Validating the Hybrid ERTMS/ETCS Level 3 Concept with Electrum

Alcino Cunha and **Nuno Macedo**
Universidade do Minho & INESC TEC, Portugal

ABZ18, Southampton, UK

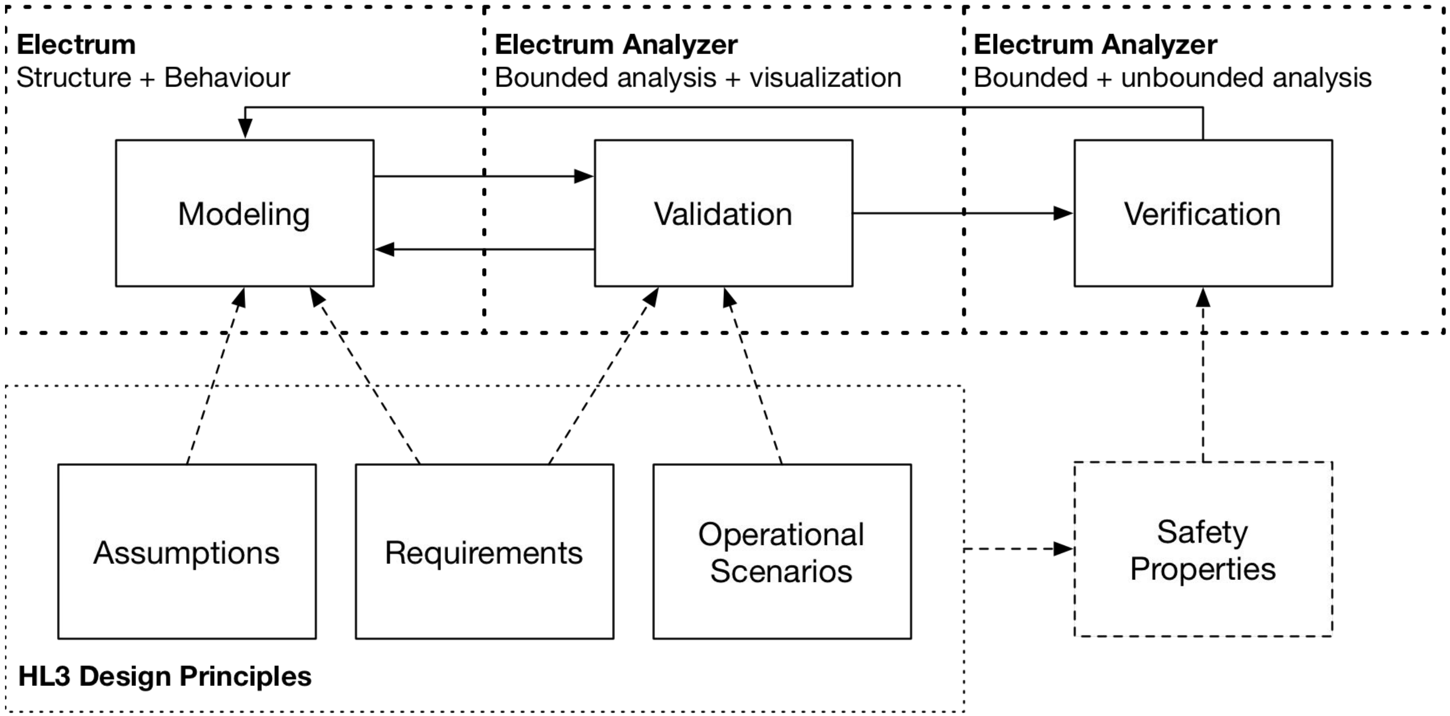
Electrum

- Alloy language extended with dynamic features inspired by TLA+
  - structural elements (sigs / fields) can be *static* or *mutable*
  - first-order relational logic extended with (future and past) *LTL* operators
  - *primed* relational expressions refer to the succeeding state
- Alloy Analyzer extended to temporal context
  - bounded model checking (through SAT)
  - unbounded model checking (through SMV)
  - visualisation of trace instances

