# Revisiting the Formal Verification of the Darlington Nuclear Generating Station Shutdown Systems 12 Years Later

Mark Lawford, P.Eng, Ph.D.

McMaster Centre for Software Certification (McSCert)
McMaster University
Hamilton, ON, Canada

*Workshop on Theorem Proving in Certification*
Cambridge University

McSCert

# Outline



McSCert

# McMaster Centre for Software Certification

- Leading 5 year $22 Million Ontario Research Fund Research Excellence project on Certification of Software Intensive Systems in collaboration with U Waterloo and York U (Canada).
- Focused on product *(not process)* oriented certification
- Working with industry and regulators to improve software in three (soon 4!) main areas:
  - Biomedical Devices,
  - Financial Systems,
  - Nuclear
  - and coming soon - Automotive!
- We are currently recruiting:
  - Post docs
  - Research Engineers
  - Graduate Students

McSCert

# Darlington SDS Redesign Project

**Note:**

This talk is not the originial Darlington Project when Ontario Hydro was forced to reverse engineer tabular specifications for requirements and the code in order to get regulatory approval.
Its about the Redesign project where formal techniques were integrated in the forward development process.

- People often cite the difficulty & cost of the original project when they want to dismiss tabular methods.
- No other formal method applied after the fact would have fared any better.

McSCert

# What is a Shut Down System (SDS)?

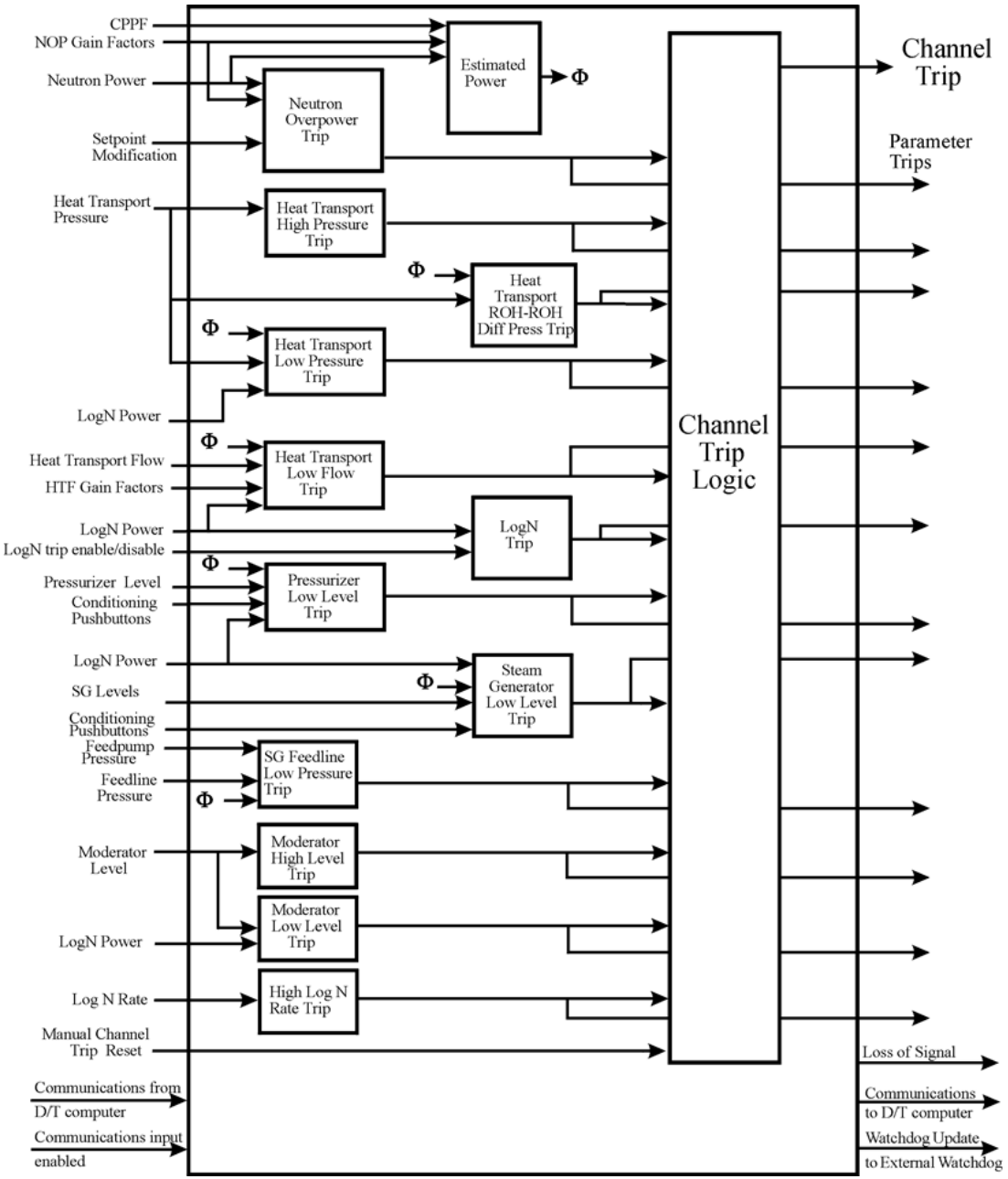
An SDS is:

- watchdog system that monitors system parameters
- shuts down (trips) reactor if it observes “bad” behavior
- process control is performed a separate Digital Control computer (DCC) - not as critical

Why use formal verification?

- Spurious trips cost $$$
- Difficult to make modifications & even more difficult to get regulatory approval for changes
- Minor changes result in another extensive (& expensive) round of testing & review
- Testing can’t cover all possible cases
- Too much detail for person to catch everything by review

McSCert




60 modules
280 access programs
40,000 lines of code (including comments)
33,000 FORTRAN
7,000 assembler
84 monitored variables
27 controlled variables

McSCert

# The Standard Used

The CANDU Computer Systems Engineering Centre for Excellence *Standard for Software Engineering of Safety Critical Software* first fundamental principle states:

> *"The required behavior of the software shall be documented using mathematical functions in a notation which has well defined syntax and semantics."*

