Validating Multiple Variants of an Automotive Light System with Electrum

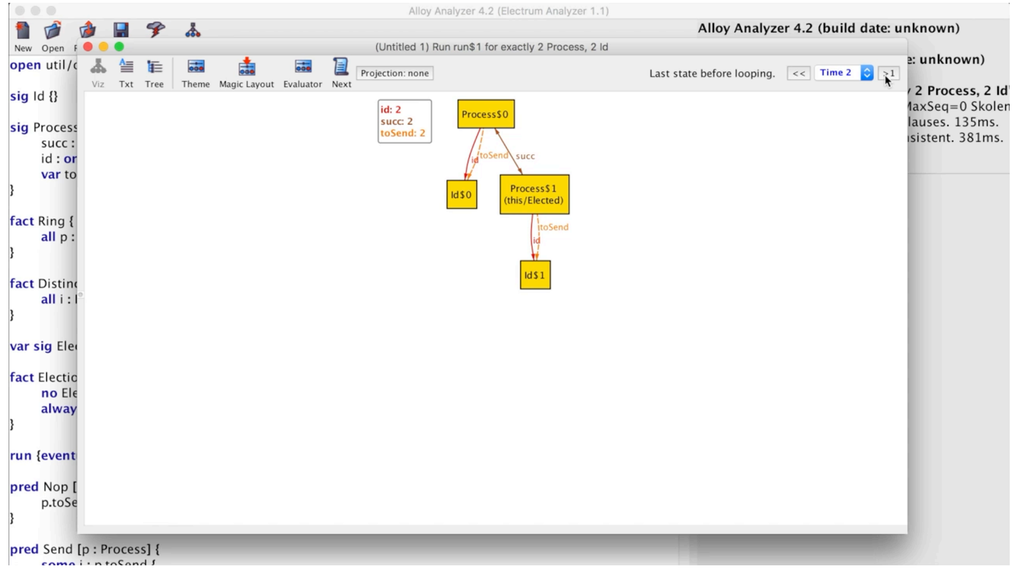
Nuno Macedo, Alcino Cunha, Chong Liu
INESC TEC & University of Porto / University of Minho

ABZ'21, June 2021

ELS in Electrum

A model checker for relational linear temporal logic

- Formal specification language with structural and dynamic constructs
- Declarative specifications, behaviour can be under-specified
- Automatic verification through solving, returns counter-example traces
- Trace visualiser and scenario exploration operations

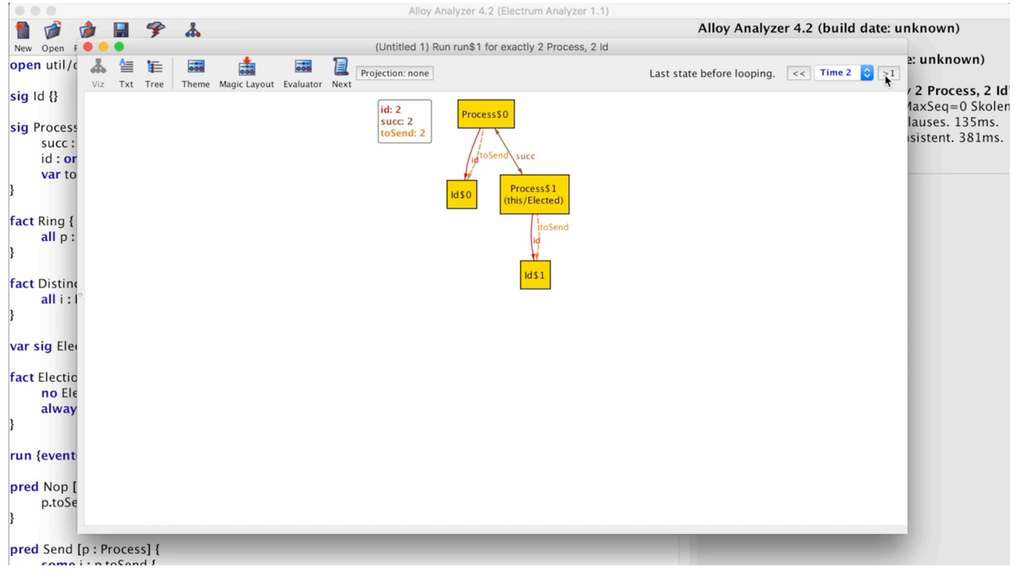


haslab.github.io/Electrum/

ELS in Electrum

A model checker for relational linear temporal logic

- Formal specification language with structural and dynamic constructs
- Declarative specifications, behaviour can be under-specified
- Automatic verification through solving, returns counter-example traces
- Trace visualiser and scenario exploration operations