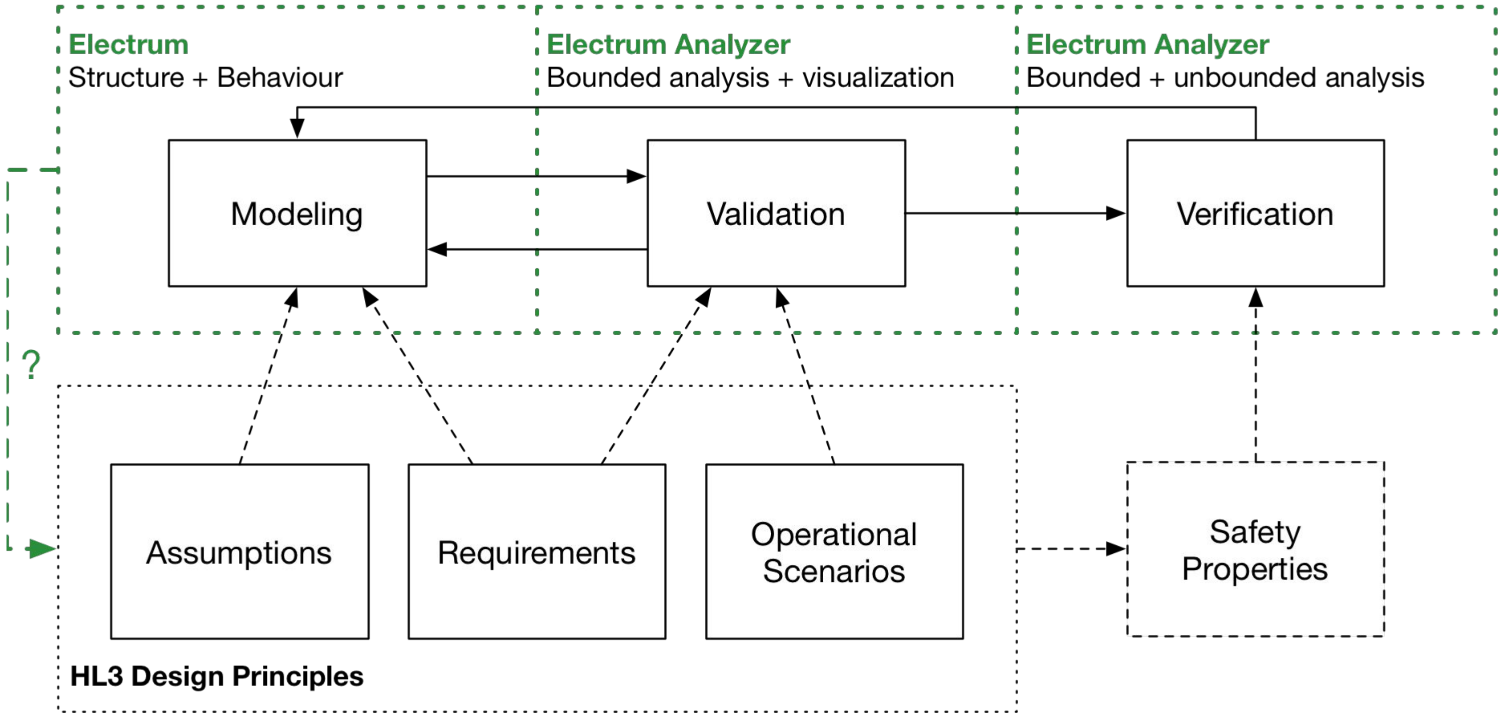
HL3 in Electrum

- Pros
    - rich structure (track configurations) and behavior (train and VSS state)
    - underspecified behavior (trains and MAs), declarative actions
    - backed by (9) scenarios
    - small universe
- Cons
    - some continuous aspects
        - communication delays, timers
        - train length/speed