**Determinism:** Want unambiguous description of safety critical behavior

**Clarity:** Easier to understand functional requirements

**Preference:** Engineers prefer to specify precise behavior and appeal to tolerances when necessary

**Sufficient:** Functional methods often sufficient - Work "most of the time" & are easily automated

McSCert

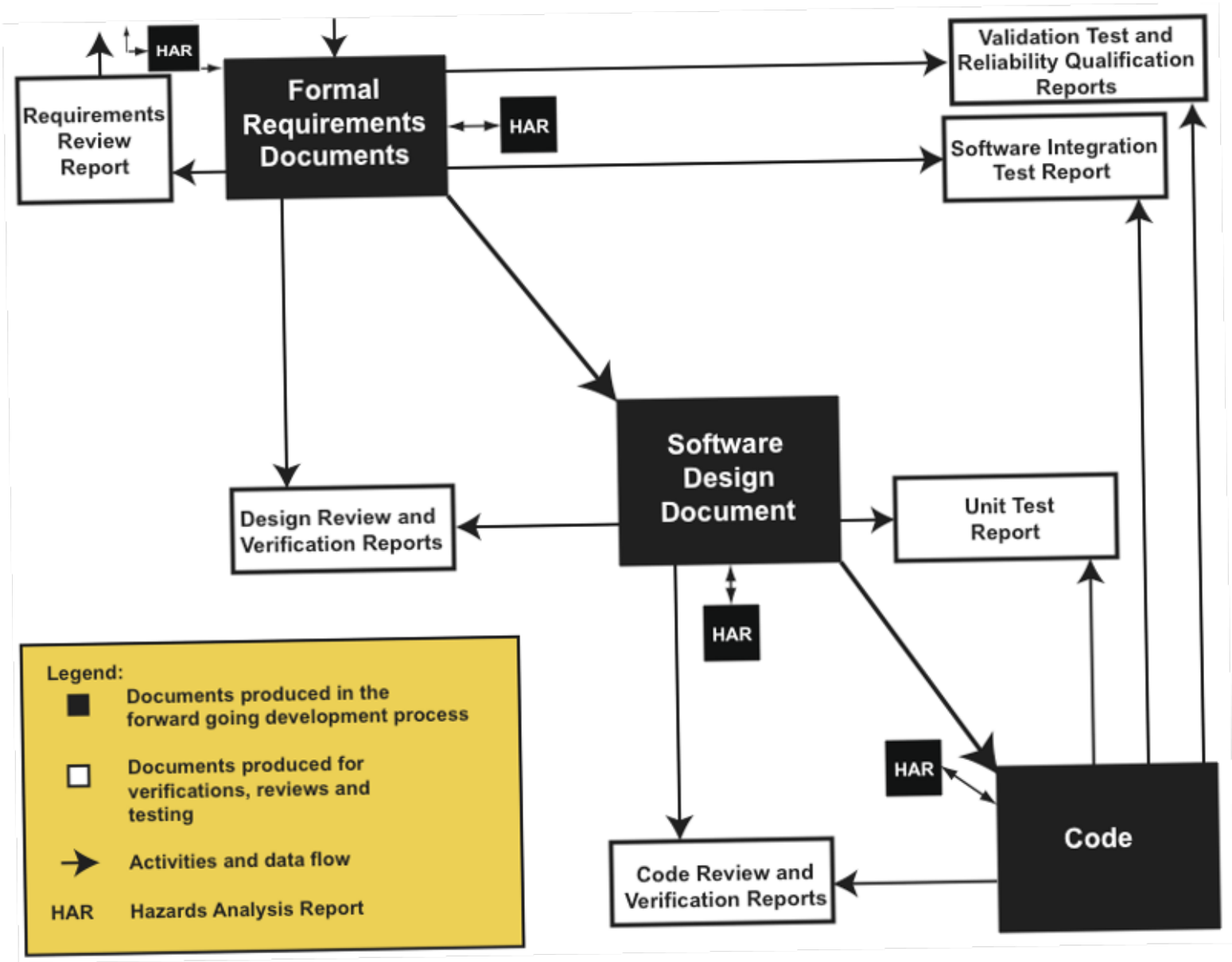
# The Assurance Case Implicit in the CANDU Standard

A part of the assurance case implicitly embodied in the standard employed in developing the SDS was as follows:

1. The requirements are specified mathematically and checked for completeness and consistency. A hazards analysis is required to document risks and especially to identify sources of single point failures. These hazards have to be mitigated in the specified requirements.
2. Compliance between requirements and software design is mathematically verified.
3. Compliance between the code and software design is verified through both mathematical verification and testing. Compliance between code and requirements is shown explicitly through testing. However, there is an implicit argument of compliance between code and requirements through the transitive closure of the mathematical verification - code to design, and design to requirements.

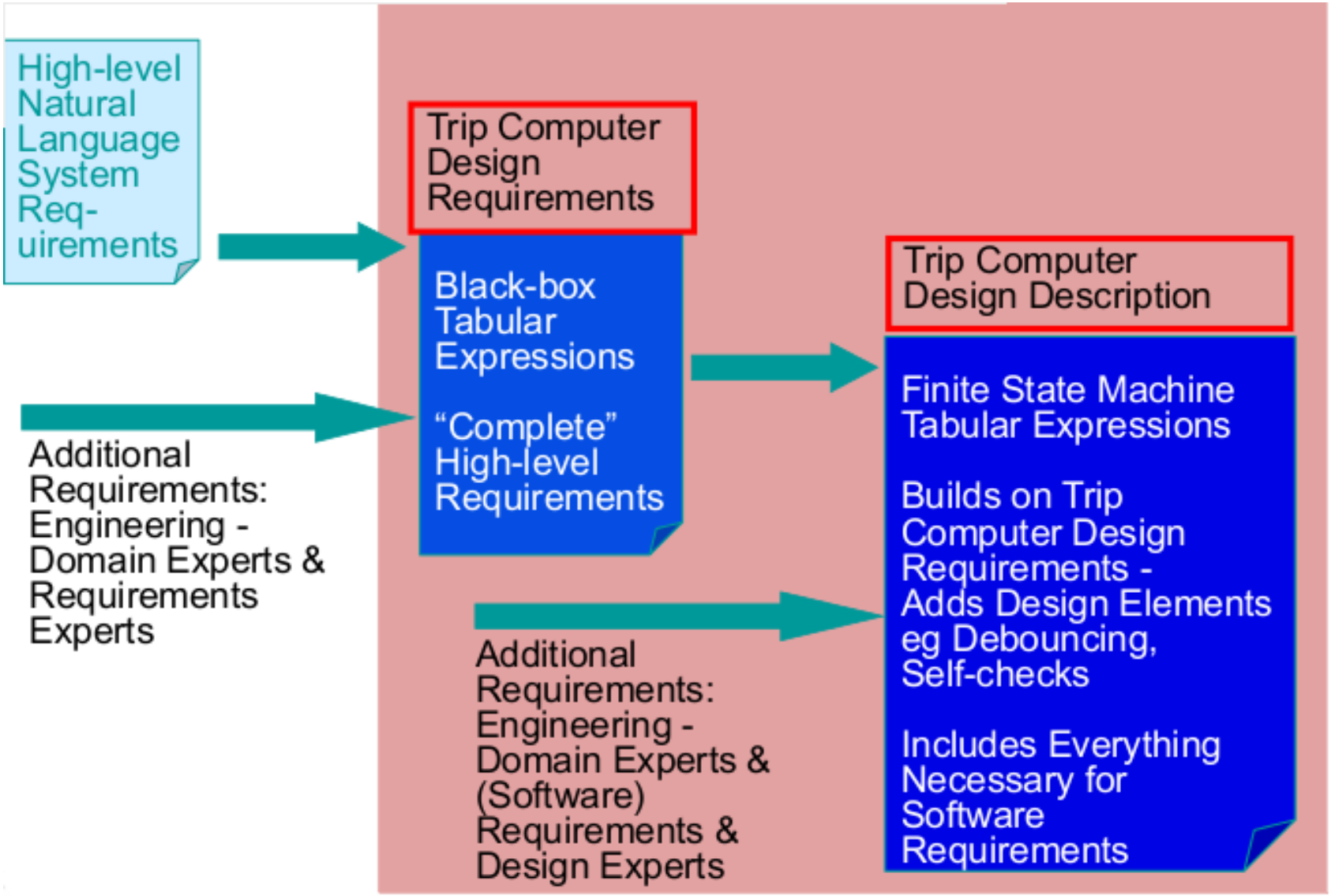
McSCert

# Idealized Development Process




McSCert

# System Requirements




McSCert

# Tabular Expressions - A Useable "Formal" Method

For the resdesign the Software Requirements Specification (SRS) made extensive use of tabular expressions to document the requirements as did the Software Design Description (SDD). Why?

