Mechanical Verification of Real-time Systems[a]

N. Shankar
shankar@csl.sri.com
URL: http://www.csl.sri.com/~shankar/

Computer Science Laboratory
SRI International
Menlo Park, CA

[a] Funded by US Naval Research Laboratory contract N00015-92-C-2177.

Overview

- **Goal:** To show how the the behavior of concurrent, real-time systems can be modelled directly in a specification language based on higher-order logic.
- PVS is used to model and verify real-time behavior:
  1. Mutual exclusion property of Fischer's protocol [Lamport87].
  2. Safety and utility of railroad crossing controller [Heitmeyer&Jeffords&Labaw93].
- Concurrent, real-time behavior is encoded in higher-order logic using traces and a *since* operator.

The Approach

Embed transition systems directly in higher-order logic.

state is an uninterpreted type.

A *program variable* with value type T has type $[\textit{state} \to T]$.

Time is a special program variable over non-negative rationals.

A *trace* is a sequence of states: $[\textit{nat} \to \textit{state}]$.

A *behavior* is a trace where time is non-decreasing and non-Zeno. (Non-Zeno property has not yet been used.)

The Since Operator

An *assertion* is a predicate on *state*.

The type *time* is taken as the nonnegative rational numbers.

$|p|$ (read "since p") has type $[\textit{state} \to \textit{time}]$: the elapsed time since p was last observed true in the behavior.

A behavior σ is *anchored* if $|p|(\sigma(0)) = \textit{Time}(\sigma(0)) + 1$.
All invariants are proved relative to anchored behaviors.

Two rules about $|p|$; for any atomic action Ac

$$\{p \wedge s = \textit{Time}\} \quad Ac \quad \{|p| = \textit{Time} - s\}$$

$$\{\neg p \wedge s = \textit{Time} \wedge t = |p|\} \quad Ac \quad \{|p| = t + \textit{Time} - s\}$$

Theorems About Since

invariant$\{p\}$ is true of behavior σ if p is true of every state of σ.

$\textbf{invariant}\{P \supset Q\} \Rightarrow \textbf{invariant}\{P\} \Rightarrow \textbf{invariant}\{Q\}$.

$\textbf{invariant}\{|(|P| \leq x)| \leq y \supset |P| \leq x + y\}$.

$\textbf{invariant}\{P \supset Q\} \Rightarrow \textbf{invariant}\{|Q| \leq |P|\}$.

$\textbf{invariant}\{(|P \wedge Q| = |P|) \vee (|P \wedge \neg Q| = |P|)\}$.

$\textbf{invariant}\{|P \vee Q| \leq \mathit{min}(|P|, |Q|)\}$.

$\textbf{invariant}\{(|P \vee Q| = |P|) \vee (|P \vee Q| = |Q|)\}$.

$\textbf{invariant}\{\mathit{max}(|P|, |Q|) \leq |P \wedge Q|\}$.

$\textbf{invariant}\{|P| \leq |P \wedge Q|\}$.

An Example: Fischer's Mutual Exclusion Protocol (in Unity Notation)

```
Program mutex

  declare  x : nat,
           PC: [posnat -> {init, try, wait, cs}]
  initially  x = 0 ‖ ⟨‖ i : posnat ::  PC(i) = init ⟩
  assign
     ⟨[]i : posnat ::
         PC(i) := try          if x = 0 ∧ PC(i) = init
       [] x, PC(i) := i, wait   if PC(i) = try
       [] PC(i) := cs           if x = i ∧ PC(i) = wait
     ⟩
end {mutex}
```


Timing Constraints

These are additional assumptions capturing timing behavior.

Time spent by any process i in its `try` state is at most `hi`. More accurately, if i's state is `try`, the time since i was in `init` state is at most `hi`.

$$\textbf{invariant}\{\forall \mathtt{i}\colon \mathtt{PC(i)} = \mathtt{try} \supset |\mathtt{PC(i)} = \mathtt{init}| \leq \mathtt{hi}\}. \qquad (1)$$

Time spent by i in `wait` state is at least `lo`. More accurately, i was last in its `try` state at least `lo` units before it entered its critical section `cs`.

$$\textbf{invariant}\{\forall \mathtt{i}\colon \mathtt{PC(i)} = \mathtt{cs} \supset |\mathtt{PC(i)} = \mathtt{wait}| + \mathtt{lo} \leq |\mathtt{PC(i)} = \mathtt{try}|\}. \qquad (2)$$

$$\mathtt{hi} < \mathtt{lo} \qquad (3)$$

Proof of the Mutex Protocol

$$\textbf{invariant}\{\forall \mathtt{i}\colon \mathtt{x} = \mathtt{i} \supset |\mathtt{PC(i)} = \mathtt{try}| \leq |\mathtt{x} = 0|\}. \qquad (4)$$

