Explainable Restart-Behaviour of Reactive Systems Software

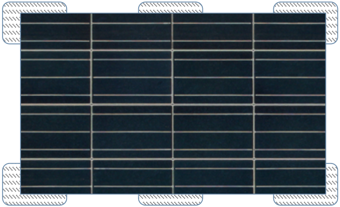
Dimitri Bohlender | Stefan Kowalewski

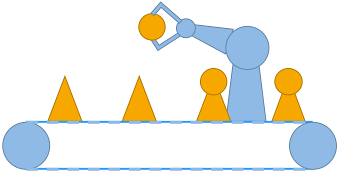
ES4CPS, GI-Dagstuhl Seminar, 7 Jan 2019

Informatik 11
Embedded Software
RWTH AACHEN UNIVERSITY

The Setting

- Control software is at the heart of many complex systems
- May be distributed over one to several controllers



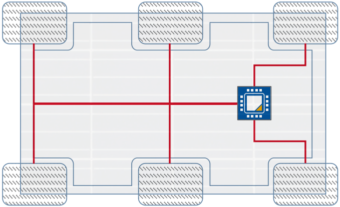


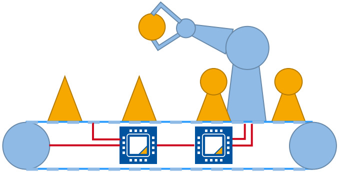
- Systems must meet high safety and reliability requirements

⇒ Correct control software is an integral part

The Setting

- Control software is at the heart of many complex systems
- May be distributed over one to several controllers



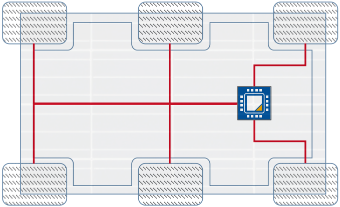


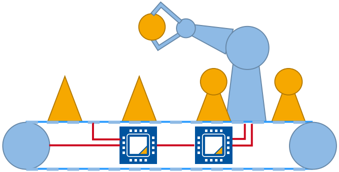
- Systems must meet high safety and reliability requirements

⇒ Correct control software is an integral part

The Setting

- Control software is at the heart of many complex systems
- May be distributed over one to several controllers



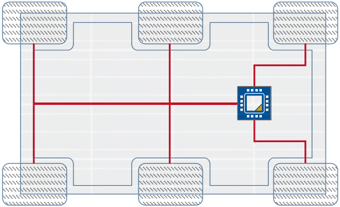


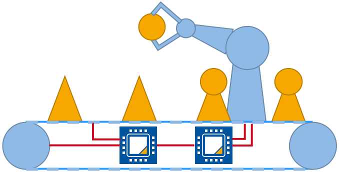
- Systems must meet high safety and reliability requirements

⇒ Correct control software is an integral part

The Setting

- Control software is at the heart of many complex systems
- May be distributed over one to several controllers





- Systems must meet high safety and reliability requirements

⇒ Correct control software is an integral part

Formal Verification

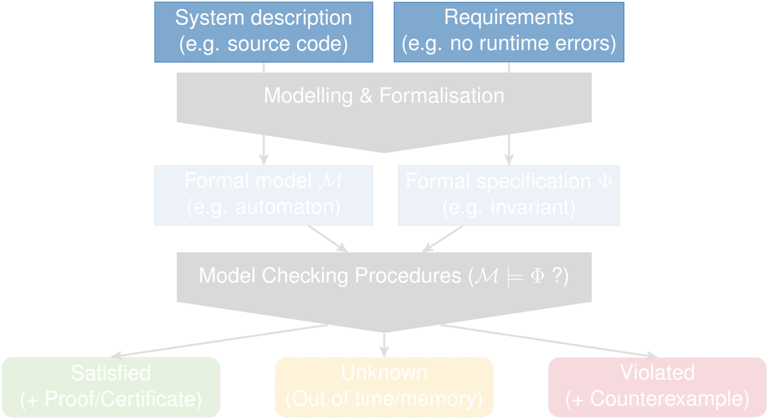
- Testing is under-approximative and gives no guarantees
- Formal verification can (dis-)prove properties of interest

Formal Verification

- Testing is under-approximative and gives no guarantees
- Formal verification can (dis-)prove properties of interest

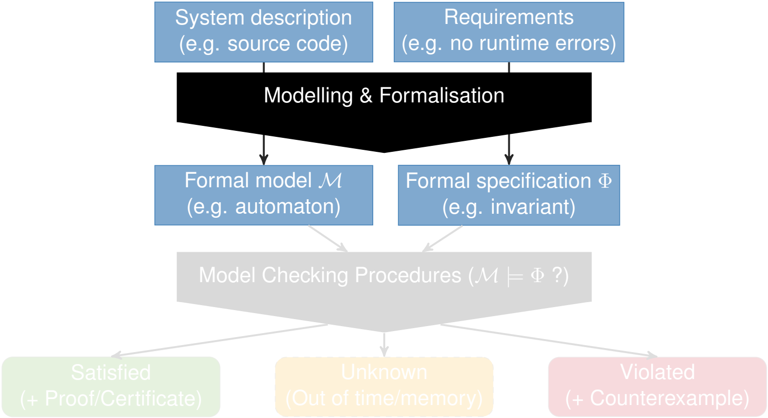
Formal Verification

- Testing is under-approximative and gives no guarantees
- Formal verification can (dis-)prove properties of interest



