A Tabular Expressions Toolbox for Critical Systems Development in Simulink

Mark Lawford

Professor, Department of Computing and Software
Associate Director
McMaster Centre for Software Certification (McSCert)
McMaster University
Hamilton, ON, Canada

SAFE & SECURE SYSTEMS & SOFTWARE SYMPOSIUM
June 12-14, 2012

Outline

1. Motivation
2. Preliminaries
   - Tabular Expressions
   - Previous Table Tools & Workflows
3. The Tabular Expression Toolbox
   - Features
   - CVC3 & PVS Integration
4. Handling IEEE Floating Point Numbers
5. Conclusions and Future Research

Formal Specifications are Good!

- Give a precise description of required behavior of a system.
- Usually involve quite a bit of mathematical notation.

Claims about formal methods:

- Can be analyzed using sophisticated tools
  - help to find design faults earlier,
  - find faults that are unlikely to be found by other methods.
- Can be used to support testing.
- Help developers to produce better systems.
- Help maintainers to evolve the system effectively.
- Can help meet DO-178c via DO-333.

McMaster University

So . . . Why Don't People Use Them?

The $W0_k(x)$ process describes a watchdog where the last input toggle was to 0 and there are x steps left until shut-down trigger. $W1_k$ is the same except that the last input toggle was to 1.

$$
\begin{aligned}
W0_k(0) &= [!\{d\}\,?X \rightarrow W0_k(0)] \\
W1_k(0) &= [!\{d\}\,?X \rightarrow W1_k(0)] \\
\forall\, x \geq 1 &: \\
W0_k(x) &= [!\emptyset\,?X \rightarrow \\
&\quad W1_k(k) \lhd (w \in X) \rhd W0_k(x-1)] \\
W1_k(x) &= [!\emptyset\,?X \rightarrow \\
&\quad W0_k(k) \lhd (w \notin X) \rhd W1_k(x-1)] \quad (29)
\end{aligned}
$$

- Writing *and* reading the specifications is hard.
- There are often errors in the specifications.
- Specifications aren't (kept) consistent with the code.
- Tools don't add enough value to justify the effort.

Tabular Expressions for Software Development

- Have been used extensively at NRL (e.g. A-7) [Heninger, 1980].
- Tabular expressions were used at Darlington Nuclear Generator for Shutdown Systems requirements and design documents.
- Readable by domain engineers, operators, testers . . . and developers!
- Used properly, a Formal development process using tabular expressions can help meet DO-178c (via DO-333).

Tabular Expressions - A Usable "Formal" Method

Example

2-Dimensional Table

$$f(x,y) = \begin{cases} x^2 - y^2, & (((y < 0) \wedge (x < 0)) \vee ((y < 0) \wedge (x > 0)) \vee ((x = 0) \wedge (y = 0))) \\ x + y, & (((y = 0) \wedge (x < 0)) \vee ((y = 0) \wedge (x > 0)) \vee ((y > 0) \wedge (x = 0))) \\ x^2 + y^2, & (((y < 0) \wedge (x = 0)) \vee ((x < 0) \wedge (y > 0)) \vee ((x > 0) \wedge (y > 0))) \end{cases}$$

f(x,y)=

	$x < 0$	$x = 0$	$x > 0$
$y < 0$	$x^2 - y^2$	$x^2 + y^2$	$x^2 - y^2$
$y = 0$	$x + y$	$x^2 - y^2$	$x + y$
$y > 0$	$x^2 + y^2$	$x + y$	$x^2 + y^2$

So why isn't everyone using tables?

They only show significant benefits when used in a process with integrated tool support - which was (generally) lacking - until now! ;-)

Tabular Expressions

- Pioneered by David Parnas.
- Represent mathematical conditional expressions formally and graphically.

Example

Let x be a real valued variable. Then the *sign* function and its equivalent tabular representation are:

$$sign(x) = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0 \end{cases} \iff$$

$x < 0$	$x = 0$	$x > 0$
-1	0	1

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 - $(c_1 \vee c_2 \vee \ldots \vee c_n) \leftrightarrow \top$

Tabular Expression Semantics

Tabular expressions have a well defined semantics. Recently [Jin and Parnas, 2010] has defined a consistent semantics for all known tabular expression types used in practice.
A table type is defined by:

1. Constituents - dimensions, indexed-sets giving condition grids and results grids
2. Auxiliary functions - how grids are evaluated or properties constituents should satisfy, e.g., predicate to evaluate if grid is "Proper"
3. Restriction schema - e.g. complete and disjoint condition headers for normal function tables
4. Evaluation schema - formal semantics of how you evaluate a table type.

Why Tables Work

Example

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

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

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

You can actually read them.

TTS: The Table Tool System [Parnas and Peters, 1999]

- Developed by Parnas et al to demonstrate possibilities of table tools
- tried to support every type of table - but did not at that time have a consistent semantics for all table types
- was an academic tool - with all that implies

NRL's SCR* Toolset

- Build on tabular methods used on the A-7 [Heninger, 1980] project.
- Heitmeyer *et al.* have made extensive use of the Software Cost Reduction (SCR) tabular methods supported by the "light-weight" SCR* tool suite
- Used extensively for the creation and analysis of requirements for industrial and military software applications (e.g., [Heitmeyer et al., 1998]).
- allows users to to incorporate more heavy duty analysis tools such as the explicit state model checker SPIN [Holzmann, 1997] with SCR*.
- Closed source, restrictive license, not commercially available toolset.

Fillmore Tools Eclipse Plugin [Peters et al., 2007]

To Do Table Tool Right We Tried to:

1. have a comprehensive table semantics Parnas *et al* 2006
2. use “Standard” way to encode semantics in documents OMDoc
3. have tools for verification, test case & code generation PVS,FCN,...
4. with a means of translating semantic content between these tools XSLT
5. in a tool developers actual use to tie it all together Eclipse

Problem: This still takes a lot of heavy lifting to get a usable tool.

How were tables used for Darlington NGS?

