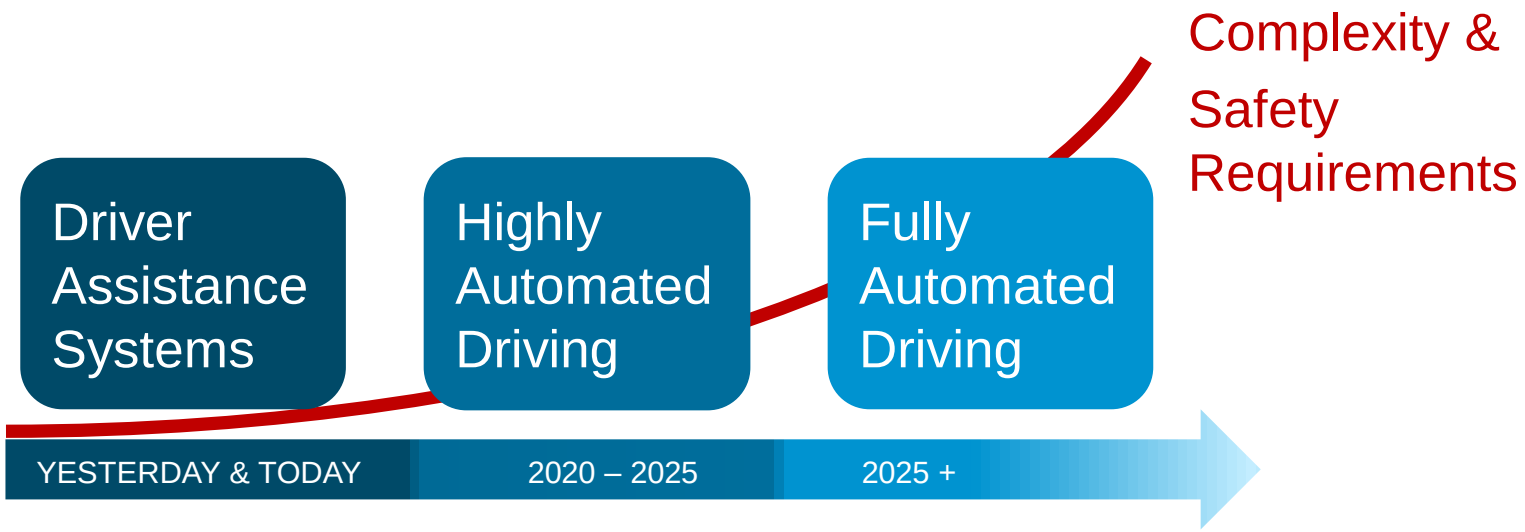
Model Checking AUTOSAR Components with CBMC

Timothée Durand [1] Katalin Fazekas [2]
Georg Weissenbacher [2] Jakob Zwirchmayr [1]

[1]TTTech AUTO AG, Vienna, Austria [2]TU Wien, Vienna, Austria

Monday 4th October, 2021



Self driving cars bring new technological challenges:

- Increase of safety requirements & complexity
- Centralized ECU architecture, Communication Systems...
- Established safety norms

Automobile Software

Driver Assistance (ADAS)

Cameras

Sensor Fusion

Computer Vision & AI

Chassis & Powertrain

Steering

Engine Mgt.

Brakes

Battery Mgt.

Safety / Performance Microcontrollers

Infotainment

Navigation

Smartphone Connectivity

Platform Services

Communication

System supervision

Synchronization

Automobile Software Components

Driver Assistance (ADAS)

SW-C SW-C SW-C SW-C SW-C

Chassis & Powertrain

SW-C SW-C SW-C SW-C SW-C

Standards:

- AUTOSAR

Safety / Performance Microcontrollers

Infotainment

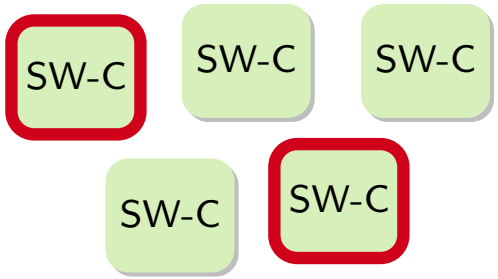
SW-C SW-C SW-C SW-C

Platform Services

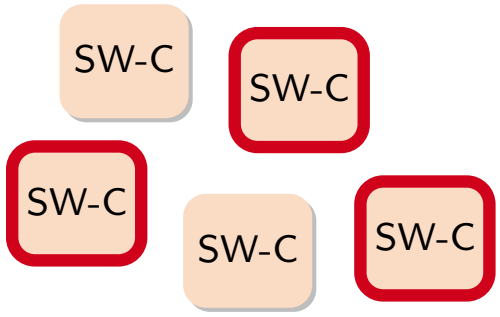
SW-C SW-C SW-C SW-C SW-C SW-C

Safety Critical Automobile Software Components

Driver Assistance (ADAS)



Chassis
& Powertrain



Standards:

- AUTOSAR
- *ISO 26262: ASIL*

Safety critical

D

B

C

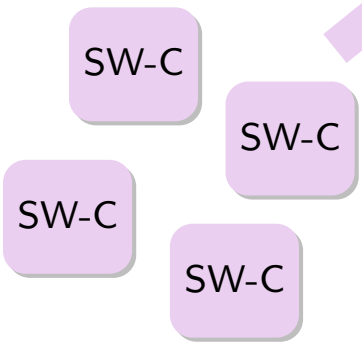
A

QM

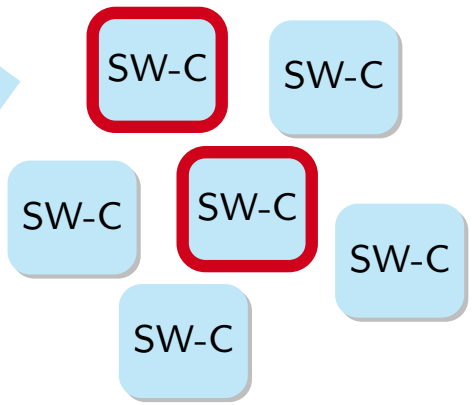
No safety guarantees

Safety / Performance
Microcontrollers

Infotainment



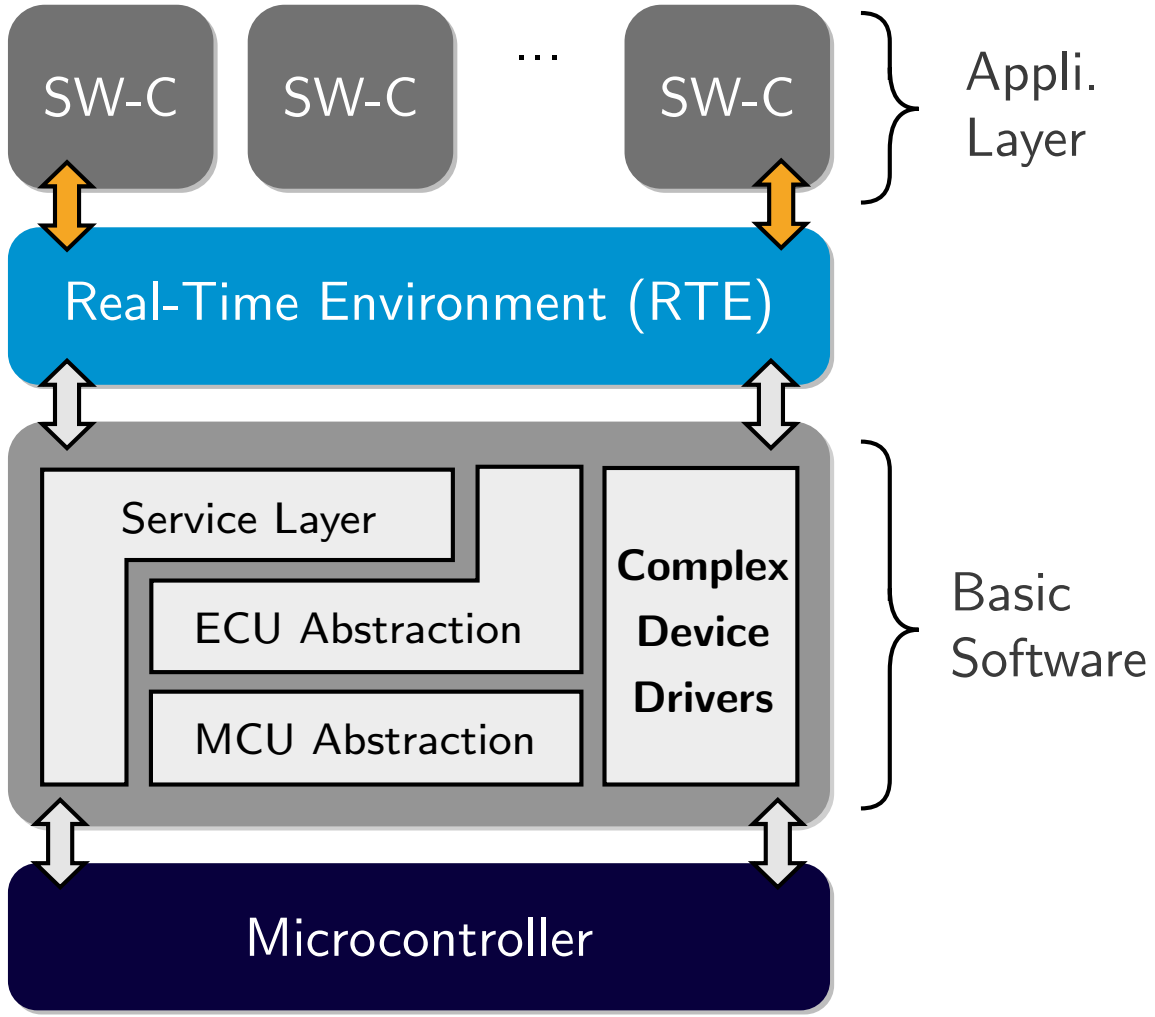
Platform Services



Automotive Software Architecture

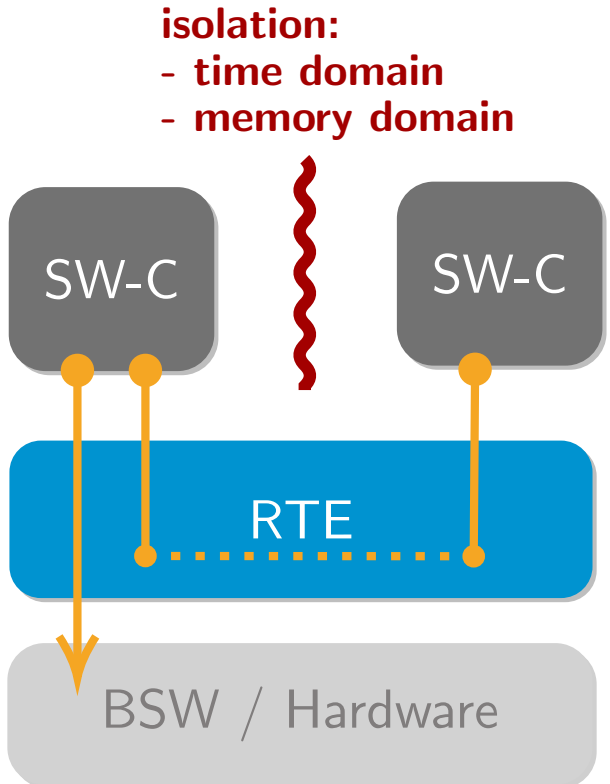
AUTOSAR - The Standardized Automotive Architecture

- AUTOSAR: Automotive Software Architecture Standard (2006)
- Defines common
  - architecture
  - interfaces
  - specifications
- Simplifies
  - integration
  - cooperation
  - code reusability