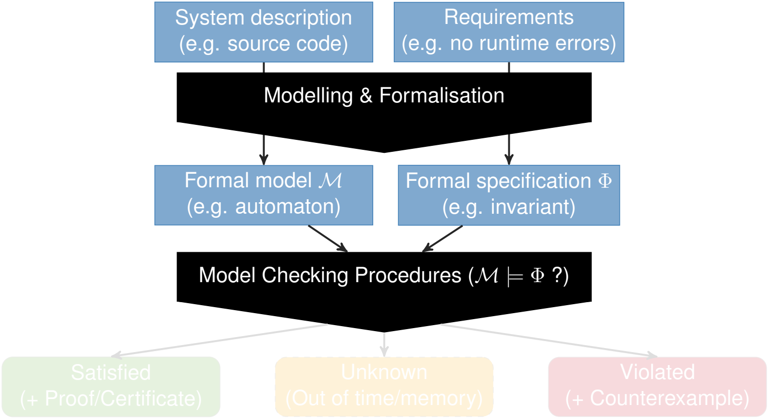
Formal Verification

- Testing is under-approximative and gives no guarantees
- Formal verification can (dis-)prove properties of interest



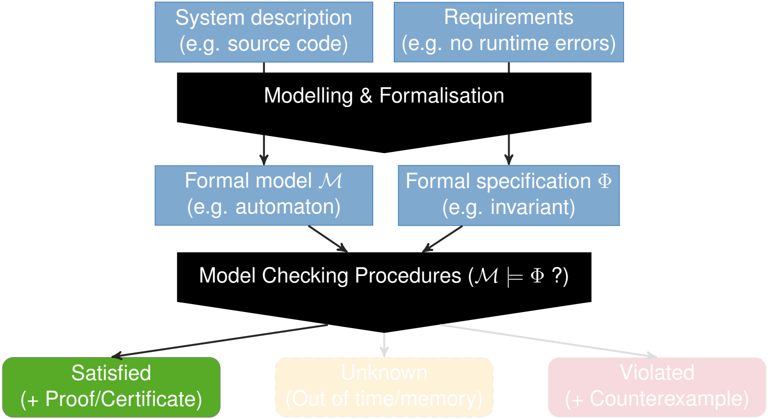
Formal Verification

- Testing is under-approximative and gives no guarantees
- Formal verification can (dis-)prove properties of interest



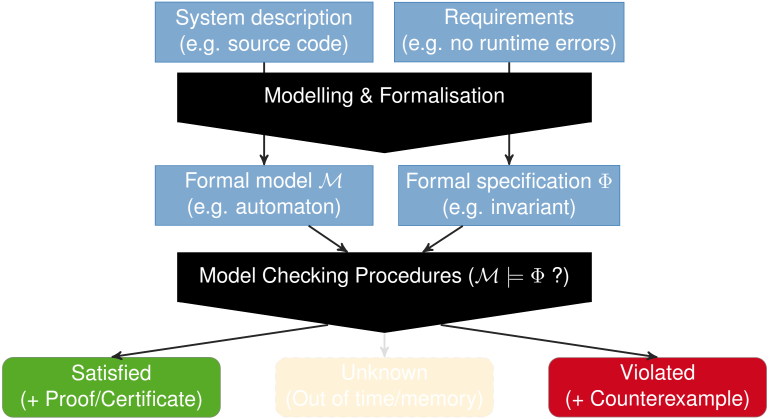
Formal Verification

- Testing is under-approximative and gives no guarantees
- Formal verification can (dis-)prove properties of interest



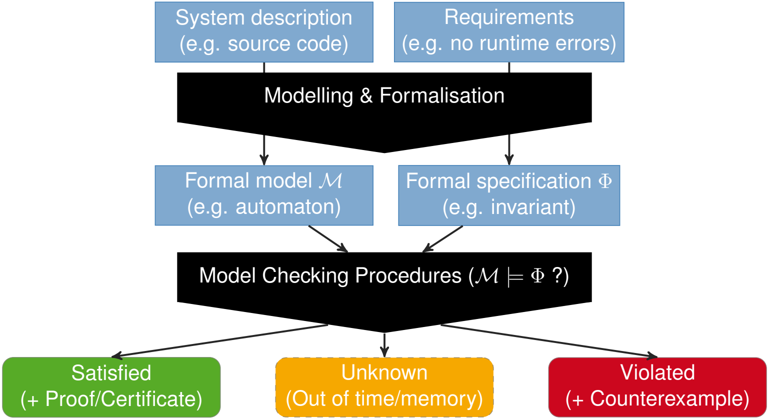
Formal Verification

- Testing is under-approximative and gives no guarantees
- Formal verification can (dis-)prove properties of interest



