#|

|#

Event: Start with the library `"fortran"` using the compiled version.

Event: For efficiency, compile those definitions not yet compiled.

Theorem: zplus-comm1
$\text{zplus}\,(x,\ y) = \text{zplus}\,(y,\ x)$

Theorem: zplus-comm2
$\text{zplus}\,(x,\ \text{zplus}\,(y,\ z)) = \text{zplus}\,(y,\ \text{zplus}\,(x,\ z))$

Theorem: zplus-assoc
$\text{zplus}\,(\text{zplus}\,(x,\ y),\ z) = \text{zplus}\,(x,\ \text{zplus}\,(y,\ z))$

Event: Disable zplus.

EVENT: Add the shell *vehicle-state*, with recognizer function symbol *vehicle-statep* and 3 accessors: w, with type restriction (none-of) and default value zero; y, with type restriction (none-of) and default value zero; v, with type restriction (none-of) and default value zero.

DEFINITION: $\text{hd}(x) = \text{car}(x)$

DEFINITION: $\text{tl}(x) = \text{cdr}(x)$

DEFINITION: $\text{empty}(x) = (\neg\ \text{listp}(x))$

THEOREM: tl-rewrite
$\text{tl}(x) = \text{cdr}(x)$

EVENT: Disable tl.

THEOREM: down-on-tl
$(\neg\ \text{empty}(x)) \rightarrow (\text{count}(\text{tl}(x)) < \text{count}(x))$

DEFINITION:
random-delta-ws (*lst*)
= **if** empty (*lst*) **then t**
  else ((hd (*lst*) = -1)
    ∨ (hd (*lst*) = 0)
    ∨ (hd (*lst*) = 1))
    ∧ random-delta-ws (tl (*lst*)) **endif**

DEFINITION:
controller (*sgn-y*, *sgn-old-y*)
= zplus (ztimes (-3, *sgn-y*), ztimes (2, *sgn-old-y*))

EVENT: Disable controller.

DEFINITION:
sgn (x)
= **if** negativep (x)
  then if negative-guts (x) = 0 **then** 0
    else -1 **endif**
  elseif $x \simeq 0$ **then** 0
  else 1 **endif**

EVENT: Disable sgn.

Definition:
next-state (*delta-w*, *state*)
= vehicle-state (zplus (w (*state*), *delta-w*),
zplus (y (*state*), zplus (v (*state*), zplus (w (*state*), *delta-w*))),
zplus (v (*state*),
controller (sgn (zplus (y (*state*),
zplus (v (*state*),
zplus (w (*state*), *delta-w*)))),
sgn (y (*state*)))))

Definition:
final-state-of-vehicle (*delta-ws*, *state*)
= **if** empty (*delta-ws*) **then** *state*
else final-state-of-vehicle (tl (*delta-ws*),
next-state (hd (*delta-ws*), *state*)) **endif**

Definition:
good-statep (*state*)
= **if** y (*state*) = 0
then (zplus (v (*state*), w (*state*)) = -1)
∨ (zplus (v (*state*), w (*state*)) = 0)
∨ (zplus (v (*state*), w (*state*)) = 1)
elseif y (*state*) = 1
then (zplus (v (*state*), w (*state*)) = -2)
∨ (zplus (v (*state*), w (*state*)) = -3)
elseif y (*state*) = 2
then (zplus (v (*state*), w (*state*)) = -1)
∨ (zplus (v (*state*), w (*state*)) = -2)
elseif y (*state*) = 3 **then** zplus (v (*state*), w (*state*)) = -1
elseif y (*state*) = -3 **then** zplus (v (*state*), w (*state*)) = 1
elseif y (*state*) = -2
then (zplus (v (*state*), w (*state*)) = 1)
∨ (zplus (v (*state*), w (*state*)) = 2)
elseif y (*state*) = -1
then (zplus (v (*state*), w (*state*)) = 2)
∨ (zplus (v (*state*), w (*state*)) = 3)
else f endif

Theorem: next-good-state
(good-statep (*state*) ∧ ((*r* = -1) ∨ (*r* = 0) ∨ (*r* = 1)))
→ good-statep (next-state (*r*, *state*))

Definition:
zero-delta-ws (*lst*)
= **if** empty (*lst*) **then t**
else (hd (*lst*) = 0) ∧ zero-delta-ws (tl (*lst*)) **endif**

