KU LEUVEN

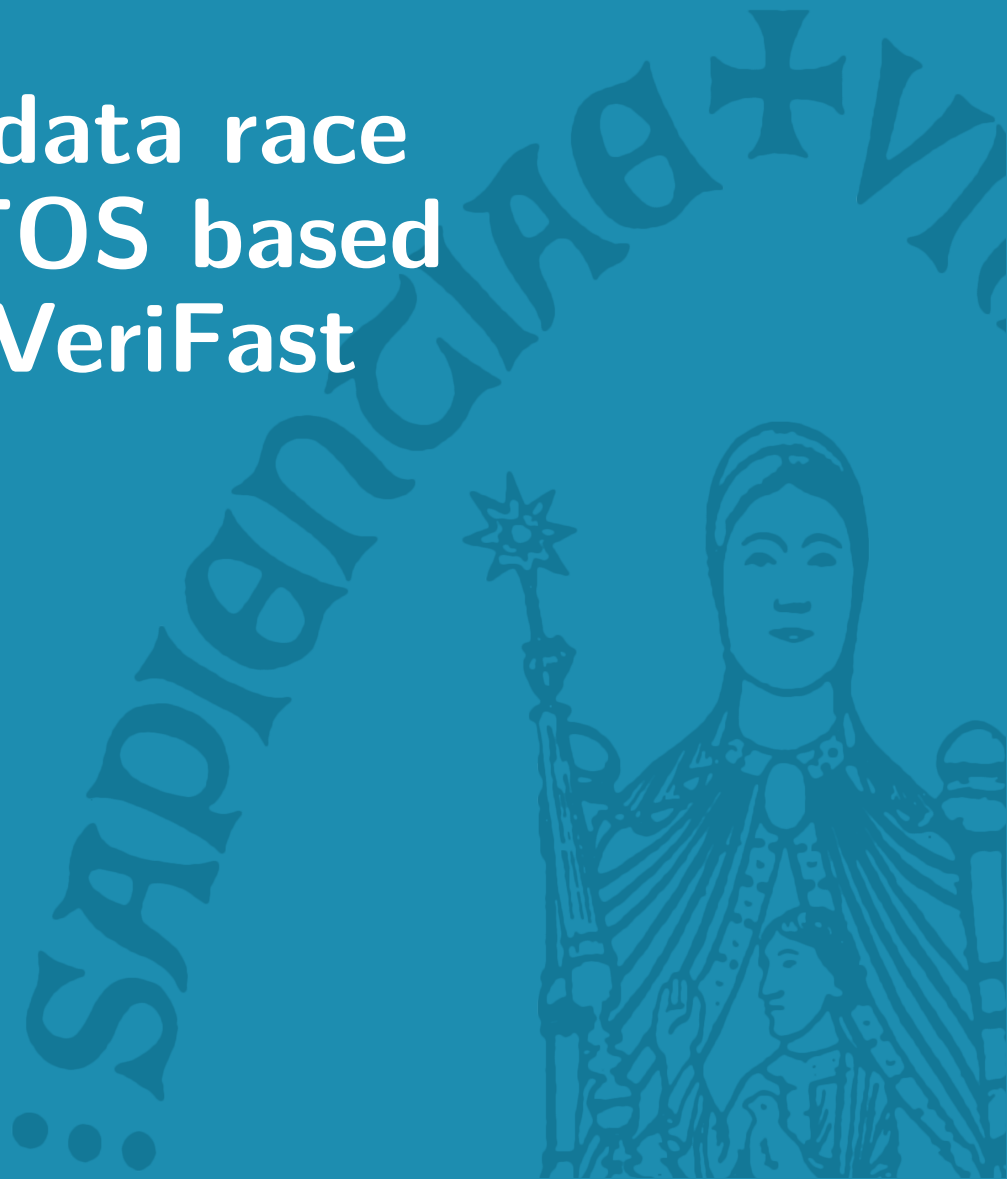
Formally verifying data race freedom of FreeRTOS based applications using VeriFast