Formal Verification

- Testing is under-approximative and gives no guarantees
- Formal verification can (dis-)prove properties of interest



Restart-Behaviour

However:

- Such proof holds w.r.t. the model – not the real system

⇒ Model is usually missing behaviour enabled by hardware

Battery-backed memory & restart-functionality are widespread

- Non-volatile state variables allow for "restart-robust" designs
- Restarts may be
  - triggered by a watchdog timer
  - the result of a power outage or voltage fluctuation
  - triggered manually, e.g. during testing

Restart-Behaviour

However:

- Such proof holds w.r.t. the model – not the real system

⇒ Model is usually missing behaviour enabled by hardware

Battery-backed memory & restart-functionality are widespread

- Non-volatile state variables allow for "restart-robust" designs
- Restarts may be
  - triggered by a watchdog timer
  - the result of a power-outage or voltage fluctuation
  - triggered manually, e.g. during testing

Restart-Behaviour

However:

- Such proof holds w.r.t. the model – not the real system

⇒ Model is usually missing behaviour enabled by hardware

Battery-backed memory & restart-functionality are widespread

- Non-volatile state variables allow for "restart-robust" designs
- Restarts may be
  - triggered by a watchdog timer
  - the result of a power outage or voltage fluctuation
  - triggered manually, e.g. during testing

Restart-Behaviour

However:

- Such proof holds w.r.t. the model – not the real system

⇒ Model is usually missing behaviour enabled by hardware

Battery-backed memory & restart-functionality are widespread

- Non-volatile state variables allow for "restart-robust" designs
- Restarts may be
  - triggered by a watchdog timer
  - the result of a power outage or voltage fluctuation
  - triggered manually, e.g. during testing

Restart-Behaviour

However:

- Such proof holds w.r.t. the model – not the real system

⇒ Model is usually missing behaviour enabled by hardware

Battery-backed memory & restart-functionality are widespread

- Non-volatile state variables allow for “restart-robust” designs
- Restarts may be
  - triggered by a watchdog timer
  - the result of a power outage or voltage fluctuation
  - triggered manually, e.g. during testing

Restart-Behaviour

However:

- Such proof holds w.r.t. the model – not the real system

⇒ Model is usually missing behaviour enabled by hardware

Battery-backed memory & restart-functionality are widespread

- Non-volatile state variables allow for “restart-robust” designs
- Restarts may be
  - triggered by a watchdog timer
  - the result of a power outage or voltage fluctuation
  - triggered manually, e.g. during testing

Managing Restart-Behaviour

- Restarts significantly increase the number of corner cases
- Different semantics of writing to battery-backed memory exist
- Choice of retain-variables left to developer

⇒ Difficult to reason about manually and explain defects

Example (Automated drilling of holes in workpieces)

- Let drill's position be volatile
- A restart may result in unintended movement and damage
- Even though restart-free operation might be as expected

Managing Restart-Behaviour

- Restarts significantly increase the number of corner cases
- Different semantics of writing to battery-backed memory exist
- Choice of retain-variables left to developer

⇒ Difficult to reason about manually and explain defects

Example (Automated drilling of holes in workpieces)

- Let drill's position be volatile
- A restart may result in unintended movement and damage
- Even though restart-free operation might be as expected

Managing Restart-Behaviour

- Restarts significantly increase the number of corner cases
- Different semantics of writing to battery-backed memory exist
- Choice of retain-variables left to developer

⇒ Difficult to reason about manually and explain defects

Example (Automated drilling of holes in workpieces)

- Let drill's position be volatile
- A restart may result in unintended movement and damage
- Even though restart-free operation might be as expected

Managing Restart-Behaviour

- Restarts significantly increase the number of corner cases
- Different semantics of writing to battery-backed memory exist
- Choice of retain-variables left to developer

⇒ Difficult to reason about manually and explain defects

Example (Automated drilling of holes in workpieces)

- Let drill's position be volatile
- A restart may result in unintended movement and damage
- Even though restart-free operation might be as expected

Managing Restart-Behaviour

- Restarts significantly increase the number of corner cases
- Different semantics of writing to battery-backed memory exist
- Choice of retain-variables left to developer

⇒ Difficult to reason about manually and explain defects

Example (Automated drilling of holes in workpieces)

- Let drill's position be volatile
- A restart may result in unintended movement and damage
- Even though restart-free operation might be as expected

Managing Restart-Behaviour

- Restarts significantly increase the number of corner cases
- Different semantics of writing to battery-backed memory exist
- Choice of retain-variables left to developer

$\Rightarrow$ Difficult to reason about manually and explain defects

Example (Automated drilling of holes in workpieces)

- Let drill's position be volatile
- A restart may result in unintended movement and damage
- Even though restart-free operation might be as expected

Managing Restart-Behaviour

- Restarts significantly increase the number of corner cases
- Different semantics of writing to battery-backed memory exist
- Choice of retain-variables left to developer