Proof: Trivially by induction on atomic actions of `mutex`.

$$\textbf{invariant}\{\forall \mathtt{i}\colon \mathtt{PC(i)} = \mathtt{try} \supset |\mathtt{x} = 0| \leq |\mathtt{PC(i)} = \mathtt{init}|\}. \qquad (5)$$

$$\textbf{invariant}\{\forall \mathtt{j}, \mathtt{i}\colon \mathtt{PC(i)} = \mathtt{cs} \wedge \mathtt{x} = \mathtt{i} \supset \mathtt{PC(j)} \neq \mathtt{try}\}. \qquad (6)$$

Proof: If $\mathtt{PC(i)} = \mathtt{cs}$, $\mathtt{x} = \mathtt{i}$, and $\mathtt{PC(j)} \neq \mathtt{try}$, then:
Invariantly, $|\mathtt{x} = 0| \leq |\mathtt{PC(j)} = \mathtt{init}| \leq \mathtt{hi}$, by (1) and (5).
Invariantly, $\mathtt{lo} \leq |\mathtt{PC(i)} = \mathtt{try}| \leq |\mathtt{x} = 0|$, by (2) and (4). By (3), *contradiction*.

$$\textbf{invariant}\{\forall \mathtt{i}\colon \mathtt{PC(i)} = \mathtt{cs} \supset \mathtt{x} = \mathtt{i}\} \qquad (7)$$

$$\textbf{invariant}\{\forall \mathtt{i}, \mathtt{j}\colon \mathtt{PC(i)} = \mathtt{cs} \wedge \mathtt{PC(j)} = \mathtt{cs} \supset \mathtt{i} = \mathtt{j}\} \qquad (8)$$

Time in PVS

```
time : THEORY
BEGIN
  state: TYPE
  statepred: TYPE = PRED[state]  %assertions
  Action: TYPE =  PRED[state, state]
  pp: VAR statepred
  action: VAR Action
  time : TYPE = {x : rational| x >= 0}
  Time: [state -> time]
  seq, seq1, seq2: VAR sequence[state]
  i, j,k: VAR nat

  nondec?(seq): bool =
    (FORALL i, j:
       i>j IMPLIES Time(seq(j)) <= Time(seq(i)))
  nonzeno?(seq): bool =
    (FORALL (x: rational): (EXISTS i: x < Time(seq(i))))
  behavior?(seq): bool = nondec?(seq) AND nonzeno?(seq)
  behavior : TYPE = (behavior?)
  aa, bb, cc: VAR behavior

  Inv(pp)(aa) : bool =
   (FORALL (n : nat) : pp(aa(n)))

  Init(pp)(aa) : bool = pp(aa(0))
END time
```


Since in PVS

```
since   : THEORY
  BEGIN
  USING time

  i, j,k: VAR nat
  pp: VAR statepred
  action: VAR Action
  aa, bb, cc: VAR behavior

  since: [statepred -> [ state-> time]]

  since_ax: AXIOM
    since(pp)(aa(i+1))
    = (IF pp(aa(i))
          THEN (Time(aa(i+1)) - Time(aa(i)))
          ELSE since(pp)(aa(i))
               + (Time(aa(i+1)) - Time(aa(i)))
       ENDIF)

  behavior0: TYPE =
      {aa: behavior |
        (FORALL pp: since(pp)(aa(0)) = Time(aa(0)) + 1)}

END since
```


Mutex in PVS

```
mutex   : THEORY

  BEGIN
  USING time, since, since_thms

  process : TYPE = posnat
  i, j, k: VAR process

  aa, bb, cc : VAR behavior0

  s, s0, s1 : VAR state

  pp, qq, rr: VAR statepred

  program_counter : TYPE = {init, try, wait, cs}

  x : [state -> nat]
  PC: [state -> [process -> program_counter]]
```


Mutex Actions

```
  Check(s0, s1): bool =
   (EXISTS i: PC(s0)(i) = init
          AND x(s0) = 0
          AND x(s1) = x(s0)
          AND PC(s1) = PC(s0) WITH [(i) := try])

  Ask(s0, s1): bool =
   (EXISTS i: PC(s0)(i) = try
          AND x(s1) = i
          AND PC(s1) = PC(s0) WITH [(i) := wait])

  Doublecheck(s0, s1) :bool =
   (EXISTS i: PC(s0)(i) = wait
          AND x(s0) = i
          AND x(s1) = x(s0)
          AND PC(s1) = PC(s0) WITH [(i) := cs])

  Stutter(s0, s1) : bool =
    (FORALL i: PC(s1)(i) = PC(s0)(i)) AND x(s1) = x(s0)
```

