```
#|
 Copyright (C) 1995 by Matthew Wilding and Computational Logic, Inc.
 All Rights Reserved.

 This script is hereby placed in the public domain, and therefore unlimited
 editing and redistribution is permitted.

 NO WARRANTY

 Matthew Wilding and Computational Logic, Inc. PROVIDES ABSOLUTELY NO
 WARRANTY.  THE EVENT SCRIPT IS PROVIDED "AS IS" WITHOUT WARRANTY OF
 ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
 ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND
 PERFORMANCE OF THE SCRIPT IS WITH YOU.  SHOULD THE SCRIPT PROVE
 DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR
 CORRECTION.

 IN NO EVENT WILL Matthew Wilding and Computational Logic, Inc. BE
 LIABLE TO YOU FOR ANY DAMAGES, ANY LOST PROFITS, LOST MONIES, OR
 OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
 USE OR INABILITY TO USE THIS SCRIPT (INCLUDING BUT NOT LIMITED TO
 LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
 THIRD PARTIES), EVEN IF YOU HAVE ADVISED US OF THE POSSIBILITY OF
 SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.
|#

;; This file contains the events that lead to the proof of the
;; optimality of an earliest-deadline-first scheduler on any set of
;; periodic tasks.  It is documented in CLI Technical Report #110.
;;
;; Report #110 will not be available for public dissemination until
;; December 1995 at the earliest, but the portion about this proof is
;; available as ftp://ftp.cli.com/home/wilding/scheduler-proof.ps
;;
;; Matt Wilding
;; September, 1995
```

Event: Start with the library "naturals" using the compiled version.

Definition: $\text{tk-name}\,(pt) = \text{car}\,(pt)$

Definition: $\text{tk-period}\,(pt) = \text{cadr}\,(pt)$

DEFINITION: tk-duration (pt) = caddr (pt)

```
; periodic task is a triple (name period duration)
```

DEFINITION:
periodic-taskp (pt)
= (listp (pt)
  ∧ litatom (tk-name (pt))
  ∧ (tk-name $(pt) \neq$ **nil**)
  ∧ (0 < tk-period (pt))
  ∧ (0 < tk-duration (pt))
  ∧ (**nil** = cdr (cdr (cdr (pt)))))

DEFINITION:
periodic-tasksp (pts)
= **if** listp (pts)
  then periodic-taskp (car (pts))
    ∧ (¬ assoc (tk-name (car (pts)), cdr (pts)))
    ∧ periodic-tasksp (cdr (pts))
  else pts = **nil endif**

```
;; a task request has the form (name request-time deadline-time duration)

;; generate the requests for a periodic task during a time period
```

DEFINITION:
periodic-task-requests $(pt,\ starting\text{-}time,\ ending\text{-}time)$
= **if** periodic-taskp (pt)
  then if $starting\text{-}time < ending\text{-}time$
    then cons (list (tk-name (pt),
        $starting\text{-}time$,
        $starting\text{-}time$ + tk-period (pt),
        tk-duration (pt)),
      periodic-task-requests $(pt,$
        $starting\text{-}time$ + tk-period (pt),
        $ending\text{-}time$))
    else nil endif
  else nil endif

```
;; we generate a list of task request lists
```

DEFINITION:
periodic-tasks-requests $(pts,\ starting\text{-}time,\ ending\text{-}time)$
= **if** periodic-tasksp (pts)
  then if listp (pts)

```
        then append (periodic-task-requests (car (pts),
                                              starting-time,
                                              ending-time),
                     periodic-tasks-requests (cdr (pts),
                                              starting-time,
                                              ending-time))
        else nil endif
    else nil endif
```

DEFINITION:
repeat $(n, \; val)$

```
=   if n ≃ 0 then nil
    else cons (val, repeat (n − 1, val)) endif
```

```
;; produce an "obvious" schedule of length c that consists of
;; (duration/period)*c calls of each task in a periodic task list
```

DEFINITION:
substring-schedule (*pts*, *bigp*)

```
=   if listp (pts)
    then append (repeat ((bigp * caddar (pts)) ÷ cadar (pts), caar (pts)),
                 substring-schedule (cdr (pts), bigp))
    else nil endif
```

DEFINITION:
make-length (*length*, *list*, *fill*)

```
=   if length ≃ 0 then nil
    elseif listp (list)
    then cons (car (list), make-length (length − 1, cdr (list), fill))
    else cons (fill, make-length (length − 1, nil, fill)) endif
```

DEFINITION:
repeat-list (*list*, *times*)

```
=   if times ≃ 0 then nil
    else append (list, repeat-list (list, times − 1)) endif
```

DEFINITION:
make-simple-schedule (*pts*, *bigp*, *length*)

```
=   repeat-list (make-length (bigp, substring-schedule (pts, bigp), nil),
                 length ÷ bigp)
```

DEFINITION:
firstn (*n*, *list*)

```
=   if n ≃ 0 then nil
    else cons (car (list), firstn (n − 1, cdr (list))) endif
```

DEFINITION:
nth $(n,\ list)$
$=$ **if** $n \simeq 0$ **then** car $(list)$
else nth $(n - 1,\ \text{cdr}\,(list))$ **endif**

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

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

THEOREM: length-append
length (append $(x,\ y)$) $=$ (length (x) $+$ length (y))

THEOREM: length-firstn
length (firstn $(n,\ list)$) $=$ fix (n)

THEOREM: length-nthcdr
length (nthcdr $(n,\ list)$) $=$ (length $(list)$ $-$ n)

THEOREM: equal-length-0
(length $(x) = 0$) $=$ ($\neg$ listp (x))

```
;; each task request period is a multiple of bigp, and bigp*duration
;; is a multiple of period
```

DEFINITION:
expanded-tasksp $(ts,\ p)$
$=$ **if** listp (ts)
then ((cadar (ts) **mod** p) $\simeq 0$)
$\wedge$ ((caddar (ts) **mod** p) $\simeq 0$)
$\wedge$ ((($p *$ caddar (ts)) **mod** cadar (ts)) $\simeq 0$)
$\wedge$ expanded-tasksp (cdr (ts), p)
else t endif

DEFINITION:
plist $(list)$
$=$ **if** listp $(list)$ **then** cons (car $(list)$, plist (cdr $(list)$))
else nil endif

DEFINITION: name (r) $=$ car (r)

DEFINITION: request-time (r) = cadr (r)

DEFINITION: deadline (r) = caddr (r)

DEFINITION: duration (r) = cadddr (r)

EVENT: Let us define the theory *task-abbr* to consist of the following events: name, request-time, deadline, duration, tk-name, tk-duration, tk-period.

DEFINITION:
good-schedule $(s,\ r)$
= **if** listp (r)
  then (occurrences (name (car (r)),
      firstn (deadline (car (r)) − request-time (car (r)),
        nthcdr (request-time (car (r)), s)))
    = duration (car (r)))
   ∧ good-schedule $(s$, cdr $(r))$
  else t endif

DEFINITION:
big-period (pts)
= **if** listp (pts) **then** tk-period (car (pts)) ∗ big-period (cdr (pts))
  else 1 **endif**

DEFINITION:
active-task-requests $(time,\ r)$
= **if** listp (r)
  then if $(time <$ deadline (car (r)))
     ∧ $(time \not<$ request-time (car (r)))
    then cons (car (r), active-task-requests $(time$, cdr $(r))$)
    else active-task-requests $(time$, cdr $(r))$ **endif**
  else nil endif

DEFINITION:
unfulfilled $(time,\ s,\ r)$
= **if** listp (r)
  then if occurrences (name (car (r)),
      firstn $(time\ -$ request-time (car (r)),
        nthcdr (request-time (car (r)), s)))
     = duration (car (r)) **then** unfulfilled $(time,\ s$, cdr $(r))$
    else cons (car (r), unfulfilled $(time,\ s$, cdr $(r))$) **endif**
  else nil endif

```
;; return a task request with least deadline
```

DEFINITION:
least-deadline (r)
$=$ **if** listp (r)
then if listp (cdr (r))
then if deadline (car (r)) $<$ deadline (car (cdr (r)))
then least-deadline (cons (car (r), cdr (cdr (r))))
else least-deadline (cdr (r)) **endif**
else car (r) **endif**
else nil endif

```
;; return location of first instance of task in s no earlier than time
```

DEFINITION:
first-instance (*time*, *task*, s)
$=$ **if** *time* $<$ length (s)
then if nth (*time*, s) $=$ *task* **then** *time*
else first-instance ($1 +$ *time*, *task*, s) **endif**
else f endif

DEFINITION:
replace-nth (n, *val*, *list*)
$=$ **if** $n \simeq$ `0` **then** cons (*val*, cdr (*list*))
else cons (car (*list*), replace-nth ($n - 1$, *val*, cdr (*list*))) **endif**

```
;; swap locations i and j in list
```

DEFINITION:
swap (i, j, *list*)
$=$ replace-nth (i, nth (j, *list*), replace-nth (j, nth (i, *list*), *list*))

THEOREM: length-replace-nth
length (replace-nth (i, *val*, *list*))
$=$ **if** $i <$ length (*list*) **then** length (*list*)
else $1 + i$ **endif**

THEOREM: length-swap
length (swap (i, j, *list*))
$=$ **if** $i <$ length (*list*)
then if $j <$ length (*list*) **then** length (*list*)
else $1 + j$ **endif**
elseif $i < j$ **then** $1 + j$
else $1 + i$ **endif**

DEFINITION:
make-element-edf (s, r, *time*)

= **let** *unfulfilled* **be** unfulfilled (*time*, *s*, active-task-requests (*time*, *r*))
in
let *first* **be** first-instance (*time*,
car (least-deadline (*unfulfilled*)),
s)
in
if listp (*unfulfilled*) $\wedge$ *first* **then** swap (*time*, *first*, *s*)
else *s* **endif endlet endlet**

THEOREM: lessp-first-instance
listp (*s*) $\rightarrow$ (first-instance (*time*, *task*, *s*) $<$ length (*s*))

THEOREM: length-make-element-edf
(*time* $<$ length (*s*)) $\rightarrow$ (length (make-element-edf (*s*, *r*, *time*)) $=$ length (*s*))

EVENT: Disable make-element-edf.

DEFINITION:
make-schedule-edf (*s*, *r*, *first*)
= **if** *first* $<$ length (*s*)
then make-schedule-edf (make-element-edf (*s*, *r*, *first*), *r*, 1 + *first*)
else *s* **endif**

EVENT: Enable make-element-edf.

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

THEOREM: plist-repeat-list
plist (repeat-list (*s*, *n*)) = repeat-list (*s*, *n*)

THEOREM: length-repeat-list
length (repeat-list (*s*, *n*)) = (*n* $*$ length (*s*))

THEOREM: append-nil
append (*list*, **nil**) = plist (*list*)

THEOREM: good-schedule-append
good-schedule (*s*, append (*r1*, *r2*))
= (good-schedule (*s*, *r1*) $\wedge$ good-schedule (*s*, *r2*))

```
;; introduce the useful notion of sublist
```

DEFINITION:
sublistp $(a,\ b)$
$=$ **if** listp (b)
then if listp (a)
then if car $(a) =$ car (b) **then** sublistp (cdr (a), cdr (b))
else sublistp (a, cdr (b)) **endif**
else t endif
else $a \simeq$ **nil endif**

DEFINITION:
remove-until $(v,\ l)$
$=$ **if** listp (l)
then if $v =$ car (l) **then** cdr (l)
else remove-until (v, cdr (l)) **endif**
else l **endif**

THEOREM: sublistp-cons-rewrite
sublistp (cons $(c,\ x),\ y) = ((c \in y) \wedge$ sublistp (x, remove-until $(c,\ y)$))

THEOREM: listp-remove-until-means-listp
listp (remove-until $(a,\ z)$) $\rightarrow$ listp (z)

THEOREM: lessp-remove-until
(length (remove-until $(a,\ y)$) $<$ length (y)) $=$ listp (y)

THEOREM: remove-until-append
remove-until (a, append $(x,\ y)$)
$=$ **if** $a \in x$ **then** append (remove-until $(a,\ x),\ y$)
else remove-until $(a,\ y)$ **endif**

DEFINITION:
list-until $(v,\ l)$
$=$ **if** listp (l)
then if $v =$ car (l) **then** list (v)
else cons (car (l), list-until (v, cdr (l))) **endif**
else l **endif**

THEOREM: append-remove-until-list-until
append (list-until $(v,\ l)$, remove-until $(v,\ l)$) $= l$

```
;; amazingly complex - easier way? (took me ~3 hours to prove this little guy)
```

DEFINITION:
sublistp-append-induct $(a,\ b,\ y,\ z)$
$=$ **if** listp (a)
then sublistp-append-induct (cdr (a),

b,
append (y, list-until (car (a), z)),
remove-until (car (a), z))

elseif $b \simeq$ **nil then t**
elseif listp (y)
then if car (b) $\in y$
then sublistp-append-induct (list (car (b)),
cdr (b),
remove-until (car (b), y),
z)
else t endif
else t endif

THEOREM: member-append
($a \in$ append (x, y)) = (($a \in x$) $\vee$ ($a \in y$))

THEOREM: listp-bagint-with-singleton-implies-member
listp (bagint (y, list (z))) $\rightarrow$ ($z \in y$)

THEOREM: bagint-singleton
bagint (x, list (y))
= **if** $y \in x$ **then** list (y)
else nil endif

THEOREM: transitivity-of-append
append (append (a, b), c) = append (a, append (b, c))

THEOREM: sublistp-append
sublistp (append (a, b), z) $\rightarrow$ sublistp (b, append (y, z))

THEOREM: sublistp-cdr1
sublistp (x, y) $\rightarrow$ sublistp (cdr (x), y)

THEOREM: sublistp-cdr2
sublistp (x, cdr (y)) $\rightarrow$ sublistp (x, y)

THEOREM: remainder-big-period-sublist
(periodic-tasksp (y) $\wedge$ sublistp (x, y))
$\rightarrow$ ((big-period (y) **mod** big-period (x)) = 0)

THEOREM: remainder-big-period-cdr
((n **mod** big-period (pts)) = 0) $\rightarrow$ ((n **mod** big-period (cdr (pts))) = 0)

THEOREM: repeat-list-plus
repeat-list (l, $a + b$) = append (repeat-list (l, a), repeat-list (l, b))

THEOREM: nthcdr-append

$\text{nthcdr}(n, \text{append}(l1, l2))$
$= \textbf{if } n < \text{length}(l1) \textbf{ then } \text{append}(\text{nthcdr}(n, l1), l2)$
$\quad \textbf{else } \text{nthcdr}(n - \text{length}(l1), l2) \textbf{ endif}$

THEOREM: firstn-append

$\text{firstn}(n, \text{append}(l1, l2))$
$= \textbf{if } \text{length}(l1) < n \textbf{ then } \text{append}(l1, \text{firstn}(n - \text{length}(l1), l2))$
$\quad \textbf{else } \text{firstn}(n, l1) \textbf{ endif}$

DEFINITION:

$\text{nthcdr-repeat-list-induct}(n1, n2, list)$
$= \textbf{if } n1 \simeq 0 \textbf{ then t}$
$\quad \textbf{else } \text{nthcdr-repeat-list-induct}(n1 - 1, n2 - \text{length}(list), list) \textbf{ endif}$

THEOREM: nthcdr-repeat-list

$(((n1 \textbf{ mod } \text{length}(list)) = 0) \land ((n2 * \text{length}(list)) \not< n1))$
$\to (\text{nthcdr}(n1, \text{repeat-list}(list, n2))$
$\quad = \text{repeat-list}(list, n2 - (n1 \div \text{length}(list))))$

THEOREM: length-make-length

$\text{length}(\text{make-length}(n, list, fill)) = \text{fix}(n)$

THEOREM: member-expanded

$((tk \in pts) \land \text{expanded-tasksp}(pts, bigp))$
$\to ((\text{cadr}(tk) \textbf{ mod } bigp) = 0)$

THEOREM: firstn-length-list

$\text{firstn}(\text{length}(x), x) = \text{plist}(x)$

THEOREM: firstn-repeat-list

$(((n1 \textbf{ mod } \text{length}(list)) = 0) \land ((n2 * \text{length}(list)) \not< n1))$
$\to (\text{firstn}(n1, \text{repeat-list}(list, n2))$
$\quad = \text{repeat-list}(list, n1 \div \text{length}(list)))$

THEOREM: lessp-remainder-special

$(((x \textbf{ mod } z) = 0) \land ((y \textbf{ mod } z) = 0))$
$\to ((y < (x + z)) = ((z \not\simeq 0) \land (x \not< y)))$

THEOREM: lessp-difference-special

$(((x \textbf{ mod } z) = 0) \land ((y \textbf{ mod } z) = 0))$
$\to (((x - y) < z) = ((z \not\simeq 0) \land (y \not< x)))$

THEOREM: occurrences-append

$\text{occurrences}(a, \text{append}(x, y)) = (\text{occurrences}(a, x) + \text{occurrences}(a, y))$

THEOREM: occurrences-repeat-list
$\text{occurrences}(v, \text{repeat-list}(\mathit{list}, n)) = (n * \text{occurrences}(v, \mathit{list}))$

