SIEMENS

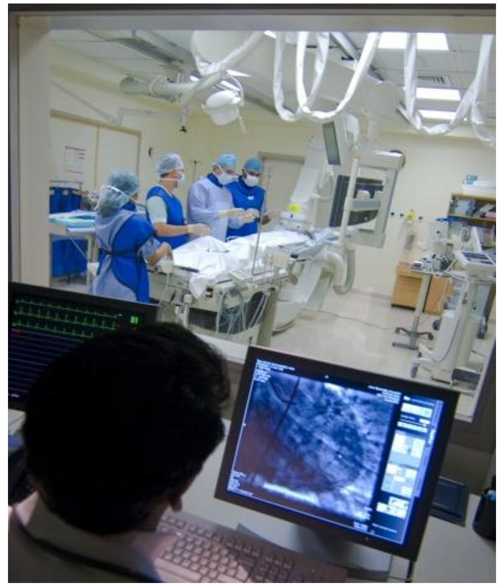
July 2015

Verification Cases: Characterizing the Completeness Degree of Incomplete Verification

Towards Using Formal Verification for Low Criticality Functions

Daniel Ratiu (Siemens CT), **Vincent Nimal** (Univ. Oxford)

Our Business Domains



State of Practice

Structural test coverage recommendations for different SIL levels (IEC 61508)

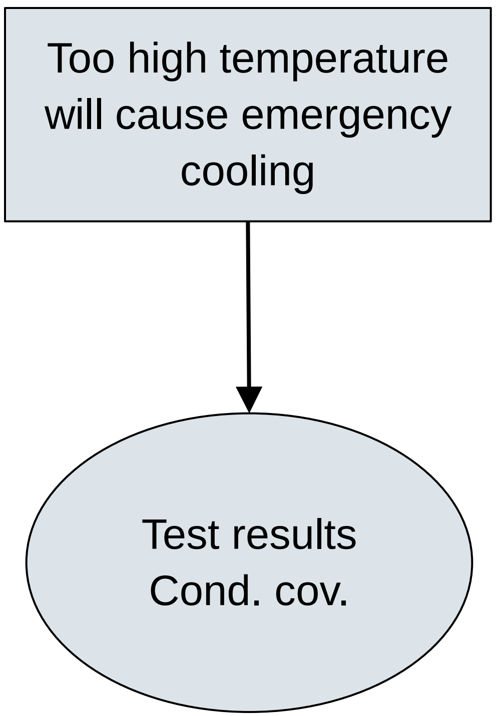
	Technique/Measure *	Ref	SIL 1	SIL 2	SIL 3	SIL 4
7a	Structural test coverage (entry points) 100 % **	C.5.8	HR	HR	HR	HR
7b	Structural test coverage (statements) 100 %**	C.5.8	R	HR	HR	HR
7c	Structural test coverage (branches) 100 %**	C.5.8	R	R	HR	HR
7d	Structural test coverage (conditions, MC/DC) 100 %**	C.5.8	R	R	R	HR

State of Practice (2)

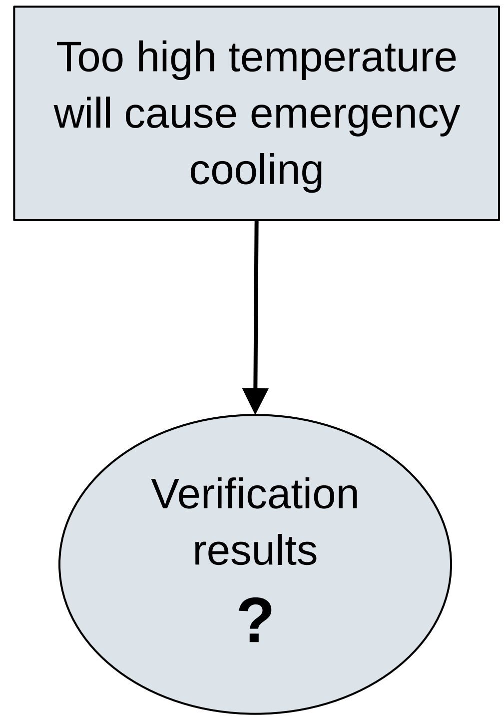
Formal verification is used only when explicitly required by safety standards (i.e. not at all for SIL1-3 functions)

… even if scalable C-level model checkers are available,
… results as good as „branch coverage testing“ are easy to achieve

Evidence in Assurance Cases



Tests results as evidence



Verification results as evidence

Why is Formal Verification not (really) Used?

SIEMENS

We need means to characterize the completeness degree of incomplete verification

… in order to qualify the evidence which is used in an assurance case

Verification Cases

Complement Unit Testing with Verification for Functions at Lower Criticality Levels

Increase the Usability of Formal Verification for Practicing Engineers

www.mbeddr.com

Causes of Incompleteness

- Tooling limitations
  - Boundness of model checkers

- Simplification assumptions to speed-up the verification
  - Restrictions on the data environment
  - Simplifying models of libraries
  - Bounding the unwinding of certain loops
  - Stubbing parts of the system
  - Bounding the number of threads
  - …

Example

The system must prevent overheating. […] The cooling system can be started manually or automatically. […] The min. and max. temperatures of coolant are entered by the operator

```
uint16 min
uint16 max
boolean coolingStarted;

cooling_cmd coolingCommand(uint16 crtTemp) {
  return
```

	crtTemp < min	crtTemp in [min..max[	crtTemp > max
coolingStarted	stop cooling	start cooling	emergency cooling
!coolingStarted	no cooling	no cooling	emergency cooling

```
} coolingCommand (function)
```

```
  exported testcase coolingTest1 {
    min = 20;
    max = 50;
    coolingStarted = true;
    assert(0) coolingCommand(25) == start_cooling;

    coolingStarted = false;
    assert(1) coolingCommand(25) == no_cooling;
  } coolingTest1(test case)
```

Example

The system must prevent overheating. […] The cooling system can be started manually or automatically. […] The min. and max. temperatures of coolant are entered by the operator

```
uint16 min
uint16 max
boolean coolingStarted;

cooling_cmd coolingCommand(uint16 crtTemp) {
  return
} coolingCommand (function)
```

	crtTemp < min	crtTemp in [min..max[	crtTemp > max
coolingStarted	stop cooling	start cooling	emergency cooling
!coolingStarted	no cooling	no cooling	emergency cooling

```
verification_case coolingVerification for :coolingCommand {
  initial state: {
    data env: min : uint8 -> min (uint16) {   };
    data env: max : uint8 -> max (uint16) { max > min; };
  }

  verification step {
    data env: myTemp : uint8 -> crtTemp (uint16) {   };
    data env: myCoolingStarted : boolean -> coolingStarted (boolean) {   };
    cooling_cmd cmd = coolingCommand(myTemp);
    assert(myTemp > max → cmd == emergency_cooling);
  }
} coolingVerification (function)
```

Verification Cases vs. Test Cases