Model-checking techniques for reactive systems

Amos Robinson, FP-Syd 2022-08-17

Steering selection system

```
def SteerSelector(
  human:   Stream[Int32],
  machine: Stream[Int32]
): Stream[Int32] =
  val human_filtered = MeanN(HISTORY, human)
  if (human_filtered >= OVERRIDE)
    human
  else
    machine
```

Steering selection system… Mean/Sum

```
def SteerSelector(
  human:   Stream[Int32],
  machine: Stream[Int32]
): Stream[Int32] =
  val human_filtered = MeanN(HISTORY, human)
  if (human_filtered >= OVERRIDE)
    human
  else
    machine


def MeanN(n: Int, v: Stream[Int32]): Stream[Int32] =
  SumN(n, v) / n

def SumN(n: Int, v: Stream[Int32]): Stream[Int32] = n match
  case 0 => 0
  case 1 => v
  case _ => v + SumN(n - 1, fby(0, v))
```

A property we want to prove

```
def SteerSelector(
  human:   Stream[Int32],
  machine: Stream[Int32]
): Stream[Int32] =
  val human_filtered = MeanN(HISTORY, human)
  if (human_filtered >= OVERRIDE)
    human
  else
    machine


property("if no human override then machine control") {
  LastN(HISTORY, human < OVERRIDE) ==> (SteerSelector(human, machine) == machine)
}
```

A property we want to prove… LastN

```
def SteerSelector(
  human:   Stream[Int32],
  machine: Stream[Int32]
): Stream[Int32] =
  val human_filtered = MeanN(HISTORY, human)
  if (human_filtered >= OVERRIDE)
    human
  else
    machine


property("if no human override then machine control") {
  LastN(HISTORY, human < OVERRIDE) ==> (SteerSelector(human, machine) == machine)
}

def LastN(n: Int, e: Stream[Bool]): Stream[Bool] = n match
  case 0 => True
  case 1 => e
  case _ => e && LastN(n - 1, fby(False, e))
```

A property we want to prove… if condition

```
def SteerSelector(
  human:   Stream[Int32],
  machine: Stream[Int32]
): Stream[Int32] =
  val human_filtered = MeanN(HISTORY, human)
  if (human_filtered >= OVERRIDE)
    human
  else // otherwise MeanN(HISTORY, human) < OVERRIDE
    machine


property("if no human override then machine control") {
  LastN(HISTORY, human < OVERRIDE) ==> MeanN(HISTORY, human) < OVERRIDE
}

def LastN(n: Int, e: Stream[Bool]): Stream[Bool] = n match
  case 0 => True
  case 1 => e
  case _ => e && LastN(n - 1, fby(False, e))
```

Simplify

```
val HISTORY  = 2

def LastN(n: Int, e: Stream[Bool]): Stream[Bool] = n match
  case 0 => True
  case 1 => e
  case _ => e && LastN(n - 1, fby(False, e))

def MeanN(n: Int, v: Stream[Int32]): Stream[Int32] =
  SumN(n, v) / n

def SumN(n: Int, v: Stream[Int32]): Stream[Int32] = n match
  case 0 => 0
  case 1 => v
  case _ => v + SumN(n - 1, fby(0, v))


property("if no human override then machine control") {
  LastN(HISTORY, human < OVERRIDE) ==> MeanN(HISTORY, human) < OVERRIDE
}
```

Simplify… substitute history

```
val HISTORY  = 2

def LastN(n: Int, e: Stream[Bool]): Stream[Bool] = n match
  case 0 => True
  case 1 => e
  case _ => e && LastN(n - 1, fby(False, e))

def MeanN(n: Int, v: Stream[Int32]): Stream[Int32] =
  SumN(n, v) / n

def SumN(n: Int, v: Stream[Int32]): Stream[Int32] = n match
  case 0 => 0
  case 1 => v
  case _ => v + SumN(n - 1, fby(0, v))


property("if no human override then machine control") {
  LastN(       2, human < OVERRIDE) ==> MeanN(       2, human) < OVERRIDE
}
```

Simplify… unfold MeanN

```
val HISTORY  = 2

def LastN(n: Int, e: Stream[Bool]): Stream[Bool] = n match
  case 0 => True
  case 1 => e
  case _ => e && LastN(n - 1, fby(False, e))

def MeanN(n: Int, v: Stream[Int32]): Stream[Int32] =
  SumN(n, v) / n

def SumN(n: Int, v: Stream[Int32]): Stream[Int32] = n match
  case 0 => 0
  case 1 => v
  case _ => v + SumN(n - 1, fby(0, v))


property("if no human override then machine control") {
  LastN(      2, human < OVERRIDE) ==> SumN(      2, human) / 2 < OVERRIDE
}
```