THEOREM: occurrences-make-length
$\text{occurrences}(v, \text{make-length}(\mathit{size}, \mathit{list}, \mathit{fill}))$
$= \ \textbf{if } \mathit{size} < \text{length}(\mathit{list}) \ \textbf{then } \text{occurrences}(v, \text{firstn}(\mathit{size}, \mathit{list}))$
$\quad \textbf{elseif } v = \mathit{fill}$
$\quad \textbf{then } \text{occurrences}(v, \mathit{list}) + (\mathit{size} - \text{length}(\mathit{list}))$
$\quad \textbf{else } \text{occurrences}(v, \mathit{list}) \ \textbf{endif}$

THEOREM: occurrences-repeat
$\text{occurrences}(x, \text{repeat}(n, y))$
$= \ \textbf{if } x = y \ \textbf{then } \text{fix}(n)$
$\quad \textbf{else } 0 \ \textbf{endif}$

THEOREM: member-repeat
$(x \in \text{repeat}(n, v)) = ((0 < n) \wedge (x = v))$

THEOREM: member-substring-schedule
$(\text{periodic-tasksp}(z) \wedge \text{expanded-tasksp}(z, \mathit{bigp}) \wedge (v \neq \textbf{nil}))$
$\rightarrow \ ((v \in \text{substring-schedule}(z, \mathit{bigp})) \leftrightarrow \text{assoc}(v, z))$

THEOREM: member-car-x-x
$(\text{car}(x) \in x) = \text{listp}(x)$

THEOREM: occurrences-substring-schedule
$((\mathit{tk} \in \mathit{pts})$
$\wedge \ (0 < \mathit{bigp})$
$\wedge \ \text{periodic-tasksp}(\mathit{pts})$
$\wedge \ \text{expanded-tasksp}(\mathit{pts}, \mathit{bigp}))$
$\rightarrow \ (\text{occurrences}(\text{car}(\mathit{tk}), \text{substring-schedule}(\mathit{pts}, \mathit{bigp}))$
$\quad = \ ((\mathit{bigp} * \text{caddr}(\mathit{tk})) \div \text{cadr}(\mathit{tk})))$

THEOREM: times-quotient-quotient-special
$(((x \ \textbf{mod} \ \mathit{bigp}) = 0) \wedge (((\mathit{bigp} * y) \ \textbf{mod} \ x) = 0) \wedge (0 < \mathit{bigp}))$
$\rightarrow \ (((x \div \mathit{bigp}) * ((\mathit{bigp} * y) \div x)) = \text{fix}(y))$

THEOREM: good-schedule-periodic-task-requests
$((\mathit{tk} \in \mathit{pts})$
$\wedge \ \text{periodic-tasksp}(\mathit{pts})$
$\wedge \ \text{expanded-tasksp}(\mathit{pts}, \mathit{bigp})$
$\wedge \ (0 < \mathit{bigp})$
$\wedge \ (\mathit{n2} \not< \mathit{n1})$
$\wedge \ (\mathit{n2} \in \mathbf{N})$
$\wedge \ (\mathit{n1} \in \mathbf{N})$

$\land \quad ((\text{cadr}(tk) \textbf{ mod } bigp) \simeq 0)$
$\land \quad (((bigp * \text{caddr}(tk)) \textbf{ mod } \text{cadr}(tk)) \simeq 0)$
$\land \quad ((n1 \textbf{ mod } bigp) \simeq 0)$
$\land \quad ((n1 \textbf{ mod } \text{cadr}(tk)) \simeq 0)$
$\land \quad ((n2 \textbf{ mod } bigp) \simeq 0)$
$\land \quad ((n2 \textbf{ mod } \text{big-period}(pts)) \simeq 0)$
$\land \quad ((n2 \textbf{ mod } \text{cadr}(tk)) \simeq 0)$
$\land \quad (bigp \not< \text{length}(\text{substring-schedule}(pts,\ bigp))))$
$\rightarrow \quad \text{good-schedule}(\text{repeat-list}(\text{make-length}(bigp,\ \text{substring-schedule}(pts,\ bigp),\ \textbf{nil}),\ n2 \div bigp),\ \text{periodic-task-requests}(tk,\ n1,\ n2))$

THEOREM: member-sublistp
$(\text{sublistp}(x,\ y) \land (e \in x)) \rightarrow (e \in y)$

THEOREM: member-expanded-tasksp-means
$(\text{expanded-tasksp}(pts,\ bigp) \land (tk \in pts))$
$\rightarrow \quad (((\text{cadr}(tk) \textbf{ mod } bigp) = 0)$
$\qquad \land \quad (((bigp * \text{caddr}(tk)) \textbf{ mod } \text{cadr}(tk)) = 0))$

THEOREM: remainder-period-if-remainder-big-period
$(\text{periodic-tasksp}(pts) \land (tk \in pts) \land ((n \textbf{ mod } \text{big-period}(pts)) \simeq 0))$
$\rightarrow \quad ((n \textbf{ mod } \text{cadr}(tk)) = 0)$

THEOREM: good-simple-schedule-sublist
$((bigp \not< \text{length}(\text{substring-schedule}(pts2,\ bigp)))$
$\land \quad (n1 = 0)$
$\land \quad \text{sublistp}(pts1,\ pts2)$
$\land \quad \text{periodic-tasksp}(pts2)$
$\land \quad \text{expanded-tasksp}(pts2,\ bigp)$
$\land \quad (0 < bigp)$
$\land \quad (n2 \in \mathbf{N})$
$\land \quad ((n2 \textbf{ mod } bigp) \simeq 0)$
$\land \quad ((n2 \textbf{ mod } \text{big-period}(pts2)) \simeq 0))$
$\rightarrow \quad \text{good-schedule}(\text{make-simple-schedule}(pts2,\ bigp,\ n2),\ \text{periodic-tasks-requests}(pts1,\ n1,\ n2))$

THEOREM: sublistp-x-x
$\text{sublistp}(x,\ x)$

THEOREM: periodic-tasks-requests-simple
$(n1 \not< n2) \rightarrow (\text{periodic-tasks-requests}(pts,\ n1,\ n2) = \textbf{nil})$

`;; big theorem`

THEOREM: good-simple-schedule
$((bigp \not< \text{length}(\text{substring-schedule}(pts,\ bigp)))$
$\wedge\quad \text{periodic-tasksp}(pts)$
$\wedge\quad \text{expanded-tasksp}(pts,\ bigp)$
$\wedge\quad (0 < bigp)$
$\wedge\quad ((n\ \textbf{mod}\ bigp) \simeq 0)$
$\wedge\quad ((n\ \textbf{mod}\ \text{big-period}(pts)) \simeq 0))$
$\rightarrow\quad \text{good-schedule}(\text{make-simple-schedule}(pts,\ bigp,\ n),$
$\qquad\qquad \text{periodic-tasks-requests}(pts,\ 0,\ n))$

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DEFINITION:
expand-tasks $(pts,\ bigp)$
$=$ **if** listp (pts)
then cons (list (caar (pts), $bigp *$ cadar (pts), $bigp *$ caddar (pts)),
expand-tasks (cdr (pts), $bigp$))
else nil endif

```
;;;; expanded-tasksp identifies expand-tasks
```

THEOREM: expanded-tasksp-expand-task-helper
$(((x\ \textbf{mod}\ \text{big-period}(pts1)) = 0) \wedge (x \not\simeq 0))$
$\rightarrow\quad \text{expanded-tasksp}(\text{expand-tasks}(pts1,\ x),\ x)$

THEOREM: zerop-big-period
$\text{periodic-tasksp}(pts) \rightarrow (0 < \text{big-period}(pts))$

THEOREM: expanded-tasksp-expand-task
$\text{periodic-tasksp}(pts)$
$\rightarrow\quad \text{expanded-tasksp}(\text{expand-tasks}(pts,\ \text{big-period}(pts)),\ \text{big-period}(pts))$

THEOREM: assoc-expand-tasks
$\text{periodic-tasksp}(x) \rightarrow (\text{assoc}(v,\ \text{expand-tasks}(x,\ n)) \leftrightarrow \text{assoc}(v,\ x))$

THEOREM: periodic-tasksp-expand-tasks
$\text{periodic-tasksp}(pts)$
$\rightarrow\quad (\text{periodic-tasksp}(\text{expand-tasks}(pts,\ n))$
$\qquad = \quad ((n \not\simeq 0) \vee (\neg\ \text{listp}(pts))))$

DEFINITION:
non-overlapping-requests3 $(request,\ request\text{-}list)$
$=$ **if** listp $(request\text{-}list)$
then $((\text{car}(request) \neq \text{caar}(request\text{-}list))$
$\quad\vee\quad (\text{cadr}(request) \not< \text{caddar}(request\text{-}list))$

$\lor$ (cadar (*request-list*) $\not<$ caddr (*request*))
$\lor$ (car (*request-list*) $=$ *request*))
$\land$ non-overlapping-requests3 (*request*, cdr (*request-list*))
else t endif

DEFINITION:
non-overlapping-requests2 (*r1*, *r2*)
= **if** listp (*r1*)
then non-overlapping-requests3 (car (*r1*), *r2*)
$\land$ non-overlapping-requests2 (cdr (*r1*), *r2*)
else t endif

DEFINITION:
non-overlapping-requests (*r*) = non-overlapping-requests2 (*r*, *r*)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DEFINITION:
double-cdr-induction (a, b)
= **if** listp (a) **then** double-cdr-induction (cdr (a), cdr (b))
else t endif

THEOREM: plist-firstn
plist (firstn (n, l)) = firstn (n, l)

THEOREM: cons-nth-nthcdr
(n < length (l)) $\rightarrow$ (cons (nth (n, l), nthcdr (n, cdr (l))) = nthcdr (n, l))

THEOREM: equal-append
(append (a, b) = y)
= **if** $y \simeq$ **nil then** ($a \simeq$ **nil**) $\land$ ($b = y$)
else (length (y) $\not<$ length (a))
$\land$ (firstn (length (a), y) = plist (a))
$\land$ (nthcdr (length (a), y) = b) **endif**

THEOREM: replace-nth-replace-nth
(fix (i) $\neq$ fix (j))
$\rightarrow$ (replace-nth (i, x, replace-nth (j, y, l))
= replace-nth (j, y, replace-nth (i, x, l)))

THEOREM: replace-nth-idempotent
(fix (i) = fix (j))
$\rightarrow$ (replace-nth (i, x, replace-nth (j, y, l)) = replace-nth (i, x, l))

THEOREM: member-nth
(n < length (*list*)) $\rightarrow$ (nth (n, *list*) $\in$ *list*)

THEOREM: swap-commutative
$\text{swap}(j,\ i,\ s) = \text{swap}(i,\ j,\ s)$

THEOREM: member-replace-nth
$(x \notin l)$
$\rightarrow\ ((x \in \text{replace-nth}(i,\ v,\ l))$
$\quad = ((x = v) \vee ((x = 0) \wedge (\text{length}(l) < i))))$

THEOREM: occurrences-replace-nth
$\text{occurrences}(x,\ \text{replace-nth}(n,\ v,\ \mathit{list}))$
$=\ ((\text{occurrences}(x,\ \mathit{list})$
$\quad +\ \textbf{if}\ v = x\ \textbf{then}\ 1$
$\quad\quad \textbf{else}\ 0\ \textbf{endif}$
$\quad +\ \textbf{if}\ x = 0\ \textbf{then}\ n - \text{length}(\mathit{list})$
$\quad\quad \textbf{else}\ 0\ \textbf{endif})$
$\quad -\ \textbf{if}\ (x = \text{nth}(n,\ \mathit{list})) \wedge (n < \text{length}(\mathit{list}))\ \textbf{then}\ 1$
$\quad\quad \textbf{else}\ 0\ \textbf{endif})$

THEOREM: nthcdr-1
$\text{nthcdr}(1,\ x) = \text{cdr}(x)$

THEOREM: nlistp-nthcdr
$(\neg\ \text{listp}(s))$
$\rightarrow\ (\text{nthcdr}(n,\ s)$
$\quad = \textbf{if}\ n \simeq 0\ \textbf{then}\ s$
$\quad\quad \textbf{else}\ 0\ \textbf{endif})$

THEOREM: nthcdr-firstn-plus
$(\text{nthcdr}(n,\ \text{firstn}(n + x,\ s)) = \text{firstn}(x,\ \text{nthcdr}(n,\ s)))$
$\wedge\ (\text{nthcdr}(n,\ \text{firstn}(x + n,\ s)) = \text{firstn}(x,\ \text{nthcdr}(n,\ s)))$

THEOREM: cdr-nthcdr-cons
$\text{cdr}(\text{nthcdr}(x,\ \text{cons}(a,\ b))) = \text{nthcdr}(x,\ b)$

DEFINITION:
$\text{add1-sub1-induct}(a,\ b)$
$=\ \textbf{if}\ b \simeq 0\ \textbf{then}\ \textbf{t}$
$\quad \textbf{else}\ \text{add1-sub1-induct}(1 + a,\ b - 1)\ \textbf{endif}$

THEOREM: nth-nthcdr
$\text{nth}(\mathit{n1},\ \text{nthcdr}(\mathit{n2},\ s)) = \text{nth}(\mathit{n1} + \mathit{n2},\ s)$

THEOREM: nthcdr-nthcdr
$\text{nthcdr}(\mathit{n1},\ \text{nthcdr}(\mathit{n2},\ s)) = \text{nthcdr}(\mathit{n1} + \mathit{n2},\ s)$

THEOREM: equal-append-a-append-a
$(\text{append}(a,\ x) = \text{append}(a,\ y)) = (x = y)$

THEOREM: nthcdr-replace-nth
nthcdr $(n,$ replace-nth $(i,\ v,\ l))$
$=$ **if** $i < n$ **then** nthcdr $(n,\ l)$
else replace-nth $(i - n,\ v,$ nthcdr $(n,\ l))$ **endif**

THEOREM: firstn-replace-nth
firstn $(n,$ replace-nth $(i,\ v,\ l))$
$=$ **if** $i < n$ **then** replace-nth $(i,\ v,$ firstn $(n,\ l))$
else firstn $(n,\ l)$ **endif**

THEOREM: cdr-firstn-cons
cdr (firstn $(n,$ cons $(a,\ b)))$
$=$ **if** $n \simeq 0$ **then** 0
else firstn $(n - 1,\ b)$ **endif**

THEOREM: nth-firstn
nth $(n,$ firstn $(n2,\ s))$
$=$ **if** $n < n2$ **then** nth $(n,\ s)$
else 0 **endif**

THEOREM: firstn-cons
firstn $(n,$ cons $(a,\ b))$
$=$ **if** $n \simeq 0$ **then nil**
else cons $(a,$ firstn $(n - 1,\ b))$ **endif**

DEFINITION:
double-sub1-induction $(a,\ b)$
$=$ **if** $a \simeq 0$ **then t**
else double-sub1-induction $(a - 1,\ b - 1)$ **endif**

THEOREM: equal-repeat-repeat
(repeat $(n,\ v) =$ repeat $(n2,\ v2))$
$=$ ((fix $(n) =$ fix $(n2)) \land ((v = v2) \lor (n \simeq 0)))$

THEOREM: firstn-too-big
(length $(x) < n)$
$\rightarrow$ (firstn $(n,\ x) =$ append $(x,$ repeat $(n -$ length $(x),\ 0)))$

THEOREM: firstn-firstn
firstn $(a,$ firstn $(b,\ x))$
$=$ **if** $b < a$ **then** append (firstn $(b,\ x),$ repeat $(a - b,\ 0))$
else firstn $(a,\ x)$ **endif**

THEOREM: plist-repeat
plist (repeat $(n,\ x)) =$ repeat $(n,\ x)$

THEOREM: firstn-1
$\text{firstn}(\mathtt{1},\ s) = \text{list}(\text{car}(s))$

THEOREM: nthcdr-cons-firstn
$\text{nthcdr}(w,\ \text{cons}(a,\ \text{firstn}(w + x,\ b)))$
$= \ \ \textbf{if}\ w \simeq \mathtt{0}\ \ \textbf{then}\ \text{cons}(a,\ \text{firstn}(x,\ b))$
$\quad \textbf{else}\ \text{nthcdr}(w - 1,\ \text{firstn}(w + x,\ b))\ \textbf{endif}$

THEOREM: nthcdr-sub1-firstn-plus
$(w \not\simeq \mathtt{0})$
$\rightarrow \ \ (\text{nthcdr}(w - 1,\ \text{firstn}(w + x,\ b)) = \text{firstn}(1 + x,\ \text{nthcdr}(w - 1,\ b)))$

THEOREM: nthcdr-repeat
$\text{nthcdr}(n1,\ \text{repeat}(n2,\ x))$
$= \ \ \textbf{if}\ n2 < n1\ \ \textbf{then}\ \mathtt{0}$
$\quad \textbf{else}\ \text{repeat}(n2 - n1,\ x)\ \textbf{endif}$

THEOREM: firstn-nlistp
$(l \simeq \textbf{nil}) \rightarrow (\text{firstn}(n,\ l) = \text{repeat}(n,\ \mathtt{0}))$

DEFINITION:
$\text{member-nth-firstn-induction}(i,\ y,\ s)$
$= \ \ \textbf{if}\ i \simeq \mathtt{0}\ \ \textbf{then}\ \textbf{t}$
$\quad \textbf{else}\ \text{member-nth-firstn-induction}(i - 1,\ y - 1,\ \text{cdr}(s))\ \textbf{endif}$

THEOREM: member-nth-firstn
$(i < j) \rightarrow (\text{nth}(i,\ s) \in \text{firstn}(j,\ s))$

THEOREM: member-car-firstn
$(\text{car}(x) \in \text{firstn}(n,\ x)) = (n \not\simeq \mathtt{0})$

