The Tabular Expressions Toolbox for Matlab/Simulink

Colin Eles and Mark Lawford

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

CASCON 2010 Affiliated Workshop
Software Certification Consortium: Certification Methods for Safety-Critical Software

McSCert

Outline

1. Introduction
2. Preliminaries
   - Table Semantics
   - Previous Table Tools
3. The Tabular Expression Toolbox
   - Features
   - PVS Integration
4. Working Example
5. Conclusions and Future Research

McSCert

Formal Specifications are Good!

- Give a precise description of required behaviour 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.

McSCert

Why Don’t People Use Them?

- 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.

McSCert

Tabular Expressions - A Useable “Formal” Method

- Previously tabular expressions were used at Darlington for Shutdown Systems requirements and design documents
- 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

Table Tools are very useful

Well, they would be really usefull if they existed in a form that was generally available.

Tabular Expression Semantics

Tabluar 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.

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:

$$sign(x) = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0 \end{cases} \quad \Longleftrightarrow \quad \begin{array}{|c|c|c|} \hline x < 0 & x = 0 & x > 0 \\ \hline \hline -1 & 0 & 1 \\ \hline \end{array}$$

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

Why Tables Work

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

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

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

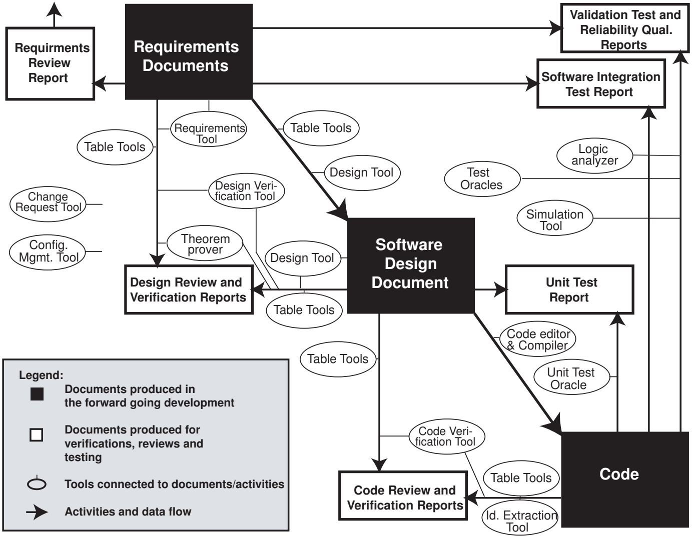
Fillmore Tools Eclipse Plugin[Peters et al., 2007]

To Do Table Tool Right We Need 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 useable tool.

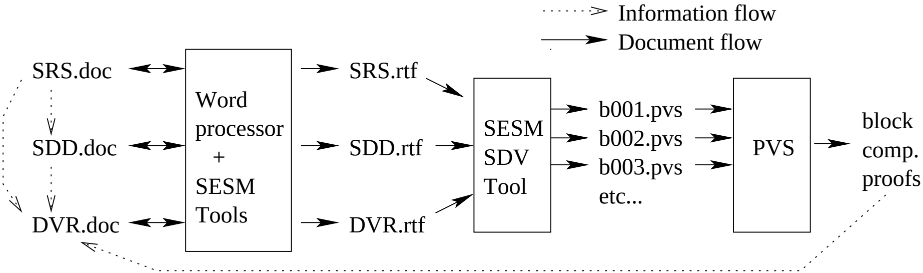
How was this done for Darlington?



Legend:

- ■ Documents produced in the forward going development
- □ Documents produced for verifications, reviews and testing
- ○ Tools connected to documents/activities
- → Activities and data flow

Tool supported tabular expressions



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!

