Practical Application of Functional and Relational Methods for the Specification and Verification of Safety Critical Software

Mark Lawford
Assistant Professor
Dept. of Computing And Software
Faculty of Engineering
McMaster University

lawford@mcmaster.ca

Credits

Joint work with:

Jeff McDougall
JKM Software Technologies Inc.
jkm@pathcom.com

Peter Froebel & Greg Moum,
Ontario Power Generation Inc.
peter.froebel@ontariopowergeneration.com
g.moum@ontariopowergeneration.com

Additional contributions:
Mike Viola
Manager
Software Engineering Standards & Methods

and

Paul Joannou
Director
Engineering Standards

both of Ontario Power Generation Inc.

Outline

- Motivation
- Mathematical Preliminaries
- Functional 4-Variable Model & its decomposition
- Example of Functional Limitations
- Relational 8-Variable Model
- Conclusions & Further Research

Motivation: Reactor Shutdown System (SDS)

What is an SDS?

- 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

Motivation: (cont)

The CANDU Computer Systems Engineering Centre of Excellence *Standard for Software Engineering of Safety Critical Software* (Joannou *et al.*) 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."*

Why use functions?

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

Motivation (cont)

Verification uses tabular methods, custom tools and SRI's PVS automated proof assistant to handle typechecking and proof details.

Over 100 verification "blocks" in system. Each block typically has multiple inputs.

Goals:

- Make formal methods practical: Minimize effort required to perform systematic design verification.
- Address functional limitations by formalizing tolerance arguments.

How?

- Use of simple algebraic properties can reduce effort required for formal verification.

- Introduce 8-variable model.

Preliminaries

Let $g : V_1 \to V_2$ and $h : V_2 \to V_3$, then $h \circ g$ denotes *functional composition*.

Let $G \subseteq V_1 \times V_2$ and $H \subset V_2 \times V_3$, then $G \bullet H$ denotes *relational composition*.

Given equivalence relations $E_1, E_2 \subseteq V \times V$, define partial order $E_1 \leq E_2$ iff
$(\forall v, v' : V)((v, v') \in E_1 \Rightarrow (v, v') \in E_2)$.

Given $f : V_1 \to V_3$, define $\text{ker}(f)$, the *equivalence kernel of* f to be the equivalence relation given by: $(v_1, v_1') \in \text{ker}(f)$ iff $f(v_1) = f(v_1')$.

Claim:

$$\begin{tikzcd}
V_1 \arrow[rr, "f"] \arrow[dr, "g"'] & & V_3 \\
& V_2 \arrow[ur, dashed, "h"'] &
\end{tikzcd}$$

$$(\exists h : V_2 \to V_3) f = h \circ g \text{ iff } \text{ker}(g) \leq \text{ker}(f)$$

4-Variable Model (Parnas & Madey)

$$\begin{array}{ccc} \mathbf{M} & \xrightarrow{REQ} & \mathbf{C} \\ {\scriptstyle IN}\downarrow & & \uparrow{\scriptstyle OUT} \\ \mathbf{I} & \xrightarrow{SOF} & \mathbf{O} \end{array}$$

$\mathbf{M}$ - Monitored Variables statespace

$\mathbf{C}$ - Controlled Variables statespace

$\mathbf{I}$ - Input Variables statespace

$\mathbf{O}$ - Output Variables statespace

$\mathbf{M}, \mathbf{C}, \mathbf{I}, \mathbf{O}$ 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:

$$REQ = OUT \circ SOF \circ IN \qquad (1)$$

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

“Vertical” Decomposition

$$
\begin{tikzcd}
\mathbf{M} \arrow[rrr, "REQ"] \arrow[d, "IN"'] \arrow[dr, "Abst_M"] & & & \mathbf{C} \arrow[dl, "Abst_C"'] \\
\mathbf{I} \arrow[r, "SOF_{in}"] & \mathbf{M}_p \arrow[r, "SOF_{req}"] & \mathbf{C}_p \arrow[r, "SOF_{out}"] & \mathbf{O} \arrow[u, "OUT"']
\end{tikzcd}
$$

$$
\begin{aligned}
Abst_C \circ REQ &= SOF_{req} \circ Abst_M && (2)\\
Abst_M &= SOF_{in} \circ IN && (3)\\
id_{\mathbf{C}} &= OUT \circ SOF_{out} \circ Abst_C && (4)
\end{aligned}
$$