Simplify… etc

```
val HISTORY  = 2

def LastN(n: Int, e: Stream[Bool]): Stream[Bool] = n match
  case 0 => True
  case 1 => e
  case _ => e && LastN(n - 1, fby(False, e))

def MeanN(n: Int, v: Stream[Int32]): Stream[Int32] =
  SumN(n, v) / n

def SumN(n: Int, v: Stream[Int32]): Stream[Int32] = n match
  case 0 => 0
  case 1 => v
  case _ => v + SumN(n - 1, fby(0, v))


property("if no human override then machine control") {
  human < OVERRIDE && fby(False, human < OVERRIDE) ==>
    ((human + fby(0, human)) / 2 < OVERRIDE)
}
```

Simplify… normal form

```
property("if no human override then machine control") {
   val human_in_bounds = human < OVERRIDE
   val last_fby        = fby(False, human_in_bounds)
   val last_in_bounds  = human_in_bounds && last_fby

   val mean_fby        = fby(0, human)
   val mean_sum        = human + mean_fby
   val mean            = mean_sum / 2
   val mean_in_bounds  = mean < OVERRIDE

   last_in_bounds ==> mean_in_bounds
 }
```

Transition systems

```
      val human_in_bounds = human < OVERRIDE
      val last_fby        = fby(False, human_in_bounds)
      val last_in_bounds  = human_in_bounds && last_fby
      val mean_fby        = fby(0, human)
      val mean_sum        = human + mean_fby
      val mean            = mean_sum / 2
      val mean_in_bounds  = mean < OVERRIDE
      last_in_bounds ==> mean_in_bounds
================================================================

  (declare-datatype Row
    ((Row
      (human           Int)
      (human_in_bounds Bool)
      (last_in_bounds  Bool)
      (mean_sum        Int)
      (mean            Int)
      (mean_in_bounds  Bool))))

  (declare-datatype State
    ((State
      (last_fby        Bool)
      (mean_fby        Int))))
```

Transition systems

```
        val human_in_bounds = human < OVERRIDE
        val last_fby        = fby(False, human_in_bounds)
        val last_in_bounds  = human_in_bounds && last_fby
        val mean_fby        = fby(0, human)
        val mean_sum        = human + mean_fby
        val mean            = mean_sum / 2
        val mean_in_bounds  = mean < OVERRIDE
        last_in_bounds ==> mean_in_bounds
=============================================================

  (declare-datatype Row
    ((Row
      (human           Int)
      (human_in_bounds Bool)
      (last_in_bounds  Bool)
      (mean_sum        Int)
      (mean            Int)
      (mean_in_bounds  Bool))))

  (declare-datatype State
    ((State
      (last_fby        Bool)
      (mean_fby        Int))))
```

Transition systems… extract pure bindings

```
      val human_in_bounds = human < OVERRIDE
      val last_fby        = fby(False, human_in_bounds)
      val last_in_bounds  = human_in_bounds && last_fby
      val mean_fby        = fby(0, human)
      val mean_sum        = human + mean_fby
      val mean            = mean_sum / 2
      val mean_in_bounds  = mean < OVERRIDE
      last_in_bounds ==> mean_in_bounds
=============================================================
```

```
(declare-datatype Row   ...)
(declare-datatype State ...)

(define-fun extract ((state State) (row Row)) Bool
  (and
    (= (human_in_bounds row) (< (human row) OVERRIDE))
    (= (last_in_bounds  row) (and (human_in_bounds row) (last_fby state)))
    (= (mean_sum        row) (+ (human row) (mean_fby state)))
    (= (mean            row) (/ (mean_sum row) 2))
    (= (mean_in_bounds  row) (< (mean row) OVERRIDE))))
```

Transition systems… extract pure bindings

```
        val human_in_bounds = human < OVERRIDE
        val last_fby        = fby(False, human_in_bounds)
        val last_in_bounds  = human_in_bounds && last_fby
        val mean_fby        = fby(0, human)
        val mean_sum        = human + mean_fby
        val mean            = mean_sum / 2
        val mean_in_bounds  = mean < OVERRIDE
        last_in_bounds ==> mean_in_bounds
=============================================================
```

```
  (declare-datatype Row   ...)
  (declare-datatype State ...)

  (define-fun extract ((state State) (row Row)) Bool
    (and
      (= (human_in_bounds row) (< (human row) OVERRIDE))
      (= (last_in_bounds  row) (and (human_in_bounds row) (last_fby state)))
      (= (mean_sum        row) (+ (human row) (mean_fby state)))
      (= (mean            row) (/ (mean_sum row) 2))
      (= (mean_in_bounds  row) (< (mean row) OVERRIDE))))
```

