```
;; >> Might try to relax the assumption that there is only
;;    one train and it always is going in the same direction.
```

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

```
;; SOME SUBSIDIARY LEMMAS
```

THEOREM: not-lessp-sub1
$(x \not< y) \rightarrow ((x < (y - 1)) = \mathbf{f})$

THEOREM: plus-add1
$((x + (1 + y)) = (1 + (x + y)))$
$\land \quad (((1 + x) + y) = (1 + (x + y)))$

THEOREM: difference-add1-sub1
$(x \not\simeq 0) \rightarrow ((x - (1 + y)) = ((x - y) - 1))$

```
;; THE SYSTEM
;; Our simple control system consists of three distinct components.
```

```
;; The train is an part of the environment; it produces in a way that
;; we cannot control but subject to certain constraints that we specify.
;; The gate is our controlled system and responds to a series of commands
;; generated by the controller.  The controller is a device that inputs
;; a series of sensor readings from the environment, giving the position of
;; the train.  It generates a series of actuator commands for the gate.
;;
;; Our task is to model the system comprised of these three components.
```

```
;; GATE SIMULATION
;; The gate is the controlled system, so we consider that it responds
;; reliably to a series of commands that are generated by the control
;; algorithm.  These commands come in the form of a sequence of atoms
;; 'open and 'close.   The simulation of the gate tells us its state
;; at each moment of time.
```

Conservative Axiom: choose-time-intro

$$
\begin{aligned}
&(\mathit{min} \le \mathit{max})\\
\to\quad &((\text{choose-time}\,(\mathit{min},\ \mathit{max},\ \mathit{oracle}) \in \mathbf{N})\\
&\quad \wedge\quad (\text{choose-time}\,(\mathit{min},\ \mathit{max},\ \mathit{oracle}) \not< \mathit{min})\\
&\quad \wedge\quad (\mathit{max} \not< \text{choose-time}\,(\mathit{min},\ \mathit{max},\ \mathit{oracle})))
\end{aligned}
$$

Simultaneously, we introduce the new function symbol *choose-time*.

Theorem: lessp-sub1-choose-sub1-max

$$
\begin{aligned}
&(((\mathit{max} - 1) < x) \wedge (\mathit{min} \le \mathit{max}))\\
\to\quad &(((\text{choose-time}\,(\mathit{min},\ \mathit{max},\ \mathit{oracle}) - 1) < x) = \mathbf{t})
\end{aligned}
$$

```
;; The gate can be in 4 states: open, closed, going-up, going-down.
```

Conservative Axiom: gate-parameters-intro

$$
\begin{aligned}
&(\text{GATE-CLOSING-MIN-TIME} \in \mathbf{N})\\
\wedge\quad &(\text{GATE-CLOSING-MAX-TIME} \in \mathbf{N})\\
\wedge\quad &(0 < \text{GATE-CLOSING-MAX-TIME})\\
\wedge\quad &(\text{GATE-CLOSING-MAX-TIME} \not< \text{GATE-CLOSING-MIN-TIME})\\
\wedge\quad &(\text{GATE-OPENING-MIN-TIME} \in \mathbf{N})\\
\wedge\quad &(0 < \text{GATE-OPENING-MAX-TIME})\\
\wedge\quad &(\text{GATE-OPENING-MAX-TIME} \in \mathbf{N})\\
\wedge\quad &(\text{GATE-OPENING-MAX-TIME} \not< \text{GATE-OPENING-MIN-TIME})
\end{aligned}
$$

Simultaneously, we introduce the new function symbols *gate-closing-min-time*, *gate-closing-max-time*, *gate-opening-min-time*, and *gate-opening-max-time*.

```
;; The gate state is a pair (current-state . n) where n is the time that
;; the gate will remain in that state.  This is only required for going
;; up and going down and tells us how long they can expect to be in that
;; condition unless they are given a countervening order.  This doesn't
;; take into account some anomolous situations as , for example, when we
;; may have just told the gate to close and immediately tell it to open,
;; so that it has only a short distance to move.
```

DEFINITION:
gate-positionp (x) = $(x \in$ `'(open closed going-up going-down)`$)$

DEFINITION: gate-current-state (state) = car (state)