- Ontario Hydro had some experience with tabular expressions since they were used to get Darlington licensed the first time.
- They are readable by domain engineers, operators, testers . . . and developers!
- Built on previous successes with tabular methods (e.g. A-7)

They eventually showed significant benefits when used in a process with integrated tool support.

McSCert

# Tabular Expressions

- In order for a table to be proper it must satisfy two properties.

$$f(x_1, \ldots, x_m) = \begin{array}{|c|c|c|c|} \hline c_1 & c_2 & \ldots & c_n \\ \hline e_1 & e_2 & \ldots & e_n \\ \hline \end{array}$$

Here each $c_i$ is a Boolean expression, when $c_i$ is true $f$ returns $e_i$

1. Disjointness - $i \neq j \rightarrow (c_i \wedge c_j \leftrightarrow \bot)$
2. Completeness - $(c_1 \vee c_2 \vee \ldots \vee c_n) \leftrightarrow \top$

McSCert

# Why Tables Work

$$\mathrm{f}(x,y) \stackrel{\mathrm{df}}{=} \begin{cases} x+y & \text{if } x>1 \wedge y<0 \\ x-y & \text{if } x\leq 1 \wedge y<0 \\ x & \text{if } x>1 \wedge y=0 \\ xy & \text{if } x\leq 1 \wedge y=0 \\ y & \text{if } x>1 \wedge y>0 \\ x/y & \text{if } x\leq 1 \wedge y>0 \end{cases} \tag{1}$$

$$\mathrm{f}(x,y) \stackrel{\mathrm{df}}{=} \tag{2}$$

| | $x>1$ | $x\leq 1$ |
|---|---|---|
| $y<0$ | $x+y$ | $x-y$ |
| $y=0$ | $x$ | $xy$ |
| $y>0$ | $y$ | $x/y$ |

You can actually read them.

McSCert

# TCDD Example - NOP Trip

**2.1.3.9.1 Neutron Overpower Parameter Trip**

**2.1.3.9.1.1 Inputs/Natural Language Expressions**

| Input | NL Expression | Reference |
|---|---|---|
| $f\_NOPsentrip_i$, i=1,..,18 | - | 2.1.3.9.2.4 |

hyperlinks

**2.1.3.9.1.2 c_NOPparmtrip**

| *Condition* | *Result* **c_NOPparmtrip** |
|---|---|
| Any ($i \in 1,..,18$) ($f\_NOPsentrip_i = e\_Trip$) *{Any NOP sensor is tripped}* | e_Trip |
| All ($i = 1,..,18$) ($f\_NOPsentrip_i = e\_NotTrip$) *{All NOP sensors are not tripped}* | e_NotTrip |

Tabular Expressions

we did not use ∀, ∃ and other dense math notation

comments

McSCert

# TCDD Example - NOP Trip

### 2.1.3.9.2 Neutron Overpower Sensor Trips

**Determines whether there is an NOP sensor trip, which is used to determine whether there is an associated parameter trip.**

natural language expression!
domain experts loved them

#### 2.1.3.9.2.1 Inputs/Natural Language Expressions

| Input | NL Expression | Reference |
|---|---|---|
| f_NOPsp | - | 2.1.3.9.3.3 |
| f_NOPGain$_i$, i=1,..,18, k_CalNOPHiLimit, k_CalNOPLoLimit, k_NOPOffset, m_NOPai$_i$, i=1,..,18 | Calibrated i$^{th}$ NOP signal, i=1,..,18 | 2.1.4.12 |

#### 2.1.3.9.2.4 f_NOPsentrip$_i$, i=1,..,18

{For each i = 1,..,18}

| *Condition* | *Result* f_NOPsentrip$_i$ |
|---|---|
| f_NOPsp ≤ Calibrated i$^{th}$ NOP signal *{Calibrated NOP signal$_i$ is now in the trip region}* | e_Trip |
| f_NOPsp - k_NOPhys < Calibrated i$^{th}$ NOP signal < f_NOPsp *{Calibrated NOP signal$_i$ is now in the deadband region}* | (f_NOPsentrip$_i$)$_{-1}$ |
| Calibrated i$^{th}$ NOP signal ≤ f_NOPsp - k_NOPhys *{Calibrated NOP signal$_i$ is now in the non-trip region}* | e_NotTrip |

complete & disjoint!

McSCert

# Module Interface Spec for NOP

## 4.23 MODULE NPParTrip

*Provides the current NOP parameter trip status to drive the NOP parameter trip output.*

| | Name | Value | Type |
|---|---|---|---|
| **Constants:** | (None) | | |

| | Name | Definition |
|---|---|---|
| **Types:** | (None) | |

**Access Programs:**

**EPTNP**

Determines the current NOP parameter trip status and posts the parameter trip output state to DigitalOutput module.

References: *c_NOPparmtrip*

**GPTSNP**

return: t_boolean

Returns the current NOP parameter trip status. A return value of $TRUE or $FALSE indicates that the parameter is tripped or not tripped respectively.

References: *c_NOPparmtrip*

**IPTNP**

Initializes all the NPParTrip module internal states.

References: *Initial Value: NPParTrip*

Module "cover page" (MIS) shows external behaviour only.

SQRL

McSCert

# Module Internal Design

- Each access program needs to be specified. We specify details of the required behaviour without going to the level of sequential code statements (most of the time)
- Two very simple rules make mathematical verification much more tractable:
  - Get
  - Process
  - Set
- Value of an asynchronous variables (e.g. a timer) stored on input - stored value used throughout module

McSCert

# Module Internal Design - NOP

**ACCESS PROGRAM EPTNP**

| | Name | Ext_value | Type | Origin |
|---|---|---|---|---|
| **Inputs:** | l_ST | GST1NP(l_ST) | ARRAY 1 TO KNUMNP OF t_boolean | NPSnrTrip |

| | Name | Ext_value | Type | Origin |
|---|---|---|---|---|
| **Updates:** | (None) | | | |

| | Name | Ext_value | Type | Origin |
|---|---|---|---|---|
| **Outputs:** | l_TrpDO | SDONP(l_TrpDO) | t_boolean | DigitalOutput |
| | PTSNP | - | t_boolean | State |