$\mathbf{M}_p$ and $\mathbf{C}_p$ are the *pseudo-monitored* and *pseudo-controlled* “variables” corresponding to software’s internal representation of monitored and controlled quantities

(2) represents main verification block

(3) and (4) represent hardware hiding modules

Hardware Hiding Example

E.g. temperature of the primary heat transport system which belongs to $\mathbf{M}$ might have a value of 500.3 Kelvin.

A/D converters map this via IN to a value of 3.4 volts in a parameter of $\mathbf{I}$.

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 $\mathbf{M}_p$.

“Vertical” Decomposition (cont)

More “vertical” decomposition obtained by isolating outputs. In effect,

i) projecting $\mathbf{C}$ onto single output

ii) restricting REQ to relevant subset of $\mathbf{M}$

Note “wrong way” $Abst_C$ arrow - used to reduce number of required abstraction functions

Can reduce by up to 1/2 number of abstraction functions required.

Proof obligation (4) precludes possibility of trivial implementations

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

"Horizontal" Decomposition

$$
\begin{tikzcd}
\mathbf{M} \arrow[r, "REQ_1"] \arrow[rrrrr, bend left, "REQ"] \arrow[d, "Abst_M"'] & \mathbf{V}_1 \arrow[r, "REQ_2"] \arrow[d, "Abst_{V_1}"'] & \mathbf{V}_2 \arrow[d, "Abst_{V_2}"'] & \dots & \mathbf{V}_{n-1} \arrow[r, "REQ_n"] \arrow[d, "Abst_{V_{n-1}}"] & \mathbf{C} \arrow[d, "Abst_C"] \\
\mathbf{M}_p \arrow[r, "SOF_1"] \arrow[rrrrr, bend right, "SOF_{req}"'] & \mathbf{V}_{1p} \arrow[r, "SOF_2"] & \mathbf{V}_{2p} & \dots & \mathbf{V}_{(n-1)p} \arrow[r, "SOF_n"] & \mathbf{C}_p
\end{tikzcd}
$$

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

$$SOF_i \circ Abst_{V_{i-1}} = Abst_{V_i} \circ REQ_i \qquad (5)$$

Cost of decomposition? Verifier must provide cross reference in form of $Abst_{V_i} : V_i \rightarrow V_{ip}$.

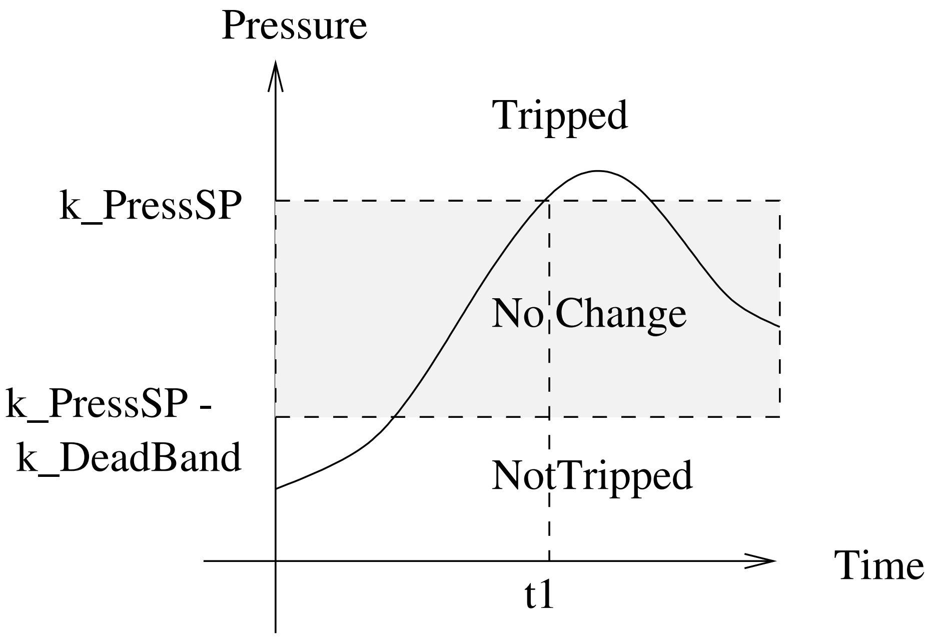
Now we see benefit of "wrong way" arrow: Same $Abst_{V_i}$ can be used on output then input of successive blocks.