DEFINITION: gate-state-duration (state) = cdr (state)

DEFINITION:
open (state) = (gate-current-state (state) = `'open`)

DEFINITION:
closed (state) = (gate-current-state (state) = `'closed`)

DEFINITION:
going-up (state) = (gate-current-state (state) = `'going-up`)

DEFINITION:
going-down (state) = (gate-current-state (state) = `'going-down`)

THEOREM: gate-state-accessors
(gate-current-state (cons $(x,\ y)$) = x)
$\land$ (gate-state-duration (cons $(x,\ y)$) = y)

DEFINITION:
legal-gate-statep (x)
= (gate-positionp (gate-current-state (x))
 $\land$ ((going-up (x) $\lor$ going-down (x))
 $\rightarrow$ (gate-state-duration $(x) \in \mathbf{N}$)))

EVENT: Disable gate-current-state.

EVENT: Disable gate-state-duration.

DEFINITION:
compute-going-down-state (state)
= **let** n **be** gate-state-duration (state)
in
if $n \simeq 0$ **then** cons (`'closed`, `0`)
else cons (`'going-down`, $n - 1$) **endif endlet**

DEFINITION:
compute-going-up-state (*state*)
= **let** n **be** gate-state-duration (*state*)
in
if $n \simeq$ `0` **then** cons (`'open`, `0`)
else cons (`'going-up`, $n - 1$) **endif endlet**

DEFINITION:
gate-next-state (*cmd*, *state*, *oracle*)
= **case on** *cmd*:
case = *close*
then case on gate-current-state (*state*):
case = *open*
then cons (`'going-down`,
choose-time (GATE-CLOSING-MIN-TIME,
GATE-CLOSING-MAX-TIME,
oracle) $- 1$)
case = *closed*
then *state*
case = *going-up*
then cons (`'going-down`,
choose-time (GATE-CLOSING-MIN-TIME,
GATE-CLOSING-MAX-TIME,
oracle) $- 1$)
case = *going-down*
then compute-going-down-state (*state*)
otherwise f endcase
case = *open*
then case on gate-current-state (*state*):
case = *open*
then *state*
case = *closed*
then cons (`'going-up`,
choose-time (GATE-OPENING-MIN-TIME,
GATE-OPENING-MAX-TIME,
oracle) $- 1$)
case = *going-up*
then compute-going-up-state (*state*)
case = *going-down*
then cons (`'going-up`,
choose-time (GATE-OPENING-MIN-TIME,
GATE-OPENING-MAX-TIME,
oracle) $- 1$)
otherwise f endcase

otherwise f endcase

```
;; TRAIN BEHAVIOR
;; The train provides our input to the system.  The gate must act in
;; response to the train.  It can do so only if the behavior of the
;; train meets certain reasonable criteria.  Therefore, the specification
;; of the train consists of a set of constraints on the possible traces of
;; the train behavior as sensed by input sensors along the track.  The
;; train can be in one of 4 states: approaching the crossing,
;; in the crossing, past the crossing (or outside of our field of view).
;; We assume that the sensors are accurate (debounced) and that trains are
;; sufficiently widely separated.
```

DEFINITION:
train-positionp $(x) = (x \in$ `'(approaching in-gate elsewhere)`$)$

DEFINITION: approaching $(x) = (x =$ `'approaching`$)$

DEFINITION: in-gate $(x) = (x =$ `'in-gate`$)$

DEFINITION: elsewhere $(x) = (x =$ `'elsewhere`$)$

DEFINITION:
legal-next-positions (x)
= **case on** x:
case = *elsewhere*
then `'(elsewhere approaching)`
case = *approaching*
then `'(approaching in-gate)`
case = *in-gate*
then `'(in-gate elsewhere)`
otherwise nil endcase

DEFINITION:
legal-transitionp $(x,\ y) = (y \in$ legal-next-positions $(x))$

DEFINITION:
legal-train-trace1 (*trace*)
= **if** *trace* $\simeq$ **nil then** *trace* = **nil**
else train-positionp (car (*trace*))
$\wedge$ **if** listp (cdr (*trace*))
then legal-transitionp (car (*trace*), cadr (*trace*))
else t endif
$\wedge$ legal-train-trace1 (cdr (*trace*)) **endif**

CONSERVATIVE AXIOM: train-constraints
$(\textsc{approaching-min-time} \in \mathbf{N})$
$\wedge \quad (\textsc{approaching-min-time} \not< (1 + \textsc{gate-closing-max-time}))$

Simultaneously, we introduce the new function symbol *approaching-min-time*.