**Local Terms:**

| | |
|---|---|
| l_NoSTrp | (ALL i=1..KNUMNP)(l_ST[i] = $FALSE) |

safe state
if there is one

VCT: EPTNP

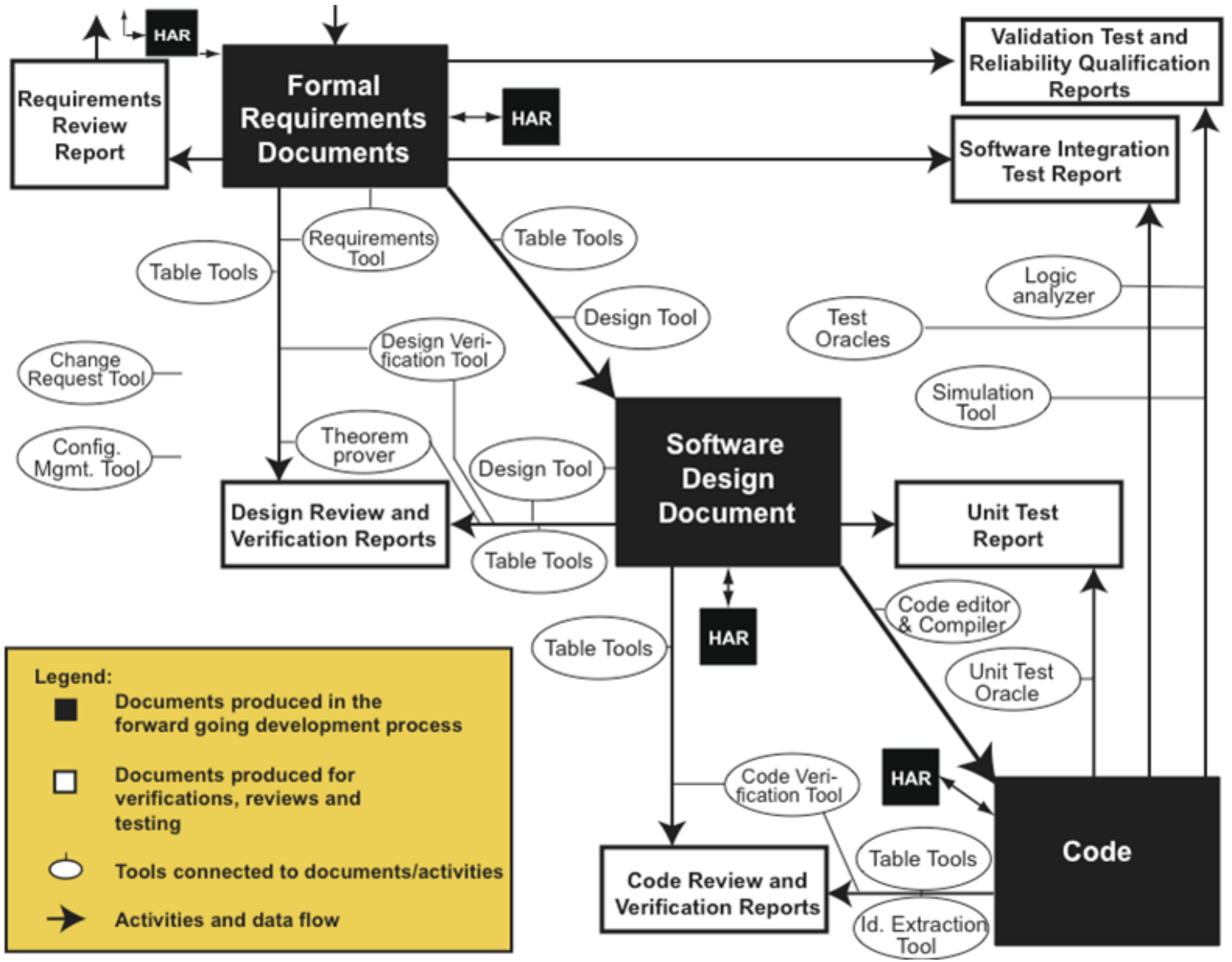
| | l_NoSTrp | NOT(l_NoSTrp) |
|---|---|---|
| l_TrpDO | $FALSE | $TRUE |
| PTSNP | $FALSE | $TRUE |

Use 16 bits for a boolean. Choose $TRUE for safe state. Then 65K to 1 chance of safe state even if there is a hardware fault.

17

McSCert

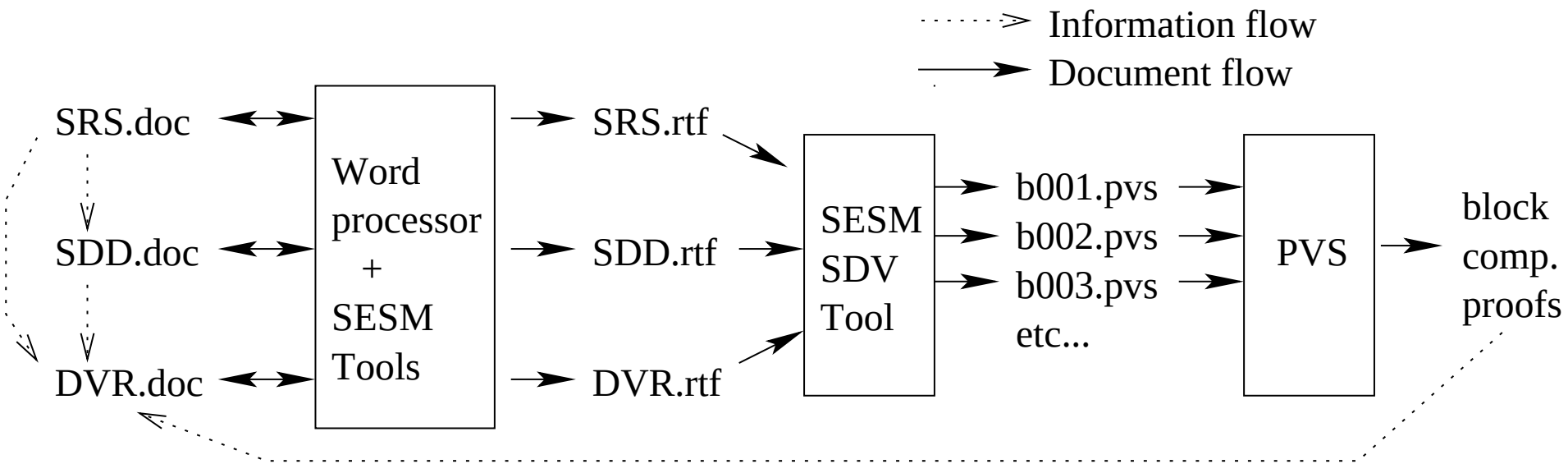
# Idealized Development Process & Tools




McSCert

# Tool Supported Formal Methods

A formal method should be tightly integrated with the software development process - i.e. it is directly applied to project documents used by all parties as part of the process.




$$SOF_i \circ Abst_{V_{i-1}} = Abst_{V_i} \circ REQ_i$$

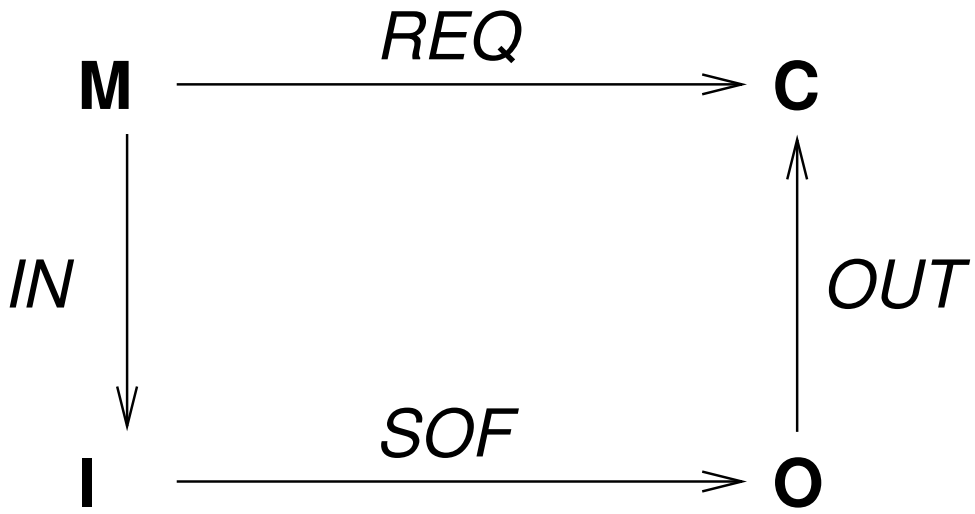
McSCert

# How much did the tools do?

- roughly 70% of the over 200 functional blocks from the two software designs of the Redesign Project were formally verified using the SDV Tool together with PVS.
- The remainder of the verification blocks that did not involve straight forward block comparisons, requiring additional reasoning about the program’s main execution thread and timing constraints, were handled by rigorous manual arguments.