Final defense

Kristof Achten

DistriNet - KU Leuven

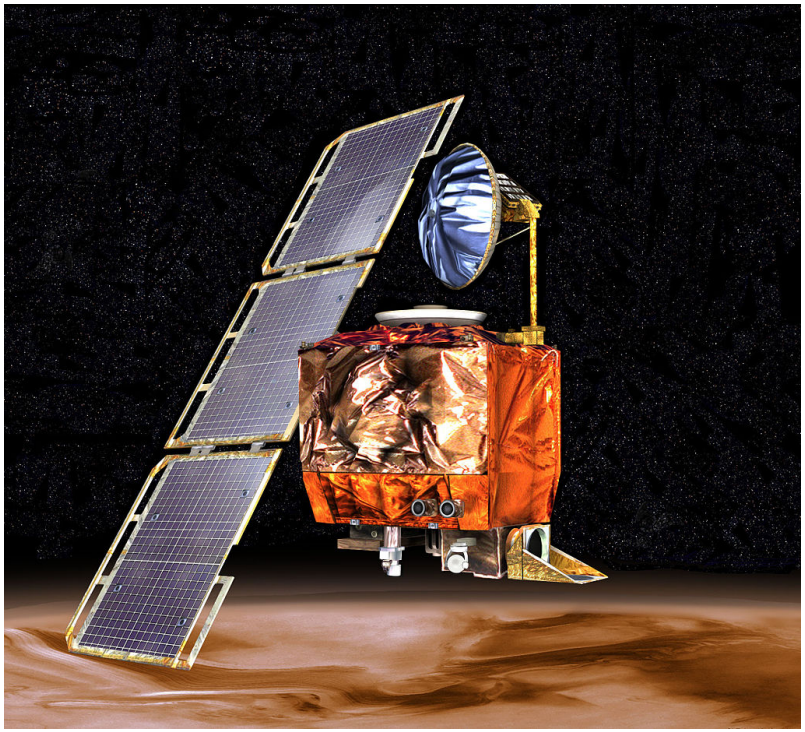
June 2019



1 Outline

1 The importance of testing

Mars Climate Orbiter crash



NASA/JPL/Corby Waste [Public domain]

1 The importance of testing

$\Rightarrow$ **Correct and thorough testing is important**

But how to do it well?

1 The importance of testing

"Program testing can be a very effective way to show the presence of bugs, but it is ***hopelessly inadequate for showing their absence****."*

– Edsger W. Dijkstra (1972)

1 Proving the absence of bugs

Formal software verification

- Prove absence instead of presence of bugs
- Intensive process $\Rightarrow$ special tools

1 Proving the absence of bugs

Formal software verification

- Prove absence instead of presence of bugs
- Intensive process $\Rightarrow$ special tools
- In general: two approaches
  1. **Model-checking approach:** exhaustive exploration
  2. **Deductive verification:** proof obligation generation

1 The Verification Grand Challenge

Challenge set out by Tony Hoare in 2003

- Automatisation of formal verification

1 The Verification Grand Challenge

Challenge set out by Tony Hoare in 2003

- Automatisation of formal verification
- Criteria for a verifying compiler[9,3]
  1. Unified framework using different methods
  2. Corpus of verified samples

1 The Verification Grand Challenge

Challenge set out by Tony Hoare in 2003

- Automatisation of formal verification
- Criteria for a verifying compiler[9,3]
  1. Unified framework using different methods
  2. Corpus of verified samples

$\Rightarrow$ **Feasibility is disputable**

1 The VeriFast Program Verifier

VeriFast[4,5] tool for static formal software verification in a modular fashion

- In development at KULeuven
- Based on separation logic on heap chunks

1 The VeriFast Program Verifier

VeriFast[4,5] tool for static formal software verification in a modular fashion

- In development at KULeuven
- Based on separation logic on heap chunks
- Focus on expressiveness and speed