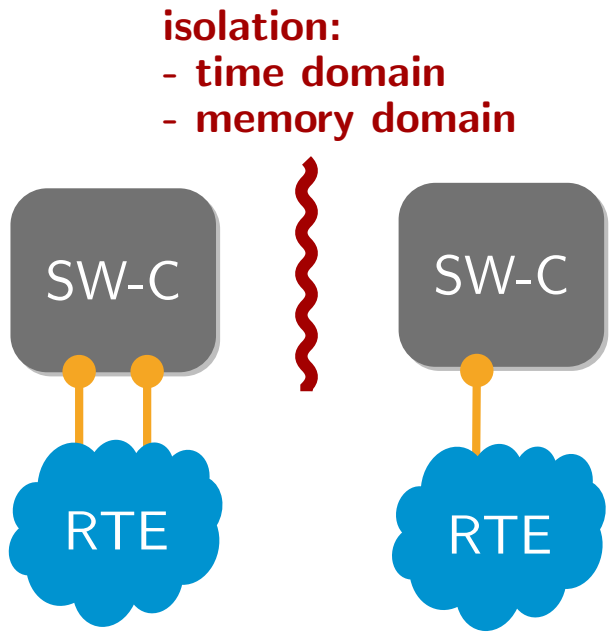
Focus on Software Components (SW-C)

- Diversity of application SW-C:
  - Safety critical services
  - Infotainement
- Composability safety concepts
  - Safely add/remove SW-C
  - *modular verification*
- Interaction through RTE
  - inter-SW-C communication
  - read sensor values
  - network communication

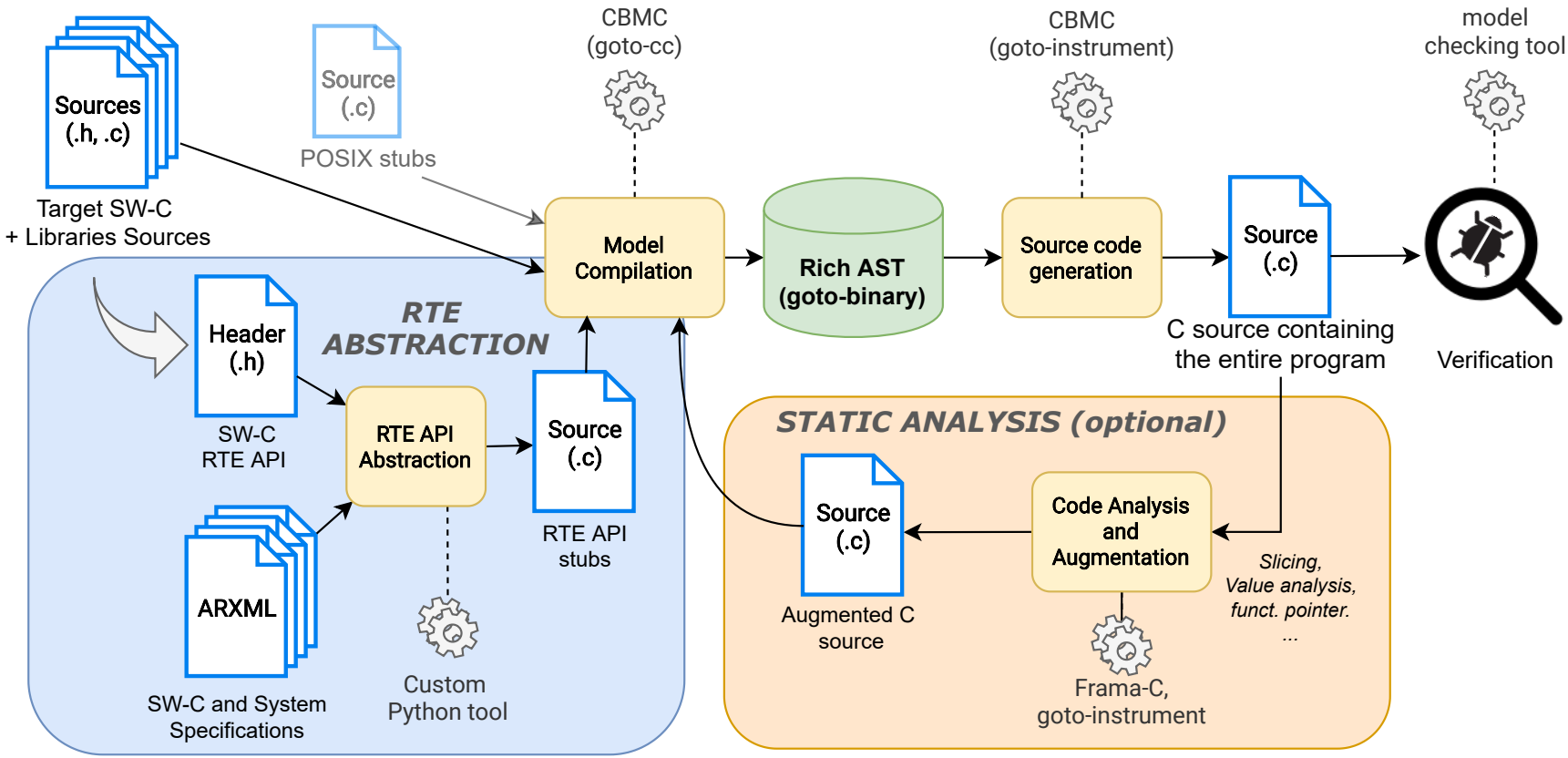


SW-C verification methodology

- Isolate and inspect one SW-C at a time
- Abstract RTE communication interfaces
- Simulate interactions using Non-Deterministic stubs
- Verify safety properties: Run-time errors



SW-C verification framework



RTE Abstraction

SW-C specifications to RTE Abstraction

ARXML (AUTOSAR-XML) provides:

- list of ports
- data-type specifications

```
<APPLICATION-SW-COMPONENT-TYPE>
  <SHORT-NAME>CtApSoftwareComponent1</SHORT-NAME>
    <PORTS>
      <R-PORT-PROTOTYPE>
        <SHORT-NAME>PpSwc1Temperature</SHORT-NAME>
        <REQUIRED-COM-SPECS>
          <NONQUEUED-RECEIVER-COM-SPEC>
            <DATA-ELEMENT-REF>
                /PortInterfaces/PiSwc1/DeTemperature
                          ...
            <TYPE-TREF>
                /DataTypes/Dt_TemperatureData
```

```
1 int Rte_Read_DeTemperature(Dt_TemperatureData* data)
```

SW-C specifications to RTE Abstraction

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>Dt_TemperatureData</SHORT-NAME>
  <CATEGORY>STRUCTURE</CATEGORY>
  <SUB-ELEMENTS>
    <SHORT-NAME>DeTemperature</SHORT-NAME>
      <IMPLEMENTATION-DATA-TYPE-REF>
        /DataTypes/Dt_Temperature
        ...
    <SHORT-NAME>DeTimestamp</SHORT-NAME>
      <IMPLEMENTATION-DATA-TYPE-REF>
        /DataTypes/Dt_Timestamp
```

```
1 struct Dt_TemperatureData {
2   Dt_Temperature DeTemperature;
3   Dt_Timestamp DeTimestamp;
4 };
```

SW-C specifications to RTE Abstraction

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>Dt_Temperature</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <BASE-TYPE-REF>/DataTypes/BaseTypes/sint16</BASE-TYPE-REF>
  <DATA-CONSTR-REF>/DataConstrs/Temp_DataConstr</DATA-CONSTR>
    ...
      <LOWER-LIMIT>-273</LOWER-LIMIT>
      <UPPER-LIMIT>1000</UPPER-LIMIT>
```

Non-deterministic RTE Datatype generator:

```
1 Dt_Temperature nondet_Dt_Temperature(){
2   Dt_Temperature res = nondet_sint16();
3   assume(res >= -273 && res <= 1000);
4   return res;
5 }
```

RTE interface stub

```
1  Dt_Temperature nondet_Dt_Temperature(){
2    Dt_Temperature res = nondet_sint16();
3    assume(res >= -273 && res <= 1000);
4    return res;
5  }
6
7  int STUB_Rte_Read_DeTemperature
8            (Dt_TemperatureData *data){
9    data->DeTemperature = nondet_Dt_Temperature();
10   data->DeTimestamp = nondet_Dt_Timestamp();
11   return 0;
12 }
```

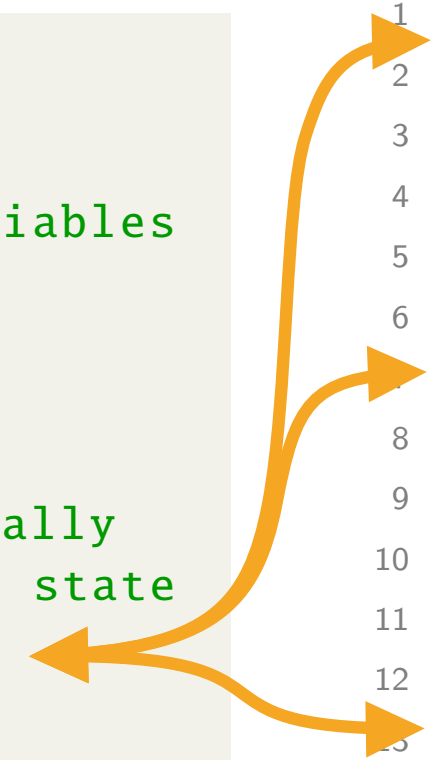
Software Component Verification Harness

SW-C codebase

```
1  int main()
2  {
3    // initialize
4    // global variables
5    initialize();
6
7    while(1)
8    {
9      // periodically
10     // maintain state
11     maintain();
12   }
13 }
```