McSCert

# 4-Variable Model (Parnas &Madey)




**M** - Monitored Variable statespace
**C** - Controlled Variable statespace
**I** - Input Variable statespace
**O** - Output Variable statespace

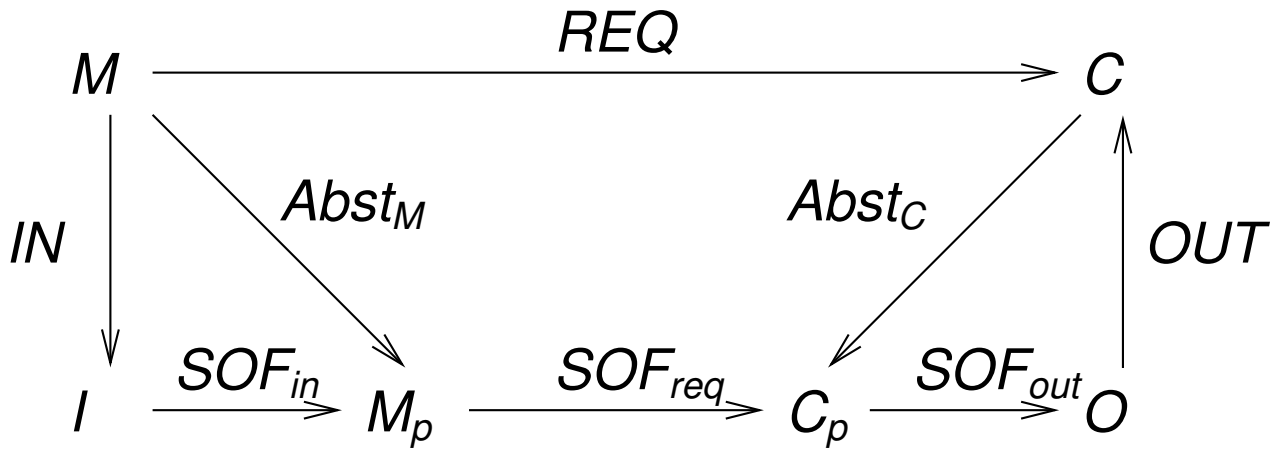
**M**, **C**, **I**, **O** are time series vectors and *REQ*, *SOF*, *IN*, *OUT* are relations. We use a special case where all relations are functional resulting in proof obligation:

$$REQ = OUT \circ SOF \circ IN \tag{3}$$

Here *REQ* and *SOF* are the one step transition functions of the requirements and design respectively.

McSCert

# “Vertical” Decomposition of Proof Obligations




$$Abst_{\mathbf{C}} \circ REQ = SOF_{req} \circ Abst_{\mathbf{M}} \quad (4)$$

$$Abst_{\mathbf{M}} = SOF_{in} \circ IN \quad (5)$$

$$id_{\mathbf{C}} = OUT \circ SOF_{out} \circ Abst_{\mathbf{C}} \ . \quad (6)$$

(5) and (6) represent verification of hardware hiding modules

$\mathbf{M}_p$ is *pseudo-monitored* variables

$\mathbf{C}_p$ is *pseudo-controlled variables*

McSCert

# “Vertical” Decomposition (cont)

More "vertical" decomposition obtained by isolating outputs. In effect,

i) projecting **C** onto single output

ii) restricting *REQ* to the relevant subset of **M**

**Note:**

“Wrong way” $Abst_{\mathbf{C}}$ arrow - used to reduce number of required abstraction functions.

Why?

- Can reduce by up to 1/2 number of abstraction functions required.
- Proof obligation (6) precludes possibility of trivial implementations

Invertibility of *OUT* not possible in all situations but applicable to majority of our safety critical requirements.

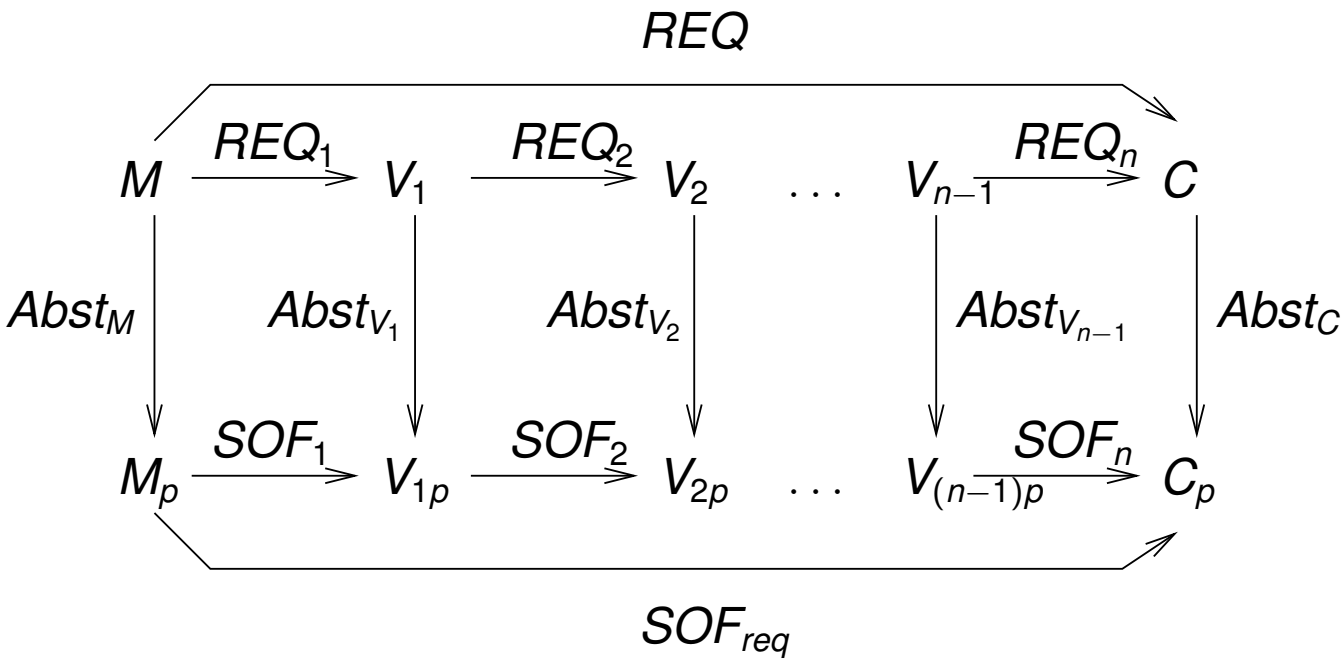
McSCert

# Hardware Hiding Example

- The temperature of the primary heat transport system which belongs to **M** might have a value of 500.3 Kelvin.
- A/D converters map this via (part of) *IN* to a value of 3.4 volts in a parameter of **I**.
- A hardware hiding module might then process this input corresponding to map $SOF_{in}$, producing a value of 500 Kelvin in the appropriate temperature variable of the software state space $\mathbf{M_p}$.