HL3 in Electrum

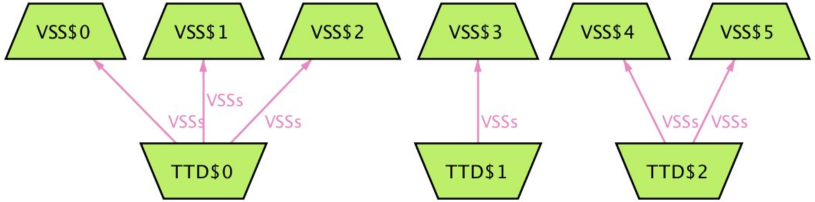


HL3 in Electrum



Modeling: Static Structure

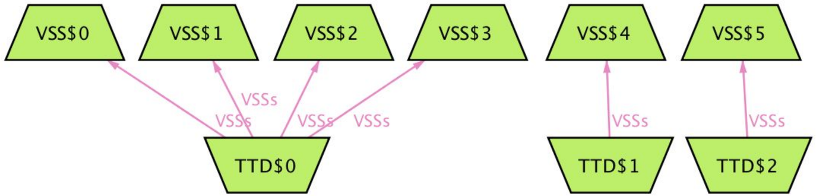
- Every possible track configuration (within scope, symmetry breaking)
- Static sigs + FOL facts (plain Alloy)
    - System *configuration*
- Abstract block/train lengths
- Minor tweak to visualizer




```
open util/ordering[VSS] as V
open util/ordering[TTD] as D

sig VSS { … }
sig TTD {
  start, end : one VSS
} { end.gte[start] }

fact trackSections {
  all ttd:TTD-D/last |
    ttd.end.V/next = (ttd.D/next).start
  D/first.start = V/first
  D/last.end = V/last }

fun VSSs[t:TTD] : set VSS {
      t.start.*V/next & t.end.*(~V/next) }

fun parent[v:VSS] : one TTD {
      max[(v.*V/prev).~start] }

run {} for 3 TTD, 6 VSS, 2 Train
```

Modeling: Static Structure

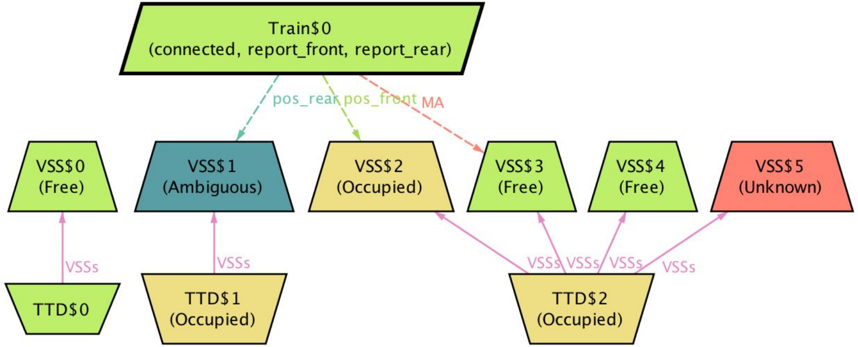
- Every possible track configuration (within scope, symmetry breaking)
- Static sigs + FOL facts (plain Alloy)
  - System *configuration*
- Abstract block/train lengths
- Minor tweak to visualizer




```
open util/ordering[VSS] as V
open util/ordering[TTD] as D

sig VSS { … }
sig TTD {
  start, end : one VSS
} { end.gte[start] }

fact trackSections {
  all ttd:TTD-D/last |
    ttd.end.V/next = (ttd.D/next).start
  D/first.start = V/first
  D/last.end = V/last }

fun VSSs[t:TTD] : set VSS {
      t.start.*V/next & t.end.*(~V/next) }

fun parent[v:VSS] : one TTD {
      max[(v.*V/prev).~start] }

run {} for 3 TTD, 6 VSS, 2 Train
```

Modeling: Dynamic Structure

- Elements change in time
  - Physical train, on-board and trackside systems
- Variable sigs + LTL facts
- Abstraction on non-integral trains and TTD state delays

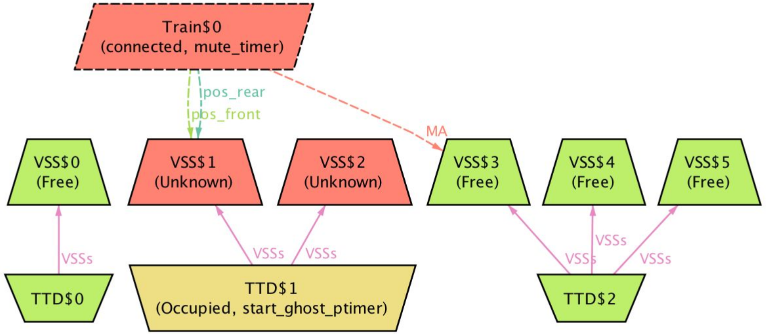



```
sig VSS { var state : one State,
          var jumping : lone Train }

sig Train { var pos_front, pos_rear : one VSS,
            var MA : one VSS }

var sig connected in Train {}
var sig report_front, report_rear in Train {}

fun occupied : set TTD {
  { ttd : TTD |
    some VSSs[ttd] & Train.(pos_rear+pos_front) } }

fun mute : set Train {
     Train-(report_rear+report_front) }

fact jumpingDef {
  always jumping = { v:VSS,t:Train | ... } }

run { Occupied + Ambiguous in VSS.state }
  for 3 but 6 VSS
```

