FF|
```

DEFINITION:
list-all $(x,\ c)$
$=$ **if** $c \simeq 0$ **then nil**
else cons (list $(x,\ c-1)$, list-all $(x,\ c-1)$) **endif**

THEOREM: list-all-gets-all
$((i \in \mathbf{N}) \wedge (i < c)) \rightarrow (\text{list}\,(x,\ i) \in \text{list-all}\,(x,\ c))$

```
; the list of all (list x i) ; with i < c
```

THEOREM: size-of-list-all
$(c \in \mathbf{N}) \rightarrow (\text{length}\,(\text{list-all}\,(x,\ c)) = c)$

DEFINITION:
all-maps $(lst,\ c)$
$=$ **if** $lst \simeq$ **nil then** `'(nil)`
else product (list-all (car (lst), c), all-maps (cdr (lst), c)) **endif**

THEOREM: all-maps-gets-all
standard-function $(g,\ lst,\ c) \rightarrow (g \in \text{all-maps}\,(lst,\ c))$

```
; now, to compute the size of (all-maps lst c)
; we first need to compute the size of the product of two lists
```

EVENT: Enable product.

THEOREM: size-of-product
$\text{length}\,(\text{product}\,(\mathit{lst1},\ \mathit{lst2})) = (\text{length}\,(\mathit{lst1}) * \text{length}\,(\mathit{lst2}))$

EVENT: Disable product.

THEOREM: size-of-all-maps
$(c \in \mathbf{N}) \rightarrow (\text{length}\,(\text{all-maps}\,(\mathit{lst},\ c)) = \text{expt}\,(c,\ \text{length}\,(\mathit{lst})))$

```
;;;;;;;  constructing standard functions
; If OP is any defined function, we may construct a standard fn from
; lst to c by running down lst and applying OP
; Since we can't quantify over all defined functions without using
; something like $eval, we just do this for the one OP needed
; in the proof of Ramsey's theorem.

; Suppose (rangep g c) ; think of g as a partition of tuples
; we are trying to get a homogeneous set for.
; In an attempt to construct a pre-homogeneous set, we fix
; a z in omega and let lst be the power set of z  (so |lst| = 2^z)
; for a y > z, we consider g to define a map lst --> c
; which takes a set S to g(S union {z,y})
; we can call this new function (power-map g z y), and
; later try to get a large set of y's for which (power-map g z y) is the same.

; (power-map g z y)  will be (augmented-map g z y (power-set (vn z)))

; first, let's define (augmented-map g z y lst)
```

DEFINITION:
$\text{end-end}\,(x,\ g,\ z,\ y) = \text{funcall}\,(g,\ \text{add-on-end}\,(y,\ \text{add-on-end}\,(z,\ x)))$

```
; i.e., we're expecting x subset z < y, so we apply the partition
; g to x union {z,y}

; for now, we only care that this defines a function of x,
; with g,z,y as parameters, and that the values are < c
; if g has range c
```

THEOREM: end-end-yields-numbers
$\text{rangep}\,(g,\ c) \rightarrow (\text{end-end}\,(x,\ g,\ z,\ y) \in \mathbf{N})$

THEOREM: bound-on-end-end
rangep $(g,\ c) \rightarrow (\text{end-end}\,(x,\ g,\ z,\ y) < c)$

EVENT: Disable end-end.

DEFINITION:
augmented-map $(g,\ z,\ y,\ lst)$
$=$ **if** $lst \simeq$ **nil then nil**
else cons (list (car (lst), end-end (car (lst), g, z, y)),
augmented-map (g, z, y, cdr (lst))) **endif**

THEOREM: augmented-map-is-standard
rangep $(g,\ c) \rightarrow$ standard-function (augmented-map (g, z, y, lst), lst, c)

THEOREM: values-of-augmented-map
$(x \in lst) \rightarrow$ (funcall (augmented-map (g, z, y, lst), x) = end-end (x, g, z, y))

```
;;;;;;;  power-map
; now, let's develop the properties of (power-map g z y)
```

DEFINITION:
power-map (g, z, y) = augmented-map (g, z, y, power-set (vn (z)))

```
; intent :  z is in omega, and y > z
; then (power-map g z y) is a function which takes the
; sub-set S of z to  g(S U {z,y})

; first, let's consider the kind of object (power-map g z y) is,
; and show that there are  only c^(2^z) of these.
```

DEFINITION:
all-fns-on-power (z, c) = all-maps (power-set (vn (z)), c)

THEOREM: size-of-all-fns-on-power
$((c \in \mathbf{N}) \wedge (z \in \mathbf{N}))$
$\rightarrow$ (length (all-fns-on-power (z, c)) = expt (c, expt (2, z)))

THEOREM: power-map-in-all-fns
rangep $(g,\ c) \rightarrow$ (power-map (g, z, y) $\in$ all-fns-on-power (z, c))

```
; now, show that power-map takes sub-sets of z where they should go:
```

THEOREM: value-of-power-map
$(\text{setp}(s) \land \text{subsetp}(s, \text{vn}(z)))$
$\rightarrow \quad (\text{funcall}(\text{power-map}(g, z, y), s) = \text{end-end}(s, g, z, y))$

```
; the main point of this is:  if we have a set YY such that
; (power-map g z y) is the same for all y in YY, then
; g(S U {z,y}) is independent of y in YY:
```

THEOREM: same-power-maps
$(\text{setp}(s) \land \text{subsetp}(s, \text{vn}(z)) \land (\text{power-map}(g, z, y1) = \text{power-map}(g, z, y2)))$
$\rightarrow \quad (\text{funcall}(g, \text{add-on-end}(y1, \text{add-on-end}(z, s)))$
$\quad\quad = \quad \text{funcall}(g, \text{add-on-end}(y2, \text{add-on-end}(z, s))))$

```
; now, in order to get such a YY, we consider (power-map g z y)
; as a function on YY, and then apply the pigeon-hole principle

; first, repeat the procedure to get a function on YY
```

DEFINITION:
power-power (g, z, yy)
$=$ **if** $yy \simeq$ **nil then nil**
else cons (list (car (yy), power-map $(g, z, \text{car}(yy))$),
power-power $(g, z, \text{cdr}(yy))$) **endif**

```
; this maps YY into (all-fns-on-power z c) -- i.e., the
; set of functions from power-set of z into c
```

THEOREM: value-of-power-power
$(y \in yy) \rightarrow (\text{funcall}(\text{power-power}(g, z, yy), y) = \text{power-map}(g, z, y))$

THEOREM: member-values-of-power-power
$(\text{rangep}(g, c) \land (y \in yy))$
$\rightarrow \quad (\text{funcall}(\text{power-power}(g, z, yy), y) \in \text{all-fns-on-power}(z, c))$

THEOREM: range-of-power-power-aux1
$(\text{rangep}(g, c) \land \text{subsetp}(k, yy))$
$\rightarrow \quad \text{mapsto}(\text{power-power}(g, z, yy), k, \text{all-fns-on-power}(z, c))$

```
; the following was a little tricky to prove, since
; we need to induct on the second YY -- with the first one fixed
; the "natural" induction will replace YY with (cdr YY) in both occurences
```

THEOREM: range-of-power-power
$\text{rangep}(g, c) \rightarrow \text{mapsto}(\text{power-power}(g, z, yy), yy, \text{all-fns-on-power}(z, c))$

EVENT: Disable range-of-power-power-aux1.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                           THE NICE SET LEMMA                           ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; assume z is a number and XX is a set
;      (we plan to have z < min XX)
; we want (nice g z XX)  to say that whenever x1,x2 are in X
; and S is a subset of z, g(S U {z,x1}) = g(S U {z,x2})

; since we will get this situation by applying the pigeon-hole
; principle, which uses constantp, we make that the official defn
; and then derive the main property
```

DEFINITION: nice $(g, z, xx) =$ constantp (power-power (g, z, xx), xx)

THEOREM: nice-implies-constant
(nice $(g, z, xx) \land (x1 \in xx) \land (x2 \in xx)$)
$\rightarrow$ (funcall (power-power (g, z, xx), $x1$) $=$ funcall (power-power (g, z, xx), $x2$))

THEOREM: nice-implies-same-values
(nice (g, z, xx)
$\land$ $(x1 \in xx)$
$\land$ $(x2 \in xx)$
$\land$ setp (s)
$\land$ subsetp $(s$, vn $(z)))$
$\rightarrow$ (funcall (g, add-on-end ($x1$, add-on-end (z, s)))
$=$ funcall (g, add-on-end ($x2$, add-on-end (z, s))))

```
;;;;;;;  lemma "nice-going-down"

; now, we plan to start with a big XX and apply the pigeon-hole
; principle to the map (power-power g z XX) to get a subset YY
; of XX such that (constantp (power-power g z XX) YY)

; we will need that this implies (constantp (power-power g z YY) YY) --
; i.e., (nice g z YY) :  lemma "nice-going-down"

; this is true because if y is in YY, a subset of XX
; then (funcall (power-power g z YY) y)  = (funcall (power-power g z XX) y)

; this is because both are equal to (power-map g z y)
```

THEOREM: nice-goes-down-aux1
$((y \in yy) \wedge \text{subsetp}\,(yy,\ xx))$
$\rightarrow\ \ (\text{funcall}\,(\text{power-power}\,(g,\ z,\ yy),\ y) = \text{funcall}\,(\text{power-power}\,(g,\ z,\ xx),\ y))$

THEOREM: nice-goes-down-aux2
$(\text{constantp}\,(\text{power-power}\,(g,\ z,\ xx),\ yy)$
$\wedge\ \ (y1 \in yy)$
$\wedge\ \ (y2 \in yy)$
$\wedge\ \ \text{subsetp}\,(yy,\ xx))$
$\rightarrow\ \ (\text{funcall}\,(\text{power-power}\,(g,\ z,\ yy),\ y1) = \text{funcall}\,(\text{power-power}\,(g,\ z,\ yy),\ y2))$

EVENT: Disable nice-goes-down-aux1.