⇒ Difficult to reason about manually and explain defects

Example (Automated drilling of holes in workpieces)

- Let drill's position be volatile
- A restart may result in unintended movement and damage
- Even though restart-free operation might be as expected

Informatik 11 Embedded Software
RWTH AACHEN UNIVERSITY

PLC Software Verification

- We work on ARCADE.PLC, a framework for
  - Static Analysis
  - Model Checking
  - Testing / Test-Generation

- Errors that only occur after restart are a common problem in industrial control code[1]

⇒ We developed procedures considering restart-robustness w.r.t. a specification, i.e. compliance in the context of restarts[2]

---

[1] Stefan Hauck-Stattelmann et al. "Analyzing the Restart Behavior of Industrial Control Applications". In: *FM 2015*. 2015, pp. 585–588.

[2] Dimitri Bohlender and Stefan Kowalewski. "Design and Verification of Restart-Robust Industrial Control Software". In: *IFM 2018*. 2018, pp. 47–68.

PLC Software Verification

- We work on ARCADE.PLC, a framework for
  - Static Analysis
  - Model Checking
  - Testing / Test-Generation

- Errors that only occur after restart are a common problem in industrial control code[1]

⇒ We developed procedures considering restart-robustness w.r.t. a specification, i.e. compliance in the context of restarts[2]

[1] Stefan Hauck-Stattelmann et al. "Analyzing the Restart Behavior of Industrial Control Applications". In: *FM 2015*. 2015, pp. 585–588.

[2] Dimitri Bohlender and Stefan Kowalewski. "Design and Verification of Restart-Robust Industrial Control Software". In: *IFM 2018*. 2018, pp. 47–68.

PLC Software Verification

- We work on ARCADE.PLC, a framework for
  - Static Analysis
  - Model Checking
  - Testing / Test-Generation

- Errors that only occur after restart are a common problem in industrial control code[1]

⇒ We developed procedures considering restart-robustness w.r.t. a specification, i.e. compliance in the context of restarts[2]

[1] Stefan Hauck-Stattelmann et al. “Analyzing the Restart Behavior of Industrial Control Applications”. In: *FM 2015*. 2015, pp. 585–588.

[2] Dimitri Bohlender and Stefan Kowalewski. “Design and Verification of Restart-Robust Industrial Control Software”. In: *IFM 2018*. 2018, pp. 47–68.

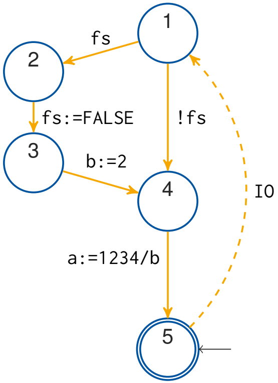
Toy Example: Invariant $a \geq 0$

- Initially $fs \mapsto true, a \mapsto 0, b \mapsto 0$
- Nominal behaviour compliant?

In context of restarts

- Let the flag `fs` be retained
- Robust with delayed[a] writes?
- Fixable for delayed writes?
- Robust with immediate writes?
- Fixable for immediate writes?

[a] until the end of the execution cycle



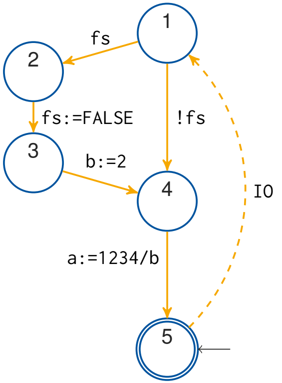
Toy Example: Invariant $a \geq 0$

- Initially $fs \mapsto true, a \mapsto 0, b \mapsto 0$
- Nominal behaviour compliant?

In context of restarts

- Let the flag `fs` be retained
- Robust with delayed[a] writes?
- Fixable for delayed writes?
- Robust with immediate writes?
- Fixable for immediate writes?

[a] until the end of the execution cycle



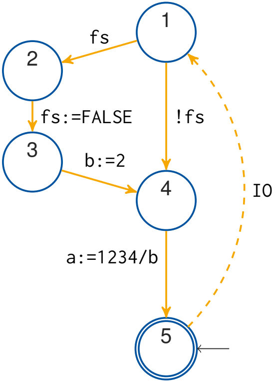
Toy Example: Invariant $a \geq 0$

- Initially $fs \mapsto true, a \mapsto 0, b \mapsto 0$
- Nominal behaviour compliant?

In context of restarts

- Let the flag `fs` be retained
- Robust with delayed[a] writes?
- Fixable for delayed writes?
- Robust with immediate writes?
- Fixable for immediate writes?

[a] until the end of the execution cycle



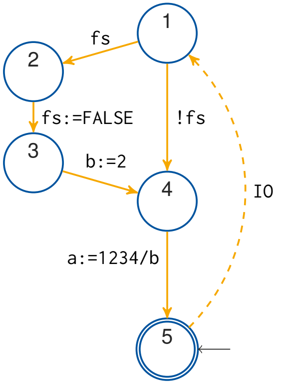
Toy Example: Invariant $a \geq 0$

- Initially $fs \mapsto true, a \mapsto 0, b \mapsto 0$
- Nominal behaviour compliant? ✓

In context of restarts

- Let the flag `fs` be retained
- Robust with delayed[a] writes?
- Fixable for delayed writes?
- Robust with immediate writes?
- Fixable for immediate writes?