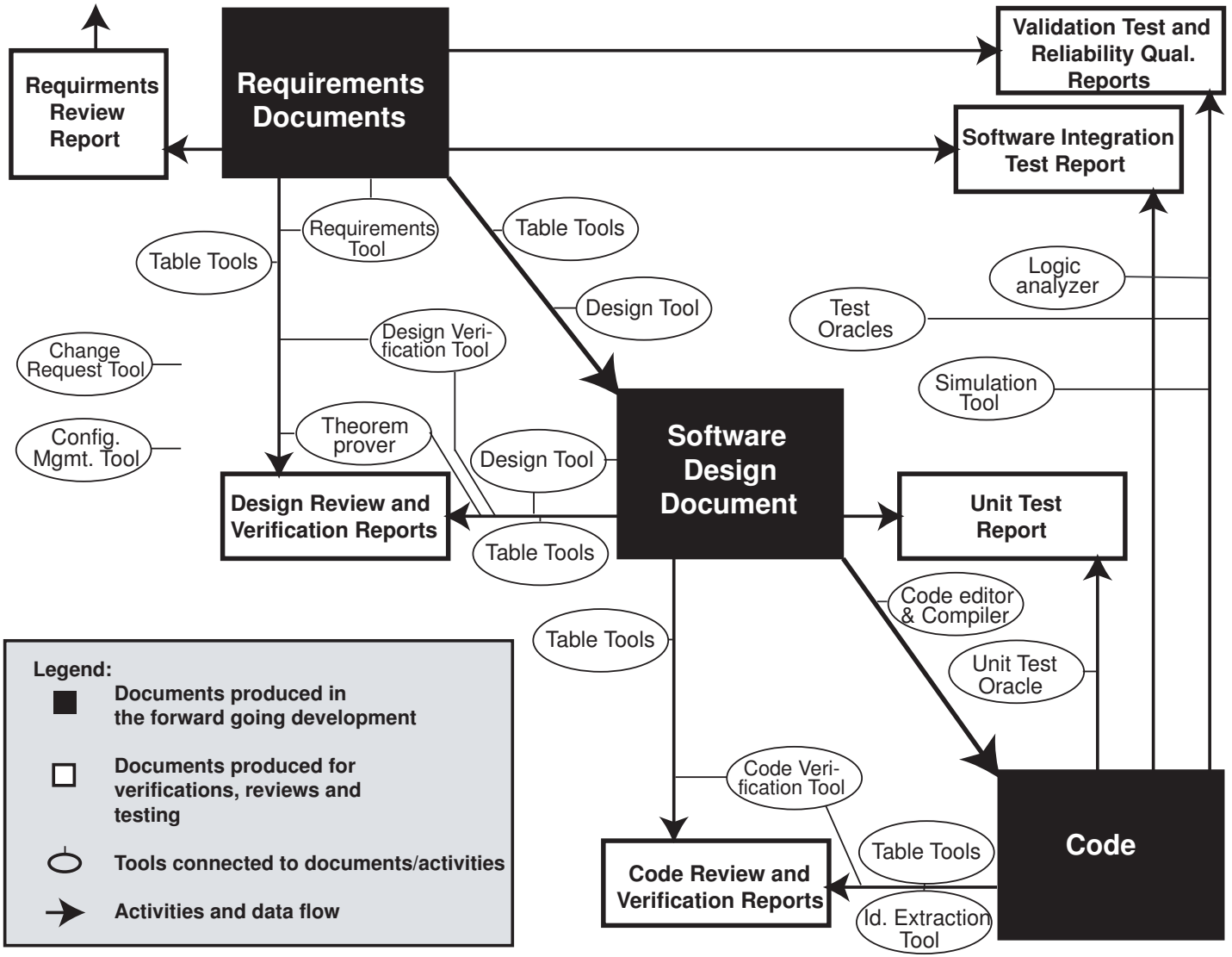
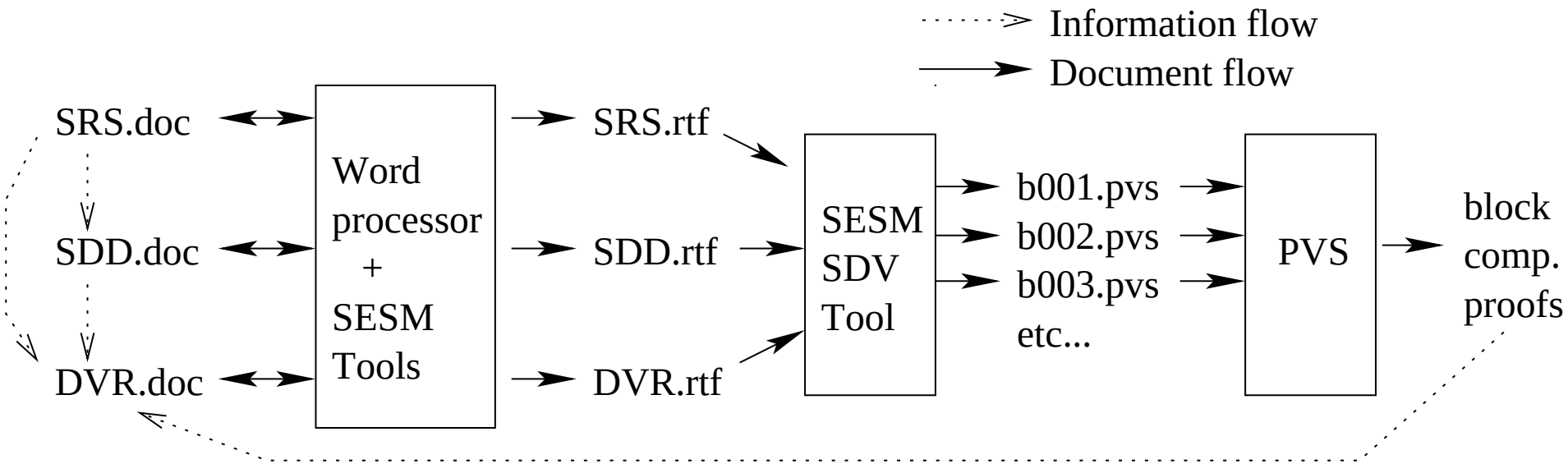


Table Tool Workflow on Darlington NGS SDS



Good:	**Bad:**
Tables were readable	but tedious to create
Tools found errors	but difficult to use & interpret errors
Had “formal” semantics	But AECL/OPG built EVERYTHING!