THEOREM: member-nth-firstn-nthcdr
$((i \not< y) \land (i < (x + y))) \rightarrow (\text{nth}(i,\ s) \in \text{firstn}(x,\ \text{nthcdr}(y,\ s)))$

THEOREM: non-overlapping-requests-means
$\text{non-overlapping-requests}(r) \rightarrow \text{non-overlapping-requests3}(\text{car}(r),\ \text{cdr}(r))$

THEOREM: non-overlapping-requests2-cdr
$\text{non-overlapping-requests2}(r1,\ r2)$
$\rightarrow \ \ \text{non-overlapping-requests2}(\text{cdr}(r1),\ \text{cdr}(r2))$

THEOREM: non-overlapping-requests-cdr
$\text{non-overlapping-requests}(r) \rightarrow \text{non-overlapping-requests}(\text{cdr}(r))$

THEOREM: member-firstn-only-if-member
$(x \notin l) \rightarrow ((x \in \text{firstn}(n,\ l)) = ((x = \mathtt{0}) \land (\text{length}(l) < n)))$

THEOREM: member-nthcdr-only-if-member
$(x \notin l) \rightarrow (x \notin \text{nthcdr}(n, l))$

THEOREM: nth-replace-nth
$$
\begin{aligned}
&\text{nth}(i, \text{replace-nth}(j, v, l)) \\
= \quad &\textbf{if } \text{fix}(i) = \text{fix}(j) \textbf{ then } v \\
&\textbf{else } \text{nth}(i, l) \textbf{ endif}
\end{aligned}
$$

THEOREM: member-x-firstn-cons-x
$(x \in \text{firstn}(n, \text{cons}(x, y))) = (\mathtt{0} < n)$

DEFINITION:
$$
\begin{aligned}
&\text{cars-non-nil-litatoms}(\textit{list}) \\
= \quad &\textbf{if } \text{listp}(\textit{list}) \\
&\textbf{then } \text{litatom}(\text{caar}(\textit{list})) \\
&\qquad \wedge \quad (\text{caar}(\textit{list}) \neq \textbf{nil}) \\
&\qquad \wedge \quad \text{cars-non-nil-litatoms}(\text{cdr}(\textit{list})) \\
&\textbf{else t endif}
\end{aligned}
$$

THEOREM: swap-preserves-good-schedule
$$
\begin{aligned}
&(\text{non-overlapping-requests3}(\textit{task-request-a}, r) \\
\wedge \quad &\text{non-overlapping-requests3}(\textit{task-request-b}, r) \\
\wedge \quad &\text{cars-non-nil-litatoms}(r) \\
\wedge \quad &(i \not< \text{cadr}(\textit{task-request-a})) \\
\wedge \quad &(j \not< \text{cadr}(\textit{task-request-a})) \\
\wedge \quad &(i < \text{caddr}(\textit{task-request-a})) \\
\wedge \quad &(j < \text{caddr}(\textit{task-request-a})) \\
\wedge \quad &(i \not< \text{cadr}(\textit{task-request-b})) \\
\wedge \quad &(j \not< \text{cadr}(\textit{task-request-b})) \\
\wedge \quad &(i < \text{caddr}(\textit{task-request-b})) \\
\wedge \quad &(j < \text{caddr}(\textit{task-request-b})) \\
\wedge \quad &(\text{nth}(i, s) = \text{car}(\textit{task-request-a})) \\
\wedge \quad &(\text{nth}(j, s) = \text{car}(\textit{task-request-b})) \\
\wedge \quad &\text{litatom}(\text{nth}(i, s)) \\
\wedge \quad &\text{litatom}(\text{nth}(j, s)) \\
\wedge \quad &\text{good-schedule}(s, r)) \\
\rightarrow \quad &\text{good-schedule}(\text{swap}(i, j, s), r)
\end{aligned}
$$

```
; return task request corresponding to task at time
```

DEFINITION:
$$
\begin{aligned}
&\text{corresponding-request}(\textit{task}, \textit{time}, r) \\
= \quad &\textbf{if } \text{listp}(r) \\
&\textbf{then if } (\text{caar}(r) = \textit{task}) \\
&\qquad\qquad \wedge \quad (\textit{time} \not< \text{cadar}(r))
\end{aligned}
$$

$\wedge \quad (\textit{time} < \text{caddar}\,(r))$ **then** car (r)
else corresponding-request $(\textit{task}, \textit{time}, \text{cdr}\,(r))$ **endif**
else f endif

DEFINITION:
all-non-nil-corresponding $(s, r, \textit{time})$
$=$ **if** $\textit{time} < \text{length}\,(s)$
then (corresponding-request (nth $(\textit{time}, s)$, $\textit{time}$, r)
$\vee \quad (\text{nth}\,(\textit{time}, s) = \textbf{nil}))$
$\wedge \quad$ all-non-nil-corresponding $(s, r, 1 + \textit{time})$
else t endif

THEOREM: non-overlapping-requests2-member
(non-overlapping-requests2 $(r1, r2) \wedge$ sublistp $(r1, r2) \wedge (e \in r1))$
$\rightarrow$ non-overlapping-requests3 $(e, r2)$

THEOREM: non-overlapping-requests3-member
(non-overlapping-requests $(r) \wedge (e \in r))$
$\rightarrow$ non-overlapping-requests3 (e, r)

THEOREM: member-corresponding-request-nth
corresponding-request $(\textit{task}, \textit{time}, r)$
$\rightarrow$ (corresponding-request $(\textit{task}, \textit{time}, r) \in r)$

DEFINITION:
all-litatoms $(\textit{list})$
$=$ **if** listp $(\textit{list})$ **then** litatom (car $(\textit{list})) \wedge$ all-litatoms (cdr $(\textit{list}))$
else t endif

THEOREM: litatom-nth
all-litatoms $(s) \rightarrow$ (litatom (nth $(i, s)) = (i < \text{length}\,(s)))$

THEOREM: lessp-first-instance-s
(first-instance $(\textit{time}, \textit{task}, s) < \text{length}\,(s)) = \text{listp}\,(s)$

THEOREM: member-nthcdr-from-cdr
$((x \notin \text{nthcdr}\,(n, \text{cdr}\,(s))) \wedge (n < \text{length}\,(s)))$
$\rightarrow \quad ((x \in \text{nthcdr}\,(n, s)) = (x = \text{nth}\,(n, s)))$

THEOREM: listp-nthcdr
$\text{listp}\,(\text{nthcdr}\,(n, x)) = (n < \text{length}\,(x))$

THEOREM: car-nthcdr
car (nthcdr (n, l))
$=$ **if** $n < \text{length}\,(l)$ **then** nth (n, l)
else 0 **endif**

THEOREM: nth-first-instance-simple
$(\text{nth}(\text{first-instance}(n, x, s), s) = x)$
$= \quad ((x \in \text{nthcdr}(n, s)) \lor (\text{car}(s) = x))$

THEOREM: equal-occurrences-firstn-nthcdr
$(n1 < n2)$
$\rightarrow \quad ((\text{occurrences}(x, \text{firstn}(n1, \text{nthcdr}(n, list))) = \text{occurrences}(x, \text{firstn}(n2, \text{nthcdr}(n, list)))) = (x \notin \text{firstn}(n2 - n1, \text{nthcdr}(n + n1, list))))$

THEOREM: member-firstn-lessp
$((x \in \text{firstn}(n1, l)) \land (n2 \not< n1)) \rightarrow (x \in \text{firstn}(n2, l))$

THEOREM: member-least-deadline
$(\text{least-deadline}(r) \in r) = \text{listp}(r)$

THEOREM: unfulfilled-task-later-in-good-schedule
$(\text{good-schedule}(s, r)$
$\land \quad \text{all-litatoms}(s)$
$\land \quad \text{cars-non-nil-litatoms}(r)$
$\land \quad (tr \in \text{unfulfilled}(element, s, \text{active-task-requests}(element, r))))$
$\rightarrow \quad (\text{car}(tr) \in \text{nthcdr}(element, s))$

THEOREM: car-corresponding-request
$\text{corresponding-request}(task, time, r)$
$\rightarrow \quad (\text{car}(\text{corresponding-request}(task, time, r)) = task)$

THEOREM: active-task-has-later-deadline
$(tr \in \text{active-task-requests}(element, r)) \rightarrow (element < \text{caddr}(tr))$

THEOREM: sublistp-unfulfilled
$\text{sublistp}(\text{unfulfilled}(e, s, r), r)$

THEOREM: least-deadline-has-later-deadline
$\text{listp}(\text{unfulfilled}(element, s, \text{active-task-requests}(element, r)))$
$\rightarrow \quad ((element < \text{caddr}(\text{least-deadline}(\text{unfulfilled}(element, s, \text{active-task-requests}(element, r))))) = \mathbf{t})$

THEOREM: active-task-hasnt-earlier-start
$(tr \in \text{active-task-requests}(element, r)) \rightarrow (element \not< \text{cadr}(tr))$

THEOREM: lessp-corresponding-request-deadline
$\text{corresponding-request}(\text{nth}(element, s), element, r)$
$\rightarrow \quad ((element < \text{caddr}(\text{corresponding-request}(\text{nth}(element, s), element, r))) = \mathbf{t})$

THEOREM: least-deadline-hasnt-earlier-start

$$\begin{array}{l} \text{listp}\,(\text{unfulfilled}\,(\textit{element},\ s,\ \text{active-task-requests}\,(\textit{element},\ r))) \\ \to\quad ((\textit{element} < \text{cadr}\,(\text{least-deadline}\,(\text{unfulfilled}\,(\textit{element},\ s,\ \text{active-task-requests}\,(\textit{element},\ r))))) \\ \qquad = \quad \mathbf{f}) \end{array}$$

THEOREM: lessp-corresponding-request-start

$$\begin{array}{l} \text{corresponding-request}\,(x,\ \textit{time},\ r) \\ \to\quad (\textit{time} \not< \text{cadr}\,(\text{corresponding-request}\,(x,\ \textit{time},\ r))) \end{array}$$

THEOREM: lessp-corresponding-request-deadline-linear

$$\begin{array}{l} \text{corresponding-request}\,(x,\ \textit{time},\ r) \\ \to\quad (\textit{time} < \text{caddr}\,(\text{corresponding-request}\,(x,\ \textit{time},\ r))) \end{array}$$

THEOREM: sublistp-active-task-requests

$$\text{sublistp}\,(\text{active-task-requests}\,(\textit{time},\ r),\ r)$$

THEOREM: member-means-all-cars-not-litatoms

$$((x \in r) \land (\neg\ \text{litatom}\,(\text{car}\,(x)))) \to (\neg\ \text{cars-non-nil-litatoms}\,(r))$$

THEOREM: member-least-deadline-unfulfilled

$$\begin{array}{l} \text{cars-non-nil-litatoms}\,(r) \\ \to\quad ((\text{least-deadline}\,(\text{unfulfilled}\,(\textit{element},\ s,\ \text{active-task-requests}\,(\textit{element},\ r))) \\ \qquad \in \quad r) \\ \qquad = \quad \text{listp}\,(\text{unfulfilled}\,(\textit{element},\ s,\ \text{active-task-requests}\,(\textit{element},\ r)))) \end{array}$$

THEOREM: not-numberp-corresponding

$$\begin{array}{l} (\textit{element} \notin \mathbf{N}) \\ \to\quad (\text{corresponding-request}\,(x,\ \textit{element},\ r) = \text{corresponding-request}\,(x,\ 0,\ r)) \end{array}$$

THEOREM: member-firstn-means-lessp-first-instance

$$\begin{array}{l} (v \in \text{firstn}\,(n - \textit{time},\ \text{nthcdr}\,(\textit{time},\ s))) \\ \to\quad ((\text{first-instance}\,(\textit{time},\ v,\ s) < n) = \mathbf{t}) \end{array}$$

THEOREM: first-instance-member-unfulfilled-not-past-deadline

$$\begin{array}{l} (\text{good-schedule}\,(s,\ r) \\ \ \land\quad (\textit{tr} \in \text{unfulfilled}\,(\textit{element},\ s,\ \text{active-task-requests}\,(\textit{element},\ r)))) \\ \to\quad ((\text{first-instance}\,(\textit{element},\ \text{car}\,(\textit{tr}),\ s) < \text{caddr}\,(\textit{tr})) = \mathbf{t}) \end{array}$$

THEOREM: lessp-firstn-instance-time

$$\begin{array}{l} (\textit{time} \not< n) \\ \to\quad ((\text{first-instance}\,(\textit{time},\ v,\ s) < n) \\ \qquad = \quad ((v \notin \text{nthcdr}\,(\textit{time},\ s)) \land (n \not\simeq 0))) \end{array}$$

THEOREM: first-instance-member-unfulfilled-not-before-start
(good-schedule $(s,\ r)$
 $\wedge$ cars-non-nil-litatoms (r)
 $\wedge$ $(tr \in$ unfulfilled (*element*, s, active-task-requests (*element*, r))))
$\rightarrow$ ((first-instance (*element*, car (tr), s) $<$ cadr (tr)) $=$ **f**)

THEOREM: member-corresponding-request
(good-schedule $(s,\ r)$ $\wedge$ corresponding-request (nth (*element*, s), *element*, r))
$\rightarrow$ (corresponding-request (nth (*element*, s), *element*, r)
 $\in$ unfulfilled (*element*, s, active-task-requests (*element*, r)))

DEFINITION:
member-deadline-induct $(x,\ l)$
= **if** listp (l)
 then if listp (cdr (l))
  then if caddar (l) $<$ caddadr (l)
   then if $x =$ cadr (l)
    then member-deadline-induct (car (l),
     cons (car (l), cddr (l)))
    else member-deadline-induct (x,
     cons (car (l), cddr (l))) **endif**
   elseif $x =$ car (l)
   then member-deadline-induct (cadr (l), cons (cadr (l), cddr (l)))
   else member-deadline-induct (x, cons (cadr (l), cddr (l))) **endif**
  else t endif
 else t endif

THEOREM: member-deadline-not-less-than-least-deadline
$(x \in l) \rightarrow$ ((caddr (x) $<$ caddr (least-deadline (l))) $=$ **f**)

THEOREM: first-instance-member-unfulfilled-not-past-deadline-better
(good-schedule $(s,\ r)$
 $\wedge$ $(tr \in$ unfulfilled (*element*, s, active-task-requests (*element*, r)))
 $\wedge$ (caddr (*tr2*) $\not<$ caddr (tr)))
$\rightarrow$ ((first-instance (*element*, car (tr), s) $<$ caddr (*tr2*)) $=$ **t**)

THEOREM: non-overlapping-requests3-simple
(cars-non-nil-litatoms (r) $\wedge$ ($\neg$ listp (tr)))
$\rightarrow$ non-overlapping-requests3 (tr, r)

THEOREM: swap-preserves-good-schedule-simple
(non-overlapping-requests3 (*task-request*, r)
 $\wedge$ cars-non-nil-litatoms (r)
 $\wedge$ ($i \not<$ cadr (*task-request*))
 $\wedge$ ($j \not<$ cadr (*task-request*))

$\wedge$ $(i <$ caddr $(task\text{-}request))$
$\wedge$ $(j <$ caddr $(task\text{-}request))$
$\wedge$ (nth $(j,\ s)$ = car $(task\text{-}request))$
$\wedge$ (nth $(i,\ s)$ = **nil**)
$\wedge$ litatom (nth $(j,\ s)$)
$\wedge$ good-schedule $(s,\ r))$
$\rightarrow$ good-schedule (swap $(i,\ j,\ s),\ r)$

Theorem: nth-too-big
$(i \not< \text{length}(s)) \rightarrow (\text{nth}(i,\ s) = 0)$

Theorem: not-corresponding-request-means-nil
(all-non-nil-corresponding $(s,\ r,\ n)$
$\wedge$ ($\neg$ corresponding-request (nth $(i,\ s),\ i,\ r$))
$\wedge$ $(i \not< n))$
$\rightarrow$ (nth $(i,\ s)$
= **if** $i <$ length (s) **then nil**
else 0 **endif**)

```
;; takes 1/2 hr without disable-theory t
```

Theorem: make-element-edf-preserves-good-schedule
(($element <$ length $(s))$
$\wedge$ non-overlapping-requests (r)
$\wedge$ cars-non-nil-litatoms (r)
$\wedge$ all-litatoms (s)
$\wedge$ all-non-nil-corresponding $(s,\ r,\ 0)$
$\wedge$ good-schedule $(s,\ r))$
$\rightarrow$ good-schedule (make-element-edf $(s,\ r,\ element),\ r)$

Theorem: all-litatoms-replace-nth
all-litatoms (replace-nth $(n,\ v,\ x)$)
= (litatom (v)
$\wedge$ (length $(x) \not< n$)
$\wedge$ all-litatoms (firstn $(n,\ x)$)
$\wedge$ all-litatoms (nthcdr $(1 + n,\ x)$)))

Theorem: all-litatoms-nthcdr
all-litatoms $(s) \rightarrow$ all-litatoms (nthcdr $(n,\ s)$)

Theorem: all-litatoms-repeat
all-litatoms (repeat $(n,\ v)$) = $((n \simeq 0) \vee$ litatom $(v))$

Theorem: all-litatoms-firstn
all-litatoms $(s) \rightarrow$ (all-litatoms (firstn $(n,\ s)$) = (length $(s) \not< n$))