haslab.github.io/Electrum/

ELS in Electrum

Main features

- Pros
  - Structural and behaviour modelling at high-level of abstraction
  - Animation (and exploration) of the reference scenarios
  - Flexible language, support for variants
- Cons
  - No numeric analysis

ELS in Electrum

Strategy overview

Adaptive Exterior Light and Speed Control System

Frank Houdek[1] and Alexander Raschke[2]

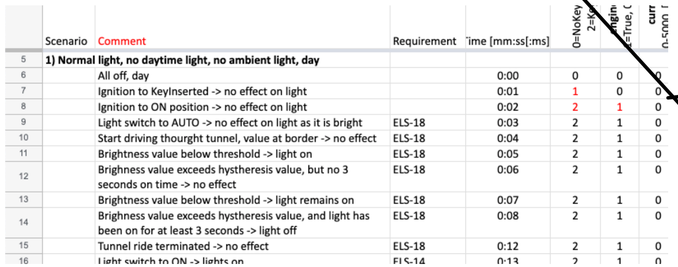
[1] Daimler AG, Research and Development, Stuttgart, Germany
frank.houdek@daimler.com
[2] Inst. of Software Engineering, Ulm University, Germany
alexander.raschke@uni-ulm.de

1 Introduction

This case study continues the successful series of case studies for formal specification and verification of the ABZ conference series, which started with the landing gear system[1] and expanded with the hemodialysis medical device[4] and the European Train Control System (ETCS)[2] in the following years. This document describes two systems from the automotive domain: an adaptive exterior light system (ELS) and a speed control system (SCS). This specification is based on the SPES XT running example[3]. Besides their general architectures, the requirements of the software based controllers are described. Both systems are only loosely coupled, which makes it possible to handle them independently.

Conventions. Throughout this document, we use the following conventions to better distinguish different terms: **Main functions** are set in bold, *sub-functions* are italicized. Predefined **signals** are written in typewriter and for the values of signals we use a font without serifs.

The structure of the document is as follows: First, the general hardware architecture of a modern car is sketched in Sect. 3. Then, the adaptive exterior light system is described in Sect. 4, followed by the requirements of the speed control system (Sect. 5). For each of the systems, the user interface, the needed sensors and the available actuators are described before the different features are explained in detail. In Sect. A, all available signals and their value ranges are summarized in a table.



	Scenario	Comment	Requirement	Time [mm:ss[:ms]	0=NoKey 2=S	Ignin 1=True, (	curr 0=Gon
5	**1) Normal light, no daytime light, no ambient light, day**						
6		All off, day		0:00	0	0	0
7		Ignition to KeyInserted -> no effect on light		0:01	1	0	0
8		Ignition to ON position -> no effect on light		0:02	2	1	0
9		Light switch to AUTO -> no effect on light as it is bright	ELS-18	0:03	2	1	0
10		Start driving thourght tunnel, value at border -> no effect	ELS-18	0:04	2	1	0
11		Brightness value below threshold -> light on	ELS-18	0:05	2	1	0
12		Brighness value exceeds hystheresis value, but no 3 seconds on time -> no effect	ELS-18	0:06	2	1	0
13		Brightness value below threshold -> light remains on	ELS-18	0:07	2	1	0
14		Brighness value exceeds hystheresis value, and light has been on for at least 3 seconds -> light off	ELS-18	0:08	2	1	0
15		Tunnel ride terminated -> no effect	ELS-18	0:12	2	1	0
16		Light switch to ON -> lights on	ELS-14	0:13	2	1	0