Tabular Expression Toolbox (TET)

Decided to develop a Matlab/Simulink toolbox:

Advantages:

- Model Based Design has been shown to reduce cost and improve quality of software development
- Focus engineer's time on early life cycle processes (modeling, simulation, analysis), and automate late life cycle activities (coding, testing)
- Matlab/Simulink industry standard.
- Advanced code generation tools for C, VHDL, Verilog.
- Existing research/tools on adding formalizations to Simulink.

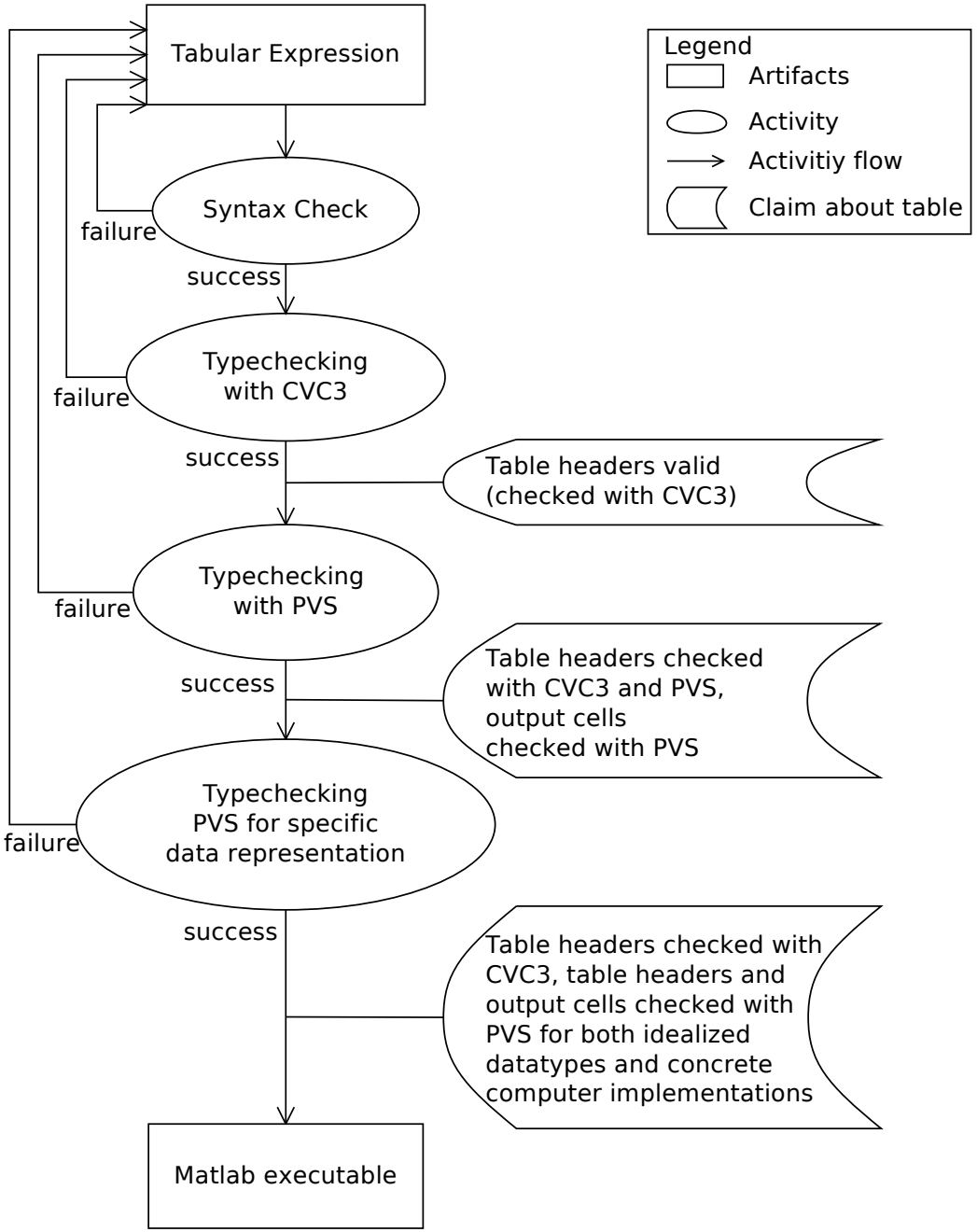
Pitfalls:

- Semantics of MBD tools are dubious and a moving target
- It's an *academic tool* (but it's open source)

Tabular Expressions Toolbox (TET)

- Provides Simulink block for creating tabular expressions.
- GUI for creating 1/2D tables with nested headers, single/multiple outputs.
- Supports code generation through embedded Matlab language.
- Integrates with CVC3 SMT solver and PVS theorem prover for checking disjointness and completeness conditions.
- For improper tables, tool attempts to generate counter example and clearly show user why table is improper.
- Available Now!

`http://www.mathworks.com/matlabcentral/fileexchange/28812-tabular-expression-toolbox`



TET Workflow

1. Perform quick Matlab syntax check
2. Checking with CVC3 on $\mathbb{R}$ is fast, often detects logic errors in headers
3. Checking with PVS on $\mathbb{R}$ provides additional assurance, type checks results grid
4. Checking with PVS on IEEE floats shows absence of under/overflow, etc.

