On the verification of autonomous systems and the role of real-time research

Bjorn Andersson and Dionisio de Niz

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM19-0589

Outline

1. Background
2. Some ongoing/previous work
3. Statement of open problem

Why Troops Don't Trust Drones

17 drone disasters that show why the FAA hates drones

White House Drone Crash Described as a U.S. Worker's Drunken Lark

Boeing 737 Max: Software patches can only do so much

Systems architects, engineers, and management can all learn from the history of the development of this complex aircraft.

RQ-4B GLOBAL HAWK ACCIDENT INVESTIGATION RELEASED

Drone Crash in Iran Reveals Secret U.S. Surveillance Effort

Robotic surgery linked to 144 deaths in the US

Death by robot: the new mechanised danger in our changing world

As the use of autonomous machines increases in society, so too has the chance of robot-related fatalities

Faulty pacemakers 'killing 2,000 a year': Third of unexpected deaths among patients thought to be caused by malfunctions

ROBOT CANNON KILLS 9, WOUNDS 14

Robot 'goes rogue and kills woman on Michigan car parts production line'

The value that autonomous systems generate to society is limited by their lack of safety.

How to achieve safety in autonomous systems?

Verifying Autonomous Systems (1)

Non-Scalable Verification (Logical Verification)

- Limit what we can verify

Verification does not account for temporal failures

- Communication, sensors
- Traditional fault-tolerant approach leads to unsatisfactory guarantees

Verification ignores adaptation

- To changing unpredictable environment

Verification assumes design-time deterministic decisions

- Ignores runtime learning / environment determined adaptation

Verifying Autonomous Systems (2)

Multi-Domain (traditionally performed independently)

- Logic
- Timing
- Control

Conflicting Cultures and Infrastructure

- Avionics, Space, Embedded
  - RTOS
  - simple static component interactions
- Silicon Valley, Machine Learning, TensorFlow
  - Linux
  - rich dynamic component interactions

Verifying Adapting (Autonomous) Systems

Guarantees changes as system adapts
- Dynamic architecture (examples)
- Different guarantees as configurations evolve

Failures should compromise performance **not** safety
- Trigger adaptation to tradeoff performance for safety

Select what to verify
- Scalable verification: reduce verified components
- Non-verified components must be monitored and enforced
- Protect verified components from unverified misbehavior

Harmonize verification domains
- Different verification domains abstract information about other domains
  - Valid under some static configurations
- Build new abstractions for dynamic architectures
  - Valid across reconfigurations
  - Support reconfiguration needs across verification domains

Need to Use Unverified Components

Scalability

- The less components verified the more scalable

Runtime learning

- Behavior changes at runtime => cannot verify at design time

Security

- A security attack may change behavior of component (e.g., code modification) at runtime