McSCert

# “Horizontal” Decomposition of Proof Obligations

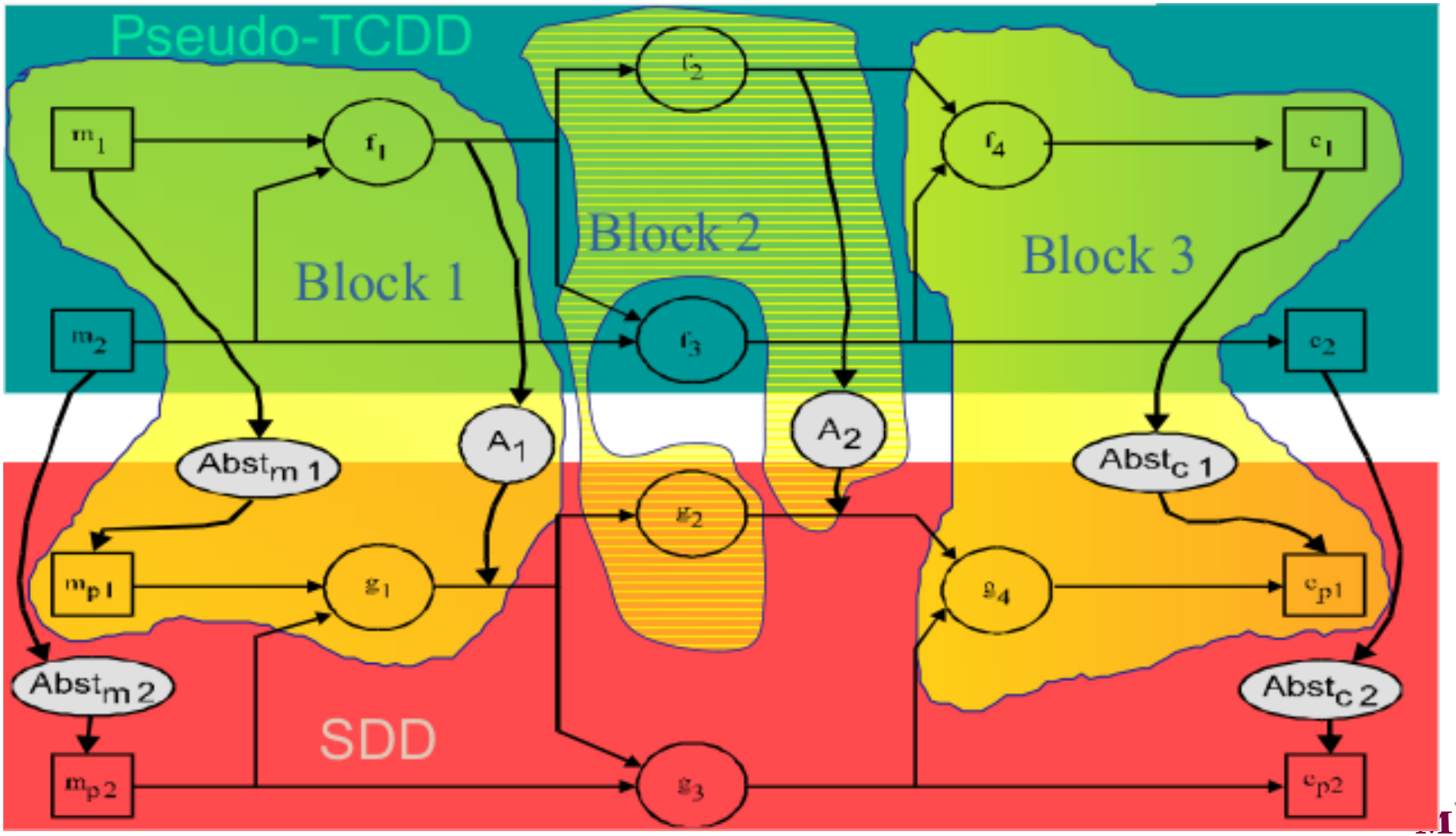



Main block comparison can be sequentially decomposed into sequence of simpler obligations of the form:

$$SOF_i \circ Abst_{V_{i-1}} = Abst_{V_i} \circ REQ_i \tag{7}$$

Cost of decomposition? Verifier must provide cross reference in form of $Abst_{V_i} : V_i \rightarrow V_{ip}$. Now we see benefit of "wrong way" arrow: Same $Abst_{V_i}$ can be used on output then input of successive blocks.

# Example: Piece-wise Verification




Block 4 is everything else.

McSCert

# Example: Piece-wise Verification

$$A1(f1(m1, m2) = g1(Abstm1(m1), Abstm2(m2)) \quad \text{(Block 1)}$$

$$A2(f2(f1*)) = g2(A1(f1*)) \quad \text{(Block 2)}$$

$$Abstc1(f4(f2*, f3*)) = g4(A2(f2*), Abstc2(f3*)) \quad \text{(Block 3)}$$

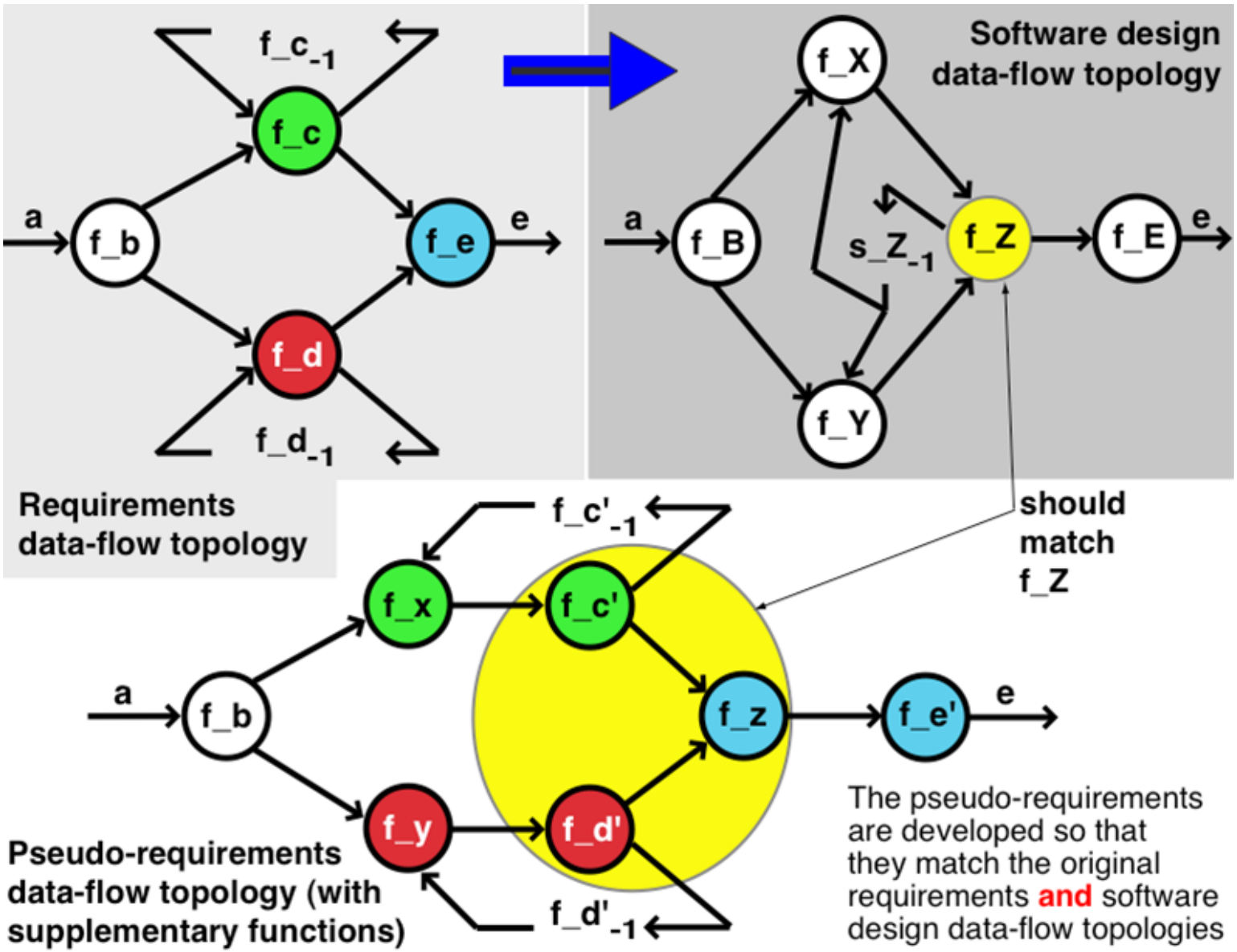
$$Abstc2(f3(f1*, m2)) = g3(A1(f1*), Abstm2(m2)) \quad \text{(Block 4)}$$