1 The VeriFast Program Verifier

VeriFast[4,5] tool for static formal software verification in a modular fashion

- In development at KULeuven
- Based on separation logic on heap chunks
- Focus on expressiveness and speed
- Guarantees in terms of:
  - Memory safety
  - Concurrency
  - Functional correctness

1 The VeriFast Program Verifier: modularity

Method contracts

```
void foo()
    //@ requires <preconditions>
    //@ ensures <postconditions>
```

1 The VeriFast Program Verifier: modularity

Method contracts

```
void foo ()
    //@ requires <preconditions>
    //@ ensures <postconditions>
```

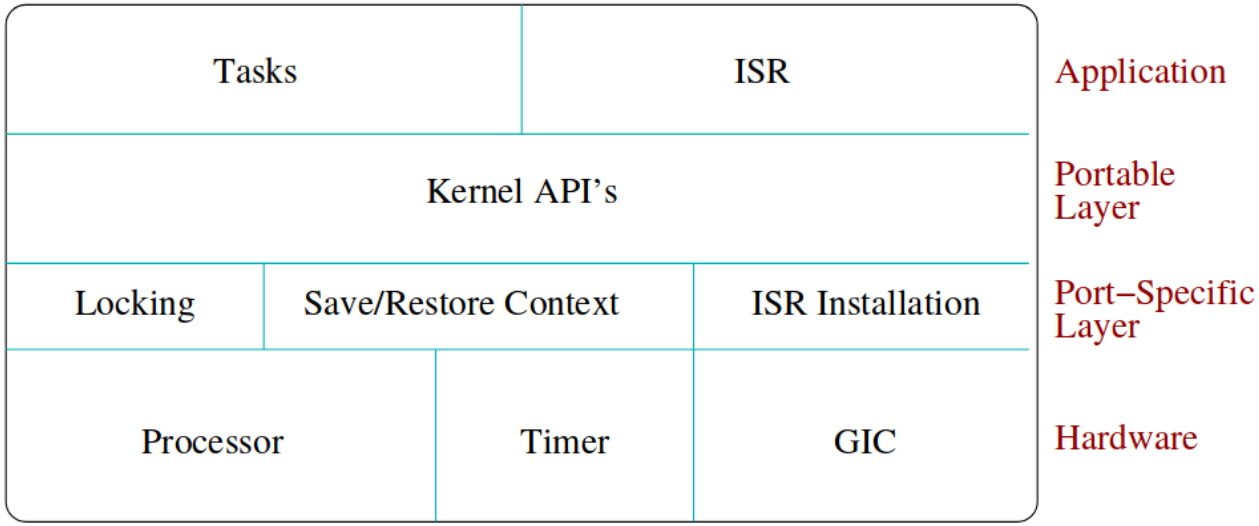
Function Simulation

```
void caller ()
    //@ requires <preconditions_caller>
    //@ ensures <postconditions_caller>
{
    call_callee();
        /* preconditions callee should hold here */
        /* postconditions callee are spawned in */
}
```

1 FreeRTOS

FreeRTOS[8] *Real-Time* operating system

- Lightweight
- Supports a broad range of micro-controllers



Source: Chandrasekaran et al. [2]

1 FreeRTOS: resource management

Two important concurrency management techniques:[6,1]

1 FreeRTOS: resource management

Two important concurrency management techniques:[6,1]

1. **Semaphores**: two flavours
   - *Binary semaphores*: used for synchronization, deferred interrupt processing
   - *Counting semaphores*: used for event counting and resource management.

   Implementation through queues

1 FreeRTOS: resource management

Two important concurrency management techniques:[6,1]

1. **Semaphores**: two flavours
   - *Binary semaphores*: used for synchronization, deferred interrupt processing
   - *Counting semaphores*: used for event counting and resource management.