Environment model

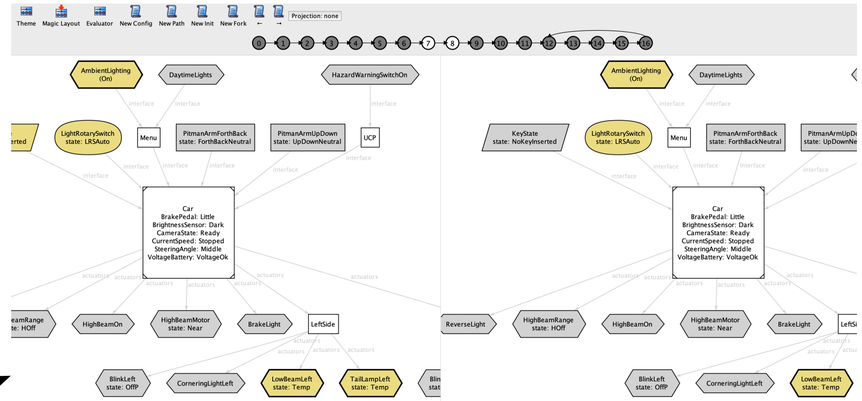
State machine definition

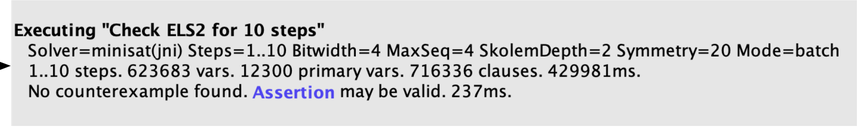
Visualization elements

Animation commands

Reference commands

Check commands





Executing "Check ELS2 for 10 steps"
Solver=minisat(jni) Steps=1..10 Bitwidth=4 MaxSeq=4 SkolemDepth=2 Symmetry=20 Mode=batch
1..10 steps. 623683 vars. 12300 primary vars. 716336 clauses. 429981ms.
No counterexample found. **Assertion** may be valid. 237ms.

Reference documents

Electrum model

Electrum Analyzer

ELS Model

Environment

- Mimics input and output signal architecture to streamline translation
- To ease modelling and maintainability:
  - Signal hierarchy introduced
  - Boolean signals distinguished
- Acceptable value discretised from the requirements
- Evolution as mutable elements

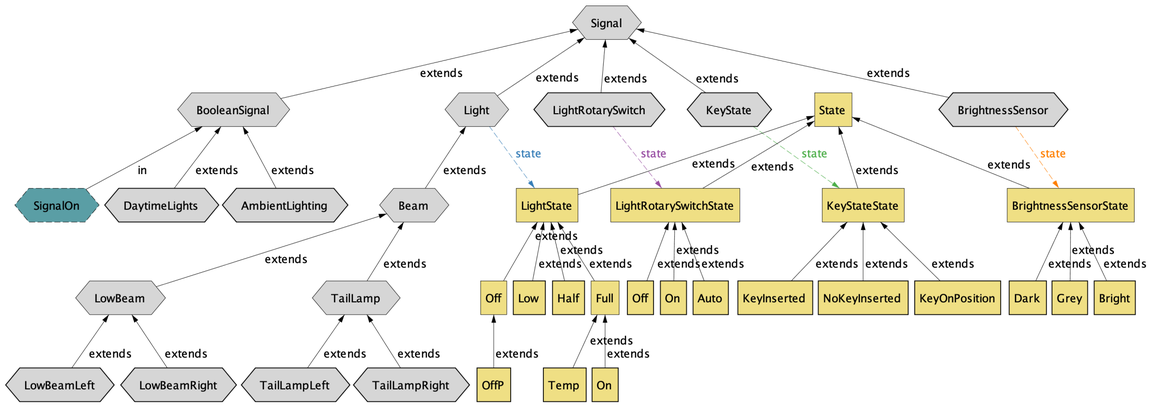
```
abstract sig Signal {}
abstract sig BooleanSignal extends Signal {}
var sig SignalOn in BooleanSignal {}

abstract sig Light extends Signal {
   var state : one LightState }
abstract sig Beam, ... extends Light {}

abstract sig LowBeam, TailLamp extends Beam {}
one sig LowBeamLeft, LowBeamRight extends LowBeam {}
one sig TailLampLeft, TailLampRight extends TailLamp {}

one sig AmbientLighting, DaytimeLights extends BooleanSignal {}

abstract sig LightState extends State {}
abstract sig Full, Off extends LightState {}
one sig Half, Low extends LightState {}
one sig On, Temp, ... extends Full {}
one sig OffP, ... extends Off {}
```