McSCert

Tabular Expression Toolbox

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 (modelling, 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

Tabular Expressions Toolbox

- 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 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`

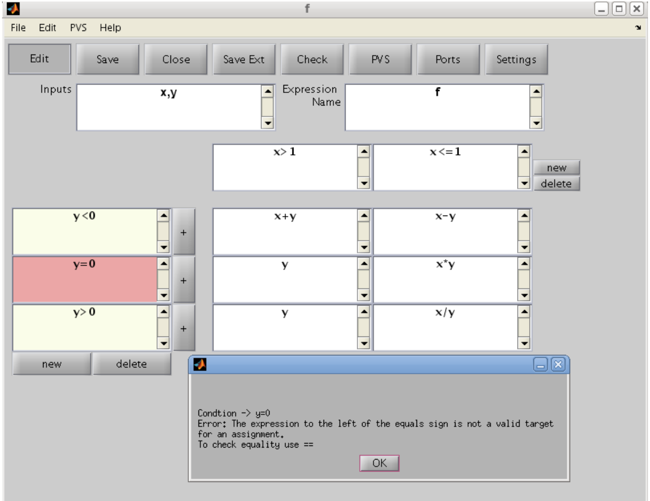
McSCert

Tabular Expression Formats & Features

- The tool supports one dimensional and two dimensional normal function tables
- tool supports multiple output for single dimensional tabels, 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.



Inputs: x,y

Expression Name: f

	x> 1	x<=1
y<0	x+y	x−y
y=0	y	x*y
y> 0	y	x/y

```
Condtion -> y=0
Error: The expression to the left of the equals sign is not a valid target
for an assignment.
To check equality use ==
```

McSCert

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
- You can apply other formal tools - e.g. Polyspace, etc.

Completeness and Consistency Checking

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

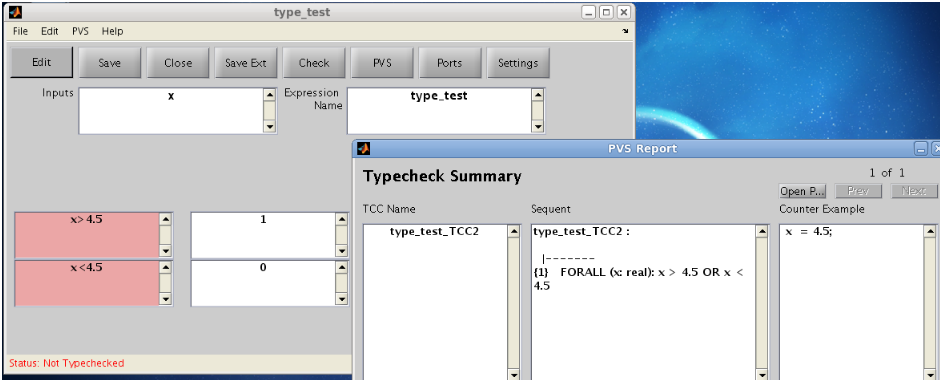
- Counter examples currently generated by PVS' "random-test" feature.
- Gives input values for counter example and
- graphical feedback highlighting the error

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.

McSCert

Counter Example Generation: Completeness



Inputs: x

Expression Name: type_test

x> 4.5	1
x<4.5	0

Status: Not Typechecked

Typecheck Summary 1 of 1

TCC Name: type_test_TCC2

Sequent:

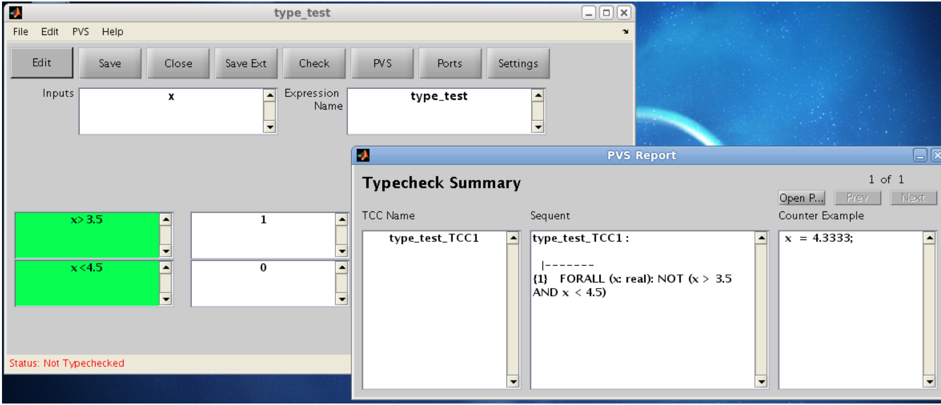
```
type_test_TCC2 :

  |-------
{1}   FORALL (x: real): x > 4.5 OR x <
4.5
```