EVENT: Disable nice-goes-down-aux2.

THEOREM: nice-goes-down-aux3
$(\text{constantp}\,(\text{power-power}\,(g,\ z,\ xx),\ yy)$
$\wedge\ \ \text{listp}\,(vv)$
$\wedge\ \ \text{listp}\,(\text{cdr}\,(vv))$
$\wedge\ \ \text{subsetp}\,(yy,\ xx)$
$\wedge\ \ \text{subsetp}\,(vv,\ yy))$
$\rightarrow\ \ (\text{funcall}\,(\text{power-power}\,(g,\ z,\ yy),\ \text{car}\,(vv))$
$\quad = \ \ \text{funcall}\,(\text{power-power}\,(g,\ z,\ yy),\ \text{cadr}\,(vv)))$

THEOREM: nice-goes-down-aux4
$(\text{subsetp}\,(vv,\ yy) \wedge \text{listp}\,(vv)) \rightarrow \text{subsetp}\,(\text{cdr}\,(vv),\ yy)$

THEOREM: nice-goes-down-aux5
$(\text{constantp}\,(\text{power-power}\,(g,\ z,\ xx),\ yy) \wedge \text{subsetp}\,(yy,\ xx) \wedge \text{subsetp}\,(vv,\ yy))$
$\rightarrow\ \ \text{constantp}\,(\text{power-power}\,(g,\ z,\ yy),\ vv)$

```
; we had the following problem in proving this:
; we want to prove  (constantp (power-power g z YY) YY) --
; but we have to induct on the second YY -- not the first --
; so we added a set VV:  assume VV subset YY subset XX
; and prove (constantp (power-power g z YY) VV) -- then set VV = YY  :
```

THEOREM: nice-goes-down
$(\text{constantp}\,(\text{power-power}\,(g,\ z,\ xx),\ yy) \wedge \text{subsetp}\,(yy,\ xx)) \rightarrow \text{nice}\,(g,\ z,\ yy)$

EVENT: Disable nice-goes-down-aux3.

EVENT: Disable nice-goes-down-aux4.

EVENT: Disable nice-goes-down-aux5.

```
;;;;;;;;  now, back to the nice set lemma
; Suppose ZZ is a set which is phi(alpha,c) -- large ; alpha >= 1 ; c >= 1
; let z = min ZZ, and alpha-hat = {alpha}(z)
; then, by cdr-lemma-1 and cdr-lemma2
; 1. (cdr ZZ) is a set
; 2. (cdr ZZ) is phi(alpha-hat,c) * c^(2^z)  --  large

; now, suppose also that g maps into c.
; then, we can apply the pigeon-hole principle to the
; function (power-power g z (cdr ZZ)) to get a subset
; YY of (cdr ZZ) such that YY is phi(alpha-hat,c)  -- large
; and (power-power g z (cdr ZZ)) is constant on YY --
; so (nice g z YY)

; we obtain YY as (extract-nice g ZZ alpha c)
; we define extract-nice so that it will return NIL if the
; hypotheses to the nice set lemma aren't met
```

DEFINITION:
extract-nice (g, zz, $alpha$, c)
= **if** ordinalp ($alpha$)
 $\land$ ($c \not\simeq 0$)
 $\land$ ord-leq (1, $alpha$)
 $\land$ rangep (g, c)
 $\land$ setp (zz)
 $\land$ o-largep (zz, phi ($alpha$, c))
 then extract-pigeon (power-power (g, car (zz), cdr (zz)),
 cdr (zz),
 all-fns-on-power (car (zz), c),
 phi (pred ($alpha$, car (zz)), c),
 expt (c, expt (2, car (zz))))
 else nil endif

```
; if ZZ is a set and YY = (extract-nice g ZZ alpha c)
; "nice-set-A" says that YY is a set
; "nice-set-B" says that if ZZ is also phi(alpha,c) -- large
;           YY is a sub-set of (cdr ZZ)
; "nice-set-C" says that (nice g z YY)
; "nice-set-D" says that YY is phi(alpha-hat,c) -- large
;           whenever alpha is an ord >= 1, c > 0, and
;           ZZ is phi(alpha,c) -- large and (rangep g c)


; Here's a trivial property of phi:
```

THEOREM: phi-non-zero
phi ($alpha$, c) $\neq$ 0

THEOREM: nice-set-a-aux1
(setp (zz) $\wedge$ o-largep (zz, phi ($alpha$, c))) $\rightarrow$ listp (zz)

THEOREM: nice-set-a-aux2
(setp (zz) $\wedge$ o-largep (zz, phi ($alpha$, c))) $\rightarrow$ setp (cdr (zz))

THEOREM: nice-set-a-aux3
setp (**nil**)

THEOREM: nice-set-a
setp (zz) $\rightarrow$ setp (extract-nice (g, zz, $alpha$, c))

EVENT: Disable nice-set-a-aux1.

EVENT: Disable nice-set-a-aux2.

EVENT: Disable nice-set-a-aux3.

THEOREM: nice-set-b
(setp (zz) $\wedge$ o-largep (zz, phi ($alpha$, c)))
$\rightarrow$ subsetp (extract-nice (g, zz, $alpha$, c), cdr (zz))

```
; for nice-set-C -- we need (contantp ...) which involves establishing
; "mapsto" hypothesis
; we have to show that the function, (power-power g (car ZZ) (cdr ZZ))
; which is mapping on (cdr ZZ), mapsto
; (all-fns-on-power (car ZZ) c)
;
```

THEOREM: nice-set-c-aux1
rangep (g, c)
$\rightarrow$ mapsto (power-power (g, car (zz), cdr (zz)),
cdr (zz),
all-fns-on-power (car (zz), c))

```
; we also need that the size of the range >= c^(2^z)
; which follows since c and z are numbers
```

EVENT: Enable nice-set-a-aux1.

```
; ZZ is non-empty, assuming (setp ZZ) and (o-largep ZZ (phi alpha c))
```

THEOREM: nice-set-c-aux2
$((c \not\simeq 0) \wedge \text{setp}\,(zz) \wedge \text{o-largep}\,(zz,\ \text{phi}\,(alpha,\ c)))$
$\rightarrow\quad (\text{length}\,(\text{all-fns-on-power}\,(\text{car}\,(zz),\ c)) = \text{expt}\,(c,\ \text{expt}\,(2,\ \text{car}\,(zz))))$

```
; putting this together with pigeon-C, we get (constantp ...)
```

THEOREM: nice-set-c-aux3
$(\text{ordinalp}\,(alpha)$
$\wedge\quad (c \not\simeq 0)$
$\wedge\quad \text{ord-leq}\,(1,\ alpha)$
$\wedge\quad \text{rangep}\,(g,\ c)$
$\wedge\quad \text{setp}\,(zz)$
$\wedge\quad \text{o-largep}\,(zz,\ \text{phi}\,(alpha,\ c)))$
$\rightarrow\quad \text{constantp}\,(\text{power-power}\,(g,\ \text{car}\,(zz),\ \text{cdr}\,(zz)),\ \text{extract-nice}\,(g,\ zz,\ alpha,\ c))$

```
; actually, if the above hypotheses fail, then
; extract-nice returns nil, so we still have constantp
```

THEOREM: nice-set-c-aux4
$(\neg\,(\text{ordinalp}\,(alpha)$
$\wedge\quad (c \not\simeq 0)$
$\wedge\quad \text{ord-leq}\,(1,\ alpha)$
$\wedge\quad \text{rangep}\,(g,\ c)$
$\wedge\quad \text{setp}\,(zz)$
$\wedge\quad \text{o-largep}\,(zz,\ \text{phi}\,(alpha,\ c))))$
$\rightarrow\quad \text{constantp}\,(\text{power-power}\,(g,\ \text{car}\,(zz),\ \text{cdr}\,(zz)),\ \text{extract-nice}\,(g,\ zz,\ alpha,\ c))$

THEOREM: nice-set-c-aux5
$\text{constantp}\,(\text{power-power}\,(g,\ \text{car}\,(zz),\ \text{cdr}\,(zz)),\ \text{extract-nice}\,(g,\ zz,\ alpha,\ c))$

```
; now, we apply nice-goes-down, since (extract-nice g ZZ alpha c)
; is a subset of (cdr ZZ) (by nice-set-B --
; assuming (and (setp ZZ) (o-largep ZZ (phi alpha c)))   )
```

EVENT: Disable nice-set-a-aux1.

EVENT: Disable nice-set-c-aux1.

EVENT: Disable nice-set-c-aux2.

EVENT: Disable nice-set-c-aux3.

EVENT: Disable nice-set-c-aux4.

EVENT: Disable nice-set-c-aux5.