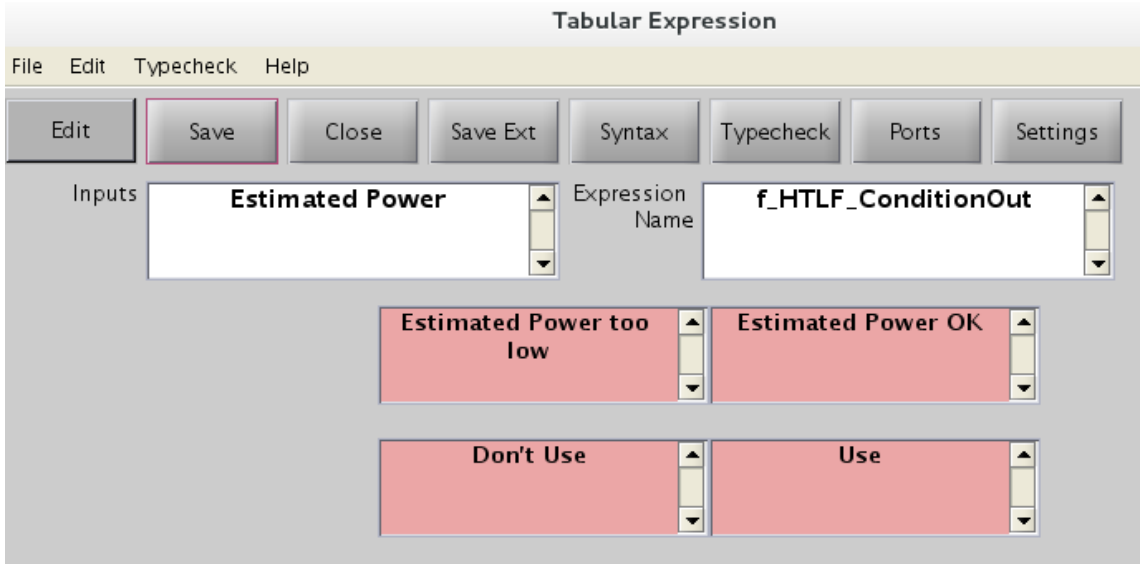
McMaster University

Tabular Expression Formats & Features

- The tool supports one dimensional and two dimensional normal function tables
- tool supports multiple output for single dimensional tables, and single output for 2 dimensional tables,
- supports nested headers (Parnas' "circular" tables) along one dimension.
- has limited "undo" feature for both expression edits and graphical (i.e. delete a row) edits.

Matlab syntax checking and Visual Highlighting

Clicking on the “Check” button uses Matlab’s Syntax checking highlights in red any cells with syntax errors or that are empty.



Requirement

Heat Transport Low Flow Trip shall be Conditioned Out at low power

Save to M-file

From the table edit window, selecting `File -> Save to M-file`

- Immediately lets you execute your specification
- You can generate C code or HDL code
- You can apply other formal tools - e.g. Polyspace, etc.

Completeness and Consistency Checking

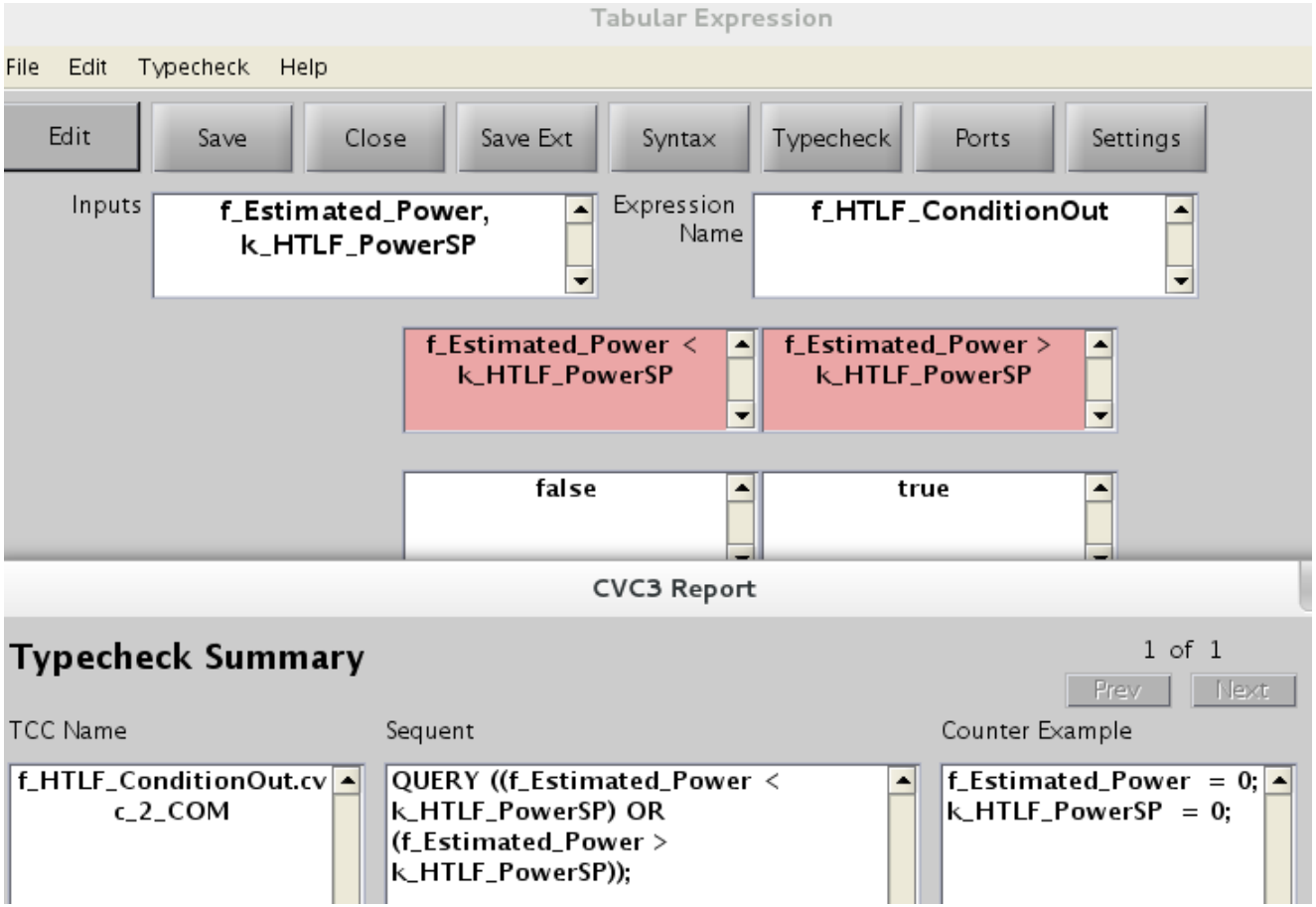
When checks fail, getting useful information about why is important.