Counter Example:

```
x = 4.5;
```

We are missing the case $x = 4.5$ because all conditions are FALSE.

Counter Example Generation: Disjointness

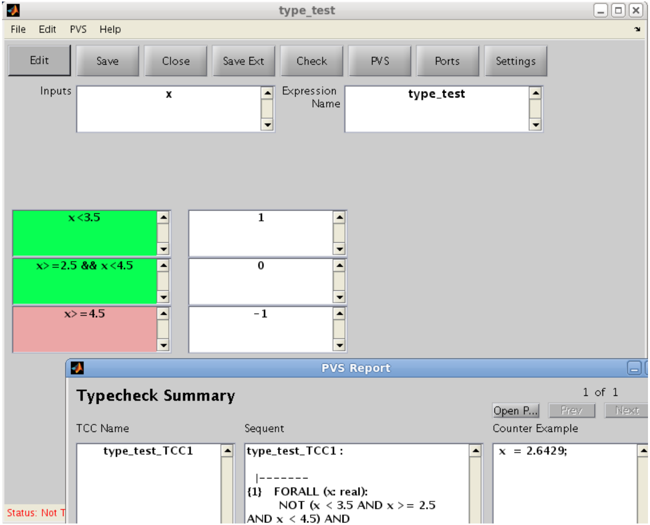


```
type_test_TCC1

type_test_TCC1 :

  |-------
{1}   FORALL (x: real): NOT (x > 3.5
AND x < 4.5)

x  =  4.3333;
```

Counter example $x = 4.3333$ makes both rows of modified table TRUE.

McSCert

Counter Example Generation: Disjointness



Inputs: x

Expression Name: type_test

x<3.5	1
x>=2.5 && x<4.5	0
x>=4.5	-1

Typecheck Summary 1 of 1

TCC Name: type_test_TCC1

Sequent:

```
type_test_TCC1 :

  |-------
{1}   FORALL (x: real):
        NOT (x < 3.5 AND x >= 2.5
AND x < 4.5) AND
```

Counter Example: x = 2.6429;

Note overlap problem is in 1st two rows since both are True.

3rd row is False. So its not part of the problem.

(Start of) Simulink Type Integration

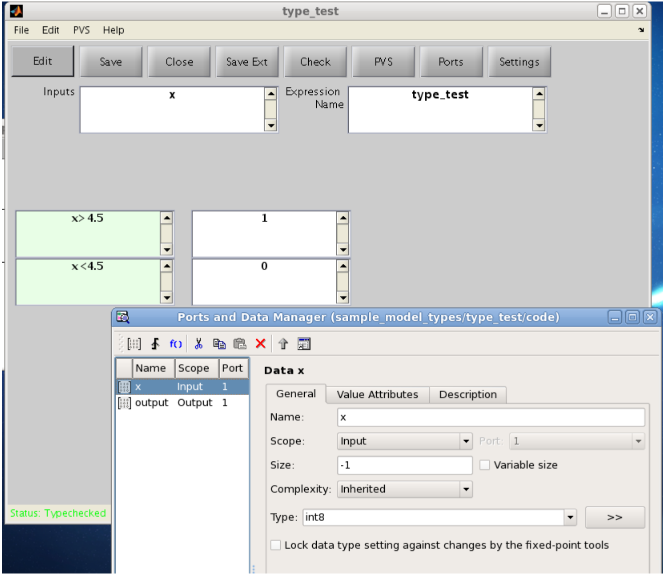
Simulink has many built-in types:

- 8, 16, 32 bit integers
- single and double precision floats
- booleans and enumerated types

The toolbox can detect the typing of the Simulink Ports and use PVS theory modeling those types in verifying the table.