Mutex Program

```
m, n : VAR nat

  program(aa): bool =
   (Init(LAMBDA s: x(s) = 0
              AND (FORALL i: PC(s)(i) = init))(aa)
  AND
   (FORALL n:
     Check(aa(n), aa(n+1)) OR
     Ask(aa(n), aa(n+1)) OR
     Doublecheck(aa(n), aa(n+1)) OR
     Stutter(aa(n), aa(n+1))))

  hi, lo: time

  TI1: AXIOM
  Inv(LAMBDA s:
      (FORALL i: PC(s)(i) = try IMPLIES
       since(LAMBDA s0: PC(s0)(i) = init)(s) <= hi))(aa)

  TI2: AXIOM
    Inv(LAMBDA s:
        (FORALL i: PC(s)(i) = cs IMPLIES
          since(LAMBDA s0: PC(s0)(i) = wait)(s) + lo <=
          since(LAMBDA s0: PC(s0)(i) = try)(s)))(aa)
```


Mutex Invariants

```
  Inv1: LEMMA
    program(aa) IMPLIES
    Inv(LAMBDA s: (FORALL i: x(s) = i  IMPLIES
                 since(LAMBDA s0: PC(s0)(i) = try)(s) <=
                 since(LAMBDA s0: x(s0) = 0)(s)))(aa)

  Inv2: LEMMA
    program(aa) IMPLIES
     Inv(LAMBDA s:
          (FORALL i: PC(s)(i) = try IMPLIES
              since(LAMBDA s0: x(s0) = 0)(s) <=
              since(LAMBDA s0: PC(s0)(i) = init)(s)))(aa)

  Inv2_2: LEMMA
    program(aa) AND hi < lo IMPLIES
    Inv(LAMBDA s:
        (FORALL i: PC(s)(i) = cs AND x(s) = i IMPLIES
           (FORALL j: PC(s)(j)/=try)))(aa)
```


Mutex Safety

```
  Inv3: LEMMA
    program(aa) AND hi < lo IMPLIES
    Inv(LAMBDA s:
        (FORALL i: PC(s)(i) = cs IMPLIES x(s) = i))(aa)

  safety: THEOREM
    program(aa) AND hi < lo IMPLIES
     Inv(LAMBDA s:
         (FORALL i, j: PC(s)(i) = cs AND PC(s)(j) = cs
                  IMPLIES i = j))(aa)
END mutex
```


RR Crossing

Three subsystems: trains, controller, and gate.

Each train goes from `safe` to `approaching` to `crossing` back to `safe`.

The controller signal oscillates between `lower` and `raise`.

Depending on the signal, the gate cycles between `up`, `moving_down`, `down`, and `moving_up`. It can also go from `moving_up` to `moving_down`, and vice-versa.

RR Crossing Program

```
Program RR

  declare train : [nat -> {safe, approaching, crossing}],
          signal : {lower, raise},
          gate : {up, down, moving_up, moving_down}
  initially ⟨∀p: train(p)  =  safe⟩
  assign
   ⟨[]p :
     train(p) := approaching, if train(p) = safe
    [] train(p) := crossing,    if train(p) = approaching
    [] train(p) := safe,        if train(p) = crossing
   ⟩
    [] signal := lower,         if Approaching
                                   ∧ signal ≠ lower
    [] signal := raise,         if Safe ∧ signal = lower
    [] gate := moving_down,     if signal = lower
                                   ∧ gate ≠ down
    [] gate := down,            if gate = moving_down
    [] gate := moving_up,       if signal = raise
                                   ∧ gate ≠ up
    [] gate := up,              if gate = moving_up
end {RR}
```


Timing Constraints

Approaching means some train is in its approaching state.

Crossing means some train is in its crossing state.