- Counter examples currently generated by
  - CVC3 SMT solver, and/or
  - PVS' "random-test" feature.
- Gives input values for counter example, and
- graphical feedback highlighting the error.

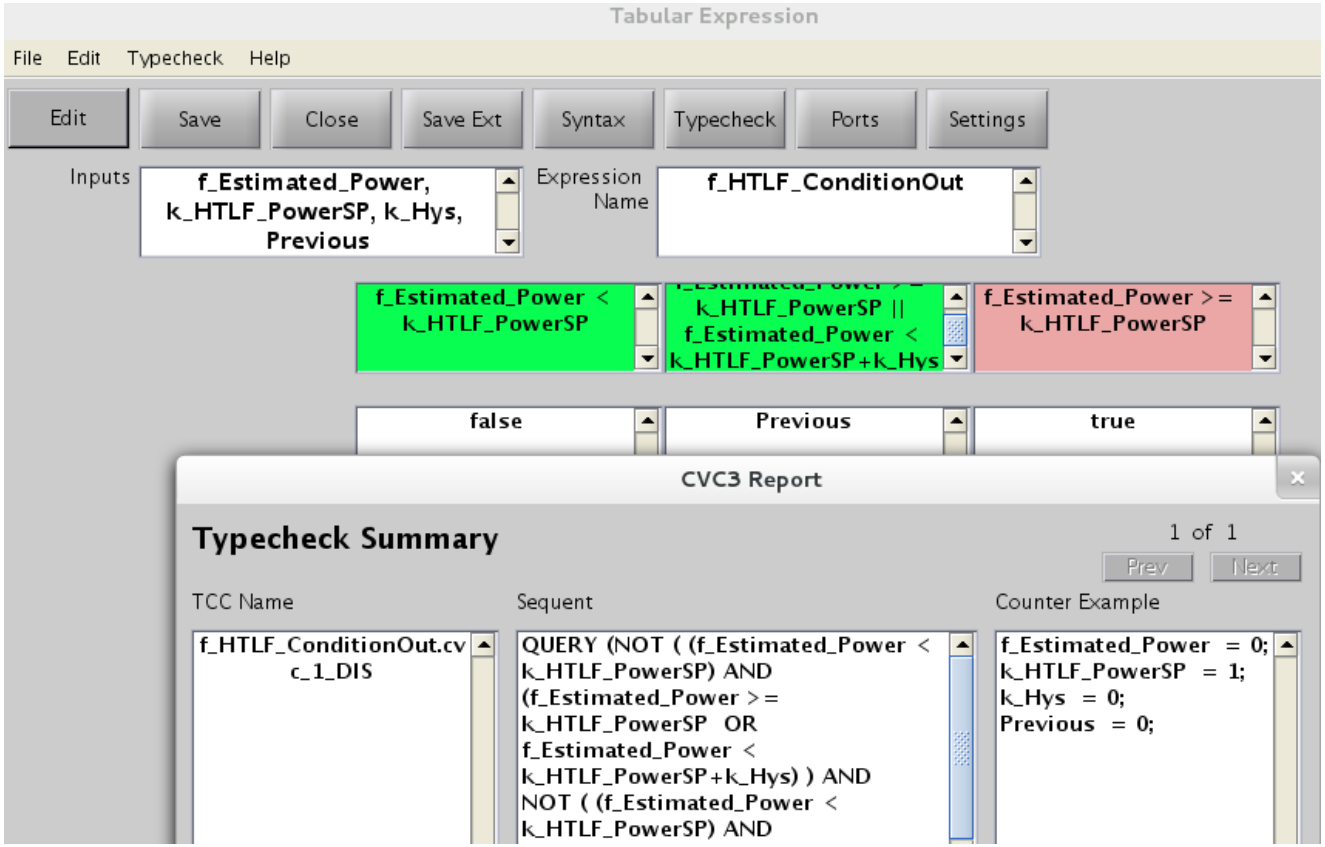
Red is used to show conditions in headers that the counter example makes FALSE.

Green is used to show conditions in headers that the counter example makes TRUE.

Counter Example Generation: Completeness



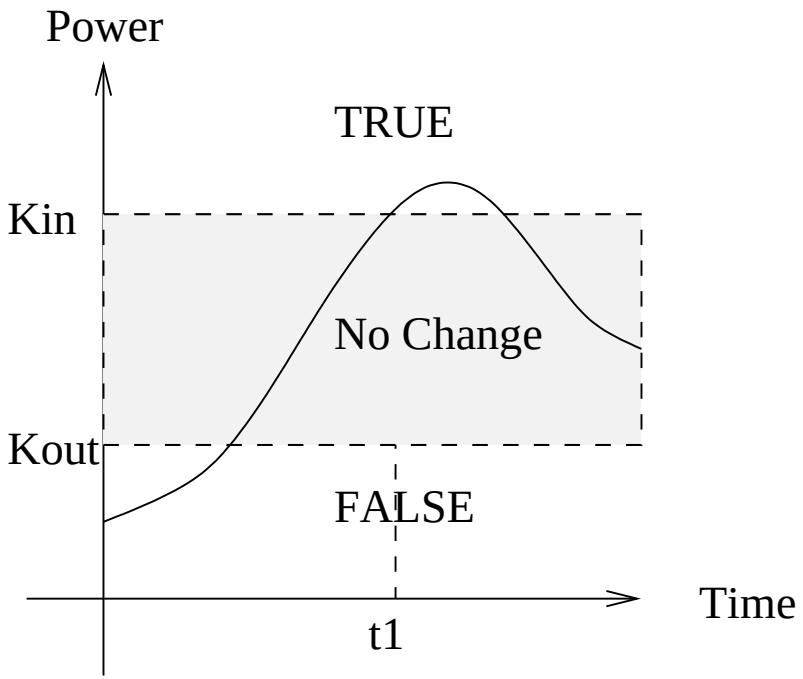
Counter Example Generation: Disjointness



Requirement

Use hysteresis on setpoints to eliminate sensor chatter.