DEFINITION:
seq-long-enough (x, *trace*, *seen-so-far*, *min*)
= **if** *seen-so-far* $\simeq$ `0`
  then if *trace* $\simeq$ **nil** **then t**
    elseif car (*trace*) = x
    then seq-long-enough (x, cdr (*trace*), `1`, *min*)
    else seq-long-enough (x, cdr (*trace*), `0`, *min*) **endif**
  elseif *trace* $\simeq$ **nil** **then** *min* $\leq$ *seen-so-far*
  elseif car (*trace*) = x
  then seq-long-enough (x, cdr (*trace*), 1 + *seen-so-far*, *min*)
  else (*min* $\leq$ *seen-so-far*)
    $\wedge$ seq-long-enough (x, cdr (*trace*), `0`, *min*) **endif**

DEFINITION:
approaches-long-enough (*trace*, *seen-so-far*)
= seq-long-enough (`'approaching`,
    trace,
    seen-so-far,
    APPROACHING-MIN-TIME)

DEFINITION:
distance-to-gate (*trace*)
= **if** *trace* $\simeq$ **nil** **then** 1 + (1 + APPROACHING-MIN-TIME)
  elseif in-gate (car (*trace*)) **then** `0`
  else 1 + distance-to-gate (cdr (*trace*)) **endif**

EVENT: Disable approaches-long-enough.

DEFINITION:
legal-train-trace (*trace*, n)
= (legal-train-trace1 (*trace*) $\wedge$ approaches-long-enough (*trace*, n))

```
;; CONTROLLER
;; The controller takes as input a sequence of readings from the track
;; sensors telling where the train is.  It's output is a sequence of
;; actuator commands of the form 'open or 'close.  The gate responds to
;; these.
```

DEFINITION:
control-output (*input*)
= **if** *input* = `'elsewhere` **then** `'open`
  else `'close` **endif**

```
;; THE SYSTEM
```

DEFINITION:
gate-behavior (*train-trace*, *gate-state*)
= **let** *next-state* **be** gate-next-state (control-output (car (*train-trace*)),
  gate-state,
  train-trace)
  in
  if *train-trace* $\simeq$ **nil then nil**
  else cons (*next-state*,
  gate-behavior (cdr (*train-trace*), *next-state*)) **endif endlet**

THEOREM: approaches-stay-long-enough
(approaches-long-enough (*train-trace*, *approaching-time*)
$\land$ (car (*train-trace*) = `'approaching`))
$\rightarrow$ approaches-long-enough (cdr (*train-trace*), 1 + *approaching-time*)

```
;; Now we state safety properties of this system.  In particular,
;; we devise constraints that assure that the gate will always be
;; closed when the train is in the gate.
```

THEOREM: approaches-long-enough-zero
listp (*train-trace*)
$\rightarrow$ (approaches-long-enough (*train-trace*, *approaching-time*)
  = **if** *approaching-time* $\simeq$ `0`
  then if car (*train-trace*) = `'approaching`
  then approaches-long-enough (cdr (*train-trace*), `1`)
  else approaches-long-enough (cdr (*train-trace*), `0`) **endif**
  elseif car (*train-trace*) = `'approaching`
  then approaches-long-enough (cdr (*train-trace*),
  1 + *approaching-time*)
  else approaches-long-enough (cdr (*train-trace*), `0`)
  $\land$ (*approaching-time* $\not<$ APPROACHING-MIN-TIME) **endif**)

DEFINITION:
distance-to-gate-induction (*train-trace*, *n*)
= **if** *train-trace* $\simeq$ **nil then t**
  elseif cdr (*train-trace*) $\simeq$ **nil then t**

elseif car ($train\text{-}trace$) = `'approaching`
then distance-to-gate-induction (cdr ($train\text{-}trace$), $1 + n$)
else distance-to-gate-induction (cdr ($train\text{-}trace$), `0`) **endif**