From the table editing widow select `PVS -> Typecheck SimTypes`

Simulink Type Integration



Inputs: x

Expression Name: type_test

x> 4.5	1
x<4.5	0

Status: Typechecked

Name	Scope	Port
x	Input	1
output	Output	1

Data x

Name: x

Scope: Input

Port: 1

Size: -1

□ Variable size

Complexity: Inherited

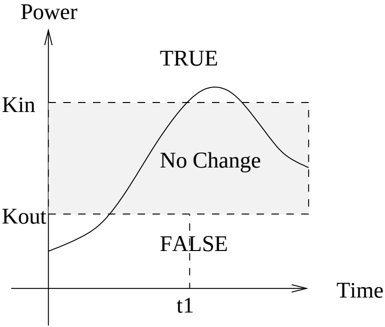
Type: int8

□ Lock data type setting against changes by the fixed-point tools

Table is complete and consistent if x is an 8-bit integer.

Previous counter example x=4.5 is not possible!

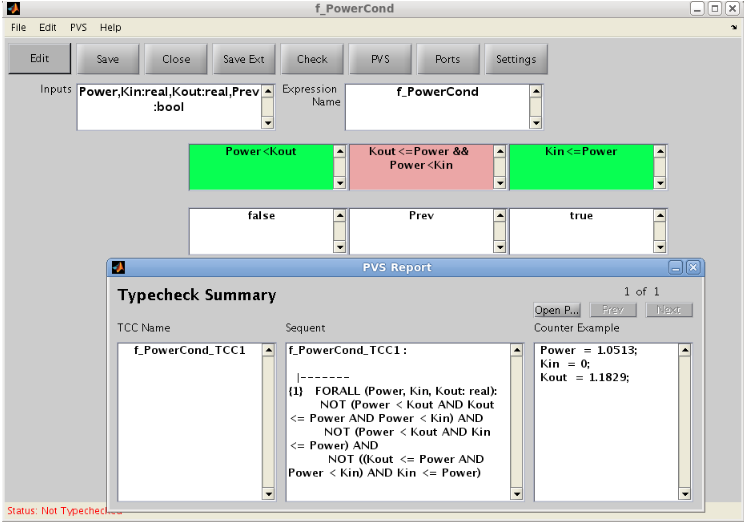
PVS (sub)Typing



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

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

PVS (sub)Typing



Inputs: Power,Kin:real,Kout:real,Prev:bool

Expression Name: f_PowerCond

Power<Kout	Kout<=Power && Power<Kin	Kin<=Power
false	Prev	true

Typecheck Summary

TCC Name: f_PowerCond_TCC1

Sequent:

```
f_PowerCond_TCC1 :

 |-------
{1}   FORALL (Power, Kin, Kout: real):
        NOT (Power < Kout AND Kout
<= Power AND Power < Kin) AND
         NOT (Power < Kout AND Kin
<= Power) AND
          NOT ((Kout <= Power AND
Power < Kin) AND Kin <= Power)
```

Counter Example:

```
Power = 1.0513;
Kin = 0;
Kout = 1.1829;
```

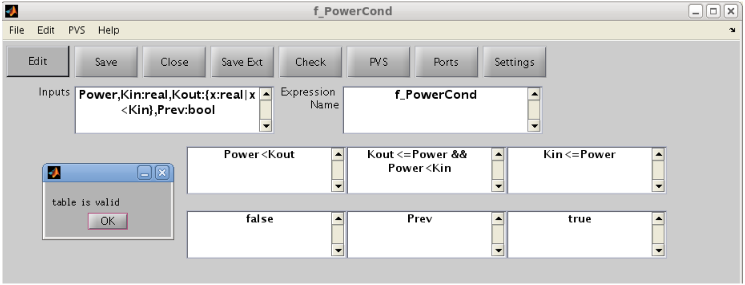
Status: Not Typechecked

McSCert

PVS (sub)Typing

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

We can make it explicit with PVS subtypes.



