Deductive verification of industrial automotive C code

Christian Lidström

clid@kth.se

KTH Royal Institute of Technology

About me

- MSc thesis at Scania, spring 2016
  - Deductive verification
- Consultant at Scania, 2016-2019
  - Research on application of Formal Methods
  - Various EU and Swedish projects
- PhD student at KTH since February
  - Funded through AVerT
    - “Automated Verificaton and Testing”
    - Vinnova FFI project
    - KTH, Scania collaboration

About Scania

- Manufacturer of heavy trucks and buses
- Worldwide production and sales
- 50,000 employees, 5,000 engineers
- >1,000,000 vehicles in operation, >300,000 connected
- 100,000 products sold/year

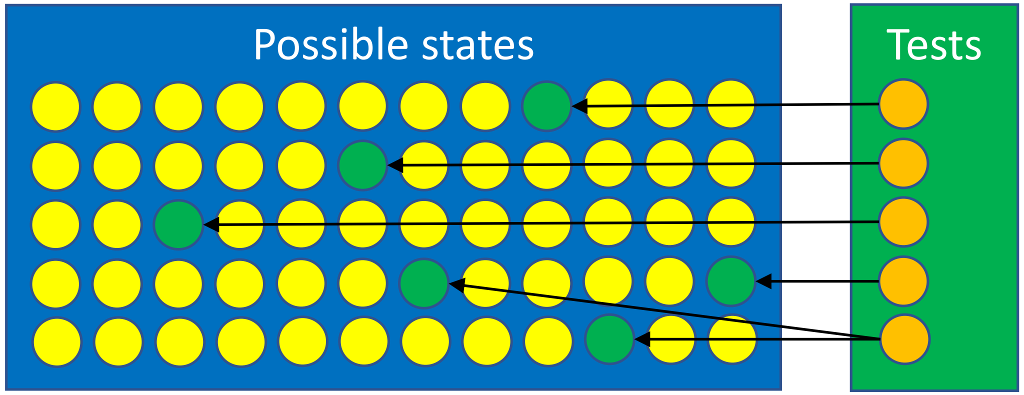
Formal Methods at Scania

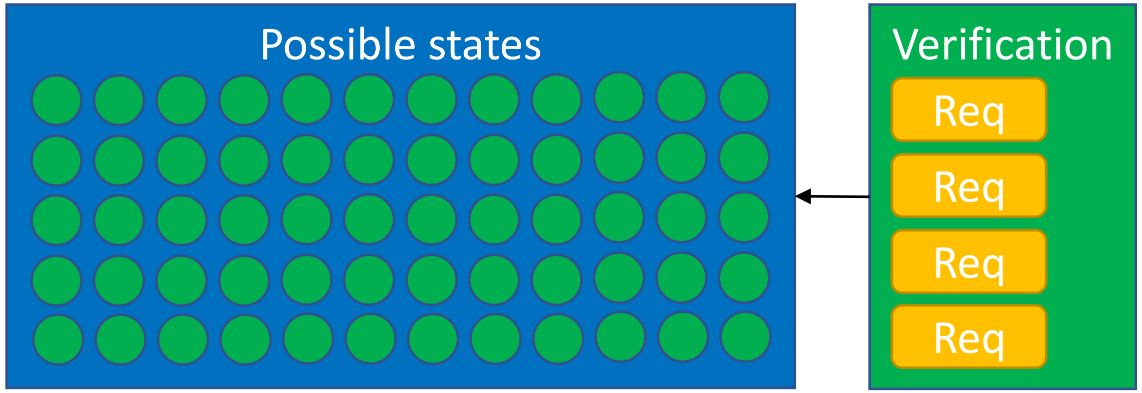
- Research >10 years
- Increased safety reqs.
  - ISO 26262
  - Autonomous vehicles
- Increased complexity
  - Autonomy / Platooning
  - Continuous integration
  - One product line, billions of variants
- More safety-critical SW

