A Verified Low-Level Implementation of the Adaptive Exterior Light and Speed Control System

Sebastian Krings
Axivion GmbH, Niederrhein University of Applied Sciences

Philipp Körner, Jannik Dunkelau, Kristin Rutenkolk
University of Düsseldorf
This one is a little different than what we usually have at ABZ.
Differences to Other Case Studies

• We did not follow a (fully?) formal approach to the case study

• Instead:
  • Implemented ELS and SCS in low-level C
  • Used test-driven development
  • After some implementation, used model checking (on C!) for verification

• Attempted an approach closer to industry
  • No offense here!
  • Understand differences to help industrial uptake of FM
  • Provide a baseline to argue about advantages of FM in other case study realizations
Why Baselining?

• Some time ago, we read
  • Seven Myths of Formal Methods – Hall
  • Seven More Myths of Formal Methods – Bowen & Hinchey

• Both discuss myths regarding the costs of using FM and disprove them

• Yet, case studies implementing a system both formally and non-formally are
  • Done only seldomly
  • Often done either by teams of FM experts or non-experts
  • Tend to the extremes: all formal or not formal at all
Testing, Validation and Verification Vectors

- Test-driven development
  - For functional requirements
  - Realized using cmockery & ctest

- Restricted language Misra C
  - To avoid common pitfalls
  - Enforced by static analysis, to some extent dataflow and pointer analyses

- Model checking
  - For functional requirements
  - Directly on the C code
  - Realized using CBMC
Implementation Language

- Development and style guidelines for C
- Introduced by the Motor Industry Software Reliability Association
- Improve safety and security by avoiding common pitfalls
- To some extent automatically checkable, others undecidable

No verification of functional requirements here!

MISRA C:2012
Guidelines for the use of the C language in critical systems
March 2013
System Overview

- Clock (external)
  - Time

- Sensors (from Car)
  - Key Status
  - Brightness
  - ...

- Input (from Driver)
  - Pitman Arm
  - ...

- ELS
  - initialize ELS
  - read sensors
  - functionality
  - store last state

- SCS
  - initialize SCS
  - read sensors
  - functionality
  - store last state

- Output (to Car)
  - ELS State
  - SCS State
```
// ignition: key inserted + ignition on
sensor = update_sensors(sensor, sensorTime, 1000);
sensor = update_sensors(sensor, sensorBrightnessSensor, 500);
sensor = update_sensors(sensor, sensorKeyState, keyInIgnitionOnPosition);
sensor = update_sensors(sensor, sensorEngineOn, 1);

mock_and_execute(sensor_states);

sensor = update_sensors(sensor, sensorTime, 2000);
pitman_vertical(pa_Downward5);
mock_and_execute(sensor_states);

assert_partial_state(blinkLeft, 100, blinkRight, 0);
pitman_vertical(pa_ud_Neutral);
sensor = update_sensors(sensor, sensorTime, 2000);
mock_and_execute(sensor);

pitman_vertical(pa_Upward7);

progress_time_partial(2000, 2499, blinkLeft, 100, blinkRight, 0);
progress_time_partial(2500, 2999, blinkLeft, 0, blinkRight, 0);

int i;
for (i = 3; i < 6; i++) {
  progress_time_partial(i * 1000, i * 1000 + 499,
                        blinkLeft, 0, blinkRight, 100);
  progress_time_partial(i * 1000 + 500, i * 1000 + 999,
                        blinkLeft, 0, blinkRight, 0);
}
```
• Requirement: Whenever the low or high beam headlights are activated, the tail lights are activated, too.
• Translate into invariant
• Place assertion in C code, will be checked at runtime and during model checking

```c
assert(implies(state.lowBeamLeft > 0,
               state.tailLampLeft > 0 &&
               state.tailLampRight > 0));
```
Model Checking using CBMC – Additional Assumptions

- CBMC unrolls transition system
- Needs to know cardinalities of enumerations, domain of variables, etc.
- Some are automatically detected, some depend on implementation (e.g. size of integers)
- Can be passed to CBMC so support it
- Again, this is a macro transparent to the C compiler

```c
keyState ks = get_key_status();
__CPROVER_assume(ks == NoKeyInserted
    || ks == KeyInserted
    || ks == KeyInIgnitionOnPosition);
```
State 59 file light/light-impl.c line 242 function light_do_step thread 0
----------------------------------------------------
ks=*/enum*/NoKeyInserted (00000000000000000000000000000000)

State 63 file light/light-impl.c line 242 function light_do_step thread 0
----------------------------------------------------
ks=*/enum*/KeyInIgnitionOnPosition (00000000000000000000000000000010)

State 65 file light/light-impl.c line 244 function light_do_step thread 0
----------------------------------------------------
engine_on=FALSE (00000000)

State 69 file light/light-impl.c line 244 function light_do_step thread 0
----------------------------------------------------
engine_on=TRUE (00000001)
Model Checking – CBMC - Remarks

- CBMC found counterexamples, e.g., to the requirement ELS-22
- Output was barely readable, i.e., 52 state transitions represent two high level states after the initialization
- As a result, we wrote our own state graph visualization tool
- A simple C-style assertion would often have tripped the test case, CBMC adds thoroughness
Model Checking - Example – Ad-hoc Visualization

A Verified Low-Level Implementation of the Adaptive Exterior Light and Speed Control System
Future Work

- Would like to follow up with an actual comparison of the different case studies
  - Discuss advantages and disadvantages of using FM
  - General lessons learned

- Idea: Make invariants from a formal model survive code generators and end up invariants in C code
  - Verify implementation again on actual system / in actual context
  - Spot errors in code generators
Summary

• We have
  • Implemented a low-level version of the case study
  • Used a language common in industry and broadly available infrastructure and techniques
  • Tried to stay close to what might have happened in industry
  • Provided a baseline to compare more formal approaches to

• We suspect
  • More rigorous approaches will show advantages and disadvantages
Thank you for your attention!

Contact:
Sebastian Krings
Axivion GmbH
Nobelstraße 15
70569 Stuttgart

Tel: +49 711 6204378-78
E-Mail: krings@axivion.com
www.axivion.com