[a] until the end of the execution cycle



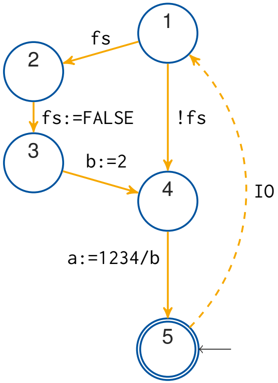
Toy Example: Invariant $a \geq 0$

- Initially $fs \mapsto true, a \mapsto 0, b \mapsto 0$
- Nominal behaviour compliant? ✓

In context of restarts

- Let the flag `fs` be retained
- Robust with delayed[a] writes?
- Fixable for delayed writes?
- Robust with immediate writes?
- Fixable for immediate writes?

[a] until the end of the execution cycle



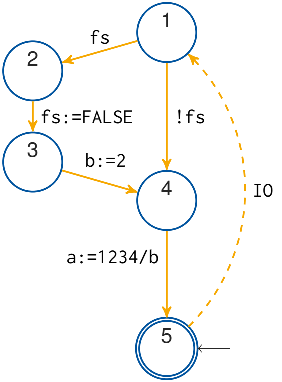
Toy Example: Invariant $a \geq 0$

- Initially $fs \mapsto true, a \mapsto 0, b \mapsto 0$
- Nominal behaviour compliant? ✓

In context of restarts

- Let the flag `fs` be retained
- Robust with delayed[a] writes?
- Fixable for delayed writes?
- Robust with immediate writes?
- Fixable for immediate writes?

[a] until the end of the execution cycle



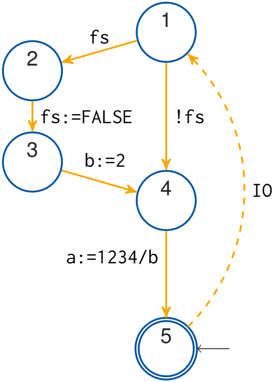
Toy Example: Invariant $a \geq 0$

- Initially $fs \mapsto true, a \mapsto 0, b \mapsto 0$
- Nominal behaviour compliant? ✓

In context of restarts

- Let the flag `fs` be retained
- Robust with delayed[a] writes? `a:=1234/0`
- Fixable for delayed writes?
- Robust with immediate writes?
- Fixable for immediate writes?

[a] until the end of the execution cycle



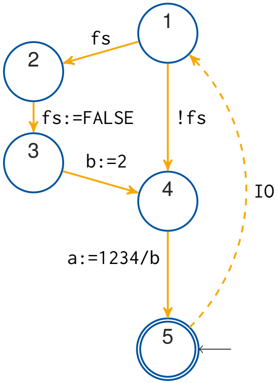
Toy Example: Invariant $a \geq 0$

- Initially $fs \mapsto true, a \mapsto 0, b \mapsto 0$
- Nominal behaviour compliant? ✓

In context of restarts

- Let the flag `fs` be retained
- Robust with delayed[a] writes? `a:=1234/0`
- Fixable for delayed writes?
- Robust with immediate writes?
- Fixable for immediate writes?

[a] until the end of the execution cycle



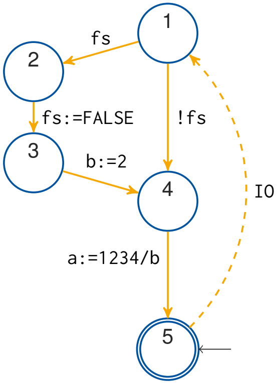
Toy Example: Invariant $a \geq 0$

- Initially $fs \mapsto true, a \mapsto 0, b \mapsto 0$
- Nominal behaviour compliant? ✓

In context of restarts

- Let the flag `fs` be retained
- Robust with delayed[a] writes? `a:=1234/0`
- Fixable for delayed writes? Retain b
- Robust with immediate writes?
- Fixable for immediate writes?

[a]until the end of the execution cycle



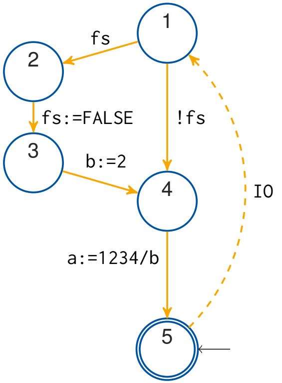
Toy Example: Invariant $a \geq 0$

- Initially $fs \mapsto true, a \mapsto 0, b \mapsto 0$
- Nominal behaviour compliant? ✓

In context of restarts

- Let the flag `fs` be retained
- Robust with delayed[a] writes? `a:=1234/0`
- Fixable for delayed writes? Retain b
- Robust with immediate writes?
- Fixable for immediate writes?