Deductive verification

- Deals with problems of:
  - Complexity
  - Number of variants
  - Amount of SW
- Increased coverage
- Increased confidence in correctness

- Tools:
  - Frama-C (WP)
  - VCC



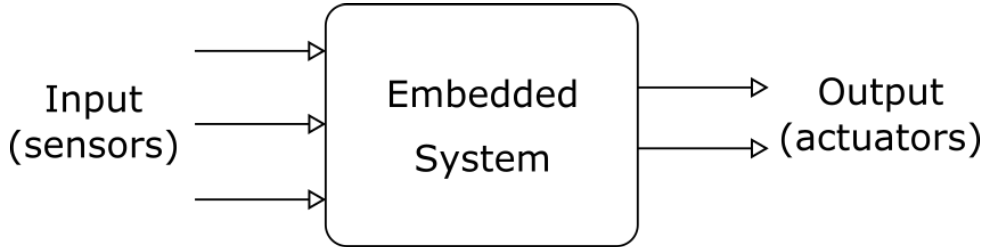


Benchmark results – Testing vs formal verification

Results of using mutation testing (inserting faults into the SW).

Fault	piTest	LBTest	Deductive verification*
1	Not terminated	Detected	Detected
2	Undetected	Detected	Detected
3	Detected	Undetected	Detected
4	Not terminated	Detected	Detected
5	Undetected	Detected	Detected
6	Not terminated	Undetected	Detected
7	Detected	Detected	Detected
8	Undetected	Detected	Detected
9	Not terminated	Detected	Detected
10	Not terminated	Detected	Detected

Case study: Dual-Circuit Steering (STEE)