Verification in the Presence of Unverified Components

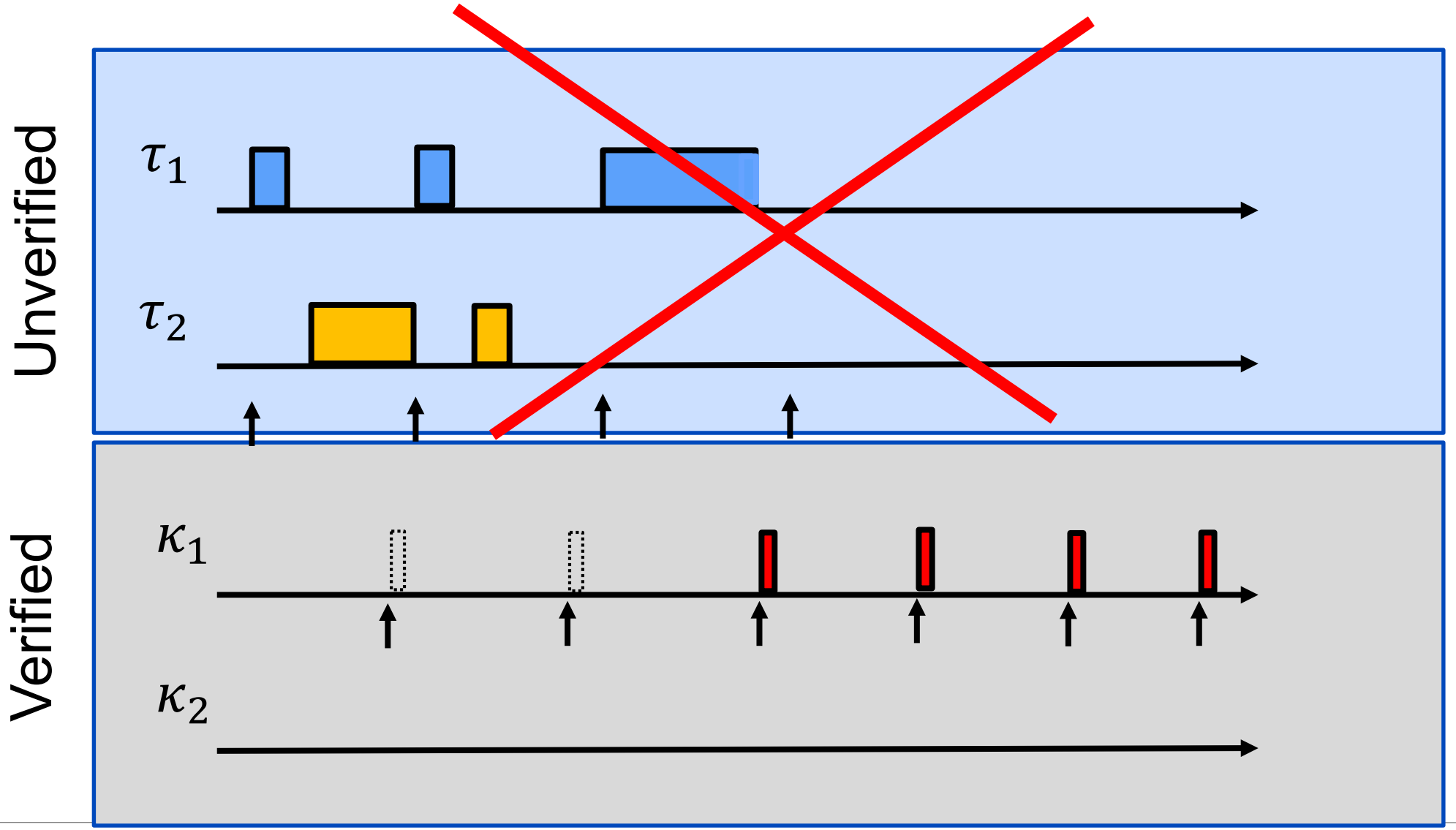
Verified component can be compromised by unverified ones

Preserving verified components guarantees

- Protection

TRUST = VERIFICATION + PROTECTION

Scheduling in Two Protection Domains



Adaptation in Different Verification Domains

Logic

- Runtime assurance:
  - Enforce safety of unverified component behavior
  - e.g., replace unsafe output

Control

- Guard complex controller (e.g., ML)
- Keep system safe and stable in case of misbehavior

Timing

- Ensure minimum functionality completed by deadline
- Accommodate control assumptions
- Guarantee execution timing across different protection domains

Adaptation Verification

Guarantees across adaptations

- Safety
  - Safe stop
- Prevent stopping mission (guarantee?)
  - Safe stop last resort
  - Anticipate to prevent safe stop as the only option

Performance across adaptations

- Stability
- Mission time

Real-Time For Autonomous Systems

Adaptable guarantees

Scheduling across protection domains

Models for other domain adaptations

Guarantees that do not compromise mission performance

Real-Time Mixed-Trust Computing
First Step

Support untrusted components

- Guarded by trusted ones
- Protecting trusted ones

Scheduling across protection domains

- Untrusted domain
  - Guarantees in absence of faults
- Verified trusted domain
  - Tamper-proof safe guarantees (fall-back)

Trusted scheduling coordination

- Untrusted domain failure does not compromise trusted domain

Open Questions

Task model

- What is a good task model for software in (Cyber-Physical) autonomous systems?

Run-time system

- What is a good run-time system for software in (Cyber-Physical) autonomous systems?

Task-parameter extraction

- What is a good method for obtaining task parameters (e.g., execution time) for software in (Cyber-Physical) autonomous systems?

Schedulability test

- What is a good schedulability test for software in (Cyber-Physical) autonomous systems?

Analysis contract

- What is a good analysis contract between different verification domains (logic, timing, control) for software in (Cyber-Physical) autonomous systems?

THANKS