THEOREM: nice-set-c
$(\text{setp}\,(zz) \land \text{o-largep}\,(zz,\ \text{phi}\,(alpha,\ c)))$
$\rightarrow \quad \text{nice}\,(g,\ \text{car}\,(zz),\ \text{extract-nice}\,(g,\ zz,\ alpha,\ c))$

```
; finally, nice-set-D assumes the full hypotheses of extract-nice
; and concludes that (extract-nice g ZZ alpha c)
; is phi(alpha-hat,c) -- large, where alpha-hat is
; (pred alpha (car ZZ))

; this uses the fact that the set we're mapping from, (cdr ZZ)
; is (phi (pred alpha (car ZZ)) c) * (expt c (expt 2 (car ZZ))) -- large
; -- this is by the cdr-lemma

; then, we have to apply pigeon-D to get a large set
```

THEOREM: nice-set-d
$(\text{ordinalp}\,(alpha)$
$\land \quad (c \not\simeq 0)$
$\land \quad \text{ord-leq}\,(1,\ alpha)$
$\land \quad \text{rangep}\,(g,\ c)$
$\land \quad \text{setp}\,(zz)$
$\land \quad \text{o-largep}\,(zz,\ \text{phi}\,(alpha,\ c)))$
$\rightarrow \quad \text{o-largep}\,(\text{extract-nice}\,(g,\ zz,\ alpha,\ c),\ \text{phi}\,(\text{pred}\,(alpha,\ \text{car}\,(zz)),\ c))$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                  THE PRE-HOMOGENEOUS SET LEMMA                           ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; we iterate extract-nice to obtain a prehomogeneous set

;;;;;;;;  Preliminaries

;;;;  nice-and-last :    a more useful version of nice
; we want this in terms of (last S)

; probably, we don't need the defn of nice any more
```

EVENT: Disable nice.

```
;;;  here's some simple fact about (last S)
```

THEOREM: last-is-member
(setp (s) $\land$ listp (s)) $\rightarrow$ (last (s) $\in$ s)

THEOREM: subset-of-last-aux1
(listp (z) $\land$ setp (cons (v, z))) $\rightarrow$ (v < last (z))

THEOREM: subset-of-last-aux2
(setp (s) $\land$ listp (s) $\land$ (x $\in$ all-but-last (s))) $\rightarrow$ (x < last (s))

THEOREM: subset-of-last-aux3
(setp (s) $\land$ listp (s) $\land$ (x $\in$ all-but-last (s))) $\rightarrow$ (x $\in$ vn (last (s)))

THEOREM: subset-of-last
(setp (s) $\land$ listp (s)) $\rightarrow$ subsetp (all-but-last (s), vn (last (s)))

EVENT: Disable subset-of-last-aux1.

EVENT: Disable subset-of-last-aux2.

EVENT: Disable subset-of-last-aux3.

THEOREM: nice-and-last
(nice (g, z, xx)
$\land$ setp (s)
$\land$ listp (s)
$\land$ ($x1$ $\in$ xx)
$\land$ ($x2$ $\in$ xx)
$\land$ (last (s) = z))
$\rightarrow$ (funcall (g, add-on-end ($x1$, s)) = funcall (g, add-on-end ($x2$, s)))

```
;;;;;;;  rephrase the nice set lemma
;; for iterating extract-nice, it will be simpler to have an
;; explicit name for the hypotheses under which it works
; then, this can be disabled when not needed
```

DEFINITION:
nice-set-hyp (g, zz, $alpha$, c)
= (ordinalp ($alpha$)

$\wedge \quad (c \not\simeq 0)$
$\wedge \quad \text{ord-leq}\,(1,\ alpha)$
$\wedge \quad \text{rangep}\,(g,\ c)$
$\wedge \quad \text{setp}\,(zz)$
$\wedge \quad \text{o-largep}\,(zz,\ \text{phi}\,(alpha,\ c)))$

```
; if the hyp fails, extract-nice returns nil
```

THEOREM: extract-nice-is-nil
$(\neg\ \text{nice-set-hyp}\,(g,\ zz,\ alpha,\ c)) \rightarrow (\text{extract-nice}\,(g,\ zz,\ alpha,\ c) = \mathbf{nil})$

```
; now, restate the four parts of the nice set lemma
; using this hyp

; probably, we don't need the defn of extract-nice any more
```

EVENT: Disable extract-nice.

THEOREM: simple-nice-set-a
$\text{nice-set-hyp}\,(g,\ zz,\ alpha,\ c) \rightarrow \text{setp}\,(\text{extract-nice}\,(g,\ zz,\ alpha,\ c))$

THEOREM: simple-nice-set-b
$\text{nice-set-hyp}\,(g,\ zz,\ alpha,\ c) \rightarrow \text{subsetp}\,(\text{extract-nice}\,(g,\ zz,\ alpha,\ c),\ \text{cdr}\,(zz))$

THEOREM: simple-nice-set-c
$\text{nice-set-hyp}\,(g,\ zz,\ alpha,\ c) \rightarrow \text{nice}\,(g,\ \text{car}\,(zz),\ \text{extract-nice}\,(g,\ zz,\ alpha,\ c))$

THEOREM: simple-nice-set-d
$\text{nice-set-hyp}\,(g,\ zz,\ alpha,\ c)$
$\rightarrow \quad \text{o-largep}\,(\text{extract-nice}\,(g,\ zz,\ alpha,\ c),\ \text{phi}\,(\text{pred}\,(alpha,\ \text{car}\,(zz)),\ c))$

```
;;;;;;  iterability

;; to prove that iterating extract-nice to a set produces
;; a subset, all we need is
```

THEOREM: extract-nice-produces-subset
$(\text{setp}\,(zz) \wedge \text{listp}\,(zz)) \rightarrow \text{subsetp}\,(\text{extract-nice}\,(g,\ zz,\ alpha,\ c),\ \text{cdr}\,(zz))$

THEOREM: extract-nice-produces-set
$(\text{setp}\,(zz) \wedge \text{listp}\,(zz)) \rightarrow \text{setp}\,(\text{extract-nice}\,(g,\ zz,\ alpha,\ c))$

```
; in the iteration, we also need that the hypotheses replicate
; themselves until alpha becomes 0
```

THEOREM: replication-of-hyp-aux1
(nice-set-hyp $(g,\ zz,\ alpha,\ c)$ $\land$ ord-leq (1, pred $(alpha,$ car $(zz))))$
$\rightarrow$ nice-set-hyp $(g,$ extract-nice $(g,\ zz,\ alpha,\ c)$, pred $(alpha,$ car $(zz)),\ c)$

THEOREM: replication-of-hyp-aux2
nice-set-hyp $(g,\ zz,\ alpha,\ c)$ $\rightarrow$ ordinalp (pred $(alpha,\ u))$

THEOREM: replication-of-hyp
(nice-set-hyp $(g,\ zz,\ alpha,\ c)$ $\land$ (pred $(alpha,$ car $(zz)) \neq 0))$
$\rightarrow$ nice-set-hyp $(g,$ extract-nice $(g,\ zz,\ alpha,\ c)$, pred $(alpha,$ car $(zz)),\ c)$

EVENT: Disable replication-of-hyp-aux1.

EVENT: Disable replication-of-hyp-aux2.

```
;;;;;;;;;  End Preliminaries
```

```
;; we get extract-prehom by iterating extract-nice
```

```
; to prove the recursion works -- we should prove
; that extract-nice returns a set of smaller length
```

THEOREM: sublists-smaller-length
(listp (zz) $\land$ sublistp $(uu,$ cdr $(zz)))$ $\rightarrow$ (length (uu) $<$ length $(zz))$

THEOREM: subsets-smaller-length
(listp (zz) $\land$ setp (zz) $\land$ setp (uu) $\land$ subsetp $(uu,$ cdr $(zz)))$
$\rightarrow$ (length (uu) $<$ length $(zz))$

THEOREM: extract-prehom-aux1
(setp (zz) $\land$ listp $(zz))$
$\rightarrow$ (length (extract-nice $(g,\ zz,\ alpha,\ c))$ $<$ length $(zz))$

```
; the following keep getting in the way of the next defn
```

EVENT: Disable o-largep.

```
; let's keep this off permanently
```

EVENT: Disable large-goes-up.

EVENT: Disable setp.

EVENT: Disable simple-nice-set-a.

DEFINITION:
extract-prehom $(g,\ zz,\ alpha,\ c)$
$=$ **if** setp $(zz) \land$ listp $(zz) \land (alpha \neq 0)$
then cons (car (zz),
extract-prehom $(g,$
extract-nice $(g,\ zz,\ alpha,\ c)$,
pred $(alpha,$ car $(zz))$,
$c))$
else nil endif

```
; use ordinary recursion, on length, not transfinite recursion on epsilon_0
```

EVENT: Enable large-goes-up.

EVENT: Enable setp.

EVENT: Enable simple-nice-set-a.

```
; this will return an alpha-large set
; so, if alpha = 0, any set will do
; so,   the intended hypotheses to extract-prehom are
; like that of extract-nice, except that alpha could be 0
```

DEFINITION:
prehom-set-hyp $(g,\ zz,\ alpha,\ c)$
$=$ (ordinalp $(alpha)$
$\land$ $(c \not\simeq 0)$
$\land$ rangep $(g,\ c)$
$\land$ setp (zz)
$\land$ o-largep $(zz,$ phi $(alpha,\ c)))$

THEOREM: compare-hyps
(prehom-set-hyp $(g,\ zz,\ alpha,\ c) \land (alpha \neq 0))$
$\rightarrow$ nice-set-hyp $(g,\ zz,\ alpha,\ c)$

```
;;; now if ZZ is a set and XX = (extract-prehom g ZZ alpha c)
; the PREHOMOGENEOUS SET LEMMA has 6 parts.  the first
; 3 don't use any properties of extract-nice -- only the
; lemma above that it returns a subset of (cdr ZZ)