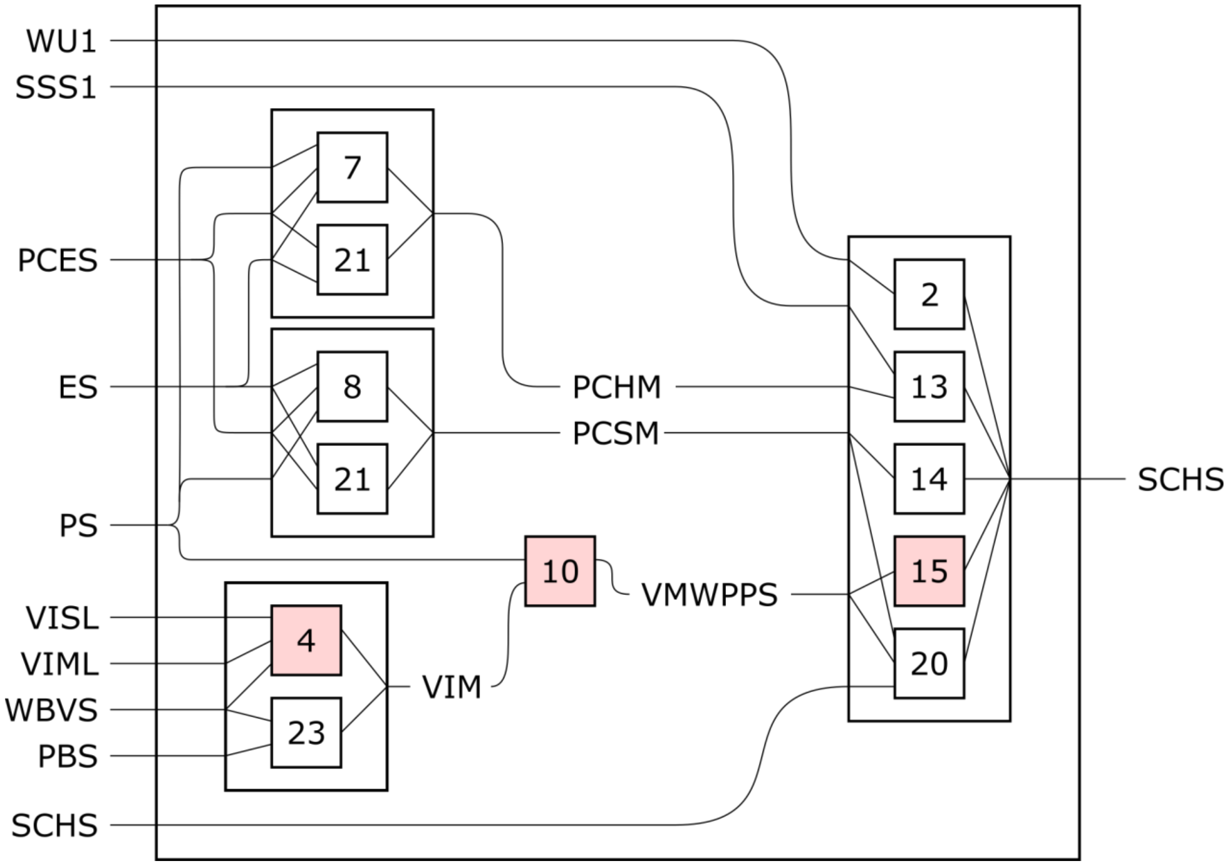


STEE requirements

Requirement #	Description
AER417_4	The vehicle is regarded as moving if vehicle speed signal is larger than 2km/h. The vehicle is regarded as stationary if the vehicle speed is below 1km/h. If WheelBasedVehicleSpeed > Vehicle Is Moving Limit Vehicle Is Moving = True If WheelBasedVehicleSpeed < Vehicle Is Stationary Limit Vehicle Is Moving = False
AER417_10	If the vehicle is moving without the primary circuit providing power steering the secondary steering circuit will be activated. If PositionSensor == NoFlow AND Vehicle Is Moving == True Vehicle Moving Without Primary Power Steering = True Else Vehicle Moving Without Primary Power Steering = False
AER417_15	If the vehicle is moving without the primary circuit providing power steering (see AER417_10) the secondary steering circuit will be activated. If Vehicle Moving Without Primary Power Steering == True Secondary Circuit Handles Steering = True

Requirements specified at module level

STEE requirements circuit



From Requirements to Contracts

- Requirement AER417_4:

> The vehicle is regarded as moving if vehicle speed signal is larger than 2km/h.
> The vehicle is regarded as stationary if the vehicle speed is below 1km/h.
>
> If WheelBasedVehicleSpeed > Vehicle Is Moving Limit
> Vehicle Is Moving = True
> If WheelBasedVehicleSpeed < Vehicle Is Stationary Limit
> Vehicle Is Moving = False

- Requirement as function contract in C source code:

```
/*@
 *   ...
 *   ensures \old(rtdb_ov_s32_astr[RTDB_VEHICLE_SPEED_E]) > STEE_V_VEHICLE_MOV_LIM_S32)
 *       ==> model_VehicleIsMoving == \true;
 *   ensures \old(rtdb_ov_s32_astr[RTDB_VEHICLE_SPEED_E]) < STEE_V_VEHICLE_STAT_LIM_S32)
 *       ==> model_VehicleIsMoving == \false;
 *   ...
 */
void Stee_10ms(tB enabled_B);
```

STEE Case study results

- 27 requirements in total
- 10 verified requirements (others not functional, module specific)

- Implementation file:
  - 10 functions, ~1400 LoC (+24 header files included)

- Verification required:
  - ~700 LoA
  - 165 seconds (< 3 minutes) for full module
  - 65 seconds for hardest function

Gurov et al (2017): *Deductive Functional Verification of Safety-Critical Embedded C-Code: An Experience Report*

What about the downsides?

- Formal methods requires:
  - Formal specifications/requirements
  - High expertise among engineers

- Deductive verification:
  - Requires even deeper knowledge (about tool/method)
  - Requires large human annotation effort
  - Tools lack features
  - Tools have scalability issues
  - Puts restrictions on code

Nyberg et al (2018): *Formal Verification in Automotive Industry: Enablers and Obstacles*

What about the downsides?