Low beam headlights signals

ELS Model

State machine

- A predicate for each ELS function, enforced to make the system evolve
- Determine the next state of the signals from the current state
- No explicit notion of time, each state has arbitrary duration
- Timed events are allowed to take an arbitrary (but finite) number of states

```
KeyState.state in KeyInIgnitionOnPosition and
   LightRotarySwitch.state in LSOn implies LowBeam.state' in On
```

ELS-14

```
DaytimeLights in SignalOn and
KeyState.state not in KeyInIgnitionOnPosition or
(LowBeam.state in On and KeyState.state in KeyInserted and
AmbientLighting not in SignalOn) implies LowBeam.state' in On
```

ELS-17

```
let low = LowBeam.state |
   LightRotarySwitch.state in LRSAuto and KeyState.state in
   KeyInIgnitionOnPosition implies
      one low' and
      BrightnessSensor.state in Dark
         implies low' in low.(univ→Temp+Temp→On++On→On) else
      BrightnessSensor.state in Bright implies
         low' in low.(univ→Off+Temp→Temp) else
      BrightnessSensor.state in Grey and low not in Temp implies
         low' in low.(iden+Temp→On)

   low in Temp implies eventually low not in Temp
```

ELS-18

Handling variability

Pure Electrum idiom

- Features are “lifted” to the language
- Structural and behavioural constraints dependent on selected features
- Supported by the regular Analyzer, but difficult to maintain

```
abstract sig Feature {}
abstract sig MarketCode extends Feature {}
one sig USA, EU, Canada extends MarketCode {}
one sig ArmoredVehicle extends Feature {}
sig Variant in Feature {}
fact FeatureModel {
   USA in Variant iff no (EU+Canada)&Variant
   Canada in Variant iff no (USA+EU)&Variant
   EU in Variant iff no (Canada+USA)&Variant }
```

Feature modelling

```
fact darknessModeSwitchOn {
   some DarknessModeSwitchOn iff ArmoredVehicle in Variant }
```

Structural variability

```
not (ArmoredVehicle in Variant and DarknessModeSwitchOn in
SignalOn) and
   AmbientLighting in SignalOn and BrightnessSensor.state in Dark
   and before KeyState.state in KeyInIgnitionOnPosition and
   KeyState.state not in KeyInIgnitionOnPosition implies
      LowBeam.state’ in Temp
```

Behavioural variability

Handling variability

Colorful Electrum

- Language extension to model variability
- Variability points with positive/ negative presence conditions
- Feature-aware analysis (through projection or feature lifting)

```
fact FeatureModel {
   ①②some none②① and ②③some none③②
   ①③some none③① and ❶❷❷some none❸❷❶ }
```

Feature modelling

```
④one sig DarknessModeSwitchOn extends BooleanSignal④
```

Structural variability

```
④not DarknessModeSwitchOn in SignalOn④ and
AmbientLighting in SignalOn and BrightnessSensor.state in Dark
and
... implies LowBeam.state’ in Temp
```

Behavioural variability

Validation & Verification

Animation and validation

- Simple commands to demonstrate functionalities
- Encode a sequence of input signals (and expected output for quick validation)
- Elements to structure visualiser (do not affect analysis)
- Operations to explore trace instances (e.g., change transition)

```
pred LowBeam2Env {
   always AmbientLighting not in SignalOn
   always KeyState.state in KeyInserted
   let lrs = LightRotarySwitch.state |
      lrs in LSOff;always lrs in LSOn }
```