Inputs: Power,Kin:real,Kout:{x:real|x<Kin},Prev:bool

Expression Name: f_PowerCond

Power<Kout	Kout<=Power && Power<Kin	Kin<=Power
false	Prev	true

table is valid

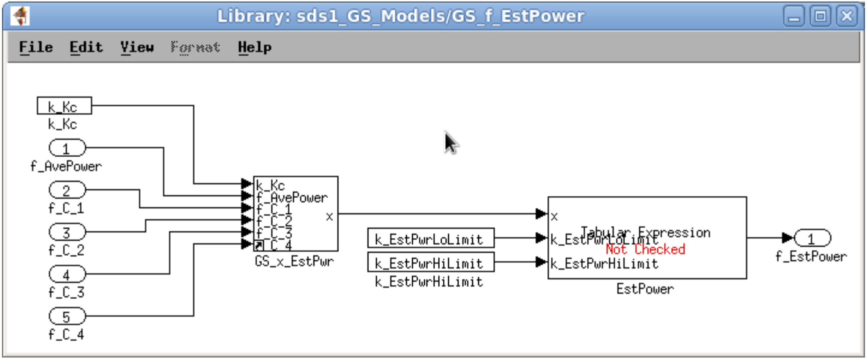
Note: We still need to develop tool to check input typing is satisfied when table with PVS subtyping is used!

SDS1 Estimated Power Module

- Design Description Document, describing system to be implemented using tabular expressions.
- Implemented by undergraduate summer student with no prior knowledge of Matlab/Simulink.
- 2802 blocks to implement this portion of system.
- 42 different tabular expressions
    - Discovered error in implementation of 2 table blocks when typechecking.
    - Typographical errors, which would not be found in compilation, but would affect functionality.
- 2 Different versions created:
    - Gold Standard - floating point, all blocks.
    - Hardware - fixed point, synthesize-able blocks.
    - No block comparison - yet.

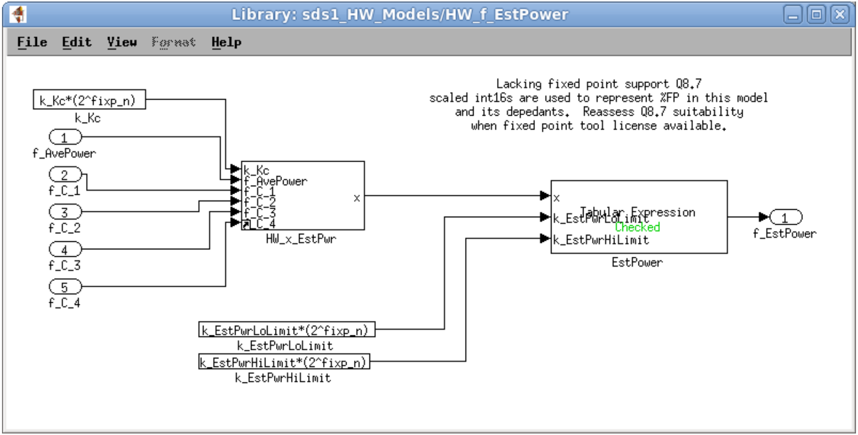
Example Blocks

From the “Gold Standard” version:



Example of Refined Block

From the Fixed point (HW) version



Lacking fixed point support Q8.7
scaled int16s are used to represent %FP in this model
and its depedants. Reassess Q8.7 suitability
when fixed point tool license available.

Want to see how it works?

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.

Parnas, D. L. and Madey, J. (1995).
Functional documents for computer systems.
Science of Computer Programming, 25(1):41–61.

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.

- 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.