; "prehom-set-A" says that if (listp ZZ) and alpha != 0 then
;           (car XX) = (car ZZ)
; "prehom-set-B" says that
;           XX is a sub-set of ZZ
; "prehom-set-C" says that if (listp ZZ) and alpha != 0  then
;            (listp XX)
; "prehom-set-D" says that XX is a set
; "prehom-set-E" says that if (prehom-set-hyp g ZZ alpha c)
;          then (o-largep XX alpha)
; "prehom-set-F" says that if (prehom-set-hyp g ZZ alpha c)
;          then (pre-homp XX g)
```

THEOREM: prehom-set-a
$(\text{setp}\,(zz) \land \text{listp}\,(zz) \land (alpha \neq 0))$
$\rightarrow \quad (\text{car}\,(\text{extract-prehom}\,(g,\, zz,\, alpha,\, c)) = \text{car}\,(zz))$

THEOREM: prehom-set-b-aux1
$(\text{listp}\,(zz) \land \text{subsetp}\,(uu,\, ww) \land \text{subsetp}\,(ww,\, \text{cdr}\,(zz)))$
$\rightarrow \quad \text{subsetp}\,(\text{cons}\,(\text{car}\,(zz),\, uu),\, zz)$

THEOREM: prehom-set-b-aux2
$(\text{setp}\,(zz) \land \text{listp}\,(zz) \land \text{subsetp}\,(uu,\, \text{extract-nice}\,(g,\, zz,\, alpha,\, c)))$
$\rightarrow \quad \text{subsetp}\,(\text{cons}\,(\text{car}\,(zz),\, uu),\, zz)$

THEOREM: prehom-set-b
$\text{setp}\,(zz) \rightarrow \text{subsetp}\,(\text{extract-prehom}\,(g,\, zz,\, alpha,\, c),\, zz)$

EVENT: Disable prehom-set-b-aux1.

EVENT: Disable prehom-set-b-aux2.

THEOREM: prehom-set-c
$(\text{setp}\,(zz) \land \text{listp}\,(zz) \land (alpha \neq 0))$
$\rightarrow \quad \text{listp}\,(\text{extract-prehom}\,(g,\, zz,\, alpha,\, c))$

```
; for D, we will have to force the correct induction
; first, consider some trivial cases

; first, when (extract-prehom g ZZ alpha c) is empty
```

THEOREM: prehom-set-d-aux1
$(\neg\ \mathrm{setp}\,(zz)) \rightarrow \mathrm{setp}\,(\text{extract-prehom}\,(g,\ zz,\ alpha,\ c))$

THEOREM: prehom-set-d-aux2
$(\neg\ \mathrm{listp}\,(zz)) \rightarrow \mathrm{setp}\,(\text{extract-prehom}\,(g,\ zz,\ alpha,\ c))$

THEOREM: prehom-set-d-aux3
$(alpha = \mathtt{0}) \rightarrow \mathrm{setp}\,(\text{extract-prehom}\,(g,\ zz,\ alpha,\ c))$

```
; next, when (extract-prehom g ZZ alpha c) is a singleton
```

THEOREM: prehom-set-d-aux4

$$\begin{array}{ll}(\mathrm{setp}\,(zz) & \\ \ \wedge & \mathrm{listp}\,(zz) \\ \ \wedge & (alpha \neq \mathtt{0}) \\ \ \wedge & (\text{extract-nice}\,(g,\ zz,\ alpha,\ c) \simeq \mathbf{nil})) \\ \rightarrow & \mathrm{setp}\,(\text{extract-prehom}\,(g,\ zz,\ alpha,\ c))\end{array}$$

```
; we can summarize the only case left to consider as:
```

THEOREM: prehom-set-d-aux5

$$\begin{array}{lll}(\neg\ \mathrm{setp}\,(\text{extract-prehom}\,(g,\ zz,\ alpha,\ c))) & & \\ \rightarrow & (\mathrm{setp}\,(zz) & \\ & \ \wedge & \mathrm{listp}\,(zz) \\ & \ \wedge & (alpha \neq \mathtt{0}) \\ & \ \wedge & \mathrm{listp}\,(\text{extract-nice}\,(g,\ zz,\ alpha,\ c)))\end{array}$$

EVENT: Disable prehom-set-d-aux1.

EVENT: Disable prehom-set-d-aux2.

EVENT: Disable prehom-set-d-aux3.

EVENT: Disable prehom-set-d-aux4.

```
; in the inductive case, we use lemma "set-builder", but we
; have to check that it applies.  It does, because
; "extract-nice" returns a subset of the cdr
; let UU = (extract-nice g ZZ alpha c)
; and let VV = (extract-prehom g  UU (pred alpha (car ZZ)) c)
; then (extract-prehom g ZZ alpha c)  =
```

```
; (cons (car ZZ) VV)

; first, state abstractly what we're using about sets:
```

THEOREM: prehom-set-d-aux6
$(\mathrm{setp}\,(zz) \wedge \mathrm{listp}\,(zz) \wedge \mathrm{listp}\,(uu) \wedge \mathrm{subsetp}\,(uu,\ \mathrm{cdr}\,(zz)))$
$\rightarrow \quad (\mathrm{car}\,(zz) < \mathrm{car}\,(uu))$

THEOREM: prehom-set-d-aux7
$(\mathrm{setp}\,(zz)$
$\wedge \quad \mathrm{listp}\,(zz)$
$\wedge \quad \mathrm{setp}\,(uu)$
$\wedge \quad \mathrm{listp}\,(uu)$
$\wedge \quad \mathrm{setp}\,(vv)$
$\wedge \quad \mathrm{listp}\,(vv)$
$\wedge \quad (\mathrm{car}\,(vv) = \mathrm{car}\,(uu))$
$\wedge \quad \mathrm{subsetp}\,(uu,\ \mathrm{cdr}\,(zz)))$
$\rightarrow \quad \mathrm{setp}\,(\mathrm{cons}\,(\mathrm{car}\,(zz),\ vv))$

EVENT: Disable prehom-set-d-aux6.

```
; now, plug in what VV is:  of form
;    (extract-prehom g  UU delta c)
;  we still assume VV is a set (that's the induction), but
; we can drop the other hypotheses, since they follow
; from parts A,B,C of the lemma
```

THEOREM: prehom-set-d-aux8
$(\mathrm{setp}\,(ee) \wedge \mathrm{setp}\,(\mathrm{cons}\,(z,\ x)) \wedge (\neg\ \mathrm{listp}\,(ee))) \rightarrow \mathrm{setp}\,(\mathrm{cons}\,(z,\ ee))$

THEOREM: prehom-set-d-aux9
$\mathrm{listp}\,(\text{extract-prehom}\,(g,\ uu,\ delta,\ c))$
$\rightarrow \quad (\mathrm{car}\,(\text{extract-prehom}\,(g,\ uu,\ delta,\ c)) = \mathrm{car}\,(uu))$

THEOREM: prehom-set-d-aux10
$(\mathrm{setp}\,(zz)$
$\wedge \quad \mathrm{listp}\,(zz)$
$\wedge \quad \mathrm{setp}\,(uu)$
$\wedge \quad \mathrm{listp}\,(uu)$
$\wedge \quad \mathrm{subsetp}\,(uu,\ \mathrm{cdr}\,(zz))$
$\wedge \quad \mathrm{setp}\,(\text{extract-prehom}\,(g,\ uu,\ delta,\ c)))$
$\rightarrow \quad \mathrm{setp}\,(\mathrm{cons}\,(\mathrm{car}\,(zz),\ \text{extract-prehom}\,(g,\ uu,\ delta,\ c)))$

EVENT: Disable prehom-set-d-aux7.

```
; now, plug in what UU is :  (extract-nice g ZZ alpha c)
; this is a set -- it may be empty, but we've already
; considered that case, so keep the hypothesis that it's non-empty
; it is always a subset of (cdr ZZ)
```

THEOREM: prehom-set-d-aux11

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\mathrm{setp}\,(zz)} \\
\wedge & \mathrm{listp}\,(zz) \\
\wedge & \mathrm{listp}\,(\text{extract-nice}\,(g,\ zz,\ alpha,\ c)) \\
\wedge & \mathrm{setp}\,(\text{extract-prehom}\,(g,\ \text{extract-nice}\,(g,\ zz,\ alpha,\ c),\ delta,\ c))) \\
\rightarrow & \mathrm{setp}\,(\mathrm{cons}\,(\mathrm{car}\,(zz), \\
& \qquad\qquad \text{extract-prehom}\,(g,\ \text{extract-nice}\,(g,\ zz,\ alpha,\ c),\ delta,\ c)))
\end{array}
$$

EVENT: Disable prehom-set-d-aux10.

```
; now, with delta =  (pred alpha (car ZZ)) , we have the induction step
```

THEOREM: prehom-set-d-aux12

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\mathrm{setp}\,(zz)} \\
\wedge & \mathrm{listp}\,(zz) \\
\wedge & \mathrm{listp}\,(\text{extract-nice}\,(g,\ zz,\ alpha,\ c)) \\
\wedge & \mathrm{setp}\,(\text{extract-prehom}\,(g, \\
& \qquad\qquad\qquad \text{extract-nice}\,(g,\ zz,\ alpha,\ c), \\
& \qquad\qquad\qquad \mathrm{pred}\,(alpha,\ \mathrm{car}\,(zz)), \\
& \qquad\qquad\qquad c))) \\
\rightarrow & \mathrm{setp}\,(\text{extract-prehom}\,(g,\ zz,\ alpha,\ c))
\end{array}
$$

```
; now, the induction should work
```

THEOREM: prehom-set-d

$\mathrm{setp}\,(\text{extract-prehom}\,(g,\ zz,\ alpha,\ c))$

EVENT: Disable prehom-set-d-aux5.

EVENT: Disable prehom-set-d-aux8.

EVENT: Disable prehom-set-d-aux9.

EVENT: Disable prehom-set-d-aux11.

EVENT: Disable prehom-set-d-aux12.