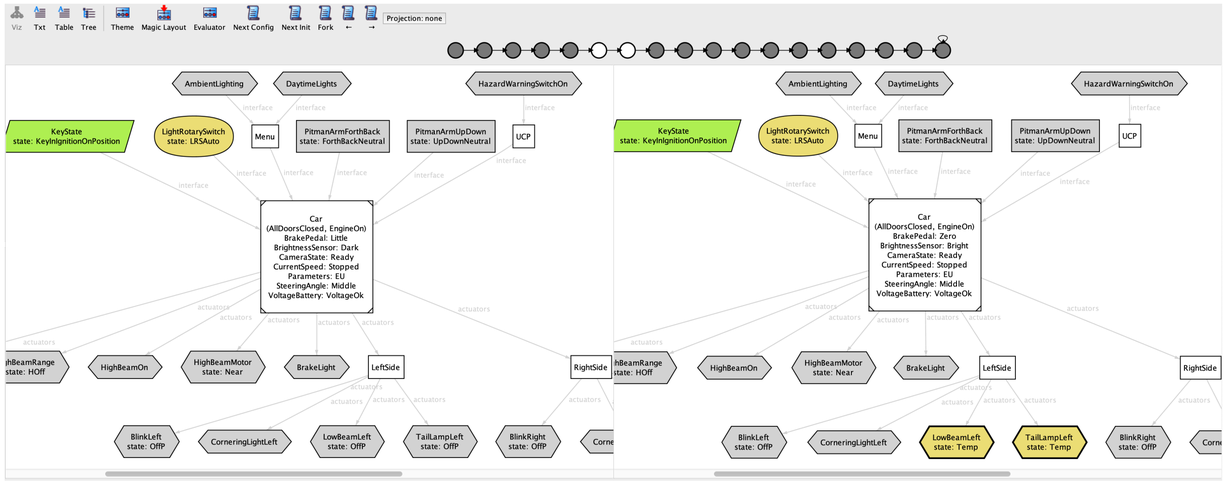
Low beams input

```
pred LowBeam2Exp {
   LowBeam.state in OffP;always LowBeam.state in Half }
```

Low beams expected output

```
run LowBeam2 {
   LowBeam2Env and after LowBeam2Exp } for 5 steps
```