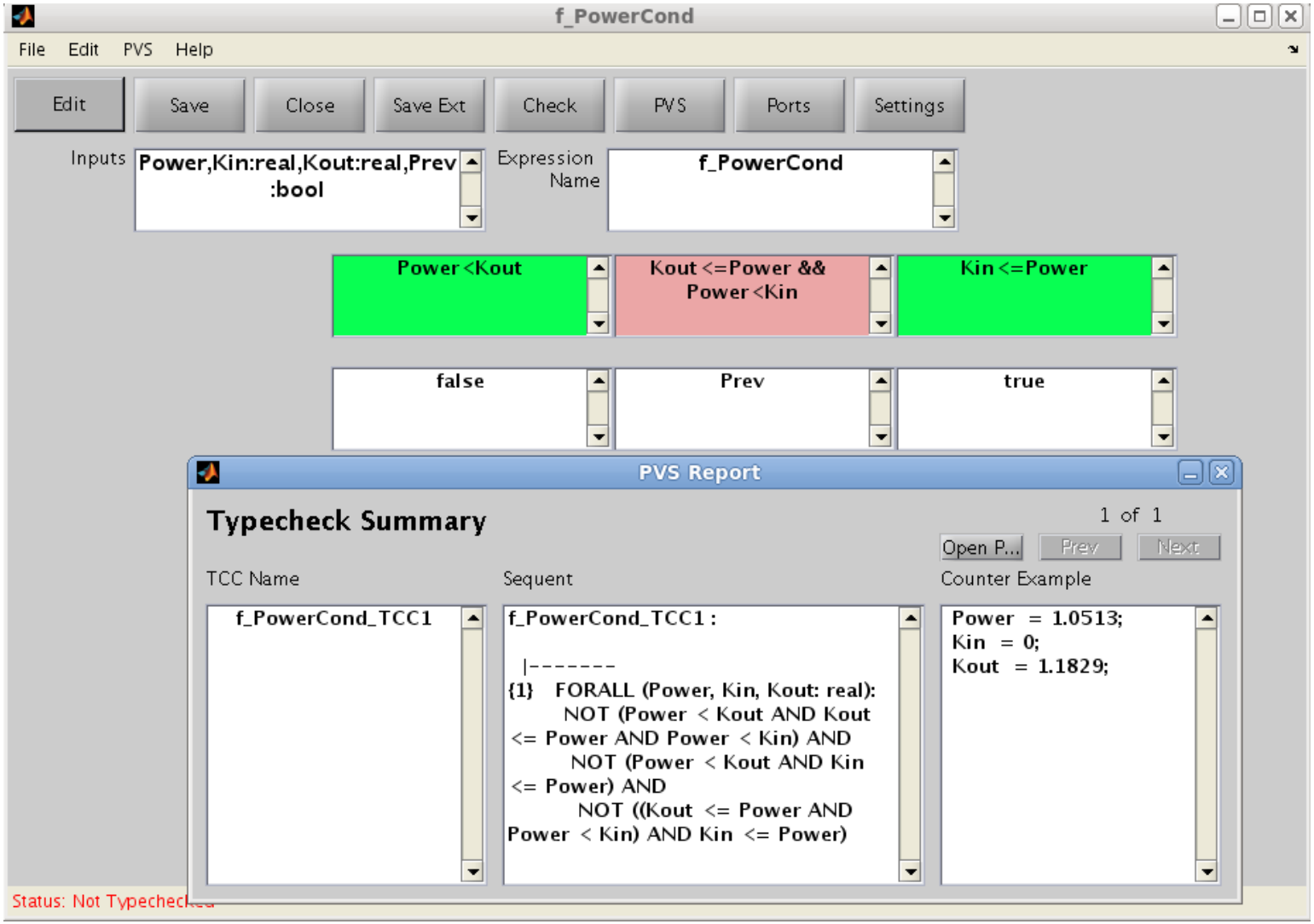
PVS (sub)Typing



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

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

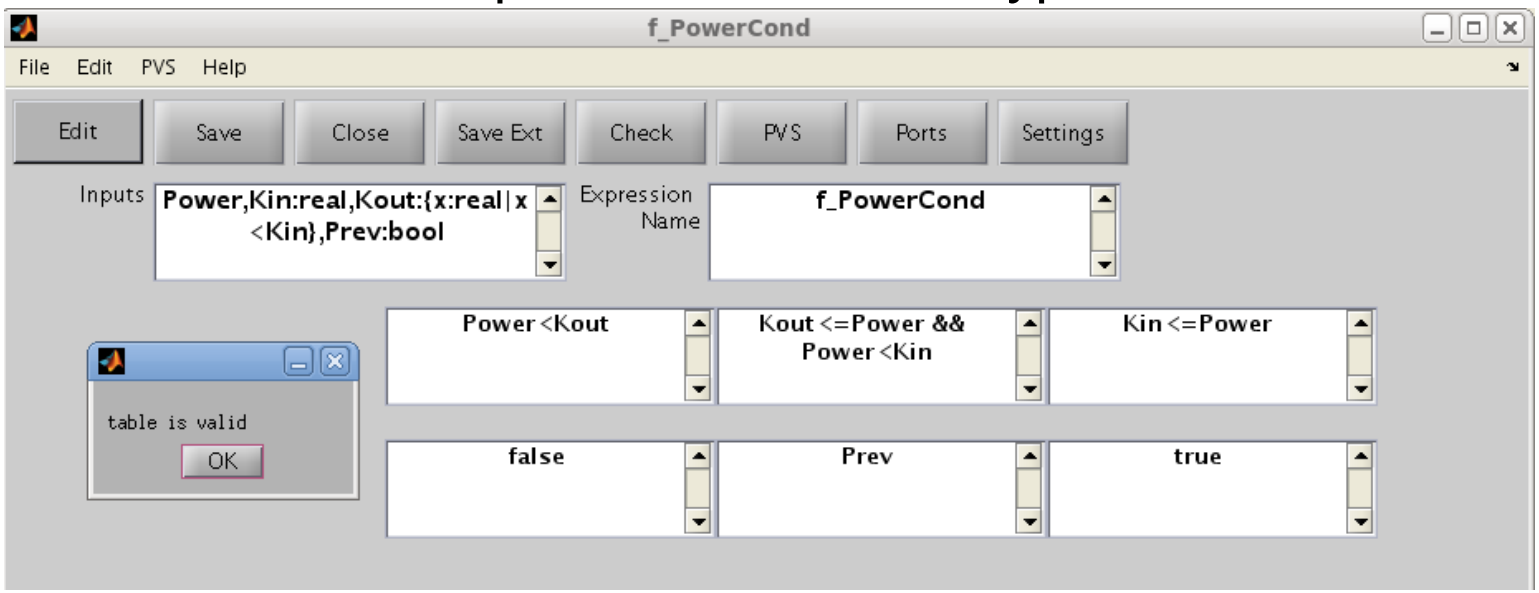
PVS (sub)Typing



PVS (sub)Typing