```
; to do the inductions in E and F, we should check that
; the hypotheses replicates to the inductive call

; the following seem useful in general:
```

THEOREM: prehom-hyp-implies-non-empty
prehom-set-hyp $(g,\ zz,\ alpha,\ c) \rightarrow$ listp (zz)

THEOREM: positive-ordinals
(ordinalp $(alpha) \wedge (alpha \neq 0)) \rightarrow$ ord-leq $(1,\ alpha)$

THEOREM: replication-of-prehom-hyp-aux1
o-largep $(zz$, phi $(alpha,\ c)) \rightarrow$ listp (zz)

THEOREM: replication-of-prehom-hyp
(prehom-set-hyp $(g,\ zz,\ alpha,\ c) \wedge (alpha \neq 0))$
$\rightarrow$ prehom-set-hyp $(g$, extract-nice $(g,\ zz,\ alpha,\ c)$, pred $(alpha$, car $(zz))$, $c)$

EVENT: Disable replication-of-prehom-hyp-aux1.

```
; now, on to E.
; for the basis:
```

THEOREM: prehom-set-e-aux1
o-largep (extract-prehom $(g,\ zz,\ alpha,\ c)$, 0)

```
; for the induction, we need
```

THEOREM: prehom-set-e-aux2
$((alpha \neq 0)$
$\wedge$ o-largep (extract-prehom $(g$,
extract-nice $(g,\ zz,\ alpha,\ c)$,
pred $(alpha$, car $(zz))$,
$c)$,
pred $(alpha$, car $(zz))))$
$\rightarrow$ o-largep (cons (car (zz),
extract-prehom $(g$,
extract-nice $(g,\ zz,\ alpha,\ c)$,

pred (*alpha*, car (*zz*)),
c)),
alpha)

THEOREM: prehom-set-e-aux3
(listp (*zz*)
∧ setp (*zz*)
∧ ($alpha \neq 0$)
∧ o-largep (extract-prehom (*g*,
extract-nice (*g*, *zz*, *alpha*, *c*),
pred (*alpha*, car (*zz*)),
c),
pred (*alpha*, car (*zz*))))
→ o-largep (extract-prehom (*g*, *zz*, *alpha*, *c*), *alpha*)

```
; to force the correct induction
```

THEOREM: prehom-set-e-aux4
(prehom-set-hyp (*g*, *zz*, *alpha*, *c*)
∧ (¬ o-largep (extract-prehom (*g*, *zz*, *alpha*, *c*), *alpha*)))
→ (¬ o-largep (extract-prehom (*g*,
extract-nice (*g*, *zz*, *alpha*, *c*),
pred (*alpha*, car (*zz*)),
c),
pred (*alpha*, car (*zz*))))

THEOREM: prehom-set-e-aux5
(prehom-set-hyp (*g*, *zz*, *alpha*, *c*)
∧ (¬ o-largep (extract-prehom (*g*, *zz*, *alpha*, *c*), *alpha*)))
→ prehom-set-hyp (*g*, extract-nice (*g*, *zz*, *alpha*, *c*), pred (*alpha*, car (*zz*)), *c*)

THEOREM: prehom-set-e
prehom-set-hyp (*g*, *zz*, *alpha*, *c*)
→ o-largep (extract-prehom (*g*, *zz*, *alpha*, *c*), *alpha*)

EVENT: Disable prehom-set-e-aux1.

EVENT: Disable prehom-set-e-aux2.

EVENT: Disable prehom-set-e-aux3.

EVENT: Disable prehom-set-e-aux4.

EVENT: Disable prehom-set-e-aux5.

```
; finally, F -- that the resulting set
;  XX = (extract-prehom g ZZ alpha c) is pre-homogeneous
; so, we have to look at all non-empty subsets of XX  --
; but in fact, the defn of pre-homogeneous works will ALL sets
; S whose last element is in XX -- regardless of whether S is
; a subset of XX.

; That's not relevant to Ramsey's Theoreom, but this stronger
; statement is a lot easier to prove by induction.

; let's summarize what we're saying can't happen:
```

DEFINITION:
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2)$
$=$ (prehom-set-hyp $(g,\ zz,\ alpha,\ c)$
$\land$ $(xx = \text{extract-prehom}\,(g,\ zz,\ alpha,\ c))$
$\land$ setp (s)
$\land$ listp (s)
$\land$ $(\text{last}\,(s) \in xx)$
$\land$ $(x1 \in xx)$
$\land$ $(x2 \in xx)$
$\land$ $(\text{last}\,(s) < x1)$
$\land$ $(\text{last}\,(s) < x2)$
$\land$ $(\text{funcall}\,(g,\ \text{add-on-end}\,(x1,\ s)) \neq \text{funcall}\,(g,\ \text{add-on-end}\,(x2,\ s))))$

```
; we show this can't happen by induction on ZZ (with a fixed SS)
; first, note that XX and ZZ are non-empty sets and 1 <= alpha
; and (car XX) = (car ZZ)
```

THEOREM: prehom-set-f-aux1
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2) \rightarrow$ listp (xx)

THEOREM: prehom-set-f-aux2
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2) \rightarrow$ setp (xx)

THEOREM: prehom-set-f-aux3
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2) \rightarrow$ listp (zz)

THEOREM: prehom-set-f-aux4
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2) \rightarrow$ setp (zz)

THEOREM: prehom-set-f-aux5
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2) \rightarrow$ ordinalp $(alpha)$

THEOREM: prehom-set-f-aux6
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2) \rightarrow$ ord-leq $(\mathtt{1},\ alpha)$

THEOREM: prehom-set-f-aux7
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2)$
$\rightarrow$ $(xx =$ cons (car (zz),
extract-prehom $(g$,
extract-nice $(g,\ zz,\ alpha,\ c)$,
pred $(alpha,$ car $(zz))$,
$c)))$

THEOREM: prehom-set-f-aux8
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2) \rightarrow$ (car $(xx) =$ car $(zz))$

THEOREM: prehom-set-f-aux9
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2)$
$\rightarrow$ (cdr (xx)
$=$ extract-prehom $(g$,
extract-nice $(g,\ zz,\ alpha,\ c)$,
pred $(alpha,$ car $(zz))$,
$c))$

```
; now, we plan to use the nice set lemma to contradict (last S) = (car XX),
; so then (last S) > (car XX), whence we can use induction on ZZ
; in either case, we need that x1,x2 are in
; (extract-prehom g (extract-nice g ZZ alpha c) (pred alpha (car ZZ))  c)
```

THEOREM: prehom-set-f-aux10
(setp $(xx) \wedge (u \in xx) \wedge (x \in xx) \wedge (u < x)) \rightarrow (x \in$ cdr $(xx))$

THEOREM: prehom-set-f-aux11
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2) \rightarrow (x1 \in$ cdr $(xx))$

THEOREM: prehom-set-f-aux12
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2) \rightarrow (x2 \in$ cdr $(xx))$

THEOREM: prehom-set-f-aux13
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2)$
$\rightarrow$ $(x1 \in$ extract-prehom $(g$,
extract-nice $(g,\ zz,\ alpha,\ c)$,
pred $(alpha,$ car $(zz))$,
$c))$

THEOREM: prehom-set-f-aux14
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2)$
$\rightarrow\ \ (x2 \in \text{extract-prehom}\,(g,$
$\text{extract-nice}\,(g,\ zz,\ alpha,\ c),$
$\text{pred}\,(alpha,\ \text{car}\,(zz)),$
$c))$

```
; now, we work on refuting the case that (last S) = (car XX) ,
; using the nice set lemma
; for this, first note that x1,x2 are in (extract-nice g ZZ alpha c)
```

THEOREM: prehom-set-f-aux15
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2)$
$\rightarrow\ \ (x1 \in \text{extract-nice}\,(g,\ zz,\ alpha,\ c))$

THEOREM: prehom-set-f-aux16
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2)$
$\rightarrow\ \ (x2 \in \text{extract-nice}\,(g,\ zz,\ alpha,\ c))$

THEOREM: prehom-set-f-aux17
(bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2) \land (\text{last}\,(s) = \text{car}\,(zz)))$
$\rightarrow\ \ (\neg\ \text{nice}\,(g,\ \text{car}\,(zz),\ \text{extract-nice}\,(g,\ zz,\ alpha,\ c)))$

```
; but it should be nice, by nice-set-C
```

THEOREM: prehom-set-f-aux18
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2) \rightarrow (\text{last}\,(s) \neq \text{car}\,(zz))$

```
; now, (car ZZ) = (car XX), so (last S) is a member of (cdr XX)
```

THEOREM: prehom-set-f-aux19
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2) \rightarrow (\text{last}\,(s) \in \text{cdr}\,(xx))$

```
; now, applying aux9, we get for (last S) what we already have for x1,x2
```

THEOREM: prehom-set-f-aux20
bad-for-prehom-set $(g,\ zz,\ alpha,\ c,\ xx,\ s,\ x1,\ x2)$
$\rightarrow\ \ (\text{last}\,(s)$
$\in\ \ \text{extract-prehom}\,(g,$
$\text{extract-nice}\,(g,\ zz,\ alpha,\ c),$
$\text{pred}\,(alpha,\ \text{car}\,(zz)),$
$c))$

```
; now, we almost have the induction, but we have to take
; car of the prehom-set-hyp part
```

THEOREM: prehom-set-f-aux21
bad-for-prehom-set (g, zz, $alpha$, c, xx, s, $x1$, $x2$)
$\rightarrow$ prehom-set-hyp (g, extract-nice (g, zz, $alpha$, c), pred ($alpha$, car (zz)), c)

THEOREM: prehom-set-f-aux22
bad-for-prehom-set (g, zz, $alpha$, c, xx, s, $x1$, $x2$)
$\rightarrow$ bad-for-prehom-set (g,
extract-nice (g, zz, $alpha$, c),
pred ($alpha$, car (zz)),
c,
extract-prehom (g,
extract-nice (g, zz, $alpha$, c),
pred ($alpha$, car (zz)),
c),
s,
$x1$,
$x2$)