Animation command



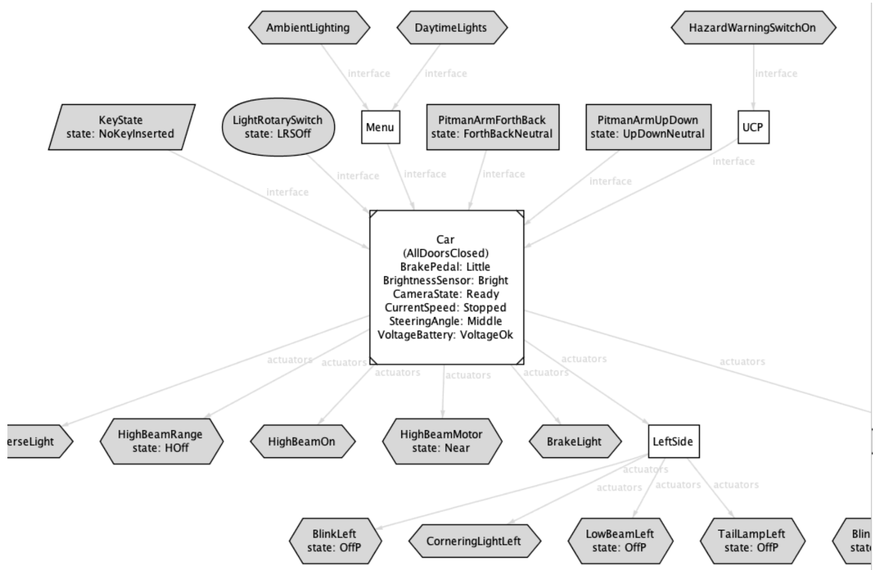
Validation & Verification

Reference scenarios

- Validator implemented to check reference sequences
- Input signals translated into commands
- Resulting output signals checked against the reference
- Relies on the discretisation of signal values

```
let s1 = not AmbientLighting in SignalOn |
   always s1
let s1 = not DarknessModeSwitchOn in SignalOn |
   always s1
let s1 = LightRotarySwitch.state in LSAuto, s0 =
LightRotarySwitch.state in LSOn, s2 = LightRotarySwitch.state in LSOff
|
   s2;s2;s2;s1;s1;s1;s1;s1;s1;s1;s0;s0;s1;s0;s0;s0;always s1
let s0 = BrightnessSensor.state in Dark, s1 = BrightnessSensor.state
in Grey, s2 = BrightnessSensor.state in Bright |
   s2;s2;s2;s2;s1;s0;s2;s0;always s2
EU in Variant
ArmoredVehicle not in Variant
...
after {
   let s2 = LowBeamLeft.state in LightLow, s3 = LowBeamLeft.state in
   LightOff, s1 = LowBeamLeft.state in LightHalf, s0 =
   LowBeamLeft.state in LightFull |
      s3;s3;s3;s3;s3;s0;s0;s0;s3;s3;s0;s3;s3;s1;s3;s2;always s3 ... }
```

Reference sequence 1



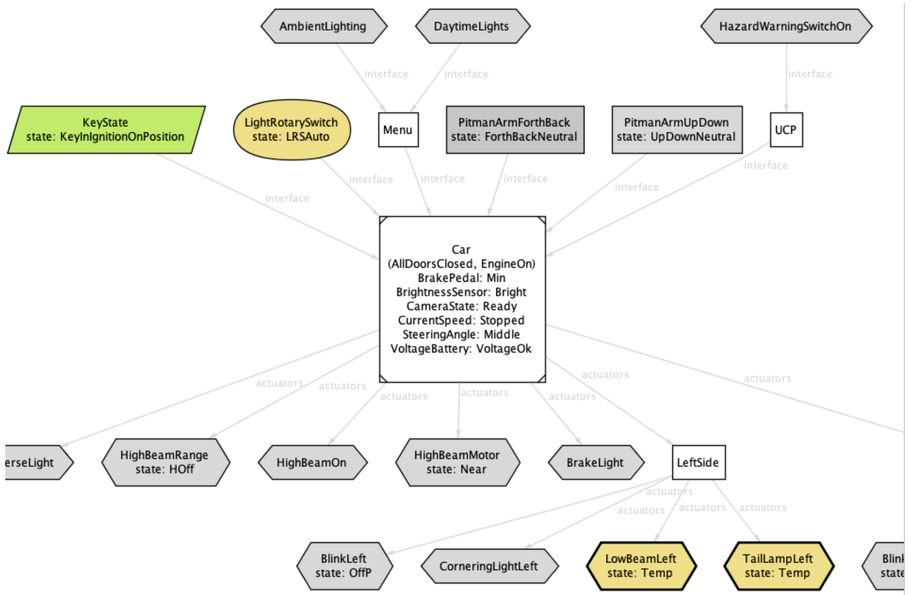
Validation & Verification

Reference scenarios

- Validator implemented to check reference sequences
- Input signals translated into commands
- Resulting output signals checked against the reference
- Relies on the discretisation of signal values

```
let s1 = not AmbientLighting in SignalOn |
   always s1
let s1 = not DarknessModeSwitchOn in SignalOn |
   always s1
let s1 = LightRotarySwitch.state in LSAuto, s0 =
LightRotarySwitch.state in LSOn, s2 = LightRotarySwitch.state in LSOff
|
   s2;s2;s2;s1;s1;s1;s1;s1;s1;s1;s0;s0;s1;s0;s0;s0;always s1
let s0 = BrightnessSensor.state in Dark, s1 = BrightnessSensor.state
in Grey, s2 = BrightnessSensor.state in Bright |
   s2;s2;s2;s2;s1;s0;s2;s0;always s2
EU in Variant
ArmoredVehicle not in Variant
...
after {
   let s2 = LowBeamLeft.state in LightLow, s3 = LowBeamLeft.state in
   LightOff, s1 = LowBeamLeft.state in LightHalf, s0 =
   LowBeamLeft.state in LightFull |
      s3;s3;s3;s3;s3;s0;s0;s0;s3;s3;s0;s3;s3;s1;s3;s2;always s3 ... }
```