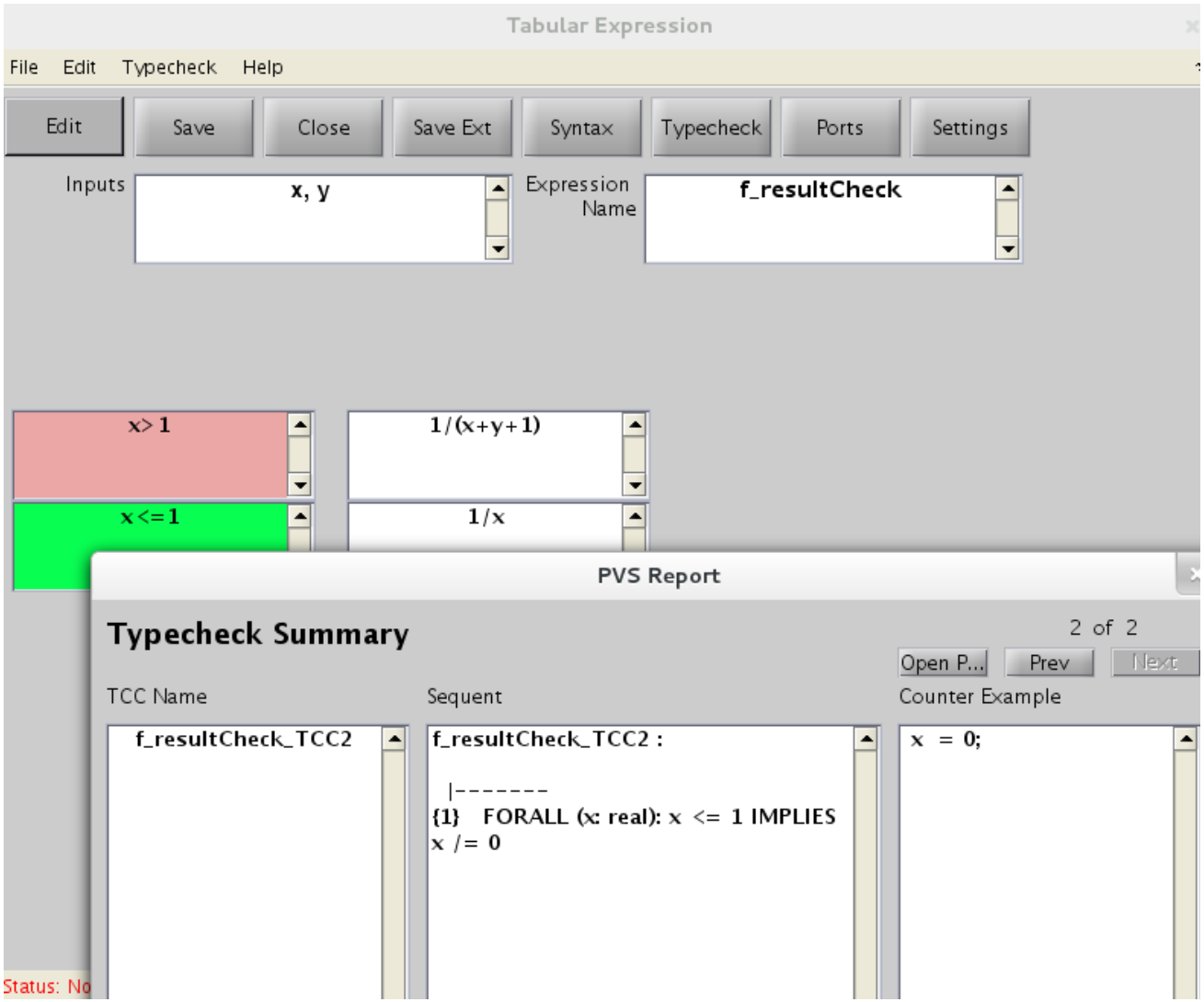
Problem occurred because developer implicitly assumed that `Kout < Kin`.

We can make it explicit with PVS subtypes.



Note: We still need to develop tool to check input typing is satisfied when table with PVS subtyping is used!