Note: Only need to check invertibility of $Abst_C$ to satisfy (4).

Pressure Sensor Trip Example

Idea: when pressure exceeds setpoint, reactor is “tripped” (shutdown).

Deadband where no change occurs is used to eliminate sensor “chatter”.



Required behavior is specified by function f_PressTrip and implemented by PTRIP.

In the function definitions, f_PressTripS1 and PREV are corresponding state variables.

Pressure Sensor Trip Example (cont)

Abstraction functions $posreal2AI$ and $Trip2bool$ map abstract datatypes to concrete implementations.

$posreal2AI$ models the A/D conversion by taking integer part of sensor values in [0, 5000] mV range.

Return type, AI, consists of integers between 0 and 5000.

Resulting block comparison commutative diagram is:

$$\begin{tikzcd}
\text{posreal} \times \text{Trip} \arrow[r, "f_PressTrip"] \arrow[d, "posreal2AI \times Trip2bool"'] & \text{Trip} \arrow[d, "Trip2Bool"] \\
\text{AI} \times \text{bool} \arrow[r, "PTRIP"] & \text{bool}
\end{tikzcd}$$

PVS for Pressure Sensor Trip

sentrip : theory
  begin

    k_PressSP : int = 2450
    k_DeadBand : int = 50
    KDB : int = k_DeadBand
    KPSP : int = k_PressSP

    Trip : type = {Tripped, NotTripped}
    AI : type = subrange(0, 5000)

f_PressTrip(Pressure : posreal, f_PressTripS1 : Trip) : Trip
  = table

Pressure $\leq$ k_PressSP − k_DeadBand	NotTripped
k_PressSP − k_DeadBand < Pressure $\wedge$ Pressure < k_PressSP	f_PressTripS1
Pressure $\geq$ k_PressSP	Tripped

endtable

  PTRIP(PRES : AI, PREV : bool) : bool = table

PRES $\leq$ KPSP − KDB	FALSE
KPSP − KDB < PRES $\wedge$ PRES < KPSP	PREV
PRES $\geq$ KPSP	TRUE

  endtable

Trip2bool(TripVal : Trip) : bool = table

TripVal = Tripped	TRUE
TripVal = NotTripped	FALSE

endtable

PVS for Pressure Sensor Trip (cont)

posreal2AI(x : posreal) : AI = table

$x \leq 0$	0
$0 < x \wedge x < 5000$	floor(x)
$x \geq 5000$	5000

endtable

Sentrip1 : theorem
    ($\forall$ (Pressure : posreal, f_PressTripS1 : Trip) :
        Trip2bool(f_PressTrip(Pressure, f_PressTripS1)) =
      PTRIP(posreal2AI(Pressure), Trip2bool(f_PressTripS1)))

end sentrip

Attempting block comparison theorem Sentrip1 reduces to proving for all inputs:

$$\neg(\text{f\_PressTripS1} = \text{Tripped} \wedge$$
$$2400 < \text{Pressure} < 2450 \wedge \text{floor(Pressure)} \leq 2400)$$

Counter examples result when Pressure $\in (2400, 2401)$ and f_PressTripS1 $= Tripped$.