[a] until the end of the execution cycle



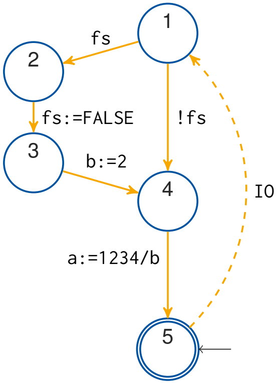
Toy Example: Invariant $a \geq 0$

- Initially $fs \mapsto true, a \mapsto 0, b \mapsto 0$
- Nominal behaviour compliant? ✓

In context of restarts

- Let the flag `fs` be retained
- Robust with delayed[a] writes? `a:=1234/0`
- Fixable for delayed writes? Retain b
- Robust with immediate writes? ✗
- Fixable for immediate writes?

[a] until the end of the execution cycle



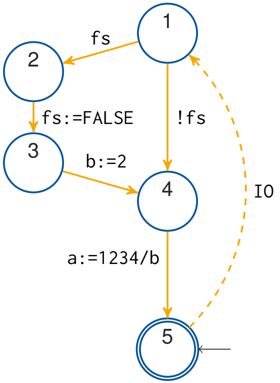
Toy Example: Invariant $a \geq 0$

- Initially $fs \mapsto true, a \mapsto 0, b \mapsto 0$
- Nominal behaviour compliant? ✓

In context of restarts

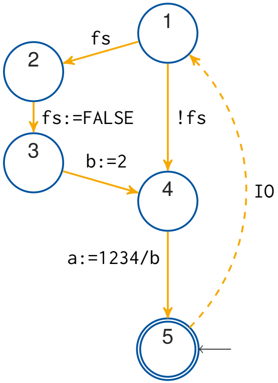
- Let the flag `fs` be retained
- Robust with delayed[a] writes? `a:=1234/0`
- Fixable for delayed writes? Retain b
- Robust with immediate writes? ✗
- Fixable for immediate writes?

[a]until the end of the execution cycle



Toy Example: Invariant $a \geq 0$

- Initially $fs \mapsto true, a \mapsto 0, b \mapsto 0$
- Nominal behaviour compliant? ✓



In context of restarts

- Let the flag `fs` be retained
- Robust with delayed[a] writes? `a:=1234/0`
- Fixable for delayed writes? Retain b
- Robust with immediate writes? ✗
- Fixable for immediate writes? ✗

[a] until the end of the execution cycle

Central Question

- We considered restart-robustness w.r.t. specifications

⇒ Require existence of specifications and checking all of them

- In practice, software is often under-specified

⇒ Many (mis-)behaviours not considered by verifier

Idea Characterise restart-robustness relational property between the nominal and the restart-augmented program behaviour

"What is a reasonable definition?"

- Insights from automation and systems engineering needed

Central Question

- ▶ We considered restart-robustness w.r.t. specifications
- ⇒ Require existence of specifications and checking all of them
- ▶ In practice, software is often under-specified
- ⇒ Many (mis-)behaviours not considered by verifier

Idea Characterise restart-robustness relational property between the nominal and the restart-augmented program behaviour

"What is a reasonable definition?"

- ▶ Insights from automation and systems engineering needed

Central Question

- We considered restart-robustness w.r.t. specifications

⇒ Require existence of specifications and checking all of them

- In practice, software is often under-specified

⇒ Many (mis-)behaviours not considered by verifier

Idea Characterise restart-robustness relational property between the nominal and the restart-augmented program behaviour

"What is a reasonable definition?"

- Insights from automation and systems engineering needed

Central Question

- We considered restart-robustness w.r.t. specifications

⇒ Require existence of specifications and checking all of them

- In practice, software is often under-specified

⇒ Many (mis-)behaviours not considered by verifier

Idea Characterise restart-robustness relational property between the nominal and the restart-augmented program behaviour

"What is a reasonable definition?"

- Insights from automation and systems engineering needed

Central Question

- ► We considered restart-robustness w.r.t. specifications
- ⇒ Require existence of specifications and checking all of them
- ► In practice, software is often under-specified
- ⇒ Many (mis-)behaviours not considered by verifier

Idea Characterise restart-robustness relational property between the nominal and the restart-augmented program behaviour

"What is a reasonable definition?"

- ► Insights from automation and systems engineering needed

Central Question

- We considered restart-robustness w.r.t. specifications

⇒ Require existence of specifications and checking all of them

- In practice, software is often under-specified

⇒ Many (mis-)behaviours not considered by verifier

Idea Characterise restart-robustness relational property between the nominal and the restart-augmented program behaviour

"What is a reasonable definition?"