Modeling: Timers

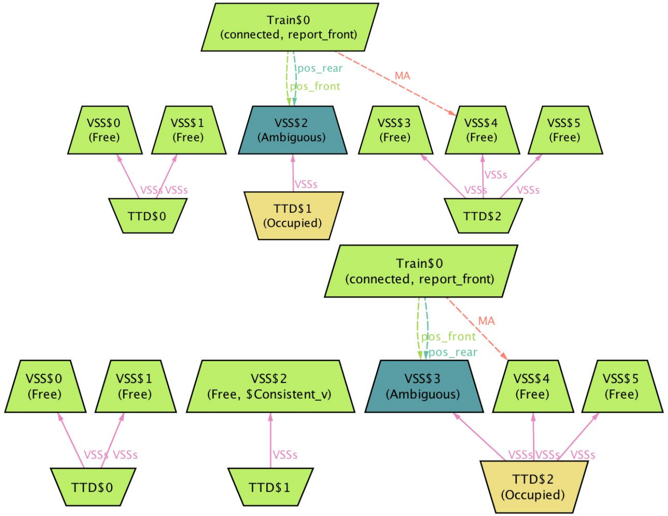
- Model only possibility of expiration (no duration imposed)
- Variable subset sigs
- Not real-time

```
var sig mute_timer in Train {}

pred set_mute_timer {
      mute_timer in mute }
...
var sig shadow_timer_A in TTD {}

pred set_shadow_timer_A {
  shadow_timer_A in start_shadow_timer_A }

fun start_shadow_timer_A : set TTD {
  { ttd : TTD | once {
    previous ttd in occupied
    ttd not in occupied
    previous ttd.end.state = Ambiguous } } }

run { some mute_timer } for 3 but 6 VSS
```



Modeling: Behavior

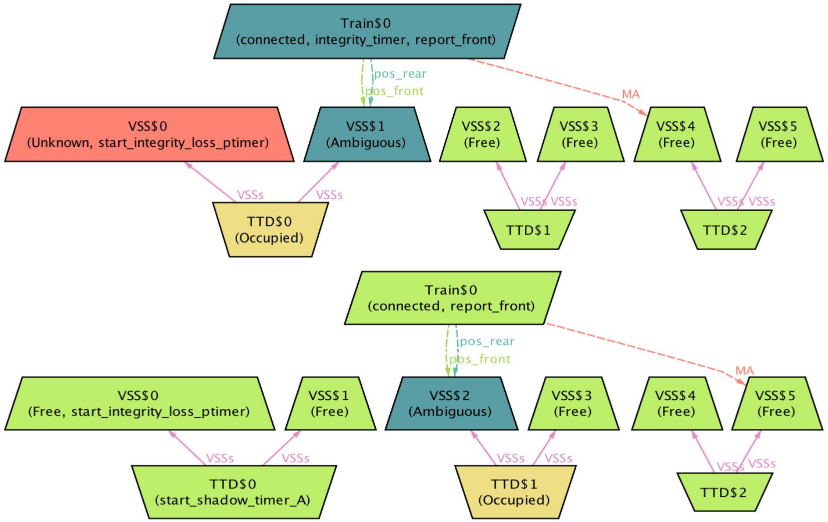
- VSS state machine, proper priorities
- Trains and MAs (non-deterministic)
- Simultaneous front/rear reports




```
pred move [t:Train] {
  t.pos_front' in t.pos_front.(iden+V/next)
  t.pos_rear' in t.pos_front'.(iden+V/prev)
  t.pos_rear' in t.pos_rear.(iden+V/next)
  { t in connected
    t in report_rear' => t in report_front' } or
  { t not in report_front'
    t not in report_rear' }
  t in connected iff t in connected' }
...

pred n09 [v:VSS] { v.state = Ambiguous
                   after (n09A[v] or n09B[v]) }
pred n09A [v:VSS] { parent[v] not in occupied }
pred n09B [v:VSS] { some tr:Train {
            tr not in disintegrated
            v not in knownLoc[tr]
            previous v in knownLoc[tr]
            parent[v] not in shadow_timer_A
            parent[v] in start_shadow_timer_A } }

run { some v:VSS | n09[v] } for 3 but 6 VSS
```