Reference sequence 1



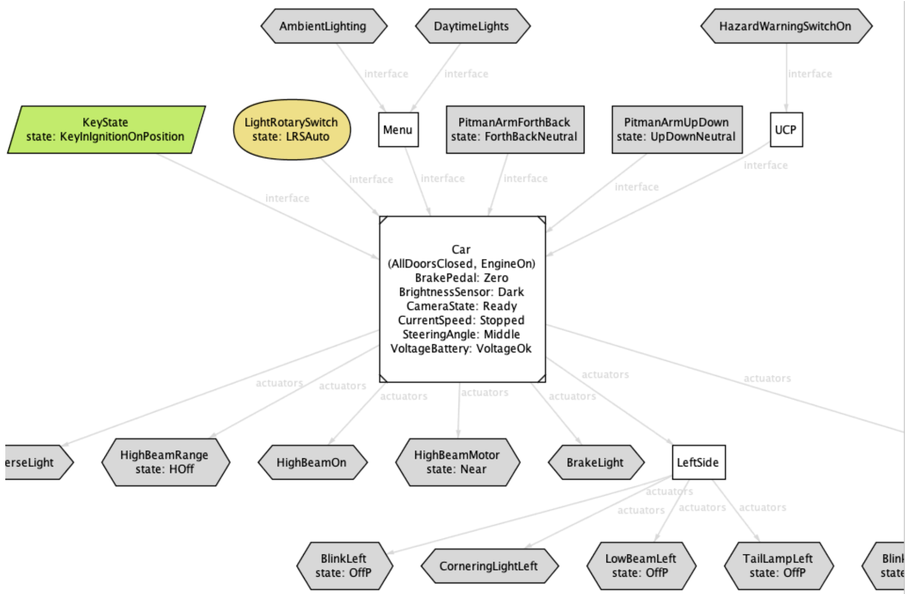
Validation & Verification

Reference scenarios

- Validator implemented to check reference sequences
- Input signals translated into commands
- Resulting output signals checked against the reference
- Relies on the discretisation of signal values

```
let s1 = not AmbientLighting in SignalOn |
   always s1
let s1 = not DarknessModeSwitchOn in SignalOn |
   always s1
let s1 = LightRotarySwitch.state in LSAuto, s0 =
LightRotarySwitch.state in LSOn, s2 = LightRotarySwitch.state in LSOff
|
   s2;s2;s2;s1;s1;s1;s1;s1;s1;s1;s0;s0;s1;s0;s0;s0;always s1
let s0 = BrightnessSensor.state in Dark, s1 = BrightnessSensor.state
in Grey, s2 = BrightnessSensor.state in Bright |
   s2;s2;s2;s2;s1;s0;s2;s0;always s2
EU in Variant
ArmoredVehicle not in Variant
...
after {
   let s2 = LowBeamLeft.state in LightLow, s3 = LowBeamLeft.state in
   LightOff, s1 = LowBeamLeft.state in LightHalf, s0 =
   LowBeamLeft.state in LightFull |
      s3;s3;s3;s3;s3;s0;s0;s0;s3;s3;s0;s3;s3;s1;s3;s2;always s3 ... }
```