THEOREM: distance-to-gate-first-approaching
(listp ($train\text{-}trace$)
 $\land$ (car ($train\text{-}trace$) = `'approaching`)
 $\land$ approaches-long-enough ($train\text{-}trace$, n))
$\rightarrow$ (APPROACHING-MIN-TIME $<$ (n + (1 + distance-to-gate ($train\text{-}trace$))))

THEOREM: distance-to-gate-first-approaching2
(listp ($train\text{-}trace$)
 $\land$ (car ($train\text{-}trace$) = `'approaching`)
 $\land$ approaches-long-enough ($train\text{-}trace$, `0`))
$\rightarrow$ ((APPROACHING-MIN-TIME $-$ 1) $<$ distance-to-gate ($train\text{-}trace$))

DEFINITION:
good-statep ($train\text{-}state$, $gate\text{-}state$, $distance\text{-}to\text{-}gate$, $time\text{-}elsewhere$)
= ((going-down ($gate\text{-}state$)
    $\rightarrow$ (gate-state-duration ($gate\text{-}state$) $<$ GATE-CLOSING-MAX-TIME))
  $\land$ (going-up ($gate\text{-}state$)
      $\rightarrow$ (gate-state-duration ($gate\text{-}state$)
          $<$ GATE-OPENING-MAX-TIME))
  $\land$ **case on** $train\text{-}state$:
    case = $in\text{-}gate$
    then closed ($gate\text{-}state$)
    case = $approaching$
     then closed ($gate\text{-}state$)
         $\lor$ (going-down ($gate\text{-}state$)
             $\land$ (gate-state-duration ($gate\text{-}state$)
                 $<$ $distance\text{-}to\text{-}gate$))
         $\lor$ (GATE-CLOSING-MAX-TIME $<$ $distance\text{-}to\text{-}gate$)
    case = $elsewhere$
     then ((GATE-OPENING-MAX-TIME $<$ $time\text{-}elsewhere$)
          $\rightarrow$ open ($gate\text{-}state$))
         $\land$ ((closed ($gate\text{-}state$) $\land$ ($time\text{-}elsewhere \simeq$ `0`))
             $\lor$ open ($gate\text{-}state$)
             $\lor$ (going-up ($gate\text{-}state$)
                 $\land$ (gate-state-duration ($gate\text{-}state$)
                     $\leq$ (GATE-OPENING-MAX-TIME
                         $-$ $time\text{-}elsewhere$))))
    otherwise f endcase)

DEFINITION:
su-invariant ($train\text{-}trace$, $gate\text{-}trace$, $time\text{-}elsewhere$)

$=$ **if** *train-trace* $\simeq$ **nil** **then** **t**
elseif *gate-trace* $\simeq$ **nil** **then** **f**
else good-statep (car (*train-trace*),
car (*gate-trace*),
distance-to-gate (*train-trace*),
time-elsewhere)
$\wedge$ su-invariant (cdr (*train-trace*),
cdr (*gate-trace*),
if elsewhere (car (*train-trace*))
then 1 + *time-elsewhere*
else `0` **endif**) **endif**

DEFINITION:
controller-induction (*train-trace*, *gate-state*, *time-approaching*, *time-elsewhere*)
$=$ **if** *train-trace* $\simeq$ **nil** **then** **t**
elseif cdr (*train-trace*) $\simeq$ **nil** **then** **t**
else controller-induction (cdr (*train-trace*),
gate-next-state (control-output (car (*train-trace*)),
gate-state,
train-trace),
if car (*train-trace*)
$=$ `'approaching`
then 1 + *time-approaching*
else `0` **endif**,
if car (*train-trace*)
$=$ `'elsewhere`
then 1 + *time-elsewhere*
else `0` **endif**) **endif**

THEOREM: controller-su-invariant
(legal-train-trace (*train-trace*, *time-approaching*)
$\wedge$ legal-gate-statep (*gate-state*)
$\wedge$ good-statep (car (*train-trace*),
gate-state,
distance-to-gate (*train-trace*),
time-elsewhere))
$\rightarrow$ su-invariant (*train-trace*,
gate-behavior (*train-trace*, *gate-state*),
time-elsewhere)

```
;; Now we define the desired safety and utility properties and prove that
;; they follow from the su-invariant property.
```