   Implementation through queues

2. **Mutexes**: binary semaphores with notion of ownership
   - *Priority inversion*: higher priority task gets no CPU cycles
   - *Deadlock*: solved through timeouts

1 Research Questions

Thesis is twofold:

1. Can the VeriFast program verifier be used to guarantee data race freedom in a FreeRTOS based application that uses mutexes and binary semaphores to guard against concurrent resource accesses?

1 Research Questions

2 Is it possible to generalize the VeriFast specifications of FreeRTOS provided resource management constructs such as mutexes and binary semaphores to be used in the verification of other FreeRTOS based applications that use the same constructs?

1 Research Questions

2 Is it possible to generalize the VeriFast specifications of FreeRTOS provided resource management constructs such as mutexes and binary semaphores to be used in the verification of other FreeRTOS based applications that use the same constructs?

- Is there a change in VeriFast annotational overhead when comparing these generalizations to the specific verification performed in question 1?
- Does the generalization facilitate the addition of other resource management constructs such as counting semaphores and recursive mutexes?

2 Outline

2 Formal verification: a case study

Application provided by SmartNodes

- Smart city solutions $\Rightarrow$ smart lighting
- Gateway - sensor network
- Careful selection of functions for case study

Source: https:/www.smartnodes.be/ [7]

2 Formal verification: a case study

Tailoring the verification to the code

- Only binary semaphores and mutexes

2 Formal verification: a case study

Tailoring the verification to the code

- Only binary semaphores and mutexes
- Time slicing disabled
- Construct sharing on same priority level

2 Formal verification: a case study

Tailoring the verification to the code

- Only binary semaphores and mutexes
- Time slicing disabled
- Construct sharing on same priority level

$\Rightarrow$ **modelled with just one priority level**

2 Formal verification: a case study

1 Specifying the binary semaphore and mutex behaviour
   - Deadlock-freedom is a requirement

2 Formal verification: a case study

1. Specifying the binary semaphore and mutex behaviour
   - Deadlock-freedom is a requirement
   - macro expansion $\Rightarrow$ **queue API** specification
   - Introduction of VeriFast predicates
     1. `queue_pred`: underlying queue structure

2 Formal verification: a case study

1. Specifying the binary semaphore and mutex behaviour
   - Deadlock-freedom is a requirement
   - macro expansion $\Rightarrow$ **queue API** specification
   - Introduction of VeriFast predicates
     1. `queue_pred`: underlying queue structure
     2. `sync_pred`: coupling of queue to protected resource

2 Formal verification: a case study

2 Developing a mock-up client

- Four FreeRTOS tasks
- Closely mimic true behaviour
- Voluntary yielding $\Rightarrow$ progress
  - required basic **task API** specification

Task API

```
BaseType_t xTaskGenericCreate( ... ); // Creation
void vTaskDelay( ... );               // Yielding
```

2 Formal verification: a case study

3 Verification of the full application

2 Formal verification: a case study

Two issues detected

1. Correct initialization of constructs is never checked
   - $\Rightarrow$ Mitigation through explicit null-pointer check

2 Formal verification: a case study

Two issues detected

1. Correct initialization of constructs is never checked
   - $\Rightarrow$ Mitigation through explicit null-pointer check
2. Arithmetic under- and overflow
   - $\Rightarrow$ Mitigation through explicit bounding of variables

2 Formal verification: a case study

Two issues detected

1. Correct initialization of constructs is never checked
   - $\Rightarrow$ Mitigation through explicit null-pointer check
2. Arithmetic under- and overflow
   - $\Rightarrow$ Mitigation through explicit bounding of variables