THEOREM: make-element-edf-preserves-all-litatoms
$((\mathit{element} < \mathrm{length}(s))$
$\land$ non-overlapping-requests (r)
$\land$ cars-non-nil-litatoms (r)
$\land$ all-litatoms (s)
$\land$ all-non-nil-corresponding $(s, r, 0)$
$\land$ good-schedule $(s, r))$
$\rightarrow$ all-litatoms (make-element-edf $(s, r, \mathit{element})$)

THEOREM: equal-lessp-sub1x-y-x-y
$(((x - 1) < y) = (x < y)) = ((x \simeq 0) \lor (\mathrm{fix}(x) \neq \mathrm{fix}(y)))$

THEOREM: all-non-nil-corresponding-replace-simple
(all-non-nil-corresponding $(s, r, n) \land (\mathit{n1} < \mathrm{length}(s)))$
$\rightarrow$ (all-non-nil-corresponding (replace-nth $(\mathit{n1}, v, s), r, n)$
$=$ (corresponding-request $(v, \mathit{n1}, r) \lor (v = \mathbf{nil}) \lor (\mathit{n1} < n)))$

THEOREM: car-replace-nth
car (replace-nth (n, v, s))
$=$ **if** $n \simeq 0$ **then** v
else car (s) **endif**

THEOREM: not-corresponding-request-means
$((\neg$ corresponding-request $(\mathit{v1}, \mathit{n1}, r))$
$\land$ $(\mathit{v1} \neq \mathbf{nil})$
$\land$ $(\mathrm{nth}(\mathit{n1}, s) = \mathit{v1})$
$\land$ $(\mathit{n1} < \mathrm{length}(s))$
$\land$ $(\mathit{n1} \not< n))$
$\rightarrow$ $(\neg$ all-non-nil-corresponding $(s, r, n))$

THEOREM: all-non-nil-corresponding-cons
all-non-nil-corresponding (cons $(a, b), r, n)$
$\rightarrow$ (all-non-nil-corresponding (cons $(x, b), r, n)$
$=$ $((x = \mathbf{nil}) \lor$ corresponding-request $(x, 0, r) \lor (n \not\simeq 0)))$

THEOREM: all-non-nil-corresponding-replace-replace
(all-non-nil-corresponding (s, r, n)
$\land$ $(\mathit{n1} < \mathrm{length}(s))$
$\land$ $(\mathit{n2} < \mathrm{length}(s)))$
$\rightarrow$ (all-non-nil-corresponding (replace-nth $(\mathit{n2}, \mathit{v2},$ replace-nth $(\mathit{n1}, \mathit{v1}, s)),$
$r,$
$n)$
$=$ ((corresponding-request $(\mathit{v1}, \mathit{n1}, r)$
$\lor$ $(\mathit{v1} = \mathbf{nil})$
$\lor$ $(\mathit{n1} < n)$

$\vee \quad (\text{fix}(n1) = \text{fix}(n2)))$
$\wedge \quad (\text{corresponding-request}(v2,\ n2,\ r)$
$\quad \vee \quad (v2 = \textbf{nil})$
$\quad \vee \quad (n2 < n))))$

THEOREM: nth-first-instance
$(v \in \text{nthcdr}(time,\ s)) \rightarrow (\text{nth}(\text{first-instance}(time,\ v,\ s),\ s) = v)$

THEOREM: car-corresponding-request-better
$\text{car}(\text{corresponding-request}(v,\ n,\ r))$
$= \quad \textbf{if } \text{corresponding-request}(v,\ n,\ r) \textbf{ then } v$
$\quad \textbf{else } 0 \textbf{ endif}$

THEOREM: equivalent-corresponding-requests
$(\text{non-overlapping-requests3}(\text{corresponding-request}(v,\ n,\ r),\ r)$
$\wedge \quad (n1 \not< \text{cadr}(\text{corresponding-request}(v,\ n,\ r)))$
$\wedge \quad (n1 < \text{caddr}(\text{corresponding-request}(v,\ n,\ r))))$
$\rightarrow \quad (\text{corresponding-request}(v,\ n1,\ r) = \text{corresponding-request}(v,\ n,\ r))$

THEOREM: lessp-n-1
$(n < 1) = (n \simeq 0)$

THEOREM: member-corresponding-request-simplify
$\text{cars-non-nil-litatoms}(r)$
$\rightarrow \quad ((\text{corresponding-request}(v,\ n,\ r) \in r)$
$\quad \leftrightarrow \quad \text{corresponding-request}(v,\ n,\ r))$

THEOREM: member-corresponding-request2
$(\text{all-non-nil-corresponding}(s,\ r,\ n1)$
$\wedge \quad (n \not< n1)$
$\wedge \quad (n < \text{length}(s))$
$\wedge \quad \text{cars-non-nil-litatoms}(r)$
$\wedge \quad (\text{nth}(n,\ s) \neq \textbf{nil}))$
$\rightarrow \quad (\text{corresponding-request}(\text{nth}(n,\ s),\ n,\ r) \in r)$

```
;;; the best thereom would be about make-element-edf's preserving
;;; all-non-nil-corresponding.  But we'll punt on that for now,
;;; and use the fact that periodic request schedules are "full"
;;; and thus all reasonable schedules have corresponding requests
```

DEFINITION:
$\text{all-nils-or-cars}(s,\ r)$
$= \quad \textbf{if } \text{listp}(s)$
$\quad \textbf{then } ((\text{car}(s) = \textbf{nil}) \vee \text{assoc}(\text{car}(s),\ r))$
$\qquad \wedge \quad \text{all-nils-or-cars}(\text{cdr}(s),\ r)$
$\quad \textbf{else t endif}$

THEOREM: corresponding-request-append
corresponding-request $(v,\ n,\ \text{append}(r1,\ r2))$
$\leftrightarrow$ (corresponding-request $(v,\ n,\ r1) \lor$ corresponding-request $(v,\ n,\ r2)$)

THEOREM: corresponding-request-different-name
$(\text{car}(pt) \neq tk)$
$\rightarrow$ ($\neg$ corresponding-request $(tk,\ n,$ periodic-task-requests $(pt,\ n1,\ n2)))$

THEOREM: corresponding-request-periodic-task
(periodic-taskp (pt)
$\land$ $((n1 \textbf{ mod } \text{cadr}(pt)) = 0)$
$\land$ $(n < n2)$
$\land$ $(n \not< n1))$
$\rightarrow$ (corresponding-request $(tk,\ n,$ periodic-task-requests $(pt,\ n1,\ n2))$
$\leftrightarrow$ $(\text{car}(pt) = tk))$

THEOREM: all-nils-or-cars-nlistp
$(\neg \text{listp}(x))$
$\rightarrow$ (all-nils-or-cars $(l,\ x) = (\text{plist}(l) = \text{repeat}(\text{length}(l),\ \textbf{nil})))$

THEOREM: assoc-nth-pts
$((n < \text{length}(s)) \land$ all-nils-or-cars $(s,\ pts) \land (\text{nth}(n,\ s) \neq \textbf{nil}))$
$\rightarrow$ assoc $(\text{nth}(n,\ s),\ pts)$

THEOREM: corresponding-request-periodic-tasks
(assoc $(v,\ pts)$
$\land$ periodic-tasksp (pts)
$\land$ $((n1 \textbf{ mod } \text{cadr}(\text{assoc}(v,\ pts))) = 0)$
$\land$ $(n < n2)$
$\land$ $(n \not< n1))$
$\rightarrow$ corresponding-request $(v,\ n,$ periodic-tasks-requests $(pts,\ n1,\ n2))$

THEOREM: all-nils-or-cars-replace-nth
(all-nils-or-cars $(s,\ r) \land (n < \text{length}(s)))$
$\rightarrow$ (all-nils-or-cars (replace-nth $(n,\ v,\ s),\ r)$
$=$ $((v = \textbf{nil}) \lor \text{assoc}(v,\ r)))$

THEOREM: all-non-nil-corresponding-periodic-requests
(all-nils-or-cars $(s,\ pts)$
$\land$ periodic-tasksp (pts)
$\land$ $(\text{fix}(n1) = \text{length}(s)))$
$\rightarrow$ all-non-nil-corresponding $(s,$ periodic-tasks-requests $(pts,\ 0,\ n1),\ n)$

DEFINITION:
double-sub1-cdr-induct $(n1,\ n2,\ s)$
$=$ **if** $n2 \simeq 0$ **then t**
else double-sub1-cdr-induct $(n1 - 1,\ n2 - 1,\ \text{cdr}(s))$ **endif**

THEOREM: all-nils-or-cars-replace-nth-replace-nth

$(\text{all-nils-or-cars}(s,\ r) \land (n < \text{length}(s)) \land (n2 < \text{length}(s)))$
$\rightarrow\ (\text{all-nils-or-cars}(\text{replace-nth}(n,\ v,\ \text{replace-nth}(n2,\ v2,\ s)),\ r)$
$\quad = (((v = \textbf{nil}) \lor \text{assoc}(v,\ r))$
$\qquad \land ((v2 = \textbf{nil})$
$\qquad\quad \lor \text{assoc}(v2,\ r)$
$\qquad\quad \lor (\text{fix}(n) = \text{fix}(n2)))))$

THEOREM: lessp-0-length-means-listp

$(n < \text{length}(l)) \rightarrow \text{listp}(l)$

THEOREM: all-nils-or-cars-make-element-edf

$(\text{all-nils-or-cars}(s,\ pts) \land \text{listp}(s) \land (n < \text{length}(s)))$
$\rightarrow\ \text{all-nils-or-cars}(\text{make-element-edf}(s,\ pts1,\ n),\ pts)$

THEOREM: make-schedule-edf-preserves-good-schedule

$(\text{non-overlapping-requests}(r)$
$\land\ \text{cars-non-nil-litatoms}(r)$
$\land\ \text{all-litatoms}(s)$
$\land\ \text{all-non-nil-corresponding}(s,\ r,\ 0)$
$\land\ \text{all-nils-or-cars}(s,\ pts)$
$\land\ \text{good-schedule}(s,\ r)$
$\land\ (r = \text{periodic-tasks-requests}(pts,\ 0,\ \text{length}(s)))$
$\land\ \text{periodic-tasksp}(pts))$
$\rightarrow\ \text{good-schedule}(\text{make-schedule-edf}(s,\ r,\ n),\ r)$

;;;;;;;;;;;

THEOREM: non-overlapping-requests2-append

$\text{non-overlapping-requests2}(\text{append}(a,\ b),\ r)$
$=\ (\text{non-overlapping-requests2}(a,\ r) \land \text{non-overlapping-requests2}(b,\ r))$

THEOREM: all-nils-or-cars-plist

$\text{all-nils-or-cars}(\text{plist}(a),\ pts) = \text{all-nils-or-cars}(a,\ pts)$

THEOREM: all-nils-or-cars-append

$\text{all-nils-or-cars}(\text{append}(a,\ b),\ pts)$
$=\ (\text{all-nils-or-cars}(a,\ pts) \land \text{all-nils-or-cars}(b,\ pts))$

THEOREM: all-nils-or-cars-repeat-list

$\text{all-nils-or-cars}(\text{repeat-list}(l,\ n),\ pts)$
$=\ ((n \simeq 0) \lor \text{all-nils-or-cars}(l,\ pts))$

THEOREM: all-nils-or-cars-make-length

$\text{all-nils-or-cars}(\text{make-length}(n,\ l,\ \textbf{nil}),\ pts)$
$=\ \textbf{if}\ n < \text{length}(l)\ \textbf{then}\ \text{all-nils-or-cars}(\text{firstn}(n,\ l),\ pts)$
$\quad \textbf{else}\ \text{all-nils-or-cars}(l,\ pts)\ \textbf{endif}$

THEOREM: all-nils-or-cars-repeat
all-nils-or-cars (repeat (n, v), pts)
$=$ ((v = **nil**) $\lor$ assoc (v, pts) $\lor$ ($n \simeq 0$))

THEOREM: all-nils-or-cars-substring-schedule
cars-non-nil-litatoms (pts)
$\rightarrow$ all-nils-or-cars (substring-schedule (pts, n), pts)

THEOREM: all-nils-or-cars-firstn
(all-nils-or-cars (l, pts) $\land$ ($n <$ length (l)))
$\rightarrow$ all-nils-or-cars (firstn (n, l), pts)

THEOREM: all-nils-or-cars-make-simple-schedule
cars-non-nil-litatoms (pts)
$\rightarrow$ all-nils-or-cars (make-simple-schedule (pts, $bigp$, n), pts)

THEOREM: length-make-simple-schedule
((n **mod** $bigp$) $= 0$)
$\rightarrow$ (length (make-simple-schedule (pts, $bigp$, n)) = fix (n))

THEOREM: periodic-tasksp-means-cars-non-nil-litatoms
periodic-tasksp (pts) $\rightarrow$ cars-non-nil-litatoms (pts)

THEOREM: all-litatoms-append
all-litatoms (append (a, b)) = (all-litatoms (a) $\land$ all-litatoms (b))

THEOREM: all-litatoms-repeat-list
all-litatoms (repeat-list (l, n)) = (($n \simeq 0$) $\lor$ all-litatoms (l))

THEOREM: all-litatoms-make-length
all-litatoms (make-length (n, l, v))
$=$ **if** length (l) $< n$ **then** all-litatoms (l) $\land$ litatom (v)
else all-litatoms (firstn (n, l)) **endif**

THEOREM: all-litatoms-substring-schedule
cars-non-nil-litatoms (pts) $\rightarrow$ all-litatoms (substring-schedule (pts, $bigp$))

THEOREM: all-litatoms-make-simple-schedule
cars-non-nil-litatoms (pts)
$\rightarrow$ all-litatoms (make-simple-schedule (pts, $bigp$, n))

THEOREM: cars-non-nil-litatoms-periodic-tasks
cars-non-nil-litatoms (pts)
$\rightarrow$ cars-non-nil-litatoms (periodic-tasks-requests (pts, $n1$, $n2$))

DEFINITION:
value-all-cars $(v,\ \mathit{list})$
$=$ **if** listp (list)
then $(\text{caar}(\mathit{list}) = v) \wedge \text{value-all-cars}(v,\ \text{cdr}(\mathit{list}))$
else t endif

THEOREM: non-overlapping-requests3-append
non-overlapping-requests3 $(v,\ \text{append}(r1,\ r2))$
$=$ (non-overlapping-requests3 $(v,\ r1) \wedge$ non-overlapping-requests3 $(v,\ r2))$

THEOREM: value-all-cars-periodic-task-requests
value-all-cars $(x,$ periodic-task-requests $(req,\ n1,\ n2))$
$=$ $((\neg$ listp (periodic-task-requests $(req,\ n1,\ n2))) \vee (x = \text{car}(req)))$

THEOREM: non-overlapping-requests2-value-all-cars
(value-all-cars $(v,\ \mathit{list}) \wedge (\neg\ \text{assoc}(v,\ r)) \wedge$ cars-non-nil-litatoms $(r))$
$\rightarrow$ (non-overlapping-requests2 $(r,\ \text{append}(\mathit{list},\ r2))$
$=$ non-overlapping-requests2 $(r,\ r2))$

THEOREM: assoc-append
$(v \neq 0) \rightarrow (\text{assoc}(v,\ \text{append}(x,\ y)) \leftrightarrow (\text{assoc}(v,\ x) \vee \text{assoc}(v,\ y)))$

THEOREM: assoc-periodic-task-requests
$(v \neq \text{car}(req)) \rightarrow (\neg$ assoc $(v,$ periodic-task-requests $(req,\ n1,\ n2)))$

THEOREM: assoc-periodic-tasks-requests
$((v \neq 0) \wedge (\neg\ \text{assoc}(v,\ \mathit{list})))$
$\rightarrow$ $(\neg$ assoc $(v,$ periodic-tasks-requests $(\mathit{list},\ n1,\ n2)))$

THEOREM: non-overlapping-requests2-append-arg2
$((\neg\ \text{assoc}(\text{car}(req),\ \mathit{list}))$
$\wedge$ $(\text{car}(req) \neq 0)$
$\wedge$ cars-non-nil-litatoms $(\mathit{list}))$
$\rightarrow$ (non-overlapping-requests2 (periodic-tasks-requests $(\mathit{list},\ n1,\ n2)$,
append (periodic-task-requests $(req,\ n3,\ n4),\ y))$
$=$ non-overlapping-requests2 (periodic-tasks-requests $(\mathit{list},\ n1,\ n2)$,
$y))$

THEOREM: non-overlapping-requests3-periodic-task
$(n1 \not< end)$
$\rightarrow$ non-overlapping-requests3 (list $(tk,\ start,\ end,\ duration)$,
periodic-task-requests $(pt,\ n1,\ n2))$

THEOREM: non-overlapping-requests3-task-name-difference
$(\text{car}(pt) \neq \text{car}(req))$
$\rightarrow$ non-overlapping-requests3 $(req,$ periodic-task-requests $(pt,\ n1,\ n2))$

THEOREM: non-overlapping-requests3-name-difference
(litatom (car (req)) $\wedge$ ($\neg$ assoc (car (req), $list$)))
$\rightarrow$ non-overlapping-requests3 (req, periodic-tasks-requests ($list$, $n1$, $n2$))

THEOREM: non-overlapping-requests2-cons-too-big
($n1 \not< $ caddr (req))
$\rightarrow$ (non-overlapping-requests2 (periodic-task-requests (tr, $n1$, $n2$),
cons (req, r))
= non-overlapping-requests2 (periodic-task-requests (tr, $n1$, $n2$), r))