```
; now, force the correct induction
```

EVENT: Enable extract-prehom-aux1.

EVENT: Disable setp.

EVENT: Disable length-of-subset.

EVENT: Disable subsetp.

DEFINITION:
prehom-set-f-kludge (g, zz, $alpha$, c, xx, s, $x1$, $x2$)
= **if** setp (zz) $\wedge$ listp (zz)
then prehom-set-f-kludge (g,
extract-nice (g, zz, $alpha$, c),
pred ($alpha$, car (zz)),
c,
extract-prehom (g,
extract-nice (g, zz, $alpha$, c),
pred ($alpha$, car (zz)),
c),

s,
$x1$,
$x2$)

else nil endif

EVENT: Disable extract-prehom-aux1.

EVENT: Enable setp.

EVENT: Enable length-of-subset.

EVENT: Enable subsetp.

THEOREM: prehom-set-f-aux23
$\neg$ bad-for-prehom-set (g, zz, $alpha$, c, xx, s, $x1$, $x2$)

```
; MAIN SUBGOAL:
```

THEOREM: prehom-set-f-aux24
(prehom-set-hyp (g, zz, $alpha$, c)
$\land$ (xx = extract-prehom (g, zz, $alpha$, c))
$\land$ setp (s)
$\land$ listp (s)
$\land$ (last (s) $\in$ xx)
$\land$ ($x1$ $\in$ xx)
$\land$ ($x2$ $\in$ xx)
$\land$ (last (s) $<$ $x1$)
$\land$ (last (s) $<$ $x2$))
$\rightarrow$ (funcall (g, add-on-end ($x1$, s)) = funcall (g, add-on-end ($x2$, s)))

EVENT: Disable bad-for-prehom-set.

EVENT: Disable prehom-set-f-kludge.

EVENT: Disable prehom-set-f-aux1.

EVENT: Disable prehom-set-f-aux2.

EVENT: Disable prehom-set-f-aux3.

EVENT: Disable prehom-set-f-aux4.

Event: Disable prehom-set-f-aux5.

Event: Disable prehom-set-f-aux6.

Event: Disable prehom-set-f-aux7.

Event: Disable prehom-set-f-aux8.

Event: Disable prehom-set-f-aux9.

Event: Disable prehom-set-f-aux10.

Event: Disable prehom-set-f-aux11.

Event: Disable prehom-set-f-aux12.

Event: Disable prehom-set-f-aux13.

Event: Disable prehom-set-f-aux14.

Event: Disable prehom-set-f-aux15.

Event: Disable prehom-set-f-aux16.

Event: Disable prehom-set-f-aux17.

Event: Disable prehom-set-f-aux18.

Event: Disable prehom-set-f-aux19.

Event: Disable prehom-set-f-aux20.

Event: Disable prehom-set-f-aux21.

Event: Disable prehom-set-f-aux22.

EVENT: Disable prehom-set-f-aux23.

THEOREM: prehom-set-f-aux25
$(\text{listp}(s) \wedge \text{subsetp}(s,\ xx)) \rightarrow (\text{last}(s) \in xx)$

THEOREM: prehom-set-f-aux26
$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{prehom-set-hyp}(g,\ zz,\ alpha,\ c)} \\
\wedge & (xx = \text{extract-prehom}(g,\ zz,\ alpha,\ c)) \\
\wedge & \text{setp}(s) \\
\wedge & \text{listp}(s) \\
\wedge & \text{subsetp}(s,\ xx) \\
\wedge & (x1 \in xx) \\
\wedge & (x2 \in xx) \\
\wedge & (\text{last}(s) < x1) \\
\wedge & (\text{last}(s) < x2)) \\
\rightarrow & (\text{funcall}(g,\ \text{add-on-end}(x1,\ s)) = \text{funcall}(g,\ \text{add-on-end}(x2,\ s)))
\end{array}
$$

EVENT: Disable prehom-set-f-aux24.

EVENT: Disable prehom-set-f-aux25.

EVENT: Disable prehom-set-f-aux26.

THEOREM: prehom-set-f
$\text{prehom-set-hyp}(g,\ zz,\ alpha,\ c) \rightarrow \text{pre-homp}(\text{extract-prehom}(g,\ zz,\ alpha,\ c),\ g)$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                     RAMSEY THEOREM -- ORDINAL VERSION                    ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; This is really the main Ramsey theorem
; It says: a partition on n-tuples of a Gamma(alpha,n,c) -- large
; set has a homogeneous alpha-large set.

; first, define Gamma:
```

DEFINITION:
$\text{gamma}(alpha,\ n,\ c)$
$$
\begin{array}{ll}
= & \textbf{if } n \leq 1 \textbf{ then } \text{star}(alpha,\ c) \\
  & \textbf{else } \text{phi}(\text{gamma}(alpha,\ n-1,\ c),\ c) \textbf{ endif}
\end{array}
$$

THEOREM: gamma-is-an-ord
$\text{ordinalp}(alpha) \rightarrow \text{ordinalp}(\text{gamma}(alpha,\ n,\ c))$

```
; now, extract-ramsey produces the homogeneous set
; assuming ZZ is Gamma(alpha,n,c) -- large
```

DEFINITION:
extract-ramsey (g, zz, $alpha$, n, c)
= **if** $n \leq 1$ **then** extract-ramsey-basis (g, zz, $alpha$, c)
else extract-ramsey (derived (g,
extract-prehom (g,
zz,
gamma ($alpha$, $n - 1$, c),
c)),
extract-prehom (g, zz, gamma ($alpha$, $n - 1$, c), c),
$alpha$,
$n - 1$,
c) **endif**

```
;;; now, if ZZ is a set and YY = (extract-prehom g ZZ alpha c)

; "ord-ramsey-A" says that YY is a set and a subset of ZZ
; "ord-ramsey-B" says that YY is alpha-large
;    assuming (ordinalp alpha) and (rangep g c) and (not (zerop n))
;       and (o-largep ZZ (Gamma alpha n c))
; "ord-ramsey-C" says that (homp YY g n)
;    assuming (ordinalp alpha) and (rangep g c) and (not (zerop n))
;       and (o-largep ZZ (Gamma alpha n c))

; Then, we put these together in one "official-looking" theorem,
; "ord-ramsey"

; note -- (not (zerop c)) follows from "rangep-is-positive"

; the details of the following are probably all irrelevant now
```

EVENT: Disable extract-prehom.

EVENT: Disable extract-nice.

EVENT: Disable extract-ramsey-basis.

THEOREM: ord-ramsey-a
setp (zz)
$\rightarrow$ (setp (extract-ramsey (g, zz, $alpha$, n, c))
$\land$ subsetp (extract-ramsey (g, zz, $alpha$, n, c), zz))

```
; for B, and C, we need that if (rangep g c), then
;  (rangep (derived g WW) c), so that (rangep g c) can be
; applied inductively to the derived partition
```

THEOREM: range-of-derived-aux1
$(\text{rangep}\,(g,\ c) \land (pr \in \text{derived-aux}\,(g,\ ww,\ lst))) \rightarrow (\text{cadr}\,(pr) < c)$

THEOREM: range-of-derived-aux2
$(\text{rangep}\,(g,\ c) \land (pr \in \text{derived-aux}\,(g,\ ww,\ lst))) \rightarrow (\text{cadr}\,(pr) \in \mathbf{N})$

DEFINITION:
counter-range $(g,\ c)$
$=$ **if** $g \simeq$ **nil then nil**
elseif $\neg\,((\text{cadar}\,(g) \in \mathbf{N}) \land (\text{cadar}\,(g) < c))$ **then** car (g)
else counter-range $(\text{cdr}\,(g),\ c)$ **endif**

```
; let's list the ways rangep can fail if c is positive:
```

EVENT: Enable rangep.

THEOREM: range-of-derived-aux3
$((0 < c) \land (\neg\ \text{rangep}\,(g,\ c))) \rightarrow \text{listp}\,(g)$

THEOREM: range-of-derived-aux4
$((0 < c) \land (\neg\ \text{rangep}\,(g,\ c)))$
$\rightarrow\ ((\neg\ \text{rangep}\,(\text{cdr}\,(g),\ c)) \lor (\text{cadar}\,(g) \notin \mathbf{N}) \lor (\text{cadar}\,(g) \not< c))$

```
; in the positive direction
```

THEOREM: range-of-derived-aux5
$((\text{cadar}\,(g) \in \mathbf{N}) \land (\text{cadar}\,(g) < c) \land \text{rangep}\,(\text{cdr}\,(g),\ c)) \rightarrow \text{rangep}\,(g,\ c)$

EVENT: Disable rangep.

THEOREM: range-of-derived-aux6
$((c \neq 0) \land (c \in \mathbf{N}) \land (\neg\ \text{rangep}\,(g,\ c))) \rightarrow \text{listp}\,(g)$

THEOREM: range-of-derived-aux7
$((0 < c) \land (\neg\ \text{rangep}\,(g,\ c))) \rightarrow (\text{counter-range}\,(g,\ c) \in g)$

THEOREM: range-of-derived-aux8
$((0 < c) \land (\neg\ \text{rangep}\,(g,\ c)))$
$\rightarrow\ (\neg\,((\text{cadr}\,(\text{counter-range}\,(g,\ c)) \in \mathbf{N})$
$\qquad \land\ (\text{cadr}\,(\text{counter-range}\,(g,\ c)) < c)))$

Theorem: range-of-derived-aux9
$\text{rangep}\,(g,\ c) \rightarrow \text{rangep}\,(\text{derived-aux}\,(g,\ ww,\ lst),\ c)$

Theorem: range-of-derived
$\text{rangep}\,(g,\ c) \rightarrow \text{rangep}\,(\text{derived}\,(g,\ ww),\ c)$

Event: Disable counter-range.