Issues will not form a realistic threat

3 Outline

3 Contributing to the verifying compiler

Specification has low reusability

1. Strong prerequisites: same priority construct interaction only

Solution: include queue predicates in sync predicates

3 Contributing to the verifying compiler

Specification has low reusability

1. Strong prerequisites: same priority construct interaction only
2. Queues have fixed semantics

Solution: include queue predicates in sync predicates

3 Contributing to the verifying compiler

Specification has low reusability

1. Strong prerequisites: same priority construct interaction only
2. Queues have fixed semantics
   - Differently used queues $\Rightarrow$ unsound verification

Solution: include queue predicates in sync predicates

3 Contributing to the verifying compiler

Specification has low reusability

1. Strong prerequisites: same priority construct interaction only
2. Queues have fixed semantics
   - Differently used queues $\Rightarrow$ unsound verification
3. Loose coupling between predicates

Solution: include queue predicates in sync predicates

3 Contributing to the verifying compiler

Specification has low reusability

1. Strong prerequisites: same priority construct interaction only
2. Queues have fixed semantics
   - Differently used queues $\Rightarrow$ unsound verification
3. Loose coupling between predicates
   - `sync_pred` and `queue_pred` together represent constructs

Solution: include queue predicates in sync predicates

3 Contributing to the verifying compiler

Mutex predicate

```
predicate mutex_pred(queue, resource, holder, state) =
    queue_pred(queue, MUTEX_TYPE, 1, state);
```

Binary semaphore predicate

```
predicate binsem_pred(queue, invar, state) =
    queue_pred(queue, BINSEM_TYPE, 1, state);
```

3 Contributing to the verifying compiler

Generalization allows for easy addition of other constructs

Counting semaphore predicate

```
predicate countingsem_pred(queue, invar, size, state) =
    queue_pred(queue, COUNTSEM_TYPE, size, state);
```

3 Contributing to the verifying compiler

Generalized specifications are larger

- Function sharing ⇒ switch modelling
- Additional construct = extra branch in switch

Switch modelling

```
BaseType_t xQueueGenericSend( ... );
    /*@ requires
          type == MUTEX_TYPE ?
            /* Mutex specific trace */
          :
            (type == BINSEM_TYPE ?
              /* Binary semaphore specific trace */
            :
              ...
    @*/
```

4 Outline

4 Discussion: comparing the specifications

Annotational overhead: original vs. generalizations

- Decrease of overhead in the application

4 Discussion: comparing the specifications

Annotational overhead: original vs. generalizations

- Decrease of overhead in the application
- Increase of overhead in the construct specifications

4 Discussion: comparing the specifications

Annotational overhead: original vs. generalizations

- Decrease of overhead in the application
- Increase of overhead in the construct specifications

  $\Rightarrow$ not relevant - reused anyway

4 Discussion: difficulties

Some difficulties were encountered:

- No prior experience with VeriFast
- No prior knowledge of FreeRTOS
- No prior knowledge of the case study

4 Discussion: difficulties

Some difficulties were encountered:

- No prior experience with VeriFast
- No prior knowledge of FreeRTOS
- No prior knowledge of the case study

$\Rightarrow$ Lots of research

4 Discussion: difficulties

VeriFast has limited C support

- Union types, constant pointers, ...
- Function attributes
- Low-level assembly code

4 Discussion: difficulties

VeriFast has limited C support

- Union types, constant pointers, ...
- Function attributes
- Low-level assembly code

$\Rightarrow$ Modifications had to be made without semantic loss

4 Discussion: future work

Some challenges remain

- Formal proofs of the specifications (E.g.: using Coq)

4 Discussion: future work

Some challenges remain

- Formal proofs of the specifications (E.g.: using Coq)
- Verification of the full FreeRTOS scheduler

4 Discussion: future work

Some challenges remain

- Formal proofs of the specifications (E.g.: using Coq)
- Verification of the full FreeRTOS scheduler
  - $\Rightarrow$ Can VeriFast be an adequate tool for this?