THEOREM: non-overlapping-requests2-periodic-task
($\neg$ assoc (car (req), x))
$\rightarrow$ non-overlapping-requests2 (periodic-task-requests (req, $n1$, $n2$),
append (periodic-task-requests (req, $n1$, $n2$),
periodic-tasks-requests (x, $n3$, $n4$)))

THEOREM: non-overlapping-requests2-periodic-tasks
periodic-tasksp (pts)
$\rightarrow$ non-overlapping-requests2 (periodic-tasks-requests (pts, 0, n),
periodic-tasks-requests (pts, 0, n))

THEOREM: non-overlapping-requests-periodic-tasks-requests
periodic-tasksp (pts)
$\rightarrow$ non-overlapping-requests (periodic-tasks-requests (pts, 0, n))

```
;;;; major lemma
```

THEOREM: edf-schedule-good-for-expanded
(($bigp \not< $ length (substring-schedule (pts, $bigp$)))
$\wedge$ periodic-tasksp (pts)
$\wedge$ expanded-tasksp (pts, $bigp$)
$\wedge$ ($0 < bigp$)
$\wedge$ ((n **mod** $bigp$) $\simeq$ 0)
$\wedge$ ((n **mod** big-period (pts)) $\simeq$ 0))
$\rightarrow$ good-schedule (make-schedule-edf (make-simple-schedule (pts, $bigp$, n),
periodic-tasks-requests (pts, 0, n),
0),
periodic-tasks-requests (pts, 0, n))

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

DEFINITION:
every-nth (n, $list$)
= **if** $n \simeq 0$ **then nil**
elseif listp ($list$) **then** cons (car ($list$), every-nth (n, nthcdr (n, $list$)))
else nil endif

DEFINITION:
expand-tasks-requests (*tasks-requests*, n)
= **if** listp (*tasks-requests*)
then cons (list (caar (*tasks-requests*),
$n *$ cadar (*tasks-requests*),
$n *$ caddar (*tasks-requests*),
$n *$ cadddar (*tasks-requests*)),
expand-tasks-requests (cdr (*tasks-requests*), n))
else nil endif

THEOREM: expand-tasks-requests-append
expand-tasks-requests (append (a, b), n)
= append (expand-tasks-requests (a, n), expand-tasks-requests (b, n))

THEOREM: listp-append
listp (append (a, b)) = (listp (a) $\vee$ listp (b))

THEOREM: listp-expand-tasks-requests
listp (expand-tasks-requests (l, n)) = listp (l)

THEOREM: listp-task-requests
listp (periodic-task-requests (*task*, *n1*, *n2*))
= ((*n1* $<$ *n2*) $\wedge$ periodic-taskp (*task*))

THEOREM: length-expand-tasks-requests
length (expand-tasks-requests (l, n)) = length (l)

THEOREM: different-lengths-mean-different
($x = y$) $\rightarrow$ (length (x) = length (y))

THEOREM: equal-nthcdr-x-x
(nthcdr (n, x) = x) = (($n \simeq 0$) $\vee$ ($x = 0$))

DEFINITION:
length-periodic-task-request-induct (*n1*, *period*, *bigp*, *n2*)
= **if** 0 $<$ (*bigp* $*$ *period*)
then if *n1* $<$ *n2*
then length-periodic-task-request-induct (*n1* + (*bigp* $*$ *period*),
period,
bigp,
n2)
else t endif
else t endif

THEOREM: length-periodic-task-requests

$$
\begin{array}{l}
((0 < bigp) \\
\ \wedge\ \ (0 < period) \\
\ \wedge\ \ ((n1 \ \mathbf{mod}\ bigp) = 0) \\
\ \wedge\ \ ((n2 \ \mathbf{mod}\ bigp) = 0)) \\
\to\ \ (\text{length}(\text{periodic-task-requests}(\text{list}(name,\ bigp * period,\ duration),\ n1,\ n2)) \\
\qquad = \ \text{length}(\text{periodic-task-requests}(\text{list}(name,\ period,\ duration),\ n1 \div bigp,\ n2 \div bigp)))
\end{array}
$$

THEOREM: plist-expand-tasks-requests

$$\text{plist}(\text{expand-tasks-requests}(task,\ n)) = \text{expand-tasks-requests}(task,\ n)$$

THEOREM: equal-plist-nil

$$(\text{plist}(l) = \mathbf{nil}) = (l \simeq \mathbf{nil})$$

THEOREM: length-cons

$$\text{length}(\text{cons}(a,\ b)) = (1 + \text{length}(b))$$

THEOREM: cons-append-hack

$$\text{cons}(a,\ \text{append}(b,\ c)) \neq c$$

THEOREM: equal-append-b-append-b

$$(\text{append}(x,\ b) = \text{append}(y,\ b)) = (\text{plist}(x) = \text{plist}(y))$$

THEOREM: plist-periodic-task-requests

$$
\begin{array}{l}
\text{plist}(\text{periodic-task-requests}(task,\ n1,\ n2)) \\
= \ \ \text{periodic-task-requests}(task,\ n1,\ n2)
\end{array}
$$

THEOREM: lessp-equal-times-x-a-x

$$((a * b) < a) = ((a \not\simeq 0) \wedge (b \simeq 0))$$

THEOREM: periodic-task-requests-expand

$$
\begin{array}{l}
(\text{periodic-taskp}(pt) \\
\ \wedge\ \ ((n1 \ \mathbf{mod}\ bigp) = 0) \\
\ \wedge\ \ ((n2 \ \mathbf{mod}\ bigp) = 0) \\
\ \wedge\ \ ((\text{cadr}(pt) \ \mathbf{mod}\ bigp) = 0) \\
\ \wedge\ \ ((\text{caddr}(pt) \ \mathbf{mod}\ bigp) = 0) \\
\ \wedge\ \ (n1 \in \mathbf{N})) \\
\to\ \ (\text{periodic-task-requests}(pt,\ n1,\ n2) \\
\qquad = \ \text{expand-tasks-requests}(\text{periodic-task-requests}(\text{cons}(\text{car}(pt),\ \text{cons}(\text{cadr}(pt) \\
\qquad\qquad \div \ \ bigp,
\end{array}
$$

```
                                        cons (caddr (pt)
                                              ÷    bigp,
                                              cdddr (pt)))),
                                   n1 ÷ bigp,
                                   n2 ÷ bigp),
                    bigp))
```

THEOREM: periodic-tasks-requests-expand

```
(((n1 mod bigp) = 0)
 ∧  ((n2 mod bigp) = 0)
 ∧  periodic-tasksp (pts)
 ∧  (n1 ∈ N))
→   (periodic-tasks-requests (expand-tasks (pts, bigp), n1, n2)
      =   expand-tasks-requests (periodic-tasks-requests (pts,
                                                          n1 ÷ bigp,
                                                          n2 ÷ bigp),
                                 bigp))
```

;;;;;

DEFINITION:

```
edf (lengthschedule, r)
=   if lengthschedule ≃ 0 then nil
    else let s be edf (lengthschedule − 1, r)
         in
         let unfulfilled be unfulfilled (lengthschedule − 1,
                                         s,
                                         active-task-requests (lengthschedule − 1,
                                                               r))
         in
         if listp (unfulfilled)
         then append (s,
                      list (name (least-deadline (unfulfilled))))
         else append (s, list (nil)) endif endlet endlet endif
```

THEOREM: length-edf-simple

length (edf (n, r)) = fix (n)

DEFINITION:

```
valid-requests (reqs)
=   if listp (reqs)
    then litatom (caar (reqs))
         ∧   (caar (reqs) ≠ nil)
         ∧   (cadar (reqs) ∈ N)
```

$\wedge$ (caddar (*reqs*) $\in$ **N**)
$\wedge$ ($0 <$ cadddar (*reqs*))
$\wedge$ (cddddar (*reqs*) $=$ **nil**)
$\wedge$ valid-requests (cdr (*reqs*))
else *reqs* $=$ **nil endif**

THEOREM: valid-requests-periodic-task
valid-requests (periodic-task-requests (*pt*, *n1*, *n2*))
$=$ ((*n1* $\in$ **N**) $\vee$ (*n1* $\not<$ *n2*) $\vee$ ($\neg$ periodic-taskp (*pt*)))

THEOREM: valid-requests-append
(plist (*a*) $=$ *a*)
$\rightarrow$ (valid-requests (append (*a*, *b*))
$=$ (valid-requests (*a*) $\wedge$ valid-requests (*b*)))

THEOREM: valid-requests-periodic-tasks
valid-requests (periodic-tasks-requests (*pts*, *n1*, *n2*))
$=$ ((*n1* $\in$ **N**)
$\vee$ (*n1* $\not<$ *n2*)
$\vee$ ($\neg$ periodic-tasksp (*pts*))
$\vee$ ($\neg$ listp (*pts*)))

THEOREM: listp-active-task-requests-0
(*n* $<$ *n2*)
$\rightarrow$ (listp (active-task-requests (*n*, periodic-task-requests (*pt*, *n1*, *n2*)))
$=$ (periodic-taskp (*pt*) $\wedge$ (*n* $\not<$ *n1*)))

THEOREM: active-task-requests-append
active-task-requests (*n*, append (*r1*, *r2*))
$=$ append (active-task-requests (*n*, *r1*), active-task-requests (*n*, *r2*))

THEOREM: listp-active-task-requests-0-multiple
(*n* $<$ *n2*)
$\rightarrow$ (listp (active-task-requests (*n*, periodic-tasks-requests (*pts*, *n1*, *n2*)))
$=$ (periodic-tasksp (*pts*) $\wedge$ (*n* $\not<$ *n1*) $\wedge$ listp (*pts*)))

THEOREM: unfulfilled-0
valid-requests (*r*) $\rightarrow$ (unfulfilled (0, *s*, *r*) $=$ *r*)

THEOREM: valid-requests-active-task-requests
valid-requests (*r*) $\rightarrow$ valid-requests (active-task-requests (*n*, *r*))

THEOREM: lessp-0-length-means
($0 <$ length (*l*)) $=$ listp (*l*)

THEOREM: listp-repeat
listp (repeat (*n*, *v*)) $=$ (*n* $\not\simeq 0$)

Theorem: length-repeat
length (repeat (n, v)) = fix (n)

Theorem: plist-append
plist (append (x, y)) = append (x, plist (y))

Theorem: length-plist
length (plist (x)) = length (x)

Theorem: listp-plist
listp (plist (x)) = listp (x)

Theorem: plist-edf-simple
plist (edf (n, r)) = edf (n, r)

Theorem: firstn-repeat
firstn ($n1$, repeat ($n2$, v))
= **if** $n2 < n1$ **then** append (repeat ($n2$, v), repeat ($n1 - n2$, 0))
else repeat ($n1$, v) **endif**

Theorem: edf-simple-nlistp
($\neg$ listp (r)) $\rightarrow$ (edf (n, r) = repeat (n, **nil**))

Theorem: make-schedule-edf-nlistp
($\neg$ listp (r)) $\rightarrow$ (make-schedule-edf (s, r, n) = s)

Theorem: replace-nth-nlistp
($l \simeq$ **nil**) $\rightarrow$ (replace-nth (n, v, l) = append (repeat (n, 0), cons (v, 0)))

Theorem: plist-replace-nth2
(plist (replace-nth (n, v, s)) = replace-nth (n, v, s))
= ((plist (s) = s) $\wedge$ ($n <$ length (s)))

Theorem: plist-swap
(plist (swap ($n1$, $n2$, s)) = swap ($n1$, $n2$, s))
= ((plist (s) = s) $\wedge$ ($n1 <$ length (s)) $\wedge$ ($n2 <$ length (s)))

Theorem: plist-make-element-edf
(plist (s) = s)
$\rightarrow$ (plist (make-element-edf (s, r, n)) = make-element-edf (s, r, n))

Theorem: length-make-schedule-edf
length (make-schedule-edf (s, r, n)) = length (s)

Theorem: equal-nthcdr-nthcdr-from-nthcdr-plus1
(($n <$ length ($s1$))
$\wedge$ ($n <$ length ($s2$))
$\wedge$ (nthcdr (n, cdr ($s1$)) = nthcdr (n, cdr ($s2$))))
$\rightarrow$ ((nthcdr (n, $s1$) = nthcdr (n, $s2$)) = (nth (n, $s1$) = nth (n, $s2$)))

THEOREM: lessp-first-instance2
first-instance $(n2,\ v,\ l) \rightarrow (\text{first-instance}\,(n2,\ v,\ l) \not< n2)$

THEOREM: nth-make-schedule-edf-simple
$(n1 < n2) \rightarrow (\text{nth}\,(n1,\ \text{make-schedule-edf}\,(s,\ r,\ n2)) = \text{nth}\,(n1,\ s))$

THEOREM: car-append
car (append $(x,\ y)$)
= **if** listp (x) **then** car (x)
else car (y) **endif**

THEOREM: listp-edf-simple
listp (edf $(n,\ r)) = (n \not\simeq 0)$

THEOREM: unfulfilled-schedule-first-part-only
$(n < \text{length}\,(s)) \rightarrow (\text{unfulfilled}\,(n,\ s,\ r) = \text{unfulfilled}\,(n,\ \text{firstn}\,(n,\ s),\ r))$

THEOREM: firstn-n-edf-simple-n
$(\text{fix}\,(n1) = \text{fix}\,(n2)) \rightarrow (\text{firstn}\,(n1,\ \text{edf}\,(n2,\ r)) = \text{edf}\,(n2,\ r))$

THEOREM: firstn-edf-simple-regular
$(n1 < n2) \rightarrow (\text{firstn}\,(n1,\ \text{edf}\,(n2,\ r)) = \text{edf}\,(n1,\ r))$

THEOREM: firstn-edf-simple-help
$(n2 \not< n1) \rightarrow (\text{firstn}\,(n1,\ \text{edf}\,(n2,\ r)) = \text{edf}\,(n1,\ r))$

THEOREM: firstn-edf-simple
firstn $(n1,\ \text{edf}\,(n2,\ r))$
= **if** $n2 < n1$ **then** append (edf $(n2,\ r)$, repeat $(n1 - n2,\ 0)$)
else edf $(n1,\ r)$ **endif**

THEOREM: nth-append
nth $(n,\ \text{append}\,(x,\ y))$
= **if** $n < \text{length}\,(x)$ **then** nth $(n,\ x)$
else nth $(n - \text{length}\,(x),\ y)$ **endif**

```
; dangerous - must use in special rules
```

THEOREM: nth-edf-simple-simpler
$(n < n2) \rightarrow (\text{nth}\,(n,\ \text{edf}\,(n2,\ r)) = \text{nth}\,(n,\ \text{edf}\,(1 + n,\ r)))$

THEOREM: active-task-requests-nnumberp
$(n \notin \mathbf{N}) \rightarrow (\text{active-task-requests}\,(n,\ r) = \text{active-task-requests}\,(0,\ r))$

THEOREM: unfulfilled-nnumberp
$(n \notin \mathbf{N}) \rightarrow (\text{unfulfilled}\,(n,\ s,\ r) = \text{unfulfilled}\,(0,\ s,\ r))$

THEOREM: nth-edf-simple
nth (n, edf ($n2$, r))
= **if** $n < n2$
then **if** listp (unfulfilled (n, edf (n, r), active-task-requests (n, r)))
then car (least-deadline (unfulfilled (n,
edf (n, r),
active-task-requests (n, r))))
else **nil** **endif**
else 0 **endif**

THEOREM: equal-cdr-cdr-means
((cdr (x) = cdr (y)) ∧ listp (x) ∧ listp (y))
→ ((x = y) = (car (x) = car (y)))

EVENT: Disable equal-cdr-cdr-means.