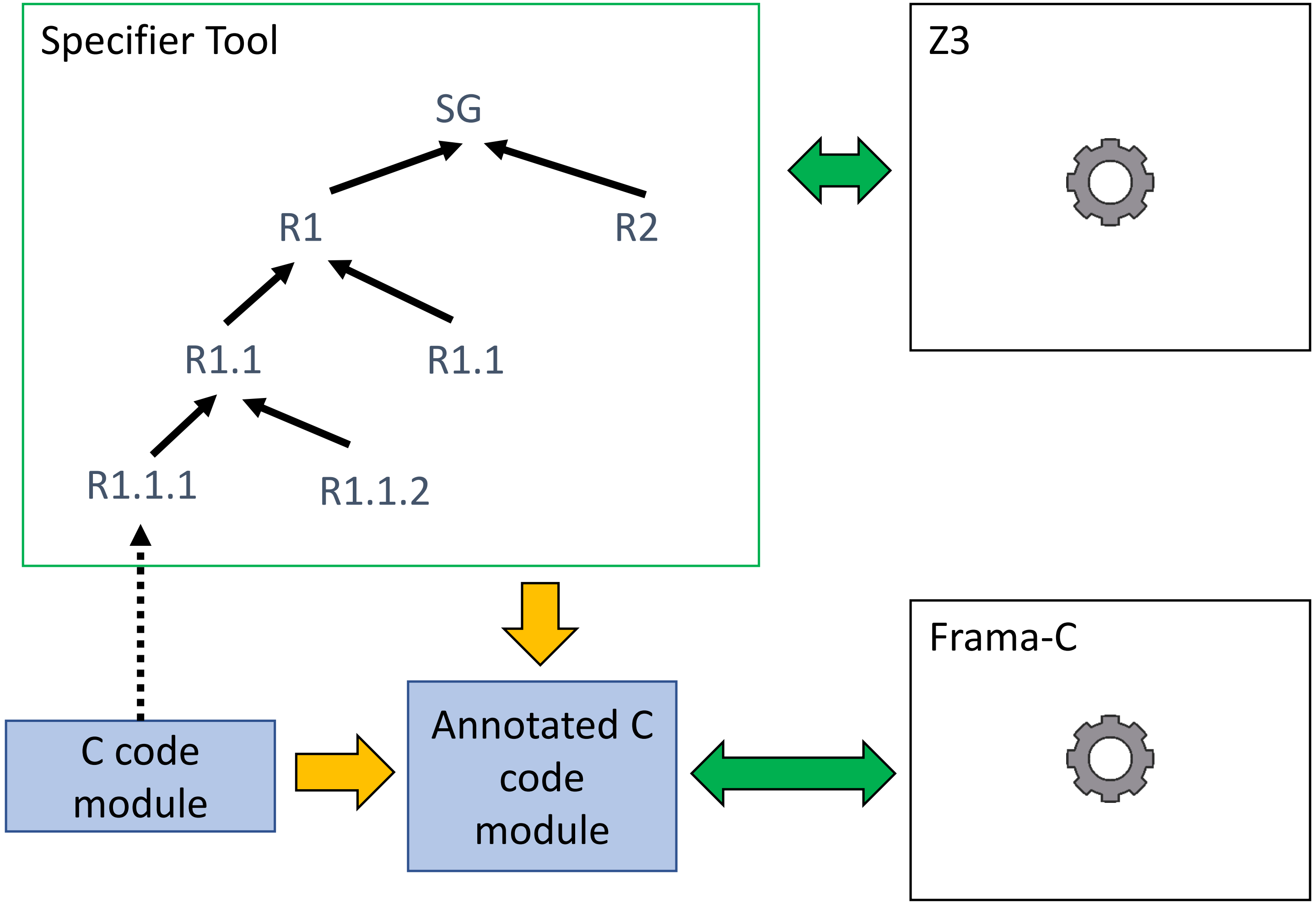
- Formal methods requires:
  - Formal specifications/requirements
  - High expertise among engineers