Transition systems… init

```
      val human_in_bounds = human < OVERRIDE
      val last_fby        = fby(False, human_in_bounds)
      val last_in_bounds  = human_in_bounds && last_fby
      val mean_fby        = fby(0, human)
      val mean_sum        = human + mean_fby
      val mean            = mean_sum / 2
      val mean_in_bounds  = mean < OVERRIDE
      last_in_bounds ==> mean_in_bounds
=============================================================

  (declare-datatype Row   ...)
  (declare-datatype State ...)
  (define-fun extract ((state State) (row Row)) Bool)

  (define-fun init ((state State)) Bool
    (and
      (= (last_fby state) false)
      (= (mean_fby state) 0)))
```

Transition systems… init

```
      val human_in_bounds = human < OVERRIDE
      val last_fby        = fby(False, human_in_bounds)
      val last_in_bounds  = human_in_bounds && last_fby
      val mean_fby        = fby(0, human)
      val mean_sum        = human + mean_fby
      val mean            = mean_sum / 2
      val mean_in_bounds  = mean < OVERRIDE
      last_in_bounds ==> mean_in_bounds
================================================================
```

```
(declare-datatype Row   ...)
(declare-datatype State ...)
(define-fun extract ((state State) (row Row)) Bool)

(define-fun init ((state State)) Bool
  (and
    (= (last_fby state) false)
    (= (mean_fby state) 0)))
```

Transition systems… step

```
      val human_in_bounds = human < OVERRIDE
      val last_fby        = fby(False, human_in_bounds)
      val last_in_bounds  = human_in_bounds && last_fby
      val mean_fby        = fby(0, human)
      val mean_sum        = human + mean_fby
      val mean            = mean_sum / 2
      val mean_in_bounds  = mean < OVERRIDE
      last_in_bounds ==> mean_in_bounds
=============================================================
```

```
(declare-datatype Row   ...)
(declare-datatype State ...)
(define-fun extract ((state State) (row Row)) Bool)
(define-fun init ((state State)) Bool)

(define-fun step ((state State) (row Row) (stateX State)) Bool
  (and
    (= (last_fby stateX) (human_in_bounds row))
    (= (mean_fby stateX) (human row))))
```

Transition systems… step

```
      val human_in_bounds = human < OVERRIDE
      val last_fby        = fby(False, human_in_bounds)
      val last_in_bounds  = human_in_bounds && last_fby
      val mean_fby        = fby(0, human)
      val mean_sum        = human + mean_fby
      val mean            = mean_sum / 2
      val mean_in_bounds  = mean < OVERRIDE
      last_in_bounds ==> mean_in_bounds
================================================================
```

```
(declare-datatype Row   ...)
(declare-datatype State ...)
(define-fun extract ((state State) (row Row)) Bool)
(define-fun init ((state State)) Bool)

(define-fun step ((state State) (row Row) (stateX State)) Bool
  (and
    (= (last_fby stateX) (human_in_bounds row))
    (= (mean_fby stateX) (human row))))
```

Transition systems… prop

```
        val human_in_bounds = human < OVERRIDE
        val last_fby        = fby(False, human_in_bounds)
        val last_in_bounds  = human_in_bounds && last_fby
        val mean_fby        = fby(0, human)
        val mean_sum        = human + mean_fby
        val mean            = mean_sum / 2
        val mean_in_bounds  = mean < OVERRIDE
        last_in_bounds ==> mean_in_bounds
=============================================================
```

```
  (declare-datatype Row   ...)
  (declare-datatype State ...)
  (define-fun extract ((state State) (row Row)) Bool)
  (define-fun init ((state State)) Bool)
  (define-fun step ((state State) (row Row) (stateX State)) Bool)

  (define-fun prop ((row Row)) Bool
    (=> (last_in_bounds row) (mean_in_bounds row)))
```

Transition systems… prop

```
        val human_in_bounds = human < OVERRIDE
        val last_fby        = fby(False, human_in_bounds)
        val last_in_bounds  = human_in_bounds && last_fby
        val mean_fby        = fby(0, human)
        val mean_sum        = human + mean_fby
        val mean            = mean_sum / 2
        val mean_in_bounds  = mean < OVERRIDE
        last_in_bounds ==> mean_in_bounds
=============================================================

  (declare-datatype Row   ...)
  (declare-datatype State ...)
  (define-fun extract ((state State) (row Row)) Bool)
  (define-fun init ((state State)) Bool)
  (define-fun step ((state State) (row Row) (stateX State)) Bool)

  (define-fun prop ((row Row)) Bool
    (=> (last_in_bounds row) (mean_in_bounds row)))
```