DEFINITION:

safety (*train-trace*, *gate-trace*)
= **if** *train-trace* ≃ **nil** **then t**
elseif in-gate (car (*train-trace*))
then closed (car (*gate-trace*))
∧ safety (cdr (*train-trace*), cdr (*gate-trace*))
else safety (cdr (*train-trace*), cdr (*gate-trace*)) **endif**

THEOREM: su-invariant-implies-safety
su-invariant (*train-trace*, *gate-trace*, *time-elsewhere*)
→ safety (*train-trace*, *gate-trace*)

DEFINITION:
utility (*train-trace*, *gate-trace*, *time-elsewhere*)
= **if** *train-trace* ≃ **nil** **then t**
elseif elsewhere (car (*train-trace*))
then if GATE-OPENING-MAX-TIME < *time-elsewhere*
then open (car (*gate-trace*))
else t endif
∧ utility (cdr (*train-trace*),
cdr (*gate-trace*),
1 + *time-elsewhere*)
else utility (cdr (*train-trace*), cdr (*gate-trace*), `0`) **endif**

THEOREM: su-invariant-implies-utility
su-invariant (*train-trace*, *gate-trace*, *time-elsewhere*)
→ utility (*train-trace*, *gate-trace*, *time-elsewhere*)

THEOREM: controller-maintains-safety-and-utility
(legal-train-trace (*train-trace*, *time-approaching*)
∧ legal-gate-statep (*gate-state*)
∧ good-statep (car (*train-trace*),
gate-state,
distance-to-gate (*train-trace*),
time-elsewhere))
→ (safety (*train-trace*, gate-behavior (*train-trace*, *gate-state*))
∧ utility (*train-trace*,
gate-behavior (*train-trace*, *gate-state*),
time-elsewhere))

```
;; We now characterize an initial state and show that our invariant
;; is true in that initial state.
```

DEFINITION:
initial-statep (*train-state*, *gate-state*)
= ((*train-state* = `'elsewhere`)
∧ (gate-current-state (*gate-state*) = `'open`))

THEOREM: controller-safety
$$(\text{initial-statep}(\text{car}(\textit{train-trace}), \textit{gate-state}) \wedge \text{legal-train-trace}(\textit{train-trace}, \textit{time-approaching})) \rightarrow \text{safety}(\textit{train-trace}, \text{gate-behavior}(\textit{train-trace}, \textit{gate-state}))$$

THEOREM: controller-utility
$$(\text{initial-statep}(\text{car}(\textit{train-trace}), \textit{gate-state}) \wedge \text{legal-train-trace}(\textit{train-trace}, \textit{time-approaching})) \rightarrow \text{utility}(\textit{train-trace}, \text{gate-behavior}(\textit{train-trace}, \textit{gate-state}), \textit{time-elsewhere})$$

```
;; AN ALTERNATIVE SPECIFICATION
;; The criticism may be raised that our specification functions
;; SAFETY and UTILITY are harder to understand than the alternatives
;; available in some other specification languages, using quantifiers
;; for example.  We offer alternative versions SAFETY2 and UTILITY2
;; that are in a quantified style and prove that they are
;; consequences of our versions.
```

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

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

DEFINITION:
safety2 $(\textit{train-trace}, \textit{gate-trace})$
$\leftrightarrow \quad \forall\, i\ (\text{in-gate}(\text{get}(i, \textit{train-trace})) \rightarrow \text{closed}(\text{get}(i, \textit{gate-trace})))$

THEOREM: safety2-suff
$$(\text{in-gate}(\text{get}(\text{i}(\textit{gate-trace}, \textit{train-trace}), \textit{train-trace})) \rightarrow \text{closed}(\text{get}(\text{i}(\textit{gate-trace}, \textit{train-trace}), \textit{gate-trace}))) \rightarrow \text{safety2}(\textit{train-trace}, \textit{gate-trace})$$

THEOREM: safety2-necc
$$(\neg\, (\text{in-gate}(\text{get}(i, \textit{train-trace})) \rightarrow \text{closed}(\text{get}(i, \textit{gate-trace})))) \rightarrow (\neg\, \text{safety2}(\textit{train-trace}, \textit{gate-trace}))$$