THEOREM: first-instance-same-as-member
n → (first-instance (n, v, s) ↔ (v ∈ nthcdr (n, s)))

THEOREM: nth-make-element-edf
n → (nth (n, make-element-edf (s, r, n))
= **if** listp (unfulfilled (n, s, active-task-requests (n, r)))
∧ first-instance (n,
car (least-deadline (unfulfilled (n,
s,
active-task-requests (n,
r)))),
s)
then car (least-deadline (unfulfilled (n,
s,
active-task-requests (n, r))))
else nth (n, s) **endif**)

THEOREM: unfulfilled-append
unfulfilled (n, s, append ($r1$, $r2$))
= append (unfulfilled (n, s, $r1$), unfulfilled (n, s, $r2$))

DEFINITION:
no-unfulfilled-active-task-induct (pts, $oldpts$)
= **if** $pts \simeq$ **nil** **then** **t**
else no-unfulfilled-active-task-induct (cdr (pts),
append ($oldpts$,
list (car (pts)))) **endif**

THEOREM: periodic-tasksp-append-car
(periodic-tasksp (append (x, y)) $\land$ listp (y))
$\rightarrow$ periodic-tasksp (append (x, list (car (y))))

THEOREM: assoc-plist
assoc (v, plist (l)) = assoc (v, l)

THEOREM: all-nils-or-cars-plist2
all-nils-or-cars ($l1$, plist ($l2$)) = all-nils-or-cars ($l1$, $l2$)

```
;; bad bad bad
```

THEOREM: assoc-append-simple
(($\neg$ assoc (v, l)) $\land$ periodic-tasksp (l))
$\rightarrow$ (assoc (v, append (l, $l2$)) = assoc (v, $l2$))

THEOREM: listp-unfulfilled-if-schedule-contains
(periodic-taskp (pt)
$\land$ good-schedule (s, periodic-task-requests (pt, $n1$, $n2$))
$\land$ (nth (n, s) = car (pt))
$\land$ ($n < n2$)
$\land$ ($n \not< n1$))
$\rightarrow$ listp (unfulfilled (n,
firstn (n, s),
active-task-requests (n,
periodic-task-requests (pt,
$n1$,
$n2$))))

THEOREM: no-unfulfilled-active-task-if-nil-help
((nth (n, s) $\neq$ **nil**)
$\land$ periodic-tasksp (append ($oldpts$, pts))
$\land$ periodic-tasksp ($oldpts$)
$\land$ periodic-tasksp (pts)
$\land$ good-schedule (s, periodic-tasks-requests (pts, $n1$, $n2$))
$\land$ all-nils-or-cars (s, append ($oldpts$, pts))
$\land$ ($n < n2$)
$\land$ ($n <$ length (s))
$\land$ ($n \not< n1$))
$\rightarrow$ (listp (unfulfilled (n,
firstn (n, s),
active-task-requests (n,
periodic-tasks-requests (pts,
$n1$,
$n2$))))
$\lor$ assoc (nth (n, s), $oldpts$))

Theorem: no-unfulfilled-active-task-if-nil
$((\text{nth}(n, s) \neq \textbf{nil})$
$\land$ periodic-tasksp (pts)
$\land$ good-schedule $(s,$ periodic-tasks-requests $(pts, n1, n2))$
$\land$ all-nils-or-cars (s, pts)
$\land$ $(n < n2)$
$\land$ $(n < \text{length}(s))$
$\land$ $(n \not< n1))$
$\rightarrow$ listp (unfulfilled $(n,$ firstn $(n, s),$ active-task-requests $(n,$ periodic-tasks-requests $(pts, n1, n2))))$

Theorem: replace-nth-first-instance-nnumberp
$(n \notin \mathbf{N})$
$\rightarrow$ (replace-nth (first-instance $(n, v, s), v2, l)$
$=$ replace-nth (first-instance $(0, v, s), v2, l))$

Theorem: make-element-nnumberp
$(n \notin \mathbf{N}) \rightarrow$ (make-element-edf $(s, r, n) =$ make-element-edf $(s, r, 0))$

```
;; versions of nth- theorems with n = 0
```

Theorem: car-make-schedule-edf-simple
$(0 < n2) \rightarrow$ (car (make-schedule-edf $(s, r, n2)) =$ car $(s))$

Theorem: car-edf-simple
car (edf $(n2, r))$
$=$ **if** $0 < n2$
then if listp (unfulfilled (0, edf (0, r), active-task-requests (0, r)))
then car (least-deadline (unfulfilled (0, edf (0, r), active-task-requests (0, r))))
else nil endif
else 0 **endif**

Theorem: car-repeat
car (repeat $(n, v))$
$=$ **if** $n \simeq 0$ **then** 0
else v **endif**

Theorem: numberp-first-instance
$(n \in \mathbf{N}) \rightarrow$ ((first-instance $(n, v, l) \in \mathbf{N}) = (v \in$ nthcdr $(n, l)))$

THEOREM: nth-make-element-simple
$(n1 < n2) \rightarrow (\text{nth}(n1, \text{make-element-edf}(s, r, n2)) = \text{nth}(n1, s))$

THEOREM: car-make-element-simple
$(n \not\simeq 0) \rightarrow (\text{car}(\text{make-element-edf}(s, r, n)) = \text{car}(s))$

THEOREM: car-make-element-edf
car (make-element-edf (s, r, n))
= **if** ($n \simeq 0$)
  ∧ listp (unfulfilled (0, s, active-task-requests (0, r)))
  ∧ first-instance (0,
    car (least-deadline (unfulfilled (0,
      s,
      active-task-requests (0,
        r)))),
    s)
  then car (least-deadline (unfulfilled (0, s, active-task-requests (0, r))))
  else car (s) **endif**

THEOREM: firstn-make-element-simple
$(n2 \not< n1) \rightarrow (\text{firstn}(n1, \text{make-element-edf}(s, r, n2)) = \text{firstn}(n1, s))$

THEOREM: firstn-sub1-cdr-make-element
$(n \not\simeq 0)$
$\rightarrow (\text{firstn}(n - 1, \text{cdr}(\text{make-element-edf}(s, r, n))) = \text{firstn}(n - 1, \text{cdr}(s)))$

THEOREM: nthcdr-n-cons-firstn-n
nthcdr (n, cons (a, firstn (n, l)))
= **if** $n \simeq 0$ **then** list (a)
  else list (nth ($n - 1$, l)) **endif**

THEOREM: nth-make-element-edf-sub1
$(n \not\simeq 0)$
→ (nth ($n - 1$, cdr (make-element-edf (s, r, n)))
  = **if** listp (unfulfilled (n, s, active-task-requests (n, r)))
    ∧ first-instance (n,
      car (least-deadline (unfulfilled (n,
        s,
        active-task-requests (n,
          r)))),
      s)
    then car (least-deadline (unfulfilled (n,
      s,
      active-task-requests (n, r))))
    else nth (n, s) **endif**)

THEOREM: equal-repeat-when-nil-not
$(\text{car}(s) \neq v) \rightarrow ((s = \text{repeat}(\text{length}(s),\ v)) = (s = \textbf{nil}))$

THEOREM: member-car-schedule
$(\text{good-schedule}(s,\ rs) \wedge \text{valid-requests}(rs) \wedge (r \in rs))$
$\rightarrow \quad (\text{car}(r) \in s)$

THEOREM: sublistp-remove-until
$\text{sublistp}(x,\ \text{remove-until}(v,\ y)) \rightarrow \text{sublistp}(x,\ y)$

THEOREM: member-nil-periodic-task-requests
$\textbf{nil} \notin \text{periodic-task-requests}(pt,\ n1,\ n2)$

THEOREM: member-nil-periodic-tasks-requests
$\textbf{nil} \notin \text{periodic-tasks-requests}(pts,\ n1,\ n2)$

THEOREM: member-least-deadline-better
$(\text{sublistp}(r1,\ r2) \wedge (\textbf{nil} \notin r2))$
$\rightarrow \quad ((\text{least-deadline}(r1) \in r2) = \text{listp}(r1))$

THEOREM: make-schedule-edf-is-edf-simple
$(\text{non-overlapping-requests}(r)$
$\wedge \quad \text{cars-non-nil-litatoms}(r)$
$\wedge \quad \text{all-litatoms}(s)$
$\wedge \quad \text{good-schedule}(s,\ r)$
$\wedge \quad (\text{plist}(s) = s)$
$\wedge \quad \text{all-non-nil-corresponding}(s,\ r,\ \mathtt{0})$
$\wedge \quad (r = \text{periodic-tasks-requests}(pts,\ \mathtt{0},\ \text{length}(s)))$
$\wedge \quad \text{all-nils-or-cars}(s,\ pts)$
$\wedge \quad \text{periodic-tasksp}(pts)$
$\wedge \quad (\text{edf}(n,\ r) = \text{firstn}(n,\ s))$
$\wedge \quad (\text{length}(s) \not< n))$
$\rightarrow \quad (\text{nthcdr}(n,\ \text{make-schedule-edf}(s,\ r,\ n)) = \text{nthcdr}(n,\ \text{edf}(\text{length}(s),\ r)))$

```
;;; big lemma
```

THEOREM: make-schedule-edf-is-edf
$(\text{periodic-tasksp}(pts)$
$\wedge \quad \text{cars-non-nil-litatoms}(\text{periodic-tasks-requests}(pts,\ \mathtt{0},\ \text{length}(s)))$
$\wedge \quad \text{all-litatoms}(s)$
$\wedge \quad \text{good-schedule}(s,\ \text{periodic-tasks-requests}(pts,\ \mathtt{0},\ \text{length}(s)))$
$\wedge \quad (\text{plist}(s) = s)$
$\wedge \quad \text{all-non-nil-corresponding}(s,$
$\qquad \text{periodic-tasks-requests}(pts,\ \mathtt{0},\ \text{length}(s)),$
$\qquad \mathtt{0})$

$\wedge$ all-nils-or-cars (s, pts)
$\wedge$ periodic-tasksp (pts)
$\wedge$ (n = length (s)))
$\rightarrow$ (make-schedule-edf (s, periodic-tasks-requests (pts, `0`, n), `0`)
= edf (length (s), periodic-tasks-requests (pts, `0`, length (s))))

THEOREM: plist-make-simple-schedule
plist (make-simple-schedule (pts, $bigp$, $length$))
= make-simple-schedule (pts, $bigp$, $length$)

DEFINITION:
cpu-utilization (pts, $bigp$)
= **if** listp (pts)
then (($bigp$ $*$ tk-duration (car (pts))) $\div$ tk-period (car (pts)))
+ cpu-utilization (cdr (pts), $bigp$)
else `0` **endif**

THEOREM: length-substring-schedule
(expanded-tasksp (pts, $bigp$) $\wedge$ periodic-tasksp (pts))
$\rightarrow$ (length (substring-schedule (pts, $bigp$)) = cpu-utilization (pts, $bigp$))

`;; combine big lemmas`

THEOREM: good-edf-for-expanded
(expanded-tasksp (pts, $bigp$)
$\wedge$ ($bigp$ $\not<$ cpu-utilization (pts, $bigp$))
$\wedge$ periodic-tasksp (pts)
$\wedge$ ((n **mod** $bigp$) = `0`)
$\wedge$ ((n **mod** big-period (pts)) = `0`))
$\rightarrow$ good-schedule (edf (n, periodic-tasks-requests (pts, `0`, n)),
periodic-tasks-requests (pts, `0`, n))

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

DEFINITION:
expand-list (n, $list$)
= **if** listp ($list$)
then append (repeat (n, car ($list$)), expand-list (n, cdr ($list$)))
else nil endif

THEOREM: expand-list-append
expand-list (n, append ($l1$, $l2$))
= append (expand-list (n, $l1$), expand-list (n, $l2$))

THEOREM: listp-expand-list
listp (expand-list (n, $list$)) = ((n $\not\simeq$ `0`) $\wedge$ listp ($list$))

THEOREM: length-expand-list
$\text{length}(\text{expand-list}(n,\ list)) = (n * \text{length}(list))$

THEOREM: plist-expand-list
$\text{plist}(\text{expand-list}(n,\ list)) = \text{expand-list}(n,\ list)$

THEOREM: nthcdr-is-nil
$(\text{nthcdr}(n,\ l) = \textbf{nil}) = ((\text{length}(l) = \text{fix}(n)) \wedge (\text{plist}(l) = l))$

THEOREM: active-task-requests-expand-tasks-requests
$(bigp \not\simeq 0)$
$\rightarrow \quad (\text{active-task-requests}(n,\ \text{expand-tasks-requests}(r,\ bigp))$
$\quad = \quad \text{expand-tasks-requests}(\text{active-task-requests}(n \div bigp,\ r),\ bigp))$

THEOREM: repeat-1
$\text{repeat}(1,\ v) = \text{list}(v)$

EVENT: Disable lessp-difference-special.

THEOREM: quotient-add1-plus-special
$((1 + (z + (bigp * v))) \div bigp)$
$= \quad \textbf{if } bigp \simeq 0 \textbf{ then } 0$
$\quad \textbf{else } v + ((1 + z) \div bigp) \textbf{ endif}$

THEOREM: remainder-add1-plus-special
$((1 + (z + (bigp * v))) \textbf{ mod } bigp)$
$= \quad \textbf{if } bigp \simeq 0 \textbf{ then } 1 + z$
$\quad \textbf{else } (1 + z) \textbf{ mod } bigp \textbf{ endif}$

THEOREM: plist-nthcdr
$((\text{length}(l) \not< n) \wedge (\text{plist}(l) = l))$
$\rightarrow \quad (\text{plist}(\text{nthcdr}(n,\ l)) = \text{nthcdr}(n,\ l))$

THEOREM: remainder-plus-add1-hack
$(((z + (v * z)) \textbf{ mod } (1 + v)) = 0)$
$\wedge \quad (((z + (z * v)) \textbf{ mod } (1 + v)) = 0)$

THEOREM: quotient-plus-add1-hack
$(((z + (v * z)) \div (1 + v)) = \text{fix}(z))$
$\wedge \quad (((z + (z * v)) \div (1 + v)) = \text{fix}(z))$

THEOREM: equal-remainder-sub1-0
$((x \textbf{ mod } y) = 0)$
$\rightarrow \quad (((x - 1) \textbf{ mod } y)$
$\quad = \quad \textbf{if } x \simeq 0 \textbf{ then } 0$
$\quad\quad \textbf{elseif } y \simeq 0 \textbf{ then } x - 1$
$\quad\quad \textbf{elseif } y = 1 \textbf{ then } 0$
$\quad\quad \textbf{else } y - 1 \textbf{ endif})$

THEOREM: lessp-round-means
$((bigp * (n \div bigp)) < n)$
$= \quad (((n \ \mathbf{mod}\ bigp) \neq 0) \vee ((bigp \simeq 0) \wedge (n \not\simeq 0)))$

THEOREM: equal-repeat-nil
$(\text{repeat}(n, v) = \mathbf{nil}) = (n \simeq 0)$

THEOREM: times-1-arg2
$(x * 1) = \text{fix}(x)$

THEOREM: firstn-0
$\text{firstn}(0, l) = \mathbf{nil}$

THEOREM: lessp-times-sub1-sub1
$((bigp * ((n \div bigp) - 1)) < (n - 1))$
$= \quad ((1 < n) \wedge (bigp \neq 1))$

THEOREM: equal-cons-repeat
$(\text{cons}(a, b) = \text{repeat}(n, v))$
$= \quad ((a = v) \wedge (n \not\simeq 0) \wedge (b = \text{repeat}(n - 1, v)))$

THEOREM: lessp-1-means
$(1 < x) = ((x \not\simeq 0) \wedge (x \neq 1))$

THEOREM: lessp-sub1-plus-hack
$(n1 < ((n2 + n1) - 1)) = (1 < n2)$

```
;; backwards
```

THEOREM: firstn-nthcdr
$\text{firstn}(x, \text{nthcdr}(n, s)) = \text{nthcdr}(n, \text{firstn}(n + x, s))$

EVENT: Disable firstn-nthcdr.

THEOREM: nthcdr-x-firstn-x
$\text{nthcdr}(n, \text{firstn}(n, l)) = \mathbf{nil}$

THEOREM: nthcdr-x-edf-x
$\text{nthcdr}(n, \text{edf}(n, r)) = \mathbf{nil}$

THEOREM: nth-cons
$\text{nth}(n, \text{cons}(a, b))$
$= \quad \mathbf{if}\ n \simeq 0\ \mathbf{then}\ a$
$\quad\quad \mathbf{else}\ \text{nth}(n - 1, b)\ \mathbf{endif}$

THEOREM: firstn-nthcdr-too-big
$(n < \text{length}(l))$
$\rightarrow \quad (\text{firstn}(n - z, \text{nthcdr}(z, l))$
$\quad = \quad \textbf{if } n < z \textbf{ then nil}$
$\quad\quad \textbf{else } \text{nthcdr}(z, \text{firstn}(n, l)) \textbf{ endif})$

THEOREM: firstn-nthcdr-edf-plus
$\text{firstn}(n1 - n2, \text{nthcdr}(n2, \text{edf}(z + n1, r)))$
$= \quad \text{firstn}(n1 - n2, \text{nthcdr}(n2, \text{edf}(n1, r)))$

DEFINITION:
$\text{nthcdr-expand-induct}(n, b, l)$
$= \quad \textbf{if } \text{listp}(l)$
$\quad \textbf{then if } b < n \textbf{ then } \text{nthcdr-expand-induct}(n - b, b, \text{cdr}(l))$
$\quad\quad \textbf{else t endif}$
$\quad \textbf{else t endif}$

THEOREM: nthcdr-expand-list
$((n \textbf{ mod } b) = 0)$
$\rightarrow \quad (\text{nthcdr}(n, \text{expand-list}(b, l))$
$\quad = \quad \textbf{if } (b * \text{length}(l)) < n \textbf{ then } 0$
$\quad\quad \textbf{else } \text{expand-list}(b, \text{nthcdr}(n \div b, l)) \textbf{ endif})$

THEOREM: expand-list-repeat
$\text{expand-list}(b, \text{repeat}(n, v)) = \text{repeat}(n * b, v)$

THEOREM: firstn-expand-list
$((n \textbf{ mod } b) = 0)$
$\rightarrow \quad (\text{firstn}(n, \text{expand-list}(b, l)) = \text{expand-list}(b, \text{firstn}(n \div b, l)))$

THEOREM: occurrences-expand-list
$\text{occurrences}(v, \text{expand-list}(bigp, l)) = (bigp * \text{occurrences}(v, l))$

THEOREM: expand-list-0
$(n \simeq 0) \rightarrow (\text{expand-list}(n, l) = \textbf{nil})$

THEOREM: listp-firstn
$\text{listp}(\text{firstn}(n, l)) = (n \not\simeq 0)$

THEOREM: equal-nil-firstn
$(\textbf{nil} = \text{firstn}(n, l)) = (n \simeq 0)$

THEOREM: firstn-difference-plus-nthcdr
$((\text{length}(l) \not< (a + b)) \wedge (b \not< c))$
$\rightarrow \quad (\text{firstn}((a + b) - c, \text{nthcdr}(c, l))$
$\quad = \quad \text{append}(\text{firstn}(b - c, \text{nthcdr}(c, l)), \text{firstn}(a, \text{nthcdr}(b, l))))$

THEOREM: lessp-occurrences-firstn
$n \not< \text{occurrences}(v, \text{firstn}(n, l))$

EVENT: Disable lessp-occurrences-firstn.