Induction

```
(declare-datatype Row   ...)
(declare-datatype State ...)
(define-fun extract ((state State) (row Row)) Bool)
(define-fun init ((state State)) Bool)
(define-fun step ((state State) (row Row) (stateX State)) Bool)
(define-fun prop ((row Row)) Bool)

(declare-const state0 State)
(declare-const row0   Row)

; is there an initial state that violates the property?
(assert (init    state0))
(assert (extract state0 row0))
(assert (not (prop row0)))
(check-sat)
```

Induction… base case ok

```
(declare-datatype Row   ...)
(declare-datatype State ...)
(define-fun extract ((state State) (row Row)) Bool)
(define-fun init ((state State)) Bool)
(define-fun step ((state State) (row Row) (stateX State)) Bool)
(define-fun prop ((row Row)) Bool)

(declare-const state0 State)
(declare-const row0   Row)

; is there an initial state that violates the property?
(assert (init    state0))
(assert (extract state0 row0))
(assert (not (prop row0)))
(check-sat)
; ...z3 says: unsat
```

Induction… step case

```
(declare-datatype Row   ...)
(declare-datatype State ...)
(define-fun extract ((state State) (row Row)) Bool)
(define-fun init ((state State)) Bool)
(define-fun step ((state State) (row Row) (stateX State)) Bool)
(define-fun prop ((row Row)) Bool)

(declare-const state0 State)
(declare-const row0   Row)
(declare-const state1 State)
(declare-const row1   Row)

; if we start in a good state, can we reach a bad one?
(assert (extract state0 row0))
(assert (step    state0 row0 state1))
(assert (extract state1 row1))
(assert (prop row0))
(assert (not (prop row1)))
(check-sat)
; z3 says: unsat
```

What if HISTORY = 3?

```
property("if no human override then machine control") {
  LastN(3, human < OVERRIDE) ==> MeanN(3, human) < OVERRIDE
}
```

HISTORY = 3… normal form

```
        val human_in_bounds = human < OVERRIDE
        val last_fby1       = fby(False, human_in_bounds)
        val last_fby2       = fby(False, last_fby1)
        val last_in_bounds  = human_in_bounds && last_fby1 && last_fby2
        val mean_fby1       = fby(i32(0), human)
        val mean_fby2       = fby(i32(0), mean_fby1)
        val mean_sum        = human + mean_fby1 + mean_fby2
        val mean            = mean_sum / 3
        val mean_in_bounds  = mean < OVERRIDE
        last_in_bounds ==> mean_in_bounds
=======================================================
```

HISTORY = 3… inductive base case ok

```
        val human_in_bounds = human < OVERRIDE
        val last_fby1       = fby(False, human_in_bounds)
        val last_fby2       = fby(False, last_fby1)
        val last_in_bounds  = human_in_bounds && last_fby1 && last_fby2
        val mean_fby1       = fby(i32(0), human)
        val mean_fby2       = fby(i32(0), mean_fby1)
        val mean_sum        = human + mean_fby1 + mean_fby2
        val mean            = mean_sum / 3
        val mean_in_bounds  = mean < OVERRIDE
        last_in_bounds ==> mean_in_bounds
=============================================================

  (assert (init    state0))
  (assert (extract state0 row0))
  (assert (not (prop row0)))
  (check-sat)
  ; ...z3 says: unsat
```

HISTORY = 3… inductive step case

```
        val human_in_bounds = human < OVERRIDE
        val last_fby1       = fby(False, human_in_bounds)
        val last_fby2       = fby(False, last_fby1)
        val last_in_bounds  = human_in_bounds && last_fby1 && last_fby2
        val mean_fby1       = fby(i32(0), human)
        val mean_fby2       = fby(i32(0), mean_fby1)
        val mean_sum        = human + mean_fby1 + mean_fby2
        val mean            = mean_sum / 3
        val mean_in_bounds  = mean < OVERRIDE
        last_in_bounds ==> mean_in_bounds
=======================================================

  (assert (extract state0 row0))
  (assert (step    state0 row0 state1))
  (assert (extract state1 row1))
  (assert (prop row0))
  (assert (not (prop row1)))
  (check-sat)
```