As long as some train is approaching, the signal is lowered within D units.

$$\textbf{invariant}\{\ \texttt{Approaching} \land (\texttt{signal} \neq \texttt{lower}) \supset |\neg\texttt{Approaching}| \leq D\}.$$

Once signal is lower, the gate is either down or moving_down within G time units.

$$\textbf{invariant}\{\ \texttt{signal} = \texttt{lower} \supset \texttt{gate} = \texttt{down} \lor \texttt{gate} = \texttt{moving\_down} \lor |\texttt{signal} \neq \texttt{lower}| \leq G\}.$$

Timing Constraints

The gate doesn't continue moving_down for more than L units.

$$\textbf{invariant}\{\ \texttt{gate} = \texttt{moving\_down} \supset |\texttt{gate} \neq \texttt{moving\_down}| \leq L\}.$$

No train goes from safe to crossing in fewer than $D+G+L$ units.

$$\textbf{invariant}\{\forall \texttt{Tr}:\ \texttt{train}(\texttt{Tr}) = \texttt{crossing} \supset |\texttt{train}(\texttt{Tr}) = \texttt{approaching}| + D + G + L < |\texttt{train}(\texttt{Tr}) = \texttt{safe}|\}.$$

RR Proof

Lemma:

$$\textbf{invariant}\{\forall \texttt{Tr}:\ \texttt{train}(\texttt{Tr}) = \texttt{approaching} \supset |\texttt{train}(\texttt{Tr}) = \texttt{safe}| \leq |\neg\texttt{Approaching}|\}.$$

Lemma:

$$\textbf{invariant}\{\forall \texttt{Tr}:\ \texttt{train}(\texttt{Tr}) = \texttt{approaching} \land \texttt{signal} \neq \texttt{lower} \supset |\texttt{train}(\texttt{Tr}) = \texttt{safe}| \leq D\}.$$

Lemma:

$$\textbf{invariant}\{\forall \texttt{Tr}:\ \texttt{train}(\texttt{Tr}) = \texttt{approaching} \land \texttt{signal} = \texttt{lower} \supset \texttt{gate} = \texttt{down} \lor \texttt{gate} = \texttt{moving\_down} \lor |\texttt{train}(\texttt{Tr}) = \texttt{safe}| \leq D + G\}.$$

Lemma:

$$\textbf{invariant}\{\ \texttt{signal} = \texttt{lower} \land \texttt{gate} \neq \texttt{down} \supset |\texttt{signal} \neq \texttt{lower}| \leq |\texttt{gate} = \texttt{down}|\}.$$

RR Proof (contd.)

Lemma:

$$\textbf{invariant}\{\forall \texttt{Tr}: \quad \texttt{train}(\texttt{Tr}) = \texttt{approaching} \supset |\texttt{train}(\texttt{Tr}) = \texttt{safe}| = |\texttt{train}(\texttt{Tr}) \neq \texttt{approaching}|\}$$

Lemma:

$$\textbf{invariant}\{\forall \texttt{Tr}: \quad \texttt{train}(\texttt{Tr}) = \texttt{approaching} \wedge \texttt{signal} = \texttt{lower} \wedge \texttt{gate} = \texttt{moving\_down} \supset |\texttt{train}(\texttt{Tr}) = \texttt{safe}| \leq D + G + L\}$$

Theorem:

$$\textbf{invariant}\{\ \texttt{Crossing} \supset \texttt{gate} = \texttt{down} \wedge \texttt{signal} = \texttt{lower}\}$$

Review

- Formalization real-time system behavior using traces and the Since operator.
- The computational model is simple: each computation is a sequence of states with constraints on the time value.
- No interleaving of time and state changes.
- The model can be used without modification to reason about systems without real-time constraints.
- The above formalization was embedded in the higher-order logic of PVS. Higher-order logic is useful for capturing temporal modalities as operators on traces.

Review (contd.)

- Interactive mechanized verification of two examples: a mutex protocol (effort = 1 day) and a railroad crossing controller (effort = 1 week).
- RR crossing model abstracts from tracks, distance, directions, etc., but multiple (infinitely many) trains are considered.
- Safety proof is quite complicated: it is not a simple matter of adding up delays.
- Utility proof is slightly less complicated. It shows that if no trains have been approaching or crossing for some duration, then the gate is up.
- PVS decision procedures for integer and real arithmetic are heavily used in the above verifications.

Other Related Approaches

- Schneider/Bloom/Marzullo: Proof outline logic with control predicates. $\uparrow cp$ records when a control predicate last became true.
- Carruth/Misra: Similar extension to the Unity logic with a *punch* operator.
- Abadi/Lamport: Use TLA with clock process and counter variables.
- Maler/Manna/Pnueli: Temporal logic with $\delta(\phi)$ to record the duration that temporal formula ϕ has been continuously true.
- Skakkebæk/Shankar: Embedding of duration calculus [Hoare&Ravn&Zhou] in PVS.
- Hybrid systems: Hytech, Kronos, timed COSPAN.

Future Work

- Further mechanization of the proofs.
- Extensions based on *integrators* as in the duration calculus.
- Use of other real-time operators: punch, duration, etc.
- Integration with hybrid model-checking systems.
- Front-end (notation and user-interface) for real-time reasoning.