Modeling: Behavior

- VSS state machine, proper priorities
- Trains and MAs (non-deterministic)
- Simultaneous front/rear reports




```
pred move [t:Train] {
  t.pos_front' in t.pos_front.(iden+V/next)
  t.pos_rear' in t.pos_front'.(iden+V/prev)
  t.pos_rear' in t.pos_rear.(iden+V/next)
  { t in connected
    t in report_rear' => t in report_front' } or
  { t not in report_front'
    t not in report_rear' }
  t in connected iff t in connected' }
...

pred n09 [v:VSS] { v.state = Ambiguous
                   after (n09A[v] or n09B[v]) }
pred n09A [v:VSS] { parent[v] not in occupied }
pred n09B [v:VSS] { some tr:Train {
            tr not in disintegrated
            v not in knownLoc[tr]
            previous v in knownLoc[tr]
            parent[v] not in shadow_timer_A
            parent[v] in start_shadow_timer_A } }

run { some v:VSS | n09[v] } for 3 but 6 VSS
```

Modeling: Tying it All Together

- Force all valid traces to act as specified
- Multiple trains may act simultaneously, to avoid trace length explosion

- Commands allow arbitrary LTL formulas
- Bounded engine requires scope on states
- Infinite looping traces
- More flexible scope supported

```
fact trace {
  always {
    timers
    MAs
    all v:VSS | states[v]
    (all tr:Train | move[tr]) or
      (some tr,ts:Train |
        split[tr,ts] or som[tr] or eom[tr])
  } }


run { eventually (some v:VSS | n09[v]) }
  for 3 but 6 VSS, 6..8 Time
```

Validation: Operational Scenarios

- All 9 operational scenarios encoded
- Verbose in Electrum
  - 8 steps, fix train movement and reporting
  - VSS state evolves according to spec
- New derived temporal operator
- Potential HL3 issues identified

```
pred S5 {
  let v11 = V/first, v12 = v11.next, ... {
  some disj tr1,tr2:Train {

  v11.state = Free
  v12.state = Occupied
  v12.nexts.state = Free
  split[tr1,tr2]

  tr1.pos_rear = v12 and after (
  tr1.pos_rear = v12 and after (
  tr1.pos_rear = v21 and after (
  tr1.pos_rear = v22 ... )))

  tr1 in report_rear;
  tr1 not in report_rear;
  tr1 not in report_rear;
  tr1 in report_rear; ...

  ... } }

run S5 for exactly 8 Time, 2 Train, 3 TTD, 8 VSS
```

Validation: Operational Scenarios

Validation: Possible Issues

- Based solely in the “principles” document
- Inconsistencies between *VSS state machine* spec and scenarios
    a. pre-condition on transition #1A and scenario 7
    b. definition of “train located” on transition #5A and scenario 4
- Inconsistencies between *timer behavior* and scenarios
    a. Ghost propagation timer permanent expiration and scenario 9

Verification

- Specified safety properties
  - Without *communication failures*, VSS state always correct?
  - With *automatic timer expiration*, VSS state always correct?
- Required additional restrictions on train behavior and MAs
- Verified with the *unbounded* model checking engine (all trace lengths)

```
assert trains_Occupied {
  (init and always (no mute+disintegrated and no
t:Train | after OS[t])) =>
    always Train.pos_rear.state = Occupied
}

check trains_Occupied
  for 8 VSS, 3 TTD, 2 Train, ~~12 Time~~
...

assert timers_Free {
  (init and always (auto_timer and (all t:Train |
t.pos_front in MAs[t] and not (after OS[t])))) =>
      always Train.pos_front.state != Free
}

check timers_Free
  for 8 VSS, 3 TTD, 2 Train, ~~12 Time~~
```

Comparison with Alloy

- Alloy model developed in parallel
- Explicit (error-prone) state modeling

```
(init[first] and all s:first.*next | (no mute[s]+disintegrated[s] and no t:Train | OS[s.next,t]))
    => all s1:first.*next Train.(pos_rear.s1).(state.s1) = Occupied
```

- No unbounded verification
- But sometimes explicit states are convenient:

`t.rep_rear = Time-(first.next)` *vs.* `t in rep_rear;t not in rep_rear;...;always t in rep_rear`

`t.pos_front.(max[s.*prev&t.rep_front])` *vs.* `{ v:VSS | t not in rep_front since (t in rep_front and v = t.pos_front) }`

Conclusions

- Promoted Electrum improvements
  - language, visualization, trace scope control
- Possible straightforward enhancements (trading scalability)
  - delays on TTD state
  - force separate front/rear reports
  - distinguish integrity-lost events
- Not so straightforward
  - real-time timers
  - speed/length issues (min-safe-rear-end)
- Potential issues: hopefully relevant to the ERTMS/ETCS community

Validating HL3 in Electrum

- Electrum Analyzer

  http://haslab.github.io/Electrum

- Models (Electrum + Alloy) + Scenarios

  https://github.com/haslab/Electrum/wiki/ERTMS