THEOREM: lessp-occurrences-edf
$n \not< \text{occurrences}(v, \text{edf}(n, r))$

THEOREM: equal-times-hack
$(a < b) \rightarrow ((a = (b * n)) = ((a = 0) \land (b \not\simeq 0) \land (n \simeq 0)))$

THEOREM: equal-occurrences-firstn-times
$(z < b)$
$\rightarrow \quad ((\text{occurrences}(v, \text{firstn}(z, l)) = (b * n))$
$\quad = \quad ((n \simeq 0) \land (\text{occurrences}(v, \text{firstn}(z, l)) \simeq 0)))$

THEOREM: firstn-plus
$\text{firstn}(a + b, l) = \text{append}(\text{firstn}(b, l), \text{firstn}(a, \text{nthcdr}(b, l)))$

THEOREM: equal-plus-times
$(z < b)$
$\rightarrow \quad ((((b * n) + z) = (b * v))$
$\quad = \quad ((z \simeq 0) \land (\text{fix}(n) = \text{fix}(v))))$

THEOREM: firstn-noop
$(\text{length}(l) = \text{fix}(n)) \rightarrow (\text{firstn}(n, l) = \text{plist}(l))$

THEOREM: lessp-x-x
$(x < x) = \mathbf{f}$

THEOREM: overlapping-non-overlapping3-means
$(\text{non-overlapping-requests3}(r, r2)$
$\land \quad (x \not< \text{cadr}(r))$
$\land \quad (\text{car}(req) = \text{car}(r))$
$\land \quad (x \not< \text{cadr}(req))$
$\land \quad (x < \text{caddr}(r))$
$\land \quad (req \in r2))$
$\rightarrow \quad ((x < \text{caddr}(req)) = (r = req))$

THEOREM: assoc-unfulfilled-expanded-non-overlapping
$((r \notin r2)$
$\land \quad \text{non-overlapping-requests3}(r, r2)$
$\land \quad (z < b)$
$\land \quad (x < \text{caddr}(r))$
$\land \quad (x \not< \text{cadr}(r)))$

$\rightarrow \quad (\neg\, \text{assoc}\,(\text{car}\,(r),$
$\qquad \text{unfulfilled}\,(z + (b * x),$
$\qquad\qquad s,$
$\qquad\qquad \text{expand-tasks-requests}\,(\text{active-task-requests}\,(x,$
$\qquad\qquad\qquad r2),$
$\qquad\qquad\qquad b))))$

THEOREM: not-assoc-car-req
$(\text{valid-requests}\,(r2)$
$\wedge \quad (req \in r2)$
$\wedge \quad (x < \text{caddr}\,(req))$
$\wedge \quad (x \not< \text{cadr}\,(req))$
$\wedge \quad \text{non-overlapping-requests}\,(r2)$
$\wedge \quad (z < b)$
$\wedge \quad (\text{occurrences}\,(\text{car}\,(req),$
$\qquad \text{firstn}\,((z + (b * x)) - (b * \text{cadr}\,(req)),$
$\qquad\qquad \text{nthcdr}\,(b * \text{cadr}\,(req),\ s)))$
$\quad = \quad (b * \text{cadddr}\,(req))))$
$\rightarrow \quad (\neg\, \text{assoc}\,(\text{car}\,(req),$
$\qquad \text{unfulfilled}\,(z + (b * x),$
$\qquad\qquad s,$
$\qquad\qquad \text{expand-tasks-requests}\,(\text{active-task-requests}\,(x,$
$\qquad\qquad\qquad r2),$
$\qquad\qquad\qquad b))))$

THEOREM: equal-car-car-hack
$((e1 \in l)$
$\wedge \quad (\neg\, \text{assoc}\,(\text{car}\,(e2),\ l))$
$\wedge \quad \text{valid-requests}\,(l)$
$\wedge \quad \text{listp}\,(e1)$
$\wedge \quad \text{listp}\,(e2))$
$\rightarrow \quad (\text{car}\,(e1) \neq \text{car}\,(e2))$

THEOREM: not-member-nthcdr-add1
$(v \notin \text{nthcdr}\,(1 + x,\ s))$
$\rightarrow \quad ((v \in \text{nthcdr}\,(x,\ s)) = ((x < \text{length}\,(s)) \wedge (v = \text{nth}\,(x,\ s))))$

THEOREM: occurrences-hack
$((v \notin \text{nthcdr}\,(n,\ s)) \wedge (n \not< b) \wedge \text{litatom}\,(v))$
$\rightarrow \quad (\text{occurrences}\,(v,\ \text{firstn}\,((a + n) - b,\ \text{nthcdr}\,(b,\ s)))$
$\quad = \quad \text{occurrences}\,(v,\ \text{firstn}\,(n - b,\ \text{nthcdr}\,(b,\ s))))$

THEOREM: occurrences-hack2
$((v \notin \text{nthcdr}\,(n,\ l)) \wedge (n < \text{length}\,(l)))$
$\rightarrow \quad (\text{occurrences}\,(v,\ \text{nthcdr}\,(n1,\ l))$
$\quad = \quad \text{occurrences}\,(v,\ \text{nthcdr}\,(n1,\ \text{firstn}\,(n,\ l))))$

THEOREM: valid-requests-unfulfilled
valid-requests $(r) \rightarrow$ valid-requests (unfulfilled (n, s, r))

THEOREM: valid-requests-expand-tasks-requests
valid-requests (r)
$\rightarrow$ (valid-requests (expand-tasks-requests (r, b))
= $((b \not\simeq 0) \vee (\neg$ listp $(r))))$

THEOREM: litatom-caar
valid-requests $(r) \rightarrow$ (litatom (caar (r)) = listp (r))

EVENT: Disable litatom-caar.

THEOREM: numberp-cadar
valid-requests $(r) \rightarrow$ (cadar $(r) \in \mathbf{N}$)

EVENT: Disable numberp-cadar.

THEOREM: not-equal-car-least-deadline
(valid-requests $(r2)$
$\wedge$ $(req \in r2)$
$\wedge$ $(x <$ caddr $(req))$
$\wedge$ $(x \not<$ cadr $(req))$
$\wedge$ non-overlapping-requests $(r2)$
$\wedge$ $(z < b)$
$\wedge$ $(b \not\simeq 0)$
$\wedge$ litatom (car (req))
$\wedge$ (occurrences (car (req),
firstn $((z + (b * x)) - (b *$ cadr $(req))$,
nthcdr $(b *$ cadr $(req), s)))$
= $(b *$ cadddr $(req))))$
$\rightarrow$ (car (req)
$\neq$ car (least-deadline (unfulfilled $(z + (b * x)$,
s,
expand-tasks-requests (active-task-requests $(x$,
$r2)$,
$b)))))$

THEOREM: not-equal-car-least-deadline-special
(valid-requests $(r2)$
$\wedge$ $(req \in r2)$
$\wedge$ $(x <$ caddr $(req))$
$\wedge$ $(x \not<$ cadr $(req))$
$\wedge$ non-overlapping-requests $(r2)$

$\wedge \quad (b \not\simeq 0)$
$\wedge \quad \text{litatom}(\text{car}(req))$
$\wedge \quad (\text{occurrences}(\text{car}(req),$
$\qquad \text{firstn}((b * x) - (b * \text{cadr}(req)),$
$\qquad\quad \text{nthcdr}(b * \text{cadr}(req), s)))$
$\quad = \quad (b * \text{cadddr}(req))))$
$\rightarrow \quad (\text{car}(req)$
$\quad \neq \quad \text{car}(\text{least-deadline}(\text{unfulfilled}(b * x,$
$\qquad s,$
$\qquad \text{expand-tasks-requests}(\text{active-task-requests}(x,$
$\qquad\quad r2),$
$\qquad b)))))$

THEOREM: occurrences-edf-satisfied
$(\text{listp}(r)$
$\wedge \quad (x < \text{caddar}(r))$
$\wedge \quad (x \not< \text{cadar}(r))$
$\wedge \quad (z < b)$
$\wedge \quad (\text{expand-list}(b, \text{edf}(x, r2))$
$\quad = \quad \text{edf}(b * x, \text{expand-tasks-requests}(r2, b)))$
$\wedge \quad \text{valid-requests}(r2)$
$\wedge \quad \text{valid-requests}(r)$
$\wedge \quad \text{non-overlapping-requests}(r2)$
$\wedge \quad \text{sublistp}(r, r2)$
$\wedge \quad (b \neq 0)$
$\wedge \quad (b \in \mathbf{N})$
$\wedge \quad (x \in \mathbf{N})$
$\wedge \quad (\text{occurrences}(\text{caar}(r),$
$\qquad \text{firstn}(x - \text{cadar}(r), \text{nthcdr}(\text{cadar}(r), \text{edf}(x, r2))))$
$\quad = \quad \text{cadddar}(r)))$
$\rightarrow \quad (\text{occurrences}(\text{caar}(r),$
$\qquad \text{firstn}(z,$
$\qquad\quad \text{nthcdr}(b * x,$
$\qquad\qquad \text{edf}(z + (b * x),$
$\qquad\qquad\quad \text{expand-tasks-requests}(r2, b)))))$
$\quad = \quad 0)$

THEOREM: unfulfilled-expanded-on-line
$((z < b)$
$\wedge \quad (\text{expand-list}(b, \text{edf}(x, r2))$
$\quad = \quad \text{edf}(x * b, \text{expand-tasks-requests}(r2, b)))$
$\wedge \quad \text{valid-requests}(r2)$
$\wedge \quad \text{valid-requests}(r)$
$\wedge \quad \text{non-overlapping-requests}(r2)$

$\land$ sublistp $(r,\ r2))$
$\rightarrow$ (unfulfilled $(z + (x * b),$
edf $(z + (x * b),$ expand-tasks-requests $(r2,\ b)),$
expand-tasks-requests (active-task-requests $(x,\ r),\ b))$
$=$ unfulfilled $(x * b,$
edf $(z + (x * b),$ expand-tasks-requests $(r2,\ b)),$
expand-tasks-requests (active-task-requests $(x,\ r),\ b)))$

Theorem: unfulfilled-expand-list
(valid-requests $(r) \land ((n\ \textbf{mod}\ b) = 0) \land (b \not\simeq 0))$
$\rightarrow$ (unfulfilled $(n,$ expand-list $(b,\ s),$ expand-tasks-requests $(r,\ b))$
$=$ expand-tasks-requests (unfulfilled $(n \div b,\ s,\ r),\ b))$

Theorem: equal-nil-expand-list
$(\textbf{nil} =$ expand-list $(b,\ l)) = ((b \simeq 0) \lor (l \simeq \textbf{nil}))$

Theorem: valid-requests-sublistp
(sublistp $(x,\ y) \land$ (plist $(x) = x) \land$ valid-requests $(y))$
$\rightarrow$ valid-requests (x)

Theorem: unfulfilled-sub1-expanded-on-line
$((a \not\simeq 0)$
$\land$ $(a < b)$
$\land$ (expand-list $(b,$ edf $(x,\ r2))$
$=$ edf $(x * b,$ expand-tasks-requests $(r2,\ b)))$
$\land$ non-overlapping-requests $(r2)$
$\land$ valid-requests $(r2)$
$\land$ valid-requests (r)
$\land$ sublistp $(r,\ r2))$
$\rightarrow$ (unfulfilled $((a + (b * x)) - 1,$
edf $((a + (b * x)) - 1,$ expand-tasks-requests $(r2,\ b)),$
expand-tasks-requests (active-task-requests $(x,\ r),\ b))$
$=$ unfulfilled $(x * b,$
edf $((a + (x * b)) - 1,$
expand-tasks-requests $(r2,\ b)),$
expand-tasks-requests (active-task-requests $(x,\ r),\ b)))$

Theorem: firstn-only-helper
(firstn $(n,\ a) =$ firstn $(n,\ b))$
$\rightarrow$ (firstn $(n - c,$ nthcdr $(c,\ a)) =$ firstn $(n - c,$ nthcdr $(c,\ b)))$

Theorem: firstn-nthcdr-edf-plus-simple
firstn $(n - a,$ nthcdr $(a,$ edf $(b + n,\ r)))$
$=$ firstn $(n - a,$ nthcdr $(a,$ edf $(n,\ r)))$

THEOREM: unfulfilled-too-big
$\text{unfulfilled}\,(n,\ \text{edf}\,(a + n,\ r1),\ r2) = \text{unfulfilled}\,(n,\ \text{edf}\,(n,\ r1),\ r2)$

THEOREM: unfulfilled-too-big-sub1
$(a \not\simeq 0)$
$\rightarrow\ (\text{unfulfilled}\,(n,\ \text{edf}\,((a + n) - 1,\ r1),\ r2)$
$\quad = \ \text{unfulfilled}\,(n,\ \text{edf}\,(n,\ r1),\ r2))$

THEOREM: car-least-deadline-expand-tasks-requests
$(b \not\simeq 0)$
$\rightarrow\ (\text{car}\,(\text{least-deadline}\,(\text{expand-tasks-requests}\,(r,\ b)))$
$\quad = \ \text{car}\,(\text{least-deadline}\,(r)))$

THEOREM: remainder-hack
$((n\ \mathbf{mod}\ p) = 0)$
$\rightarrow\ ((((x + n) - 1)\ \mathbf{mod}\ p)$
$\quad = \ \mathbf{if}\ x \simeq 0\ \mathbf{then}\ (n - 1)\ \mathbf{mod}\ p$
$\quad\quad \mathbf{else}\ (x - 1)\ \mathbf{mod}\ p\ \mathbf{endif})$

THEOREM: equal-car-least-deadline-nil
$\text{valid-requests}\,(r) \rightarrow (\text{car}\,(\text{least-deadline}\,(r)) \neq \mathbf{nil})$

THEOREM: unfulfilled-0-expand-tasks-requests
$(\text{valid-requests}\,(r) \land (b \not\simeq 0))$
$\rightarrow\ (\text{unfulfilled}\,(0,\ s,\ \text{expand-tasks-requests}\,(r,\ b))$
$\quad = \ \text{expand-tasks-requests}\,(\text{unfulfilled}\,(0,\ s,\ r),\ b))$

THEOREM: unfulfilled-0-edf
$\text{unfulfilled}\,(0,\ \text{edf}\,(n,\ r),\ r2) = \text{unfulfilled}\,(0,\ \mathbf{nil},\ r2)$

THEOREM: unfulfilled-expanded-on-line-beginning
$((z < b)$
$\land\ \text{non-overlapping-requests}\,(r2)$
$\land\ (z \neq 0)$
$\land\ \text{sublistp}\,(r,\ r2)$
$\land\ \text{valid-requests}\,(r)$
$\land\ \text{valid-requests}\,(r2))$
$\rightarrow\ (\text{unfulfilled}\,(z,$
$\quad\quad \text{edf}\,(z,\ \text{expand-tasks-requests}\,(r2,\ b)),$
$\quad\quad \text{expand-tasks-requests}\,(\text{active-task-requests}\,(0,\ r),\ b))$
$\quad = \ \text{unfulfilled}\,(0,$
$\quad\quad\quad \text{edf}\,(z,\ \text{expand-tasks-requests}\,(r2,\ b)),$
$\quad\quad\quad \text{expand-tasks-requests}\,(\text{active-task-requests}\,(0,\ r),$
$\quad\quad\quad\quad b)))$

```
;(prove-lemma edf-simple-expand-tasks-requests (rewrite)
;  (implies
;   (and
;    (lessp 1 bigp)
;    (non-overlapping-requests r)
;    (valid-requests r))
;  (equal
;   (edf n (expand-tasks-requests r bigp))
;   (append
;    (expand-list bigp (edf (quotient n bigp) r))
;    (repeat (remainder n bigp)
;            (if (listp (unfulfilled
;                        (quotient n bigp)
;                        (edf (quotient n bigp) r)
;                        (active-task-requests (quotient n bigp) r)))
;                (car (least-deadline
;                      (unfulfilled
;                       (quotient n bigp)
;                       (edf (quotient n bigp) r)
;                       (active-task-requests (quotient n bigp) r))))
;              nil)))))
;  ((induct (plus n bigp))
;   (disable-theory t)
;   (enable-theory naturals ground-zero task-abbr)
;   (disable times-add1)
;   (enable edf expand-list listp-edf-simple listp-expand-list
;           valid-requests-unfulfilled unfulfilled-expanded-on-line-beginning
;           equal-car-least-deadline-nil
;           length-edf-simple length-expand-list plist-expand-list
;           firstn-edf-simple equal-append equal-repeat-repeat
;           equal-cons-repeat lessp-1-means remainder-add1-plus-special
;           equal-repeat-nil  firstn-0 lessp-times-sub1-sub1 firstn
;           unfulfilled-sub1-expanded-on-line   unfulfilled-expanded-on-line
;           sublistp-x-x unfulfilled-too-big
;           car-least-deadline-expand-tasks-requests
;           nthcdr-expand-list listp-expand-tasks-requests
;           valid-requests-active-task-requests
;           expand-list-repeat unfulfilled-too-big-sub1
;           firstn-expand-list unfulfilled-0-edf
;           remainder-hack unfulfilled-0-expand-tasks-requests
;           occurrences-expand-list
;           unfulfilled-expand-list
;           equal-remainder-sub1-0 lessp-sub1-plus-hack
;           active-task-requests-expand-tasks-requests append-nil
```

```
;           remainder-plus-add1-hack quotient-plus-add1-hack
;           firstn-append firstn-edf-simple length-nthcdr plist-repeat
;           plist-edf-simple listp-edf-simple lessp-round-means
;           firstn-repeat length-repeat nthcdr-repeat length
;           plist-expand-list quotient-add1-plus-special repeat-1
;           plist-nthcdr nthcdr-is-nil length-append expand-list-append
;           nlistp-nthcdr listp-nthcdr nthcdr-append nthcdr listp-repeat
;           equal-nil-expand-list
;           times-1-arg2
;           repeat expand-list)))
```