HISTORY = 3… inductive step case not ok!

```
        val human_in_bounds = human < OVERRIDE
        val last_fby1       = fby(False, human_in_bounds)
        val last_fby2       = fby(False, last_fby1)
        val last_in_bounds  = human_in_bounds && last_fby1 && last_fby2
        val mean_fby1       = fby(i32(0), human)
        val mean_fby2       = fby(i32(0), mean_fby1)
        val mean_sum        = human + mean_fby1 + mean_fby2
        val mean            = mean_sum / 3
        val mean_in_bounds  = mean < OVERRIDE
        last_in_bounds ==> mean_in_bounds
=============================================================

  (assert (extract state0 row0))
  (assert (step    state0 row0 state1))
  (assert (extract state1 row1))
  (assert (prop row0))
  (assert (not (prop row1)))
  (check-sat)
  ; ...z3 says: sat
```

HISTORY = 3… counterexample-to-induction

```
        val human_in_bounds = human < OVERRIDE
        val last_fby1       = fby(False, human_in_bounds)
        val last_fby2       = fby(False, last_fby1)
        val last_in_bounds  = human_in_bounds && last_fby1 && last_fby2
        val mean_fby1       = fby(i32(0), human)
        val mean_fby2       = fby(i32(0), mean_fby1)
        val mean_sum        = human + mean_fby1 + mean_fby2
        val mean            = mean_sum / 3
        val mean_in_bounds  = mean < OVERRIDE
        last_in_bounds ==> mean_in_bounds
=======================================================

  (assert (extract state0 row0))
  (assert (step    state0 row0 state1))
  (assert (extract state1 row1))
  (assert (prop row0))
  (assert (not (prop row1)))
  (check-sat)
  (get-value OVERRIDE state0)
  ; OVERRIDE            = 10
  ; (last_fby1 state0) = true
  ; (mean_fby1 state0) = 21
  ; (last_fby2 state0) = true
  ; (mean_fby2 state0) = (- 21)
```

HISTORY = 3… counterexample-to-induction

```
        val human_in_bounds = human < OVERRIDE
        val last_fby1       = fby(False, human_in_bounds)
        val last_fby2       = fby(False, last_fby1)
        val last_in_bounds  = human_in_bounds && last_fby1 && last_fby2
        val mean_fby1       = fby(i32(0), human)
        val mean_fby2       = fby(i32(0), mean_fby1)
        val mean_sum        = human + mean_fby1 + mean_fby2
        val mean            = mean_sum / 3
        val mean_in_bounds  = mean < OVERRIDE
        last_in_bounds ==> mean_in_bounds
=============================================================

  (assert (extract state0 row0))
  (assert (step    state0 row0 state1))
  (assert (extract state1 row1))
  (assert (prop row0))
  (assert (not (prop row1)))
  (check-sat)
  (get-value OVERRIDE state0)
  ; OVERRIDE            = 10
  ; (last_fby1 state0) = true
  ; (mean_fby1 state0) = 21
  ; (last_fby2 state0) = true
  ; (mean_fby2 state0) = (- 21)
```

K-induction: k=2 step case

```
        val human_in_bounds = human < OVERRIDE
        val last_fby1       = fby(False, human_in_bounds)
        val last_fby2       = fby(False, last_fby1)
        val last_in_bounds  = human_in_bounds && last_fby1 && last_fby2
        val mean_fby1       = fby(i32(0), human)
        val mean_fby2       = fby(i32(0), mean_fby1)
        val mean_sum        = human + mean_fby1 + mean_fby2
        val mean            = mean_sum / 3
        val mean_in_bounds  = mean < OVERRIDE
        last_in_bounds ==> mean_in_bounds
=======================================================
```

```
(assert (extract state0 row0))
(assert (step    state0 row0 state1))
(assert (extract state1 row1))
(assert (step    state1 row1 state2))
(assert (extract state2 row2))
(assert (prop row0))
(assert (prop row1))
(assert (not (prop row2)))
(check-sat)
```

K-induction: k=2 base case

```
        val human_in_bounds = human < OVERRIDE
        val last_fby1       = fby(False, human_in_bounds)
        val last_fby2       = fby(False, last_fby1)
        val last_in_bounds  = human_in_bounds && last_fby1 && last_fby2
        val mean_fby1       = fby(i32(0), human)
        val mean_fby2       = fby(i32(0), mean_fby1)
        val mean_sum        = human + mean_fby1 + mean_fby2
        val mean            = mean_sum / 3
        val mean_in_bounds  = mean < OVERRIDE
        last_in_bounds ==> mean_in_bounds
=======================================================

  (assert (init state0))
  (assert (extract state0 row0))
  (assert (step    state0 row0 state1))
  (assert (extract state1 row1))
  ; can we reach a bad state in 0 or 1 steps?
  (assert (or (not (prop row0)) (not (prop row1))))
  (check-sat)
```

Nice birds I have met