4 Discussion: future work

Some challenges remain

- Formal proofs of the specifications (E.g.: using Coq)
- Verification of the full FreeRTOS scheduler
  - $\Rightarrow$ Can VeriFast be an adequate tool for this?
- Recursive mutexes

5 Outline

5 Conclusion: answering the research questions

1. Can the VeriFast program verifier be used to guarantee data race freedom in a FreeRTOS based application that uses mutexes and binary semaphores to guard against concurrent resource accesses?

5 Conclusion: answering the research questions

1. Can the VeriFast program verifier be used to guarantee data race freedom in a FreeRTOS based application that uses mutexes and binary semaphores to guard against concurrent resource accesses?

Yes, but a verification of the FreeRTOS structures such as the scheduler remains important

5 Conclusion: answering the research questions

2 Is it possible to generalize the VeriFast specifications of FreeRTOS provided resource management constructs such as mutexes and binary semaphores to be used in the verification of other FreeRTOS based applications that use the same constructs?

5 Conclusion: answering the research questions

2 Is it possible to generalize the VeriFast specifications of FreeRTOS provided resource management constructs such as mutexes and binary semaphores to be used in the verification of other FreeRTOS based applications that use the same constructs?

Yes, but some semantic loss was noted

5 Conclusion: answering the research questions

- Is there a change in VeriFast annotational overhead when comparing these generalizations to the specific verification performed in question 1?

5 Conclusion: answering the research questions

- Is there a change in VeriFast annotational overhead when comparing these generalizations to the specific verification performed in question 1?
  - $\Rightarrow$ **Lower for the application, higher for the services**

5 Conclusion: answering the research questions

- Is there a change in VeriFast annotational overhead when comparing these generalizations to the specific verification performed in question 1?
  - $\Rightarrow$ **Lower for the application, higher for the services**
- Does the generalization facilitate the addition of other resource management constructs such as counting semaphores and recursive mutexes?

5 Conclusion: answering the research questions

- Is there a change in VeriFast annotational overhead when comparing these generalizations to the specific verification performed in question 1?
  - $\Rightarrow$ **Lower for the application, higher for the services**
- Does the generalization facilitate the addition of other resource management constructs such as counting semaphores and recursive mutexes?
  - $\Rightarrow$ **Yes, as shown by the addition of counting semaphores. Recursive mutexes remain tricky.**

Thank you for your attention!

Questions?

6 References I

[1] R. Barry. *Mastering the FreeRTOS Real Time Kernel: A Hands-On Tutorial Guide*. Pre-release 161204. 2016.

[2] P. Chandrasekaran et al. "A multi-core version of FreeRTOS verified for datarace and deadlock freedom". In: *2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE)*. Oct. 2014, pp. 62–71. DOI: 10.1109/MEMCOD.2014.6961844.

[3] Tony Hoare. "The Verifying Compiler: A Grand Challenge for Computing Research". In: *Compiler Construction*. Ed. by Görel Hedin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 262–272. ISBN: 978-3-540-36579-2.

[4] B. Jacobs and F. Piessens. *The VeriFast Program Verifier*. Tech. rep. Department of Computer Science, Katholieke Universiteit Leuven, Belgium, Aug. 2008.

6 References II

[5] B. Jacobs, J. Smans, and F. Piessens. *A Quick Tour of the VeriFast Program Verifier*. Tech. rep. Department of Computer Science, Katholieke Universiteit Leuven, Belgium, Nov. 2010.

[6] Amazon Web Services. *The FreeRTOS Reference Manual: API Functions and Configuration Options*. 10.0.0 issue 1. 2017.

[7] *SmartNodes: Smart Cities through Smart Lighting*. https://www.smartnodes.be/.

[8] *The FreeRTOS Kernel*. https://www.freertos.org/.

[9] Jim Woodcock and Richard Banach. "The Verification Grand Challenge". In: *Journal of Universal Computer Science* 13.5 (May 28, 2007). http://www.jucs.org/jucs_13_5/the_verification_grand_challenge, pp. 661–668.