DEFINITION:
concat $(x,\ y)$
$=$ **if** empty (x) **then** y
else cons (hd (x), concat (tl $(x),\ y)$) **endif**

DEFINITION:
length (x)
$=$ **if** empty (x) **then** `0`
else $1 +$ length (tl (x)) **endif**

THEOREM: length-0
(length $(x) =$ `0`) $=$ empty (x)

THEOREM: decompose-list-of-length-4
zero-delta-ws (*lst*)
$\rightarrow$ ((length (*lst*) $<$ `4`) $=$ (*lst* $\neq$ concat (`'(0 0 0 0)`, cddddr (*lst*))))

THEOREM: drift-to-0-in-4
good-statep (*state*)
$\rightarrow$ (y (final-state-of-vehicle (`'(0 0 0 0)`, *state*)) $=$ `0`)

THEOREM: cancel-wind-in-4
good-statep (*state*)
$\rightarrow$ (zplus (v (final-state-of-vehicle (`'(0 0 0 0)`, *state*)),
w (final-state-of-vehicle (`'(0 0 0 0)`, *state*)))
$=$ `0`)

THEOREM: once-0-always-0
(zero-delta-ws (*lst*)
$\wedge$ (y (*state*) $=$ `0`)
$\wedge$ (zplus (w (*state*), v (*state*)) $=$ `0`))
$\rightarrow$ ((y (final-state-of-vehicle (*lst*, *state*)) $=$ `0`)
$\wedge$ (zplus (v (final-state-of-vehicle (*lst*, *state*)),
w (final-state-of-vehicle (*lst*, *state*)))
$=$ `0`))

THEOREM: final-state-of-vehicle-concat
final-state-of-vehicle (concat $(a,\ b)$, *state*)
$=$ final-state-of-vehicle (b, final-state-of-vehicle (a, *state*))

THEOREM: zero-delta-ws-concat
zero-delta-ws (concat (`'(0 0 0 0)`, v)) $=$ zero-delta-ws (v)

EVENT: Disable concat.

EVENT: Disable next-state.

THEOREM: good-statep-bounded-above
good-statep $(state) \rightarrow (\neg$ zlessp $(3,\ y\,(state)))$

THEOREM: good-statep-bounded-below
good-statep $(state) \rightarrow (\neg$ zlessp $(y\,(state),\ -3))$

EVENT: Disable good-statep.

THEOREM: zlessp-is-lessp
$((x \in \mathbf{N}) \land (y \in \mathbf{N})) \rightarrow (\text{zlessp}\,(x,\ y) = (x < y))$

EVENT: Disable zlessp.

DEFINITION:
fsv $(d,\ s)$
= **if** empty (d) **then** s
else fsv (tl (d), next-state (hd (d), s)) **endif**

THEOREM: all-good-states
(random-delta-ws (lst) $\land$ good-statep $(state)$)
$\rightarrow$ good-statep (final-state-of-vehicle $(lst,\ state)$)

THEOREM: vehicle-stays-within-3-of-course
(random-delta-ws (lst)
$\land$ ($state$ = final-state-of-vehicle (lst, vehicle-state (0, 0, 0))))
$\rightarrow$ (zlesseqp (-3, y $(state)$) $\land$ zlesseqp (y $(state)$, 3))

EVENT: Disable final-state-of-vehicle.

THEOREM: zero-delta-ws-cddddr
zero-delta-ws $(x) \rightarrow$ zero-delta-ws (cddddr (x))

THEOREM: good-states-find-and-stay-at-0
(good-statep $(state)$ $\land$ zero-delta-ws $(lst2)$ $\land$ (length $(lst2) \not< 4$))
$\rightarrow$ (y (final-state-of-vehicle $(lst2,\ state)$) = 0)

THEOREM: vehicle-gets-on-course-in-steady-wind
(random-delta-ws $(lst1)$
$\land$ zero-delta-ws $(lst2)$
$\land$ zgreatereqp (length $(lst2)$, 4)
$\land$ ($state$ = final-state-of-vehicle (concat $(lst1,\ lst2)$,
vehicle-state (0, 0, 0))))
$\rightarrow$ (y $(state)$ = 0)

Index