Counter Example Generation: Results Defined



Proof of well defined-ness of divide operation fails for $x = 0$

Problems with Floating Point values

- floating point numbers have maximums and minimums
- floating point arithmetic does not have the same axioms as under reals, ie. associativity does not hold.
- Operations on floats will, under certain scenarios, produce a reserved value NaN which cannot be compared to other values.

Reals vs Floating Points

- Both PVS and CVC3 have support for the theoretical reals.
- In the case of PVS almost all the existing theories and strategies involve the real type.
- Using the reals is much faster and more intuitive than attempting a floating point representation for proving tables.
- Our approach involves using step-wise refinement:
  1. Initial type checking shall be performed using the real type, this will uncover many of the logical errors.
  2. Further refinement involves using lower level representations (floats, integers, etc.) This step will discover possible overflow, rounding, and other errors common with numerical representations.

An Example: Numerically improved quadratic solver

Example

Conditions			Results: **solution caption**	x_1	x_2
$a = 0$	$b = 0$		Ill defined	*NaN*	*NaN*
	$b \neq 0$		One Root	$-\frac{c}{b}$	*NaN*
$a \neq 0$	$b^2 - 4ac < 0$		No Real Roots	*NaN*	*NaN*
	$b^2 - 4ac = 0$		One Root	$-\frac{b}{2a}$	*NaN*
	$b^2 - 4ac > 0$	$b \geq 0$	Two Roots	$\frac{-b-\sqrt{b^2-4ac}}{2a}$	$\frac{c}{ax_1}$
		$b < 0$	Two Roots	$\frac{-b+\sqrt{b^2-4ac}}{2a}$	$\frac{c}{ax_1}$

Example: Quadratic Solver (continued)

- Table will typecheck using reals on PVS (using dependent subtyping)
- Using floating point numbers the completeness condition fails
  - $b^2 - 4ac$
  - for input scenario $b^2 \rightarrow +\infty \wedge 4ac \rightarrow +\infty$ we get $\infty - \infty$ which produces NaN.
  - When NaN compared to 0 always false
  - Can occur if

$$b > \pm\sqrt{2^{Emax+1} - 2^{Emax+1-p}} \vee ac > 2^{Emax+1} - 2^{Emax+1-p}$$

  - Need to subtype the inputs to restrict their range in order to guarantee that table is complete and disjoint.
- We can test this example in Matlab and show that the wrong result will be given.

Solution

- Using the NASA floating point libraries for PVS we've developed some lemmas to assist with proving the correctness of floating point expressions
- For purposes of checking completeness and disjointness we want to make sure that expressions evaluate to either a finite number or infinity.
- proved the above table is proper for single floating point numbers with subtype `b^2:finite`
- A sample lemma:

```
op_add_inf_fin_l: LEMMA FORALL (mode:rounding_mode,
  x:{fp:fp_num|finite?(fp) OR infinite?(fp)},
y:{fp:fp_num| finite?(fp)}):
finite?(fp_add(x,y,mode)) OR infinite?(fp_add(x,y,mode))
```

On Fixing the Weakest Link for TET

- Currently Tabular Expression Toolbox only provides “local” block check that tabular expression is well defined.
- Really need to type check block interconnections for values, measures & units
- SRI’s SimCheck tool [Roy and Shankar, 2011] provides these features for Simulink diagrams
- Plan to work with SRI to integrate TET & SimCheck to provide graphical local and global verification capabilities

Conclusions

- Tabular expressions toolbox makes it easier to use tabular expressions and increases confidence in models
- You can “Hide the Formal Verification” under the hood so the software developers will use them!
- Need to verify inter-block typing with something like SimCheck [Roy and Shankar, 2010].
- Use of formal verification at design time is very useful . . . what are the implications for independence of design and verification teams?
- I am sweeping a lot of the nasty Matlab semantics issues under the rug - though I think you can restrict to a safe subset of Matlab as in [Whalen et al., 2008].


Heitmeyer, C., Kirby, Jr., J., Labaw, B., Archer, M., and Bharadwaj, R. (1998).
Using abstraction and model checking to detect safety violations in requirements specifications.
IEEE Transactions on Software Engineering, 24(11):927–948.


Heninger, K. L. (1980).
Specifying software requirements for complex systems: New techniques and their applications.
IEEE Transactions on Software Engineering, 6(1):2–13.


Holzmann, G. J. (1997).
The model checker SPIN.
IEEE Transactions on Software Engineering, 23(5):279–295.
Special Issue: Formal Methods in Software Practice.


Jin, Y. and Parnas, D. L. (2010).
Defining the meaning of tabular mathematical expressions.
Science of Computer Programming, 75(11):980 – 1000.

Special Section on the Programming Languages Track at the 23rd ACM Symposium on Applied Computing - ACM SAC 08.


Parnas, D. and Peters, D. (1999).
An easily extensible toolset for tabular mathematical expressions.
Tools and Algorithms for the Construction and Analysis of Systems, pages 345–359.


Peters, D. K., Lawford, M., and y Widemann, B. T. (2007).
An IDE for software development using tabular expressions.
In *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 248–251, New York, NY, USA. ACM.


Roy, P. and Shankar, N. (2010).
SimCheck: An expressive type system for Simulink.
In *Methods Symposium*, page 149. Citeseer.


Roy, P. and Shankar, N. (2011).
Simcheck: a contract type system for simulink.

McMaster University

Innovations in Systems and Software Engineering, 7:73–83. 10.1007/s11334-011-0145-4.

Wassyng, A. and Lawford, M. (2003).
Lessons learned from a successful implementation of formal methods in an industrial project.
In Araki, K., Gnesi, S., and Mandrioli, D., editors, *FME 2003: International Symposium of Formal Methods Europe Proceedings*, volume 2805 of *Lecture Notes in Computer Science*, pages 133–153, Pisa, Italy. Springer-Verlag.

Whalen, M., Cofer, D., Miller, S., Krogh, B., and Storm, W. (2008).
Integration of formal analysis into a model-based software development process.
Formal Methods for Industrial Critical Systems, pages 68–84.

McMaster University