- Insights from automation and systems engineering needed

Central Question

- We considered restart-robustness w.r.t. specifications
- ⇒ Require existence of specifications and checking all of them
- In practice, software is often under-specified
- ⇒ Many (mis-)behaviours not considered by verifier

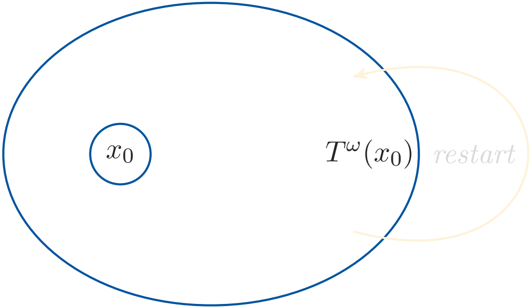
Idea Characterise restart-robustness relational property between the nominal and the restart-augmented program behaviour

"What is a reasonable definition?"

- Insights from automation and systems engineering needed

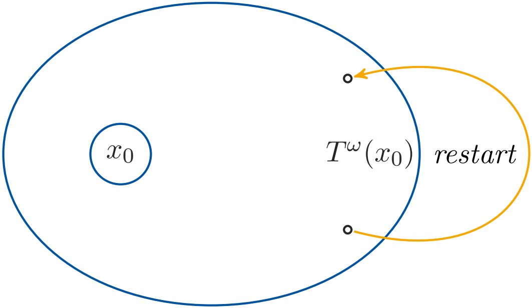
Informatik 11
Embedded Software
RWTH AACHEN UNIVERSITY

Relate State Spaces



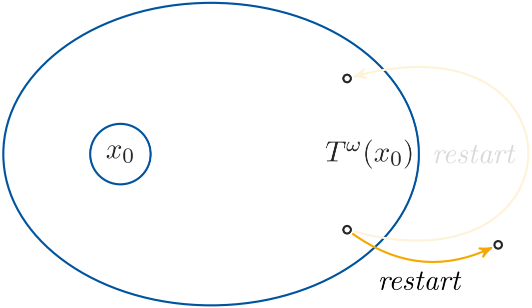
- Restart-augmented program must stay within original states
- Might need grace period of k program cycles

Relate State Spaces



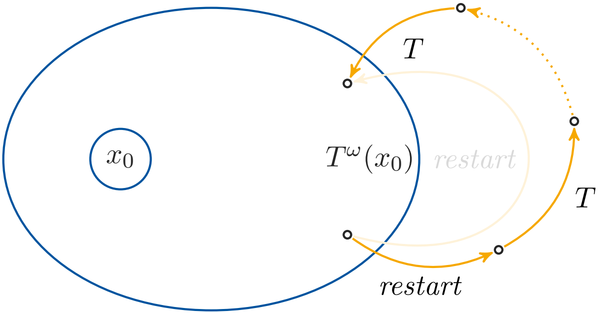
- Restart-augmented program must stay within original states
- Might need grace period of k program cycles

Relate State Spaces



- Restart-augmented program must stay within original states
- Might need grace period of k program cycles

Relate State Spaces



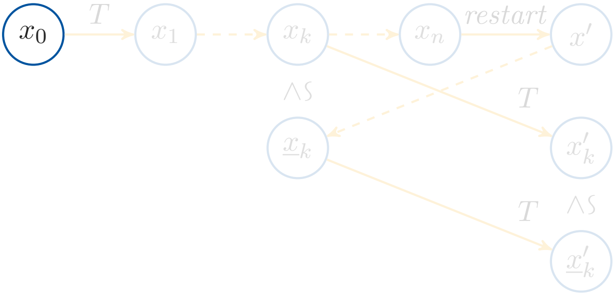
- Restart-augmented program must stay within original states
- Might need grace period of k program cycles

Relate Executions

- After restarting, a state that is (simulation-)equivalent to a pre-restart state will eventually be reached[3]

[3] Eric Koskinen and Junfeng Yang. “Reducing crash recoverability to reachability”. In: *POPL 2016*. 2016, pp. 97–108.

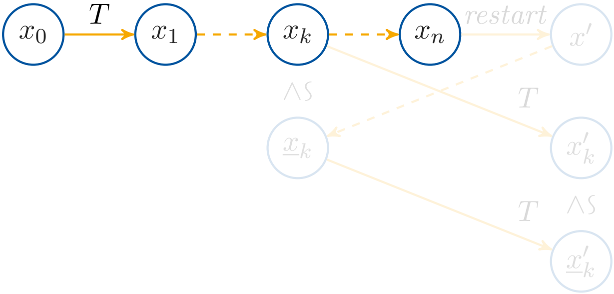
Relate Executions



- After restarting, a state that is (simulation-)equivalent to a pre-restart state will eventually be reached[3]

[3] Eric Koskinen and Junfeng Yang. "Reducing crash recoverability to reachability". In: *POPL 2016*. 2016, pp. 97–108.

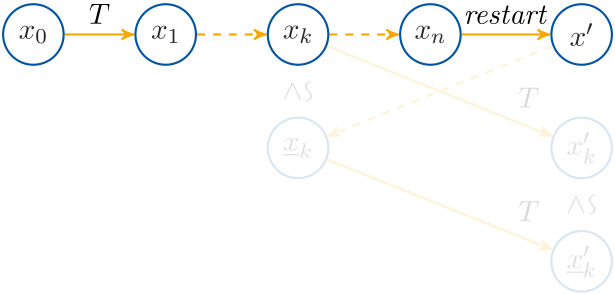
Relate Executions



- After restarting, a state that is (simulation-)equivalent to a pre-restart state will eventually be reached[3]

[3] Eric Koskinen and Junfeng Yang. "Reducing crash recoverability to reachability". In: *POPL 2016*. 2016, pp. 97–108.