RTE abstraction

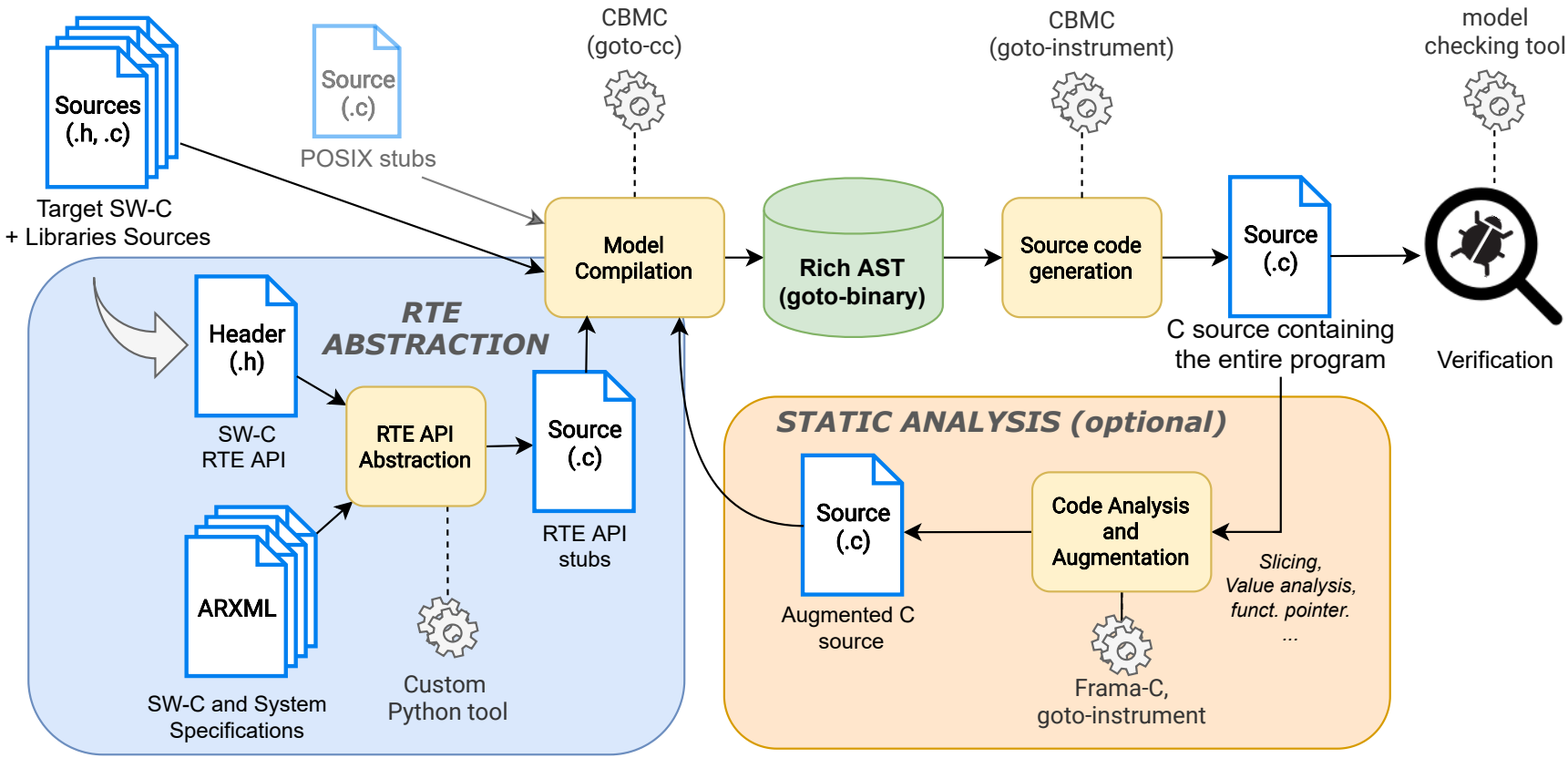
```
1  int STUB_Rte_Read_Engine_RPM
2    (Dt_EngineRPM* data)
3  {
4    //...
5  }
6
7  int STUB_Rte_Read_DeTemperature
8    (Dt_TemperatureData* data)
9  {
10   //...
11 }
12
13 int STUB_Rte_Write_FanSpeed
14 ...
```



$\Rightarrow$ Over-approximates the reachable state space

Verification methods

SW-C verification framework



Induction Proof Instrumentation

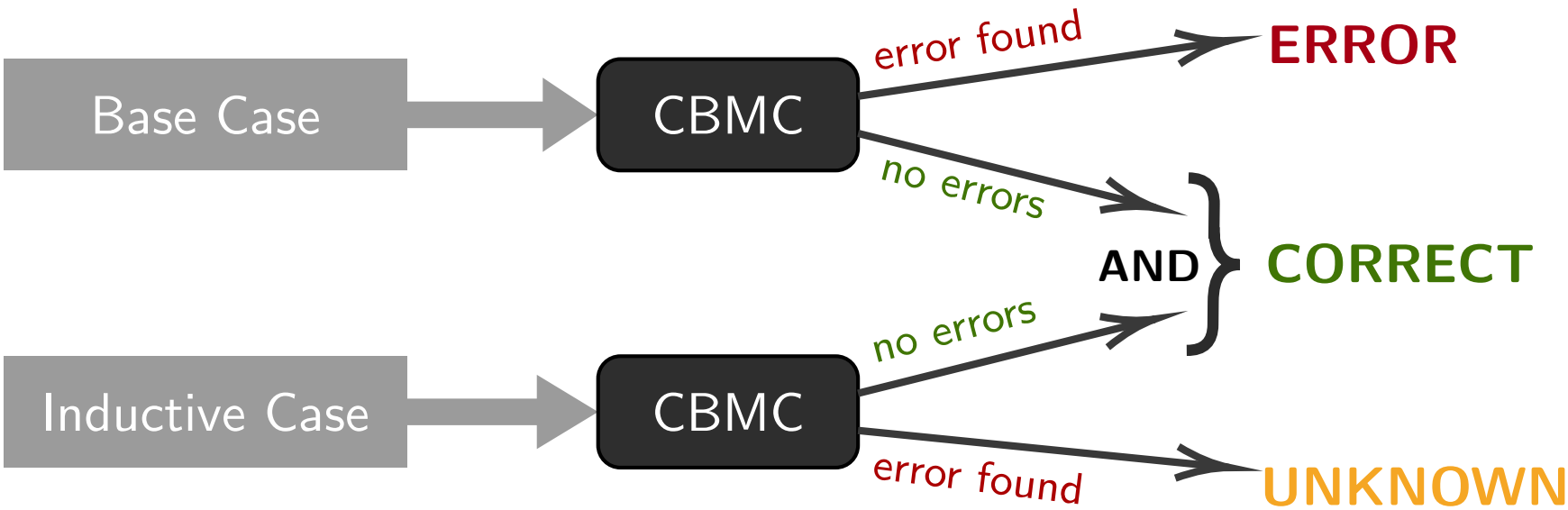
Base Case / Bounded model checking

```
1  int main()
2  {
3    initialize();
4    int i;
5    for(i=0; i < k; i++)
6    {
7      maintain();
8      assert(/*no errors*/);
9    }
10 }
```

Inductive Case

```
1  int main()
2  {
3    initialize();
4
5    // simulate arbitrary
6    // loop iteration
7    global_var = nondet();
8
9    int i;
10   for(i=0; i < k+1; i++)
11   {
12     assume(/*no errors*/);
13     maintain();
14   }
15   assert(/*no errors*/);
16 }
```

Both versions of the code are fed to CBMC which determines the reachability of error states.