In fact there does not exist a design that can satisfy requirement `f_PressTrip` since

$$\text{ker}(posreal2AI \times Trip2bool) \not\leq \text{ker}(f\_PressTrip)$$

Pressure Sensor Trip Example (cont)

While it is possible to “fix” specification so that it is implementable by changing inequalities as follows:

f_PressTrip(Pressure : posreal, f_PressTripS1 : Trip) : Trip
$=$ table

Pressure $<$ k_PressSP $-$ k_DeadBand	NotTripped
k_PressSP $-$ k_DeadBand $\leq$ Pressure $\wedge$ Pressure $<$ k_PressSP	f_PressTripS1
Pressure $\geq$ k_PressSP	Tripped

endtable

in practice this difference is a mathematical irrelevancy since A/D converters have $\pm$5mV accuracy.

Want to:

- maintain functional requirements specification and design description, and
- use formal mathematical proofs incorporating tolerances when necessary without an excessive increase in proof complexity and documentation.

8-Variable Model

$$\begin{tikzcd}
\mathbf{FM} \arrow[r, "MTRAN"] & \mathbf{BM} \arrow[r, "MTOL"] \arrow[d, "IN"'] & \mathbf{IM} \arrow[r, "REQ"] & \mathbf{IC} \arrow[r, "CTOL"] & \mathbf{BC} \arrow[r, "CTRAN"] & \mathbf{FC} \\
 & \mathbf{I} \arrow[rrr, "SOF"] & & & \mathbf{O} \arrow[u, "OUT"'] &
\end{tikzcd}$$

REQ and SOF still functions $MTOL$, $CTOL$ are Input/Output tolerance relations

$$REQUIREMENTS = MTOL \bullet REQ \bullet CTOL$$

$$DESIGN = IN \bullet SOF \bullet OUT$$

Design verification amounts to showing:

$$\begin{array}{lll} DESIGN & \subseteq & REQUIREMENTS, \text{ and} \quad (6) \\ DESIGN & & \text{is total} \quad (7) \end{array}$$

8-Variable Model (cont)

In the standard 4-variable model this is modeled by the relation $NAT \subseteq \mathbf{M} \times \mathbf{C}$. In the case of the proposed 8-variable model we could have $NAT \subseteq \mathbf{BM} \times \mathbf{BC}$. In this case (7) could be replaced by the requirement:

$$dom(NAT) \subseteq dom(DESIGN) \qquad (8)$$

Sensor Trip Revisited

Revised block comparison theorem is easily proved:

$$
\begin{array}{l}
\mathsf{Sentrip1:\ theorem} \\
(\forall\ (\mathsf{Pressure:\ posreal}, \mathsf{f\_PressTripS1:\ Trip}) \\
(\exists\ (\mathsf{Pressure2} : \{(x : \mathsf{posreal}) | \mathsf{Pressure} - 5 \leq x \leq \mathsf{Pressure} + 5\}) : \\
\quad \mathsf{Trip2bool(f\_PressTrip(Pressure2, f\_PressTripS1))} = \\
\qquad \mathsf{PTRIP(posreal2AI(Pressure), Trip2bool(f\_PressTripS1))))}
\end{array}
$$

Conclusions

- Tool supported functional 4-variable model using tabular methods has been successfully applied to Darlington SDS

- Decomposition of 4-var model reduces effort required to perform and document verification process

- Simple algebraic "tricks" can make formal methods more practical for industry

- 8-variable Relational model preserves functional specification of requirements and implementation & addresses functional limitations

Future Work

- automating verification and re-verification tasks

- modeling & verification of concurrent real-time properties with tolerances

- equivalence verification, model-checking & model reduction