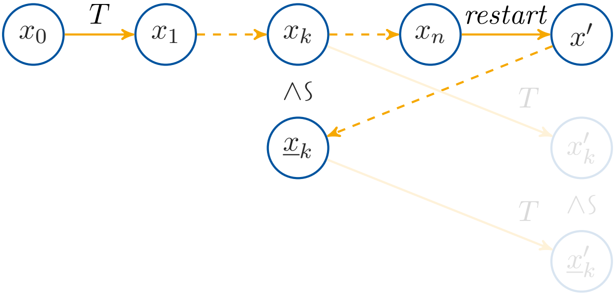
Relate Executions



- After restarting, a state that is (simulation-)equivalent to a pre-restart state will eventually be reached[3]

[3] Eric Koskinen and Junfeng Yang. "Reducing crash recoverability to reachability". In: *POPL 2016*. 2016, pp. 97–108.

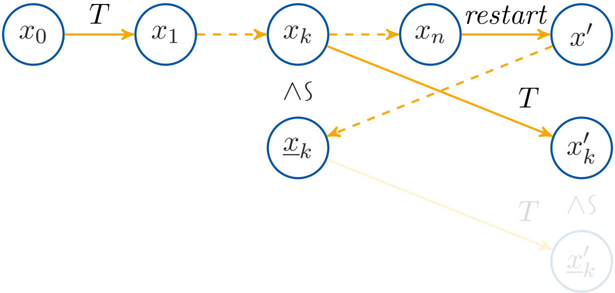
Relate Executions



- After restarting, a state that is (simulation-)equivalent to a pre-restart state will eventually be reached[3]

[3] Eric Koskinen and Junfeng Yang. "Reducing crash recoverability to reachability". In: *POPL 2016*. 2016, pp. 97–108.

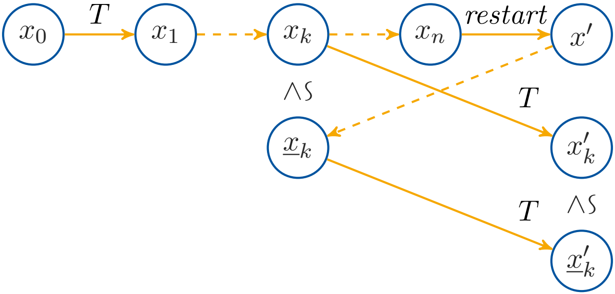
Relate Executions



- After restarting, a state that is (simulation-)equivalent to a pre-restart state will eventually be reached[3]

[3] Eric Koskinen and Junfeng Yang. "Reducing crash recoverability to reachability". In: *POPL 2016*. 2016, pp. 97–108.

Relate Executions



- After restarting, a state that is (simulation-)equivalent to a pre-restart state will eventually be reached[3]

[3] Eric Koskinen and Junfeng Yang. "Reducing crash recoverability to reachability". In: *POPL 2016*. 2016, pp. 97–108.

Summary

- Battery-backed memory & restart funcitonality are common features that enable the design of systems resilient to restarts
- Currently, we consider restart-robustness w.r.t. a specification
- A relational definition of restart-robustness is more practical

But Insight needed to define to what extent the restart-augmented program is allowed to deviate from the original

Summary

- Battery-backed memory & restart funcitonality are common features that enable the design of systems resilient to restarts
- Currently, we consider restart-robustness w.r.t. a specification
- A relational definition of restart-robustness is more practical

But Insight needed to define to what extent the restart-augmented program is allowed to deviate from the original

Summary

- Battery-backed memory & restart funcitonality are common features that enable the design of systems resilient to restarts
- Currently, we consider restart-robustness w.r.t. a specification
- A relational definition of restart-robustness is more practical

But Insight needed to define to what extent the restart-augmented program is allowed to deviate from the original

Summary

- Battery-backed memory & restart funcitonality are common features that enable the design of systems resilient to restarts
- Currently, we consider restart-robustness w.r.t. a specification
- A relational definition of restart-robustness is more practical

But Insight needed to define to what extent the restart-augmented program is allowed to deviate from the original

References I

[BK18] Dimitri Bohlender and Stefan Kowalewski. “Design and Verification of Restart-Robust Industrial Control Software”. In: *IFM 2018*. 2018, pp. 47–68.

[Hau+15] Stefan Hauck-Stattelmann et al. “Analyzing the Restart Behavior of Industrial Control Applications”. In: *FM 2015*. 2015, pp. 585–588.

[KY16] Eric Koskinen and Junfeng Yang. “Reducing crash recoverability to reachability”. In: *POPL 2016*. 2016, pp. 97–108.