- Deductive verification:
  - Requires even deeper knowledge (about tool/method)
  - Requires large human annotation effort
  - Tools lack features
  - Tools have scalability issues
  - Puts restrictions on code

Solution: automation

Nyberg et al (2018): *Formal Verification in Automotive Industry: Enablers and Obstacles*

Automated tool chain



Modular verification

```
/*@
    requires \valid(p) && \valid(q);
    assigns *p, *q;
    ensures \old(*p > *q) ==> *p == \old(*q) && *q == \old(*p);
    ensures \old(*p <= *q) ==> *p == \old(*p) && *q == \old(*q);
*/
void swap_if_gt(int * p, int * q) {
    if (*p > *q)
        swap(p, q);
}
```

```
/*@
    requires \valid(p) && \valid(q);
    assigns *p, *q;
    ensures *p == \old(*q)
         && *q == \old(*p);
*/
void swap(int * p, int * q)
{
    int tmp = *p;
    *p = *q;
    *q = tmp;
}
```

- On function call: assert precondition, assume postcondition
- Helps with scalability, but adds contracting effort

Inlining vs Contracting

- Inlining:
  - Replacing function call with body of called function
  - Preferable when possible
  - But… performance issues, no longer modular verification
  - “Barrier” modules helps
  - Ongoing MSc thesis on heuristic to predict “inlinable” functions

```c
void swap_if_gt(int * p, int * q) {
    if (*p > *q)
        swap(p, q);
}
void swap(int * p, int * q) {
    int tmp = *p;
    *p = *q;
    *q = tmp;
}
```

Inlined swap()

```c
void swap_if_gt(int * p, int * q) {
    if (*p > *q)
        int tmp = *p;
        *p = *q;
        *q = tmp;
}
```

Automated annotation

```c
/*@
    requires \valid(p) && \valid(q);
    assigns *p, *q;
    ensures \old(*p > *q) ==> *p == \old(*q) && *q == \old(*p);
    ensures \old(*p <= *q) ==> *p == \old(*p) && *q == \old(*q);
*/
void swap_if_gt(int * p, int * q) {
    if (*p > *q)
        swap(p, q);
}
```

```c
/*@
    requires \valid(p) && \valid(q);
    assigns *p, *q;
    ensures *p == \old(*q)
         && *q == \old(*p);
*/
void swap(int * p, int * q)
{
    int tmp = *p;
    *p = *q;
    *q = tmp;
}
```

- Can generate relatively easy: entry-point contract, auxiliary annotations
- May require large human effort: helper function contract

Contract generation

- Use SMC to generate functional annotations
- Ongoing MSc thesis
- Uses Eldarica (horn clause solver with C interface)
- Results promising, contract generation in seconds

Contract generation

- Use SMC to generate functional annotations

```
/*@ contract @*/
swap(int * p, int * q);

// MC harness function
void main() {
    int x, y = nondet();
    int old_x = x;
    int old_y = y;
    swap(&x, &y);
    assert(x == old_y && y == old_x);
}
```

Eldarica

```
/*@
    // Functional contract generated
    ensures *p == \old(*q)
         && *q == \old(*p);
*/
void swap(int * p, int * q)
{
    int tmp = *p;
    *p = *q;
    *q = tmp;
}
```

Conclusion

- Formal methods needed
  - Complexity, software amount, autonomy, safety standards
- Deductive verification a great tool
  - But requires automation (to nearly 100%)
- Other issues:
  - Frama-C performance lacking
  - Automated verification of floating-point arithmetic