Event: Disable range-of-derived-aux1.

Event: Disable range-of-derived-aux2.

Event: Disable range-of-derived-aux3.

Event: Disable range-of-derived-aux4.

Event: Disable range-of-derived-aux5.

Event: Disable range-of-derived-aux6.

Event: Disable range-of-derived-aux7.

Event: Disable range-of-derived-aux8.

Event: Disable range-of-derived-aux9.

```
; for B and C, which require a non-trivial induction,
; it is convenient to summarize the hypotheses.
```

Definition:
$$
\begin{aligned}
&\text{ord-ramsey-hyp}\,(g,\ zz,\ alpha,\ n,\ c)\\
=\quad &(\text{ordinalp}\,(alpha)\\
&\quad\land\ \ \text{rangep}\,(g,\ c)\\
&\quad\land\ \ (n \not\simeq \mathtt{0})\\
&\quad\land\ \ \text{setp}\,(zz)\\
&\quad\land\ \ \text{o-largep}\,(zz,\ \text{gamma}\,(alpha,\ n,\ c)))
\end{aligned}
$$

```
; since some of the earlier hyps had c > 0,
```

THEOREM: ord-ramsey-hyp-positive
ord-ramsey-hyp $(g,\ zz,\ alpha,\ n,\ c) \rightarrow (c \not\simeq \mathtt{0})$

```
;;;;;;;  replication results
; these should be useful in inductions
```

THEOREM: recursive-case-for-gamma
$(\mathtt{1} < n) \rightarrow (\text{gamma}\,(alpha,\ n,\ c) = \text{phi}\,(\text{gamma}\,(alpha,\ n-1,\ c),\ c))$

```
; first, note that ZZ satisfies the prehom set hyp
```

THEOREM: ord-ramsey-to-prehom-set-hyp
$((\mathtt{1} < n) \wedge \text{ord-ramsey-hyp}\,(g,\ zz,\ alpha,\ n,\ c))$
$\rightarrow \quad \text{prehom-set-hyp}\,(g,\ zz,\ \text{gamma}\,(alpha,\ n-1,\ c),\ c)$

```
; now, consider XX = (extract-prehom g ZZ (Gamma alpha (sub1 n) c) c)
; XX will be (Gamma alpha (sub1 n) c) -- large and a set, so:
```

THEOREM: replication-of-ord-ramsey-hyp
$((\mathtt{1} < n)$
$\wedge \quad \text{ord-ramsey-hyp}\,(g,\ zz,\ alpha,\ n,\ c)$
$\wedge \quad (xx = \text{extract-prehom}\,(g,\ zz,\ \text{gamma}\,(alpha,\ n-1,\ c),\ c)))$
$\rightarrow \quad \text{ord-ramsey-hyp}\,(\text{derived}\,(g,\ xx),\ xx,\ alpha,\ n-1,\ c)$

```
; also, formalize the recursive case for extract-ramsey:
```

THEOREM: recursive-case-for-extract-ramsey
$((\mathtt{1} < n) \wedge (xx = \text{extract-prehom}\,(g,\ zz,\ \text{gamma}\,(alpha,\ n-1,\ c),\ c)))$
$\rightarrow \quad (\text{extract-ramsey}\,(g,\ zz,\ alpha,\ n,\ c)$
$\quad = \quad \text{extract-ramsey}\,(\text{derived}\,(g,\ xx),\ xx,\ alpha,\ n-1,\ c))$

```
; the basis for ramsey-B is:
```

THEOREM: ord-ramsey-b-aux1
ord-ramsey-hyp $(g,\ zz,\ alpha,\ \mathtt{1},\ c)$
$\rightarrow \quad \text{o-largep}\,(\text{extract-ramsey}\,(g,\ zz,\ alpha,\ \mathtt{1},\ c),\ alpha)$

```
; this is immediate by ramsey-basis-D

; now, the induction should work because a counter-example with
; n will generate a counter-example with n-1
```

THEOREM: ord-ramsey-b-aux2
$((1 \leq n) \land$ ord-ramsey-hyp $(g,\ zz,\ alpha,\ n,\ c))$
$\rightarrow$ o-largep (extract-ramsey $(g,\ zz,\ alpha,\ n,\ c),\ alpha)$

```
; but of course, the (leq 1 n) is irrelevant
```

THEOREM: ord-ramsey-b
ord-ramsey-hyp $(g,\ zz,\ alpha,\ n,\ c)$
$\rightarrow$ o-largep (extract-ramsey $(g,\ zz,\ alpha,\ n,\ c),\ alpha)$

EVENT: Disable ord-ramsey-b-aux1.

EVENT: Disable ord-ramsey-b-aux2.

```
; the basis for ramsey-C is:
```

THEOREM: ord-ramsey-c-aux1
ord-ramsey-hyp $(g,\ zz,\ alpha,\ 1,\ c)$
$\rightarrow$ homp (extract-ramsey $(g,\ zz,\ alpha,\ 1,\ c),\ g,\ 1)$

```
; this is immediate by ramsey-basis-C

; now, the induction should work because a counter-example with
; n will generate a counter-example with n-1

; the induction will be:  if
; XX =  (extract-prehom g ZZ (Gamma alpha (sub1 n) c) c)
; YY = (extract-ramsey g ZZ alpha n c) =
;      (extract-ramsey (derived g XX) XX alpha (sub1 n) c)
;         (by recursive-case-for-extract-ramsey)
; and inductively,
;  (homp  YY  (derived g XX) (sub1 n))
; then by the derived partition lemma
; (homp YY g n)

; let's phrase the derived-partition-lemma in the current notation
```

THEOREM: ord-ramsey-c-aux2
$((1 < n)$
$\land$ pre-homp $(xx,\ g)$
$\land$ sublistp $(yy,\ xx)$
$\land$ homp $(yy,$ derived $(g,\ xx),\ n - 1))$
$\rightarrow$ homp $(yy,\ g,\ n)$

```
; now, let's check that the XX and YY as above really
; satisfy the (pre-homp XX g)  and (sublistp YY XX)
```

THEOREM: ord-ramsey-c-aux3
$((1 < n)$
$\land$ ord-ramsey-hyp $(g,\ zz,\ alpha,\ n,\ c)$
$\land$ $(xx =$ extract-prehom $(g,\ zz,$ gamma $(alpha,\ n - 1,\ c),\ c)))$
$\rightarrow$ pre-homp $(xx,\ g)$

THEOREM: ord-ramsey-c-aux4
$((1 < n)$
$\land$ ord-ramsey-hyp $(g,\ zz,\ alpha,\ n,\ c)$
$\land$ $(xx =$ extract-prehom $(g,\ zz,$ gamma $(alpha,\ n - 1,\ c),\ c))$
$\land$ $(yy =$ extract-ramsey $(g,\ zz,\ alpha,\ n,\ c)))$
$\rightarrow$ sublistp $(yy,\ xx)$

```
; it follows that aux2 gives
```

THEOREM: ord-ramsey-c-aux5
$((1 < n)$
$\land$ ord-ramsey-hyp $(g,\ zz,\ alpha,\ n,\ c)$
$\land$ $(xx =$ extract-prehom $(g,\ zz,$ gamma $(alpha,\ n - 1,\ c),\ c))$
$\land$ $(yy =$ extract-ramsey $(g,\ zz,\ alpha,\ n,\ c))$
$\land$ homp $(yy,$ derived $(g,\ xx),\ n - 1))$
$\rightarrow$ homp $(yy,\ g,\ n)$

THEOREM: ord-ramsey-c-aux6
$((1 \leq n) \land$ ord-ramsey-hyp $(g,\ zz,\ alpha,\ n,\ c))$
$\rightarrow$ homp (extract-ramsey $(g,\ zz,\ alpha,\ n,\ c),\ g,\ n)$

```
; but of course, the (leq 1 n) is irrelevant
```

THEOREM: ord-ramsey-c
ord-ramsey-hyp $(g,\ zz,\ alpha,\ n,\ c) \rightarrow$ homp (extract-ramsey $(g,\ zz,\ alpha,\ n,\ c),\ g,\ n)$

EVENT: Disable ord-ramsey-c-aux1.

EVENT: Disable ord-ramsey-c-aux2.

EVENT: Disable ord-ramsey-c-aux3.

EVENT: Disable ord-ramsey-c-aux4.

EVENT: Disable ord-ramsey-c-aux5.