- If we verify that (Block 1) through (Block 4) is true, then we can show that we have met the total proof obligation.[1]
- Doing it piece-wise is MUCH easier than trying to do the total proof. But somtimes you need Supplementary Function Tables to help with verification.

[1] The decomposition was done and verified manually

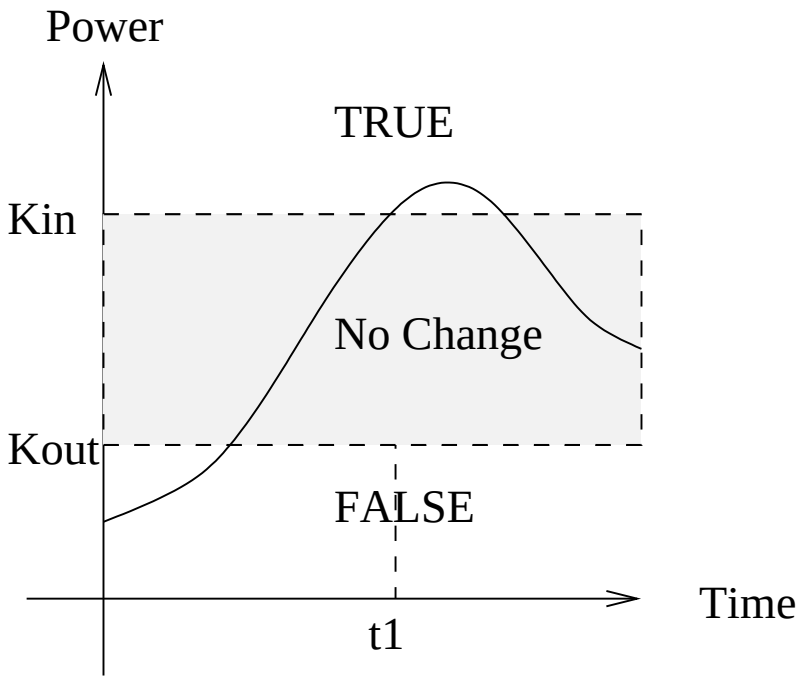
McSCert

# . . . and when dataflow is not isomorphic




McSCert

# Power Conditioning




PwrCond(Prev:bool, Power, Kin, Kout:posreal):bool =

| $Power \leq Kout$ | $Kout < Power < Kin$ | $Power \geq Kin$ |
|---|---|---|
| *FALSE* | *Prev* | *TRUE* |

McSCert

# General Power Conditioning Function

When Power:

- drops below *Kout*, sensor is unreliable so it’s “conditioned out” (*PwrCond* = *FALSE*).
- exceeds *Kin*, the sensor is “conditioned in” and is used to evaluate the system.
- is between *Kout* and *Kin*, the value of *PwrCond* is left unchanged by setting it to its previous value, *Prev*.

E.g. For the graph of *Power* above, *PwrCond* would start out FALSE, then become TRUE at time $t1$ and remain TRUE.

McSCert

# PVS Specification of a General *PwrCond* Function

```
PwrCond(Prev:bool, Power, Kin, Kout:posreal):bool = TABLE
   %-------------------------------------------------------%
   |[Power<=Kout  | Power>Kout & Power<Kin | Power>=Kin]|
   %-------------------------------------------------------%
   |    FALSE     |          Prev          |    TRUE    ||
   %-------------------------------------------------------%
ENDTABLE
```

The above PVS specification of the *PwrCond* table produces the following proof obligations or “TCCs”.

McSCert

# Step 1: Type-checking PwrCond

```
% Disjointness TCC generated (at line 14, column 55) for
  % unfinished
PwrCond_TCC1: OBLIGATION
  FORALL (Kin, Kout: posreal, Power):
    NOT (Power <= Kout AND Power > Kout & Power < Kin) AND
     NOT (Power <= Kout AND Power >= Kin) AND
      NOT ((Power > Kout & Power < Kin) AND Power >= Kin);

% Coverage TCC generated (at line 14, column 55) for
  % proved - complete
PwrCond_TCC2: OBLIGATION
  FORALL (Kin, Kout: posreal, Power):
    (Power <= Kout OR                    % Column1
     (Power > Kout & Power < Kin)        % Column2
     OR Power >= Kin)                    % Column3
```

McSCert

# Type-checking PwrCond

The coverage TCC is easily proved by PVS. Thus we conclude that at least one column is always satisfied for every input.
But attempting the Disjointness TCC fails, indicating that the columns overlap. The resulting unprovable sequent for the disjointness TCC is:

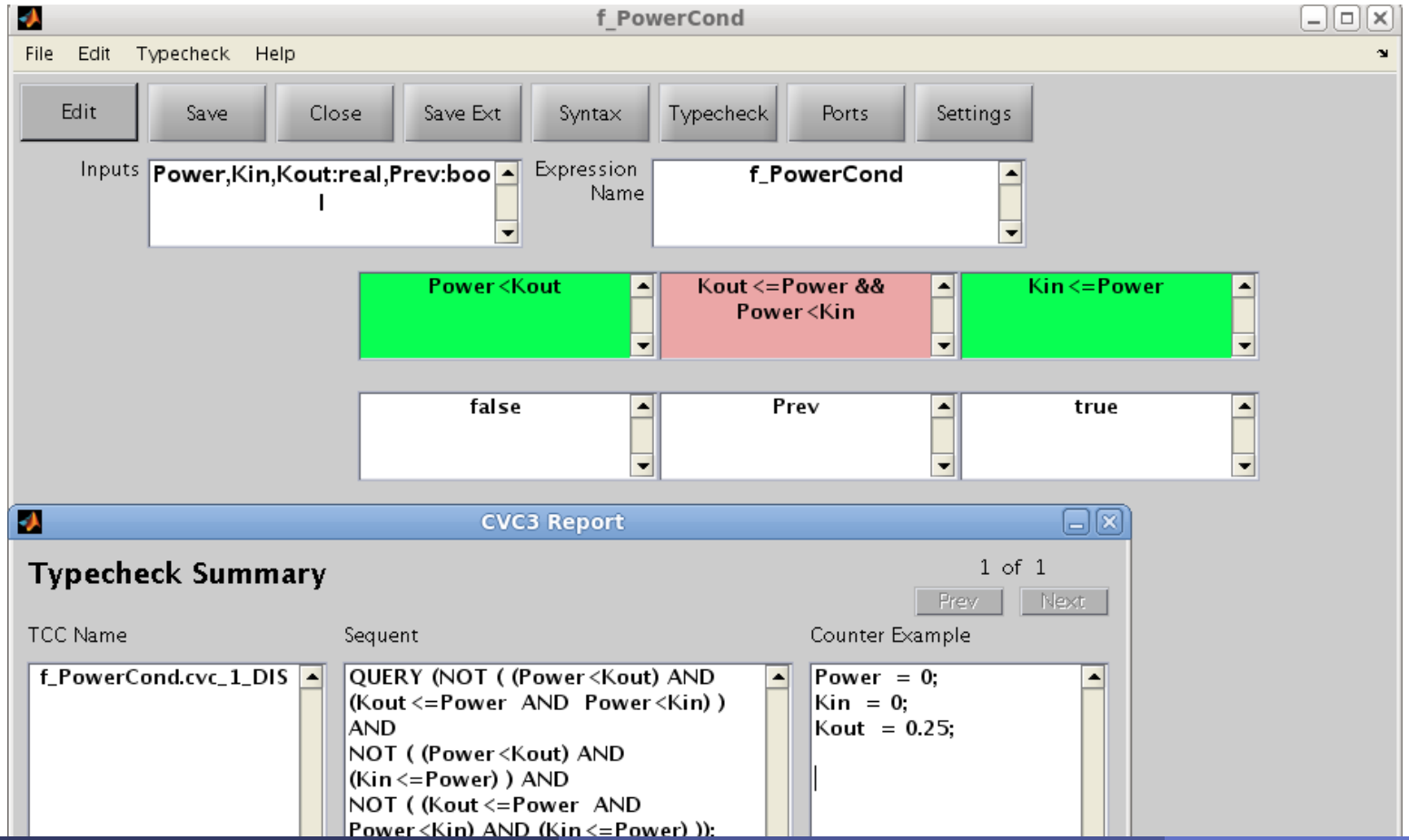
```
PwrCond_TCC1 :
[-1]     Kin!1 > 0
[-2]     Kout!1 > 0
[-3]     Power!1 > 0
[-4]     Power!1 <= Kout!1
[-5]     (Kin!1 <= Power!1)
   |-------
[1]      FALSE
Rule?
```

McSCert

# How times have changed!

The unprovable sequent was all I could get out of PVS in 1998 as a verifier.

This is a very simple one, but still not very user friendly. Now as a developer I can get in Matlab/Simulink:




McSCert

# Making the assumptions explicit

- Problem occurred because developer implicitly assumed that $\texttt{Kout} < \texttt{Kin}$.
- We can easily make it explicit with PVS subtypes.

McSCert

# Other Examples?

See URL `http://www.cas.mcmaster.ca/~lawford/papers` - in particular the references:

M. Lawford, P. Froebel, and G. Moum, “Application of Tabular Methods to the Specification and Verification of a Nuclear Reactor Shutdown System,” Submitted to Formal Methods in System Design (Accepted subject to minor revision June 2004). (download)

M. Lawford, J. McDougall, P. Froebel and G. Moum “Practical application of functional and relational methods for the specification and verification of safety critical software,” In T. Rus, editor, Proceedings of Algebraic Methodology and Software Technology, 8th International Conference, AMAST 2000, LNCS 1816, Springer, Iowa City, Iowa, USA, May 2000, pp. 7–88. (download)

McSCert

# We Avoided the "Two Model Trap"

## A Tale of Two Models

Many times formal methods have been "bolted onto the side" of an existing S/W development process or applied to a project after the product is developed by having the FM guru come and create another (formal) version of the requirements and/or design.

## The result

Short term: Errors were found! Papers written! FM are good!
Long term: FM guru moves on to another source of publications, the "formal" version of documents is not understood by anyone and bit rots aways into oblivion.

## The solution

One set of documents that are formal AND readable by domain experts AND easily maintained AND have tool support that integrates with the company's existing S/W process.

# But We Could Have Done so Much More!

We only scratched the surface of what could be done.
The really difficult stuff:

- Real Time Properties,
- tolerances,
- sequential behavior,
- numerical analysis results for fixed point arithmetic

was done manually - because it wasn't in the budget, and the regulator did not require it

**NOTE:**

At the time that was probably the right decision - but times have changed.

McSCert

# Everything was done manually too!

## Say what?

Tools are great, but they don't buy you as much as you think if they can be a single point of failure.

- In this case standards often will require the tool to be qualified to the level of the system they are being used on.
- We won't be seeing formally verified matlab/simulink any time soon

## All tools developed in house

OPG & AECL had to build custom one off tools to support their development method. They got the theorem prover PVS "off the shelf" .

McSCert

# Solving the Tool Qualification Problem

## The bad news:

You will, in all likelihood, need two different tools in order to avoid having to do it manually because “demonstrating soundness of the tools” will likely be difficult or impossible

## The good news:

Its not as hard as you might think to knock the tool qualification requirements down a level by doing the same thing with 2+ tools.

- DSLs can be used to generate code for multiple theorem provers, or SMT solvers, or model checkers
- There is often more than one way to get a result
- This can help avoid vendor lock-in

Consider this in developing your tools and process.

McSCert

# Is an Independent Verification Team Needed

**Question:**

If tools perform automated verification as part of the forward process, do we really need an independent Verification team?

McSCert

# Do we really need ATP in Certification?

**Question:**

Can't model checkers and/or SAT & SMT solvers do everything you need for certification?

Theorem provers will be part of certification for

- diversity - to help mitigate against bugs in the above tools
- abstraction - a detailed network of timed automata is closer to a software design than a requirements specification and it is non-trivial to write down TL formulas for the complete I/O behaviour of a system
- generality - the ability to verify things like convergence of numeric algorithms, properties of some nonlinear expression, the computation of a derivative used in deriving a requirement or as part of the design, dealing with complex data structures, etc.

McSCert

# Do I still need to test for certification?

Yes. Don't sell formal verification as a way to reduce testing, it shouldn't.

- Formal verification is done on models of the system.
- Testing is done on the real system.

Theorem proving and other formal verification tools can help:

- generate test cases (e.g. creative use of PVS random test, proofs $\rightarrow$ tests, etc.)
- SMT solvers to hit all cells of a table
- & model checkers to generate longer test sequences with specific properties
- formal models as oracles - e.g. PVSio can be used to execute most of the Darlington tables

FM can certainly help reduce the cost of testing!

McSCert

# How do Formal Verification & Certification relate?

- Certifying (licensing, regulatory) authorities audit - be it process or product based, by looking at samples or checking parts of the work
- The regulator on Darlington (the Atomic Energy Control Board - now the CNSC) only audited the verification results by checking samples

**But . . .**

Automated tools let you “audit everything” - just rerun all the tools.

McSCert

# Lessons Learned

- Mathematically based requirements - a crucial first step. If we don't do this we cannot perform mathematical verifications
  - Not forced to do this - but helps with certification! Long term: better if done in forward going process
- A priority was that domain experts would be able to read and understand all the details
  - Not forced to do this - but helps with certification and increases likelihood documents will be used!
- Tools make regression verification possible.

McSCert

# Research problems

Guaranteeing semantic consistency between

- models for different provers
- formal models and binary running on the system
- Formal models used for V&V and engineering modeling tools (e.g. Matlab/Simulink, MapleSim, etc)

How to best integrate the tools with:

- the whole software engineering process,
- the certification process

McSCert