THEOREM: edf-simple-expand-tasks-requests

$((1 < bigp) \land \text{non-overlapping-requests}(r) \land \text{valid-requests}(r))$
$\rightarrow\ (\text{edf}(n,\ \text{expand-tasks-requests}(r,\ bigp))$
$\quad = \text{append}(\text{expand-list}(bigp,\ \text{edf}(n \div bigp,\ r)),$
$\qquad \textbf{let}\ undone\ \textbf{be}\ \text{unfulfilled}(n \div bigp,\ \text{edf}(n \div bigp,\ r),\ \text{active-task-requests}(n \div bigp,\ r))$
$\qquad \textbf{in}$
$\qquad \text{repeat}(n\ \textbf{mod}\ bigp,\ \textbf{if}\ \text{listp}(undone)\ \textbf{then}\ \text{car}(\text{least-deadline}(undone))\ \textbf{else nil endif})\ \textbf{endlet}))$

THEOREM: quotient-difference-special

$(((a * b) - (a * c)) \div a)$
$= \textbf{if}\ a \simeq 0\ \textbf{then}\ 0\ \textbf{else}\ b - c\ \textbf{endif}$

THEOREM: good-schedule-expand-tasks-reduces

$((1 < bigp)$
$\land\ \text{non-overlapping-requests}(r1)$
$\land\ \text{sublistp}(r2,\ r1)$
$\land\ \text{valid-requests}(r1)$
$\land\ \text{valid-requests}(r2)$
$\land\ ((n\ \textbf{mod}\ bigp) = 0)$
$\land\ \text{good-schedule}(\text{edf}(n,\ \text{expand-tasks-requests}(r1,\ bigp)),\ \text{expand-tasks-requests}(r2,\ bigp)))$
$\rightarrow\ \text{good-schedule}(\text{edf}(n \div bigp,\ r1),\ r2)$

THEOREM: expand-tasks-1

$\text{periodic-tasksp}(pts) \rightarrow (\text{expand-tasks}(pts,\ 1) = pts)$

THEOREM: cpu-utilization-expand-tasks
periodic-tasksp (pts)
$\rightarrow$ (cpu-utilization (expand-tasks (pts, n), $n2$)
= **if** $n \simeq 0$ **then** 0
else cpu-utilization (pts, $n2$) **endif**)

THEOREM: big-period-expand-tasks
big-period (expand-tasks (pts, n))
= (exp (n, length (pts)) $*$ big-period (pts))

THEOREM: remainder-from-exp
(((n **mod** exp (x, y)) = 0) $\wedge$ ($y \not\simeq 0$)) $\rightarrow$ ((n **mod** x) = 0)

THEOREM: good-edf-help
((big-period (pts) $\not<$ cpu-utilization (pts, big-period (pts)))
$\wedge$ periodic-tasksp (pts)
$\wedge$ listp (pts)
$\wedge$ (1 < big-period (pts))
$\wedge$ ((n **mod** exp (big-period (pts), length (pts))) = 0))
$\rightarrow$ good-schedule (edf (n, periodic-tasks-requests (pts, 0, n)),
periodic-tasks-requests (pts, 0, n))

THEOREM: good-edf-with-remainder
((big-period (pts) $\not<$ cpu-utilization (pts, big-period (pts)))
$\wedge$ periodic-tasksp (pts)
$\wedge$ (1 < big-period (pts))
$\wedge$ ((n **mod** exp (big-period (pts), length (pts))) = 0))
$\rightarrow$ good-schedule (edf (n, periodic-tasks-requests (pts, 0, n)),
periodic-tasks-requests (pts, 0, n))

DEFINITION:
requests-deadlines-not-greater (r, $time$)
= **if** listp (r)
then if $time$ < caddar (r)
then requests-deadlines-not-greater (cdr (r), $time$)
else cons (car (r),
requests-deadlines-not-greater (cdr (r), $time$)) **endif**
else nil endif

DEFINITION:
firstn-nthcdr-firstn-induct (a, b, n, l)
= **if** $a \simeq 0$ **then t**
else firstn-nthcdr-firstn-induct ($a - 1$, $b - 1$, $n - 1$, l) **endif**

THEOREM: firstn-nthcdr-firstn-simple
($n \not< (a + b)$)
$\rightarrow$ (firstn (a, nthcdr (b, firstn (n, l))) = firstn (a, nthcdr (b, l)))

THEOREM: good-schedule-requests-not-greater
(good-schedule $(s,\ r)$ $\land$ valid-requests (r))
$\rightarrow$ good-schedule (firstn $(n,\ s)$, requests-deadlines-not-greater $(r,\ n)$)

THEOREM: requests-deadlines-append
requests-deadlines-not-greater (append $(a,\ b)$, $time$)
= append (requests-deadlines-not-greater $(a,\ time)$,
requests-deadlines-not-greater $(b,\ time)$)

THEOREM: plist-requests-deadlines
plist (requests-deadlines-not-greater $(a,\ time)$)
= requests-deadlines-not-greater $(a,\ time)$

THEOREM: requests-deadlines-periodic-task-simple
$(n1 \not< n2)$
$\rightarrow$ (requests-deadlines-not-greater (periodic-task-requests $(pt,\ n1,\ n)$, $n2$)
= **nil**)

THEOREM: equal-nil-periodic-task-requests
(periodic-task-requests $(pt,\ n1,\ n2)$ = **nil**)
= (($\neg$ periodic-taskp (pt)) $\lor$ $(n1 \not< n2)$)

THEOREM: requests-deadlines-not-greater-periodic-task
(((n **mod** cadr (pt)) = 0)
$\land$ (($n2$ **mod** cadr (pt)) = 0)
$\land$ (($n1$ **mod** cadr (pt)) = 0))
$\rightarrow$ (requests-deadlines-not-greater (periodic-task-requests $(pt,\ n1,\ n2)$, n)
= **if** $n < n2$ **then** periodic-task-requests $(pt,\ n1,\ n)$
else periodic-task-requests $(pt,\ n1,\ n2)$ **endif**)

THEOREM: equal-remainder-big-period
$(e \in l)$ $\rightarrow$ ((big-period (l) **mod** cadr (e)) = 0)

THEOREM: remainder-period-0-if-big-period
(periodic-tasksp (pts) $\land$ ((n **mod** big-period (pts)) = 0) $\land$ $(pt \in pts)$)
$\rightarrow$ ((n **mod** cadr (pt)) = 0)

THEOREM: equal-nil-periodic-tasks-requests
(**nil** = periodic-tasks-requests $(pts,\ n1,\ n2)$)
= (($pts \simeq$ **nil**) $\lor$ ($\neg$ periodic-tasksp (pts)) $\lor$ $(n1 \not< n2)$)

THEOREM: requests-deadlines-not-greater-periodic-help
(periodic-tasksp $(pts2)$
$\land$ sublistp $(pts1,\ pts2)$
$\land$ $(n1 = 0)$
$\land$ $(n2 \not< n3)$

$\land \quad ((n2 \textbf{ mod } \text{big-period}(pts2)) = 0)$
$\land \quad ((n3 \textbf{ mod } \text{big-period}(pts2)) = 0))$
$\rightarrow \quad (\text{requests-deadlines-not-greater}(\text{periodic-tasks-requests}(pts1, n1, n2), n3)$
$\quad = \quad \text{periodic-tasks-requests}(pts1, n1, n3))$

THEOREM: requests-deadlines-not-greater-periodic-rewrite
(periodic-tasksp (pts)
$\land \quad (n1 = 0)$
$\land \quad (n2 \not< n3)$
$\land \quad ((n2 \textbf{ mod } \text{big-period}(pts)) = 0)$
$\land \quad ((n3 \textbf{ mod } \text{big-period}(pts)) = 0))$
$\rightarrow \quad (\text{requests-deadlines-not-greater}(\text{periodic-tasks-requests}(pts, n1, n2), n3)$
$\quad = \quad \text{periodic-tasks-requests}(pts, n1, n3))$

THEOREM: plist-active-task-requests
$\text{plist}(\text{active-task-requests}(n, r)) = \text{active-task-requests}(n, r)$

THEOREM: lessp-plus-times-hack
$((x + (x * y)) < (x * z)) = ((x \not\simeq 0) \land ((1 + y) < z))$

THEOREM: periodic-task-requests-simple
$(n1 \not< n2) \rightarrow (\text{periodic-task-requests}(pt, n1, n2) = \textbf{nil})$

THEOREM: active-task-requests-periodic-task-requests-simple
$(n < n1)$
$\rightarrow \quad (\text{active-task-requests}(n, \text{periodic-task-requests}(pt, n1, n2)) = \textbf{nil})$

THEOREM: active-task-requests-periodic-task-requests
$(((n1 \textbf{ mod } \text{cadr}(pt)) = 0)$
$\land \quad ((n2 \textbf{ mod } \text{cadr}(pt)) = 0)$
$\land \quad ((n \textbf{ mod } \text{cadr}(pt)) = 0)$
$\land \quad (n1 \in \mathbf{N}))$
$\rightarrow \quad (\text{active-task-requests}(n, \text{periodic-task-requests}(pt, n1, n2))$
$\quad = \quad \textbf{if } (\neg \text{periodic-taskp}(pt)) \lor (n < n1) \lor (n \not< n2)$
$\quad\quad \textbf{then nil}$
$\quad\quad \textbf{else } \text{list}(\text{list}(\text{car}(pt),$
$\quad\quad\quad \text{cadr}(pt) * (n \div \text{cadr}(pt)),$
$\quad\quad\quad \text{cadr}(pt)$
$\quad\quad\quad + \quad (\text{cadr}(pt) * (n \div \text{cadr}(pt))),$
$\quad\quad\quad \text{caddr}(pt))) \textbf{ endif})$

THEOREM: active-task-requests-not-greater
$\text{active-task-requests}(n1, \text{requests-deadlines-not-greater}(r, n2))$
$= \quad \text{requests-deadlines-not-greater}(\text{active-task-requests}(n1, r), n2)$

Definition:

requests-starts-earlier $(r,\ time)$

$= \quad$ **if** listp (r)
then if cadar $(r) < time$
then cons (car (r), requests-starts-earlier (cdr (r), $time$))
else requests-starts-earlier (cdr (r), $time$) **endif**
else nil endif

Theorem: requests-starts-earlier-append

requests-starts-earlier (append $(a,\ b)$, n)
$=$ append (requests-starts-earlier $(a,\ n)$, requests-starts-earlier $(b,\ n)$)

Theorem: active-task-requests-requests-starts-earlier

$(n < n1)$
$\rightarrow$ (active-task-requests (n, requests-starts-earlier $(r,\ n1)$)
$=$ active-task-requests $(n,\ r)$)

Theorem: edf-requests-starts-earlier

$(n1 \not< n) \rightarrow$ (edf (n, requests-starts-earlier $(r,\ n1)$) $=$ edf $(n,\ r)$)

Theorem: plist-requests-starts-earlier

plist (requests-starts-earlier $(r,\ n)$) $=$ requests-starts-earlier $(r,\ n)$

Theorem: requests-starts-earlier-periodic-task

(((n **mod** cadr (pt)) $= 0$)
$\wedge$ (($n1$ **mod** cadr (pt)) $= 0$)
$\wedge$ (($n2$ **mod** cadr (pt)) $= 0$))
$\rightarrow$ (requests-starts-earlier (periodic-task-requests $(pt,\ n1,\ n2)$, n)
$=$ periodic-task-requests (pt,
$n1$,
if $n < n2$ **then** n
else $n2$ **endif**))

Theorem: requests-starts-earlier-periodic-tasks

(((n **mod** big-period $(pts2)$) $= 0$)
$\wedge$ (($n1$ **mod** big-period $(pts2)$) $= 0$)
$\wedge$ (($n2$ **mod** big-period $(pts2)$) $= 0$)
$\wedge$ sublistp $(pts1,\ pts2)$
$\wedge$ periodic-tasksp $(pts2)$)
$\rightarrow$ (requests-starts-earlier (periodic-tasks-requests $(pts1,\ n1,\ n2)$, n)
$=$ periodic-tasks-requests ($pts1$,
$n1$,
if $n < n2$ **then** n
else $n2$ **endif**))

THEOREM: good-edf-almost

$$((\text{big-period}(pts) \not< \text{cpu-utilization}(pts, \text{big-period}(pts))) \land \text{periodic-tasksp}(pts) \land (1 < \text{big-period}(pts)) \land ((n \ \mathbf{mod}\ \text{big-period}(pts)) = 0)) \to \text{good-schedule}(\text{edf}(n, \text{periodic-tasks-requests}(pts, 0, n)), \text{periodic-tasks-requests}(pts, 0, n))$$

THEOREM: equal-plus-1

$$((a + b) = 1) = (((a \simeq 0) \land (b = 1)) \lor ((b \simeq 0) \land (a = 1)))$$

THEOREM: equal-cpu-utilization-0

$$(((n \ \mathbf{mod}\ \text{big-period}(x)) = 0) \land (n \not\simeq 0) \land \text{periodic-tasksp}(x)) \to ((\text{cpu-utilization}(x, n) = 0) = (x = \mathbf{nil}))$$

THEOREM: equal-1-big-period

$$((\text{big-period}(pts) \not< \text{cpu-utilization}(pts, \text{big-period}(pts))) \land (\text{big-period}(pts) = 1)) \to (\text{periodic-tasksp}(pts) = ((pts = \mathbf{nil}) \lor ((\text{plist}(pts) = pts) \land (\text{length}(pts) = 1) \land \text{periodic-taskp}(\text{car}(pts)) \land (\text{cadar}(pts) = 1) \land (\text{caddar}(pts) = 1))))$$

DEFINITION:

simple-requests $(v, n1, n2)$
= **if** $n1 < n2$
then cons (list $(v, n1, 1 + n1, 1)$, simple-requests $(v, 1 + n1, n2)$)
else nil endif

THEOREM: periodic-task-requests-as-simple-requests

periodic-task-requests (list $(v, 1, 1)$, $n1$, $n2$)
= **if** $(\neg$ litatom $(v)) \lor (v = \mathbf{nil})$ **then nil**
else simple-requests $(v, n1, n2)$ **endif**

THEOREM: active-task-requests-simple-requests

$((n1 \in \mathbf{N}) \land (n \in \mathbf{N}))$
$\to$ (active-task-requests (n, simple-requests $(v, n1, n2)$)
= **if** $n < n1$ **then nil**
elseif $n < n2$ **then** list (list $(v, n, 1 + n, 1)$)
else nil endif)

THEOREM: nthcdr-cons-cons-repeat
nthcdr $(n,$ cons $(a,$ cons $(a,$ repeat $(m,\ a))))$
$=$ **if** $(2 + m) < n$ **then** 0
else repeat $((2 + m) - n,\ a)$ **endif**

THEOREM: requests-starts-earlier-simple-requests
requests-starts-earlier (simple-requests $(v,\ n1,\ n2),\ n)$
$=$ **if** $n < n2$ **then** simple-requests $(v,\ n1,\ n)$
else simple-requests $(v,\ n1,\ n2)$ **endif**

THEOREM: edf-simple-requests-too-big
$(z < n2)$
$\rightarrow$ (edf $(z,$ simple-requests $(v,\ n1,\ n2)) =$ edf $(z,$ simple-requests $(v,\ n1,\ z)))$

THEOREM: edf-n-simple-requests
edf $(n,$ simple-requests $(v,\ 0,\ n)) =$ repeat $(n,\ v)$

THEOREM: good-schedule-repeat-simple
good-schedule (repeat $(n,\ v)$, simple-requests $(v,\ n1,\ n))$

THEOREM: plist-simple-requests
plist (simple-requests $(v,\ n1,\ n2)) =$ simple-requests $(v,\ n1,\ n2)$

THEOREM: good-edf-bigp-1
((big-period $(pts) \not< $ cpu-utilization $(pts,$ big-period $(pts)))$
$\wedge$ periodic-tasksp (pts)
$\wedge$ (big-period $(pts) = 1)$
$\wedge$ $((n$ **mod** big-period $(pts)) = 0))$
$\rightarrow$ good-schedule (edf $(n,$ periodic-tasks-requests $(pts,\ 0,\ n)),$
periodic-tasks-requests $(pts,\ 0,\ n))$

THEOREM: good-edf-periodic
((big-period $(pts) \not< $ cpu-utilization $(pts,$ big-period $(pts)))$
$\wedge$ periodic-tasksp (pts)
$\wedge$ $((n$ **mod** big-period $(pts)) = 0))$
$\rightarrow$ good-schedule (edf $(n,$ periodic-tasks-requests $(pts,\ 0,\ n)),$
periodic-tasks-requests $(pts,\ 0,\ n))$

EVENT: Make the library `"scheduler"` and compile it.

Index