DEFINITION:
get-induct (i, x, y)
$= $ **if** $i \simeq 0$ **then t**
else get-induct $(i - 1, \text{cdr}(x), \text{cdr}(y))$ **endif**

THEOREM: safety-implies-safety2-get
(safety (*train-trace*, *gate-trace*) $\land$ in-gate (get (i, *train-trace*)))
$\rightarrow$ closed (get (i, *gate-trace*))

THEOREM: safety-implies-safety2
safety (*train-trace*, *gate-trace*) $\rightarrow$ safety2 (*train-trace*, *gate-trace*)

THEOREM: controller-safety2
(initial-statep (car (*train-trace*), *gate-state*)
$\land$ legal-train-trace (*train-trace*, *time-approaching*))
$\rightarrow$ safety2 (*train-trace*, gate-behavior (*train-trace*, *gate-state*))

DEFINITION:
time-elsewhere (i, *found-so-far*, *train-trace*)
= **if** $i \simeq 0$ **then** *found-so-far*
elseif elsewhere (car (*train-trace*))
then time-elsewhere ($i - 1$, $1 +$ *found-so-far*, cdr (*train-trace*))
else time-elsewhere ($i - 1$, 0, cdr (*train-trace*)) **endif**

DEFINITION:
utility2 (*train-trace*, *gate-trace*, n)
$\leftrightarrow$ $\forall\, i$ ((elsewhere (get (i, *train-trace*))
$\land$ (GATE-OPENING-MAX-TIME
$<$ time-elsewhere (i, n, *train-trace*)))
$\rightarrow$ open (get (i, *gate-trace*)))

THEOREM: utility2-suff
((elsewhere (get (i-1 (*gate-trace*, n, *train-trace*), *train-trace*))
$\land$ (GATE-OPENING-MAX-TIME
$<$ time-elsewhere (i-1 (*gate-trace*, n, *train-trace*), n, *train-trace*)))
$\rightarrow$ open (get (i-1 (*gate-trace*, n, *train-trace*), *gate-trace*)))
$\rightarrow$ utility2 (*train-trace*, *gate-trace*, n)

THEOREM: utility2-necc
($\neg$ ((elsewhere (get (i, *train-trace*))
$\land$ (GATE-OPENING-MAX-TIME $<$ time-elsewhere (i, n, *train-trace*)))
$\rightarrow$ open (get (i, *gate-trace*))))
$\rightarrow$ ($\neg$ utility2 (*train-trace*, *gate-trace*, n))

DEFINITION:
get-induct2 (i, x, y, n)
= **if** $i \simeq 0$ **then t**
else get-induct2 ($i - 1$,
cdr (x),
cdr (y),
if elsewhere (car (x)) **then** $1 + n$
else 0 **endif**) **endif**

Theorem: utility-implies-utility2-get
$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{utility}\,(\textit{train-trace},\ \textit{gate-trace},\ n)} \\
\ \wedge & \text{elsewhere}\,(\text{get}\,(i,\ \textit{train-trace})) \\
\ \wedge & (\textsc{gate-opening-max-time} < \text{time-elsewhere}\,(i,\ n,\ \textit{train-trace}))) \\
\rightarrow & \text{open}\,(\text{get}\,(i,\ \textit{gate-trace}))
\end{array}
$$

Theorem: utility-implies-utility2
$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{utility}\,(\textit{train-trace},\ \textit{gate-trace},\ \textit{time-elsewhere})} \\
\rightarrow & \text{utility2}\,(\textit{train-trace},\ \textit{gate-trace},\ \textit{time-elsewhere})
\end{array}
$$

Theorem: controller-utility2
$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{initial-statep}\,(\text{car}\,(\textit{train-trace}),\ \textit{gate-state})} \\
\ \wedge & \text{legal-train-trace}\,(\textit{train-trace},\ \textit{time-approaching})) \\
\rightarrow & \text{utility2}\,(\textit{train-trace}, \\
 & \qquad\qquad \text{gate-behavior}\,(\textit{train-trace},\ \textit{gate-state}), \\
 & \qquad\qquad \textit{time-elsewhere})
\end{array}
$$

Index