Reference sequence 1



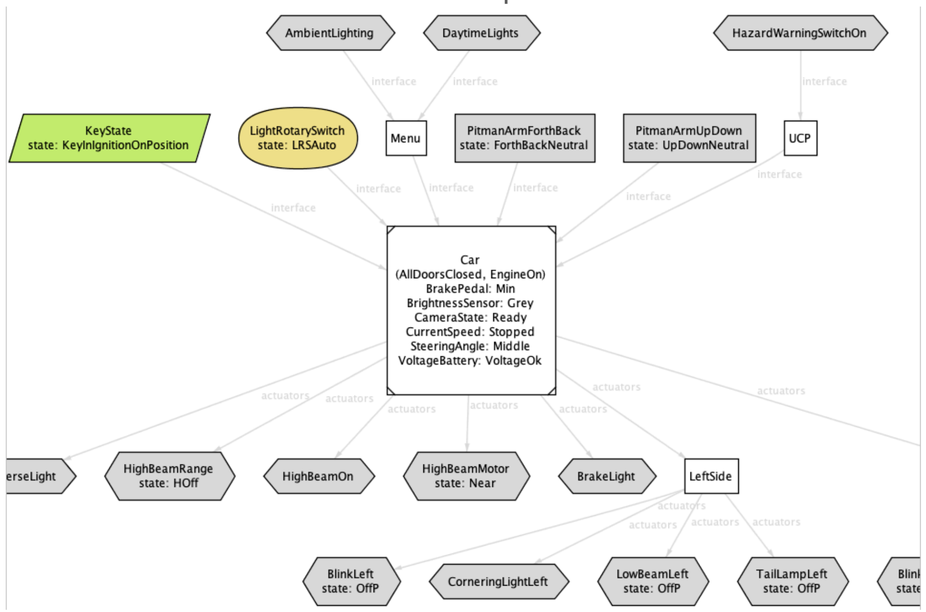
Validation & Verification

Reference scenarios

- Validator implemented to check reference sequences
- Input signals translated into commands
- Resulting output signals checked against the reference
- Relies on the discretisation of signal values

```
let s1 = not AmbientLighting in SignalOn |
   always s1
let s1 = not DarknessModeSwitchOn in SignalOn |
   always s1
let s1 = LightRotarySwitch.state in LSAuto, s0 =
LightRotarySwitch.state in LSOn, s2 = LightRotarySwitch.state in LSOff
|
   s2;s2;s2;s1;s1;s1;s1;s1;s1;s1;s0;s0;s1;s0;s0;s0;always s1
let s0 = BrightnessSensor.state in Dark, s1 = BrightnessSensor.state
in Grey, s2 = BrightnessSensor.state in Bright |
   s2;s2;s2;s2;s1;s0;s2;s0;always s2
EU in Variant
ArmoredVehicle not in Variant
...
after {
   let s2 = LowBeamLeft.state in LightLow, s3 = LowBeamLeft.state in
   LightOff, s1 = LowBeamLeft.state in LightHalf, s0 =
   LowBeamLeft.state in LightFull |
      s3;s3;s3;s3;s3;s0;s0;s0;s3;s3;s0;s3;s3;s1;s3;s2;always s3 ... }
```

Reference sequence 1



Validation & Verification

Checking requirements

- Assertions written in arbitrary first-order linear temporal logic
- Scope both on universe and maximum trace length
- For traces of length 15, most take a few seconds (some up to a minute)

```
assert ELS14 { always {
   KeyState.state in KeyInIgnitionOnPosition
   LightRotarySwitch.state in LSOn implies LowBeam.state’ in
   Full) } }
```

ELS-14

```
assert ELS17 {
   let keyPos = KeyState.state in KeyInIgnitionOnPosition,
      amb = AmbientLighting in SignalOn,
      day = DaytimeLights in SignalOn |
   always (day and not amb) implies
      always ((LowBeam.state’ in Full+Half until not keyPos) or
      always keyPos) }
```

ELS-17

Executing "Check ELS2 for 10 steps"
Solver=minisat(jni) Steps=1..10 Bitwidth=4 MaxSeq=4 SkolemDepth=2 Symmetry=20 Mode=batch
1..10 steps. 623683 vars. 12300 primary vars. 716336 clauses. 429981ms.
No counterexample found. Assertion may be valid. 237ms.

Conclusions

Outcome

- Modelled the 9 main functions
- Modelled the 12 variants (4 distinct behaviour)
- Validated against the 9 reference scenarios
- Checked most of the 48 requirements
- Model available at:

https://github.com/haslab/Electrum2/wiki/els

Conclusions

Issues identified in ELS

- Found 14 issues, resulted in fixes to the reference document and sequences
  - Inconsistencies detecting during early modelling stages
  - Ambiguities when modelling the state machine
  - Invalid outputs in the reference sequences
- Mostly related with dark cycles of blinking lights and handling of high beams

Conclusions

Limitations of the approach

- Main limitation is handling the 2 numeric requirements
  - e.g., ELS-17, calculate calculating illumination distance of high beams
- The time abstraction also disallows fine reasoning about timed requirements
  - e.g., ELS-10, forcing blinking cycles to take 1s
- This also rendered periodic functionalities the most cumbersome to encode
  - encoding cycles of arbitrary bounded duration