Static Analysis of Systems Using PVS

References:

- D.L. Parnas, Tabular representation of relations. Tech. Report CRL Report 260, Telecommunications Research Institute of Ontario, McMaster University, Hamilton, Canada, 1992.
- D.L. Parnas and J. Madey, Functional documentation for computer systems (Ver. 2). Tech. Report CRL Report 237, Telecommunications Research Institute of Ontario, McMaster University, Hamilton, Canada, 1991.
- S. Owre et al., Analyzing tabular and state−transition requirements specifications in PVS. Tech. Report CSL−95−12, Computer Science Laboratory, SRI International, Melno Park, CA, 1995 (revised 1996). pp. 1−50.

Wish list:

- Need precise, unambiguous Software Requirements Specification (SRS) (high level specification) and Software Design Description (SDD) (low level, detailed implementation)
- Method to rigorously (mathematically) verify that the implementation satisfies the system requirements in an understandable way that "scales up" to realistic systems (e.g. tool support)
- Notation must be easily understood by domain experts, developers, verifiers, testers and maintainers

Proposed Solution:

Use mathematical functions described by tables to represent SRS and SDD and then verify functional equality.

Software Verification Commutative Diagram

$$\begin{tikzcd}
M \arrow[r, "f_{REQ}"] \arrow[d, "Abst_M"'] & C \\
I \arrow[r, "f_{SOF}"'] & O \arrow[u, "Abst_O"']
\end{tikzcd}$$

M− Monitored variables range space

C − Controlled variables range space

I − (software) Input variables range space

O − (software) Output variables range space

f_{REQ} − Software Requirements Specification function

f_{SOF} − Software Requirements Specification function

$Abst_M$ − Monitored variable abstraction function

$Abst_O$ − Output variable abstraction function

Proof Obligation: For all $\underline{m} \in M$

$$f_{REQ}(\underline{m}) = Abst_O(f_{SOF}(Abst_M(\underline{m})))$$

Note:

- If we take A/D and D/A imprecisions into account then $Abst_M$ and $Abst_O$ become relations
- f_{REQ} often a relations to specify tolerances
- f_{SOF} is almost always a function for Safety Critical Systems to provide deterministic behavior, unambiuous implementation, etc.

Why use functional equality:

- requirements can be decomposed into functional blocks
- controlled variables for many blocks depend only on current value of block’s monitored variables
- Other requirements blocks can be specified by a state machine with internal state space X_{REQ} that is isomorphic to implentation’s state space X_{SOF} − then let:

  $M' := M \times X_{REQ}$

  $C' := C \times X_{REQ}$

and verify functional equality of state transition functions.

Why use tables to describe functions?

First recall the distinction between:

1. A function
2. The function's description
3. A practical means of computing the function's values

$$F(x)=x+1$$

$$\lambda(y) : y+1$$

$$F(x)=(x=1 \to 2,\ x\neq 1 \to x+1)$$

$$\{(x,y) \mid y = x + 1\}$$

are all descriptions of the same function with many possible means of computation!

Advantages of Tables:

- Visual & hence easily understood
- Supports "divide & conquer" approach
- SRS & SDD functions typically piecewise w/ possibly many discontinuities at arbitrary points and input domain partitioned into discrete subdomains
- Do not imply a particular implementation, unlike flow charts or psuedo−code
- Can provide domain coverage and determinism checks
- Have semi−automated tool support in PVS theorem prover

Tabular Representation of Functions

Simple Table:

dbStatus(temp)=

	temp<SP−50	***SP−50≤temp<SP***	***temp≥SP***
dbstatus	normal	db	high

Want to guarantee:

- All cases covered
- No overlap between different outputs (i.e. defines a proper function)

How?

Coverage: Check disjunctions of conditions are TRUE

$$c_1 \text{ or } c_2 \text{ or } \ldots \text{ or } c_n = \text{TRUE}$$

Disjointness: Check conjunct of each pair of conditions is FALSE

$$c_1 \text{ and } c_2 = \text{FALSE}$$