Challenges for model checking:

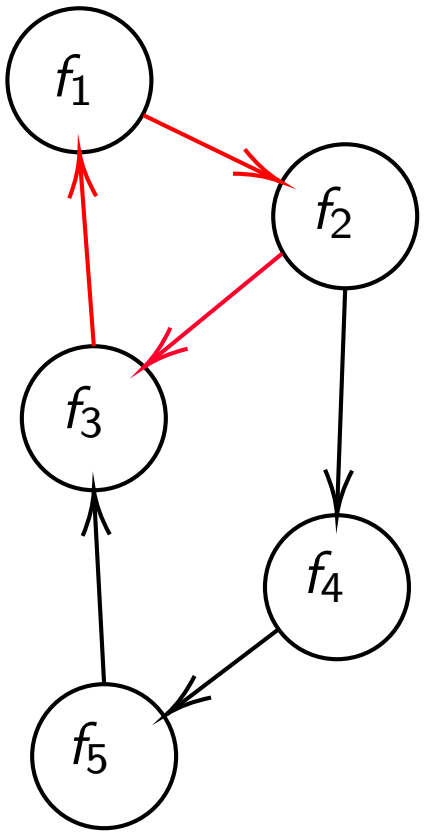
- Numerous function pointers
- Weak induction hypothesis
- Unclear SW-C boundaries

Frama-C's EVA plugin (Abstract Interpretation) provides:

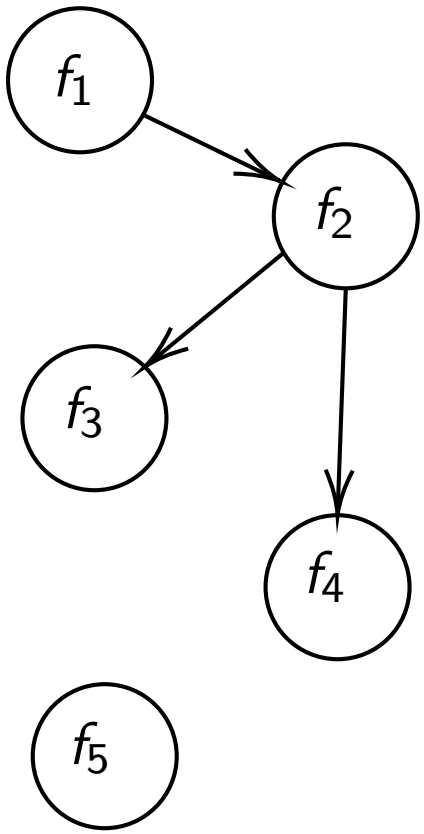
- Value-ranges for global variables
- Call graph

Static analysis: Resolve function pointers

Without static analysis



With static analysis



Note: Correctness of function pointer restriction is verified (Induction proof)

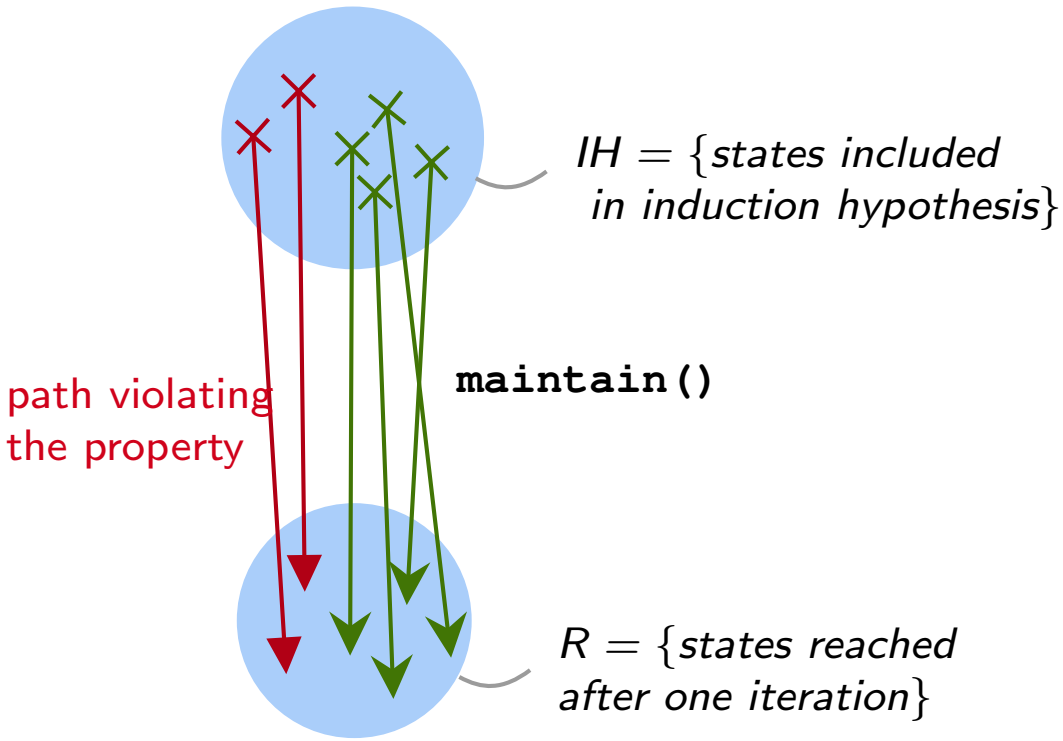
Static analysis: Strengthen induction hypothesis

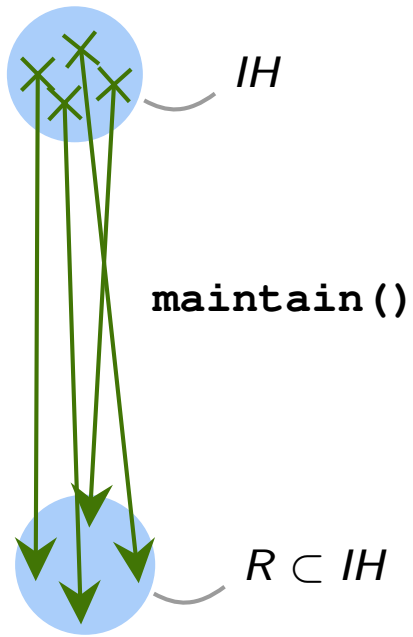
Without static analysis

```
1 // simulate arbitrary
2 // loop iteration
3 global_var = nondet();
```