EVENT: Disable ord-ramsey-c-aux6.

```
; Finally, a version suitable for framing:
```

THEOREM: ord-ramsey
(ordinalp (*alpha*)
 $\wedge$ rangep (g, c)
 $\wedge$ ($n \not\simeq 0$)
 $\wedge$ setp (zz)
 $\wedge$ o-largep (zz, gamma ($alpha$, n, c))
 $\wedge$ (yy = extract-ramsey (g, zz, $alpha$, n, c)))
$\rightarrow$ (setp (yy) $\wedge$ subsetp (yy, zz) $\wedge$ o-largep (yy, $alpha$) $\wedge$ homp (yy, g, n))

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                   RAMSEY THEOREM -- PARIS-HARRINGTON VERSION             ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; FINALLY, we use epsilon_0 recursion/induction.  We define
; a function (lambda alpha k) such that ; (lambda alpha k) >= k,
; and the interval [k, (lamda alpha k)] is alpha - large
; This will show that applying the ordinal version of Ramsey's
; theorem to a suitable large interval will produce an
; omega-large (hence large) homogeneous set.
```

DEFINITION:
lambda_kludge (*alpha*, *k*)
= **if** ordinalp (*alpha*) **then** *alpha*
  else 0 **endif**

```
; just to get the prover to accept the following definition
```

DEFINITION:
lambda (*alpha*, *k*)
= **if** ord-lessp (0, *alpha*) $\wedge$ ordinalp (*alpha*)
  then lambda (pred (*alpha*, *k*), $1 + k$)
  else *k* **endif**

EVENT: Disable lambda_kludge.

THEOREM: lambda-is-a-number
$(k \in \mathbf{N}) \rightarrow (\text{lambda}\,(alpha,\ k) \in \mathbf{N})$

THEOREM: lambda-is-big
$(k \in \mathbf{N}) \rightarrow (\text{lambda}\,(alpha,\ k) \not< k)$

THEOREM: base-case-for-lambda
$(\neg\ \text{ord-lessp}\,(\mathtt{0},\ alpha)) \rightarrow (\text{lambda}\,(alpha,\ k) = k)$

THEOREM: recursive-case-for-lambda
$(\text{ordinalp}\,(alpha) \land \text{ord-lessp}\,(\mathtt{0},\ alpha))$
$\rightarrow \quad (\text{lambda}\,(alpha,\ k) = \text{lambda}\,(\text{pred}\,(alpha,\ k),\ 1 + k))$

EVENT: Disable lambda.

```
; let's prove that (o-largep (segment k (lambda alpha k)) alpha)
; this is by induction

; base case:
```

THEOREM: lambda-gives-large-segments-aux1
$((k \in \mathbf{N}) \land \text{ordinalp}\,(alpha) \land (\neg\ \text{ord-lessp}\,(\mathtt{0},\ alpha)))$
$\rightarrow \quad \text{o-largep}\,(\text{segment}\,(k,\ \text{lambda}\,(alpha,\ k)),\ alpha)$

```
; for the recursive case:
```

THEOREM: lambda-gives-large-segments-aux2
$((k \in \mathbf{N}) \land \text{ordinalp}\,(alpha) \land \text{ord-lessp}\,(\mathtt{0},\ alpha))$
$\rightarrow \quad (\text{segment}\,(k,\ \text{lambda}\,(alpha,\ k))$
$\qquad = \quad \text{cons}\,(k,\ \text{segment}\,(1 + k,\ \text{lambda}\,(\text{pred}\,(alpha,\ k),\ 1 + k))))$

THEOREM: lambda-gives-large-segments-aux3
$((k \in \mathbf{N})$
$\land \quad \text{ordinalp}\,(alpha)$
$\land \quad \text{ord-lessp}\,(\mathtt{0},\ alpha)$
$\land \quad \text{o-largep}\,(\text{segment}\,(1 + k,\ \text{lambda}\,(\text{pred}\,(alpha,\ k),\ 1 + k)),\ \text{pred}\,(alpha,\ k)))$
$\rightarrow \quad \text{o-largep}\,(\text{segment}\,(k,\ \text{lambda}\,(alpha,\ k)),\ alpha)$

THEOREM: lambda-gives-large-segments
$((k \in \mathbf{N}) \land \text{ordinalp}\,(alpha))$
$\rightarrow \quad \text{o-largep}\,(\text{segment}\,(k,\ \text{lambda}\,(alpha,\ k)),\ alpha)$

EVENT: Disable lambda-gives-large-segments-aux1.

EVENT: Disable lambda-gives-large-segments-aux2.

EVENT: Disable lambda-gives-large-segments-aux3.

```
; so, the Ramsey number is:
```

DEFINITION: $\mathrm{r}(k, n, c) = \mathrm{lambda}(\mathrm{gamma}(\texttt{'(1 . 0)}, n, c), k)$

```
; this is a number:
```

THEOREM: r-is-a-number
$(k \in \mathbf{N}) \rightarrow (\mathrm{r}(k, n, c) \in \mathbf{N})$

THEOREM: r-is-a-big
$(k \in \mathbf{N}) \rightarrow (\mathrm{r}(k, n, c) \not< k)$

```
; now, we will get YY as an omega--large subset of ZZ = [k, (R k n c)]
; we want to show that that implies that YY is
; large (in Paris-Harrington sense) and has size at least k
; (which follows from the defn of large)
```

THEOREM: result-is-large-aux1
$(\mathrm{setp}(yy) \land \mathrm{o\text{-}largep}(yy, \texttt{'(1 . 0)}))$
$\rightarrow \quad (\mathrm{largep}(yy) \land \mathrm{listp}(yy) \land (\mathrm{length}(yy) \not< \mathrm{car}(yy)))$

THEOREM: result-is-large-aux2
$(x \in \mathrm{segment}(k, j)) \rightarrow (\mathrm{car}(\mathrm{segment}(k, j)) = k)$

THEOREM: result-is-large-aux3
$(x \in \mathrm{segment}(k, j)) \rightarrow (x \not< k)$

THEOREM: result-is-large-aux4
$(\mathrm{listp}(yy) \land \mathrm{subsetp}(yy, \mathrm{segment}(k, j))) \rightarrow (\mathrm{car}(yy) \not< k)$

THEOREM: result-is-large
$(\mathrm{setp}(yy) \land \mathrm{subsetp}(yy, \mathrm{segment}(k, j)) \land \mathrm{o\text{-}largep}(yy, \texttt{'(1 . 0)}))$
$\rightarrow \quad (\mathrm{largep}(yy) \land (\mathrm{length}(yy) \not< k))$

EVENT: Disable result-is-large-aux1.

EVENT: Disable result-is-large-aux2.

EVENT: Disable result-is-large-aux3.

EVENT: Disable result-is-large-aux4.

```
;;;;;;;;  extracting the homogeneous set

; (extract-ramsey-P-H g k n c)  finds a subset
; of (segment 0 (R k n c))  which is large and of size at least k
; actually, it just finds an omega-large subset of [k, (R k n c)]
```

DEFINITION:
extract-ramsey-p-h $(g,\ k,\ n,\ c)$
$=$ extract-ramsey $(g$, segment $(k$, r $(k,\ n,\ c))$, `'(1 . 0)`, $n,\ c)$

```
; first, let's see what we get by a direct application
; of the ord-ramsey theorem.  To apply that, we need that
; the segment  (segment k (R k n c))  is
; (Gamma omega n c) -- large
```

THEOREM: r-segment-is-large
$(k \in \mathbf{N}) \rightarrow$ o-largep (segment $(k$, r $(k,\ n,\ c))$, gamma (`'(1 . 0)`, $n,\ c$))

```
; applying ord-ramsey,
```

THEOREM: ramsey-p-h-aux1
(rangep $(g,\ c)$
$\wedge$ $(n \not\simeq 0)$
$\wedge$ $(k \in \mathbf{N})$
$\wedge$ $(r = \mathrm{r}\,(k,\ n,\ c))$
$\wedge$ $(yy =$ extract-ramsey-p-h $(g,\ k,\ n,\ c)))$
$\rightarrow$ (setp (yy)
$\wedge$ subsetp $(yy$, segment $(k,\ r))$
$\wedge$ homp $(yy,\ g,\ n)$
$\wedge$ o-largep $(yy$, `'(1 . 0)`$))$

EVENT: Disable ord-ramsey.

EVENT: Disable ord-ramsey-a.

EVENT: Disable ord-ramsey-b.

EVENT: Disable ord-ramsey-c.

```
; to convert to intended Ramsey theorem:
```

```
; we need that (subsetp YY (segment k R))
; implies (subsetp YY (segment 0 R))
```

THEOREM: ramsey-p-h-aux2
$((k \in \mathbf{N}) \land (r \in \mathbf{N}) \land (k \leq r) \land (x \in \text{segment}\,(k,\ r)))$
$\rightarrow \quad (x \in \text{segment}\,(\mathtt{0},\ r))$

THEOREM: ramsey-p-h-aux3
$((k \in \mathbf{N}) \land (r \in \mathbf{N}) \land (k \leq r)) \rightarrow \text{subsetp}\,(\text{segment}\,(k,\ r),\ \text{segment}\,(\mathtt{0},\ r))$

THEOREM: ramsey-p-h-aux4
$((k \in \mathbf{N}) \land (r \in \mathbf{N}) \land (k \leq r) \land \text{subsetp}\,(yy,\ \text{segment}\,(k,\ r)))$
$\rightarrow \quad \text{subsetp}\,(yy,\ \text{segment}\,(\mathtt{0},\ r))$

EVENT: Disable segment.

EVENT: Disable subsetp.

EVENT: Disable setp.

EVENT: Disable extract-ramsey.

EVENT: Disable extract-ramsey-p-h.

EVENT: Disable recursive-case-for-segment.

```
; Paris - Harrington version of Ramsey Theorem:
```

THEOREM: ramsey-p-h
$(\text{rangep}\,(g,\ c)$
$\land \quad (n \not\simeq \mathtt{0})$
$\land \quad (k \in \mathbf{N})$
$\land \quad (r = \text{r}\,(k,\ n,\ c))$
$\land \quad (yy = \text{extract-ramsey-p-h}\,(g,\ k,\ n,\ c)))$
$\rightarrow \quad (\text{setp}\,(yy)$
$\quad \land \quad \text{subsetp}\,(yy,\ \text{segment}\,(\mathtt{0},\ r))$
$\quad \land \quad \text{homp}\,(yy,\ g,\ n)$
$\quad \land \quad (k \leq \text{length}\,(yy))$
$\quad \land \quad \text{largep}\,(yy))$

```
;;; NOTE! : without the (largep YY), this would just be the
;;; standard Ramsey theorem, which is provable in PRA
```

Event: Make the library "paris-harrington" and compile it.

Index