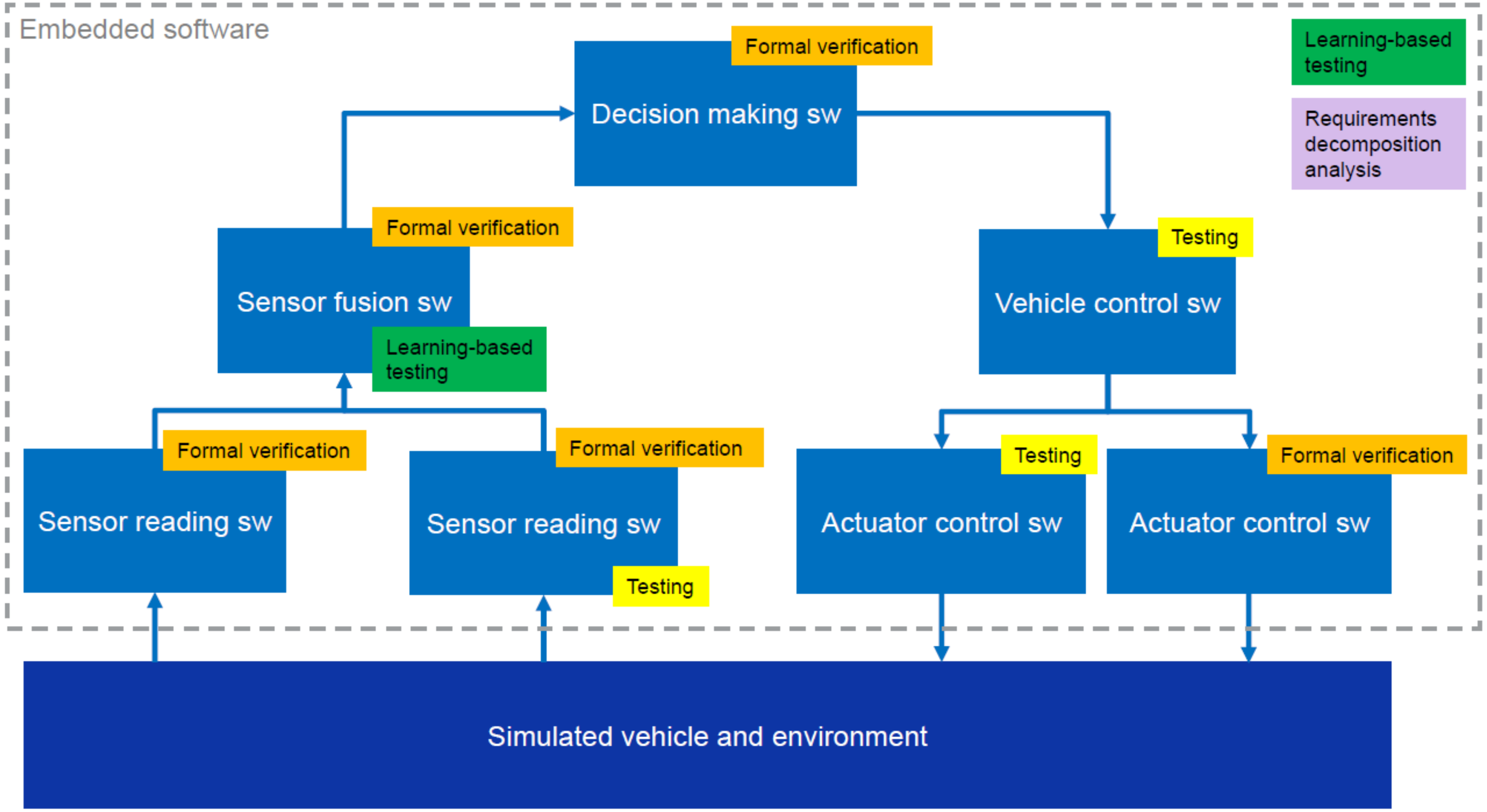
Future work

Fully automated verification

- Continue work on automation
  - Contract generation (under way)
  - Loop invariant inference
  - Model for temporal requirements
- Combine into automated toolchain

Integration into specification/requirements framework

Future work



Integration into specification/requirements framework

The end

Thanks for listening!

Any questions?