With static analysis

```
1 // simulate arbitrary
2 // loop iteration
3 global_var = nondet();
4 assume(
5  global_var ∈ Range)
```





Note: Correctness of states restriction is verified (Induction proof)

Static analysis: Strengthen induction hypothesis

Base Case / Bounded model checking

```
1  int main()
2  {
3    initialize();
4    int i;
5    for(i=0; i < k; i++)
6    {
7      maintain();
8      assert(
9        global_vars ∈ Range);
10   }
11 }
```

Inductive Case

```
1  int main()
2  {
3    initialize();
4
5    // simulate arbitrary
6    // loop iteration
7    global_var = nondet();
8
9    int i;
10   for(i=0; i < k+1; i++)
11   {
12     assume(
13       global_vars ∈ Range);
14     maintain();
15   }
16   assert(
17     global_vars ∈ Range);
18 }
```

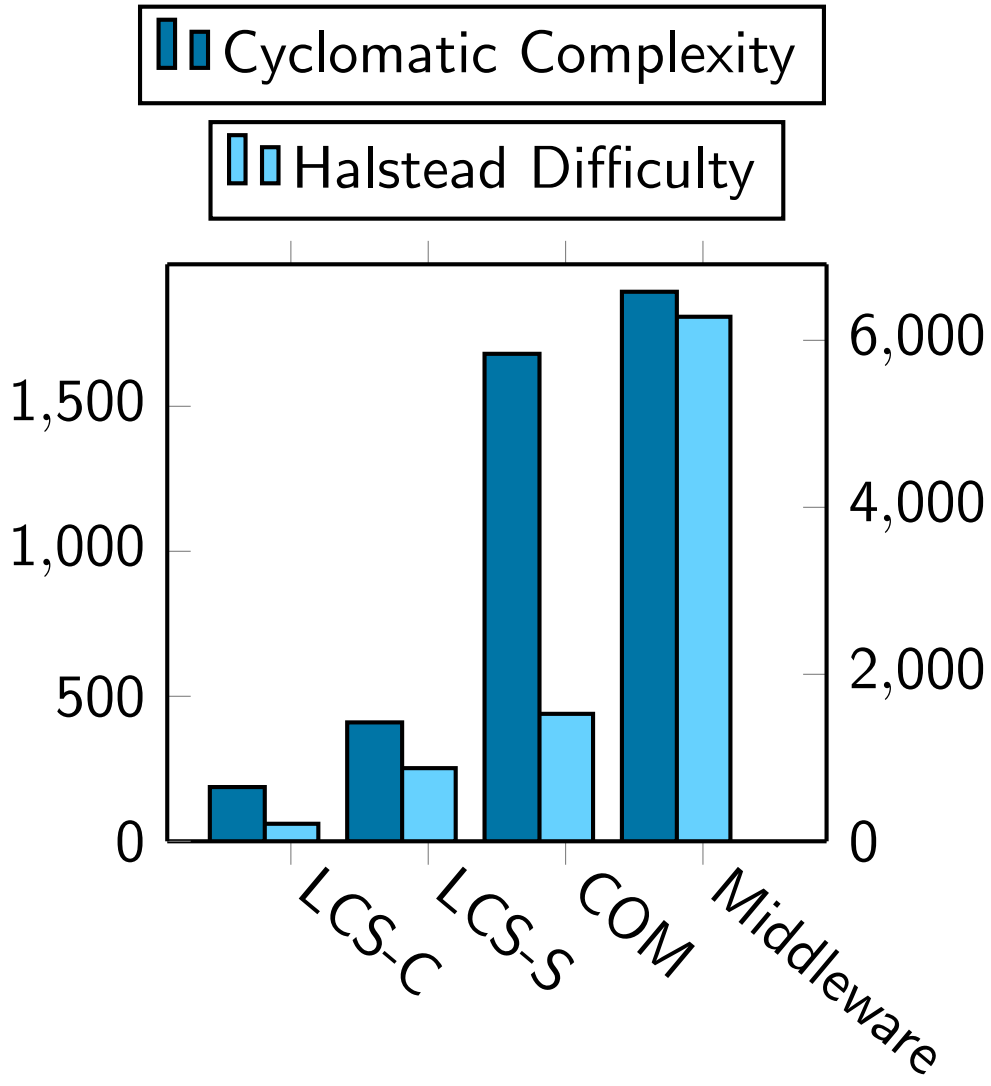
Experiments and Results

Targeted Software Components

TTTech

Four software components / CDD:

- Life Cycle Service
  - client
  - server
- Communication services:
  - COM: reading / writing to vehicle bus
  - Middleware: data tranfer between hosts



Measurements:

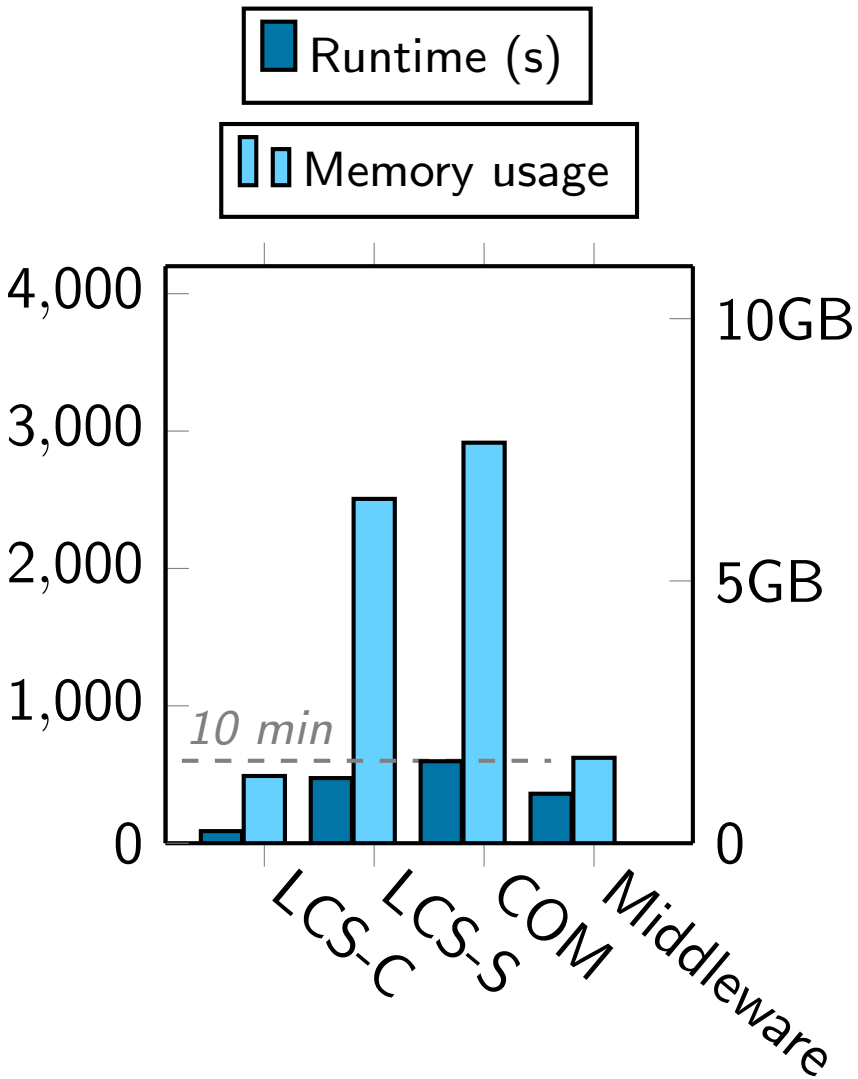
- Frama-C (static analysis) time & mem.
- CBMC (model checking) time & mem. (Base case)
- CBMC (model checking) time & mem. (Inductive case)

Resources:

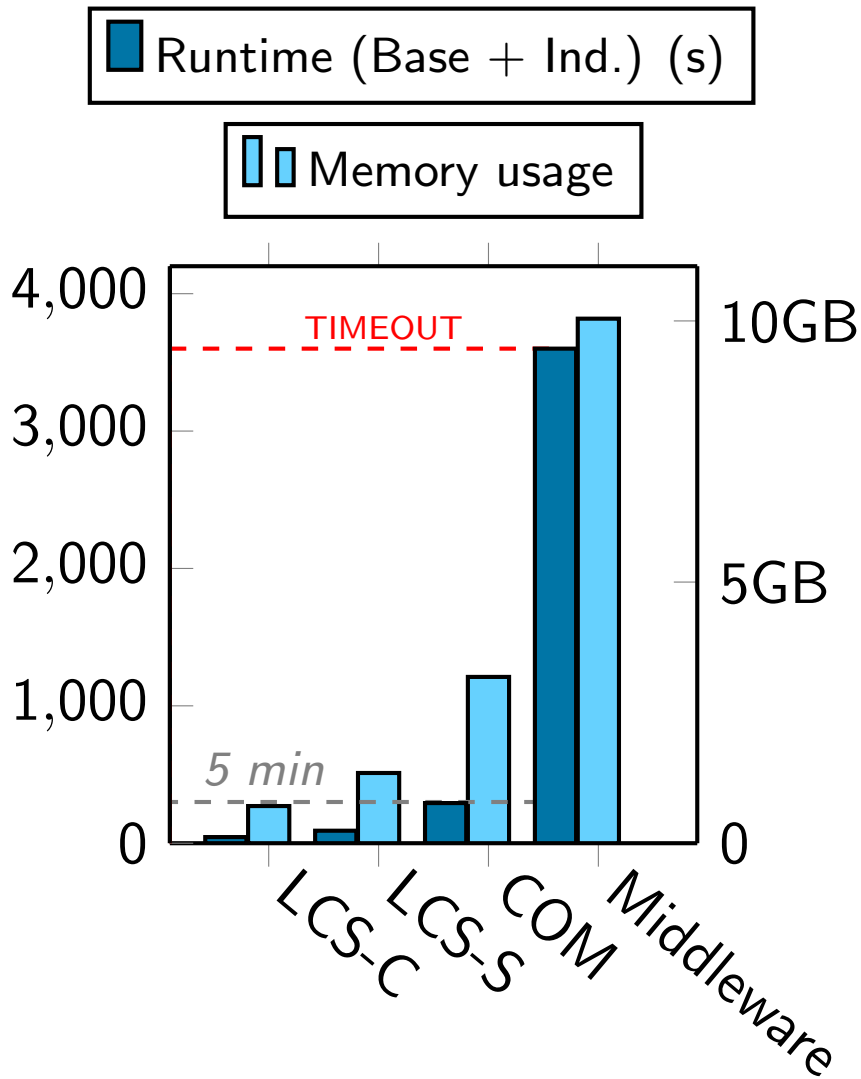
- Intel(R) Xeon(R) CPU E5345@2.33GHz
- 47.2 GB of memory
- timeout: one hour

Experiment Results

Static Analysis:



Model Checking:



Overview

Summary of Contributions

- Methodology for verification of AUTOSAR SW-C
  - Environment model
  - RTE Interface abstraction
  - k-induction encoding adapted for this problem
- Automated verification framework performing:
  1. Parsing of AUTOSAR Specification
  2. Generation of RTE interfaces abstraction
  3. Generation of a test harness (including base and inductive case of k-Induction)
  4. Parsing and Encoding of Static Analysis results (Frama-C)
  5. Verification with various tools
- Case Study on four real-world software components developed by TTTech

Automated Framework for verifying AUTOSAR SW-C

- Fully automated framework
- Suited to continuous integration
- Works for most SW-C
- Improvements necessary for the most complex cases

Further works

- Performance improvements
  - Better memory model (scaling issues)
  - Explore other verification tools (Depth-K, ESBMC...)
- Prove more advanced properties
  - Higher level properties (functional verification)
  - Non-interference between SW-C

Thank you!