Evidential Tool Bus for Flight Critical Systems

Natarajan Shankar
SRI International

Devesh Bhatt and Kirk Schloegel
Honeywell International

2011 Annual Technical Meeting
May 10–12, 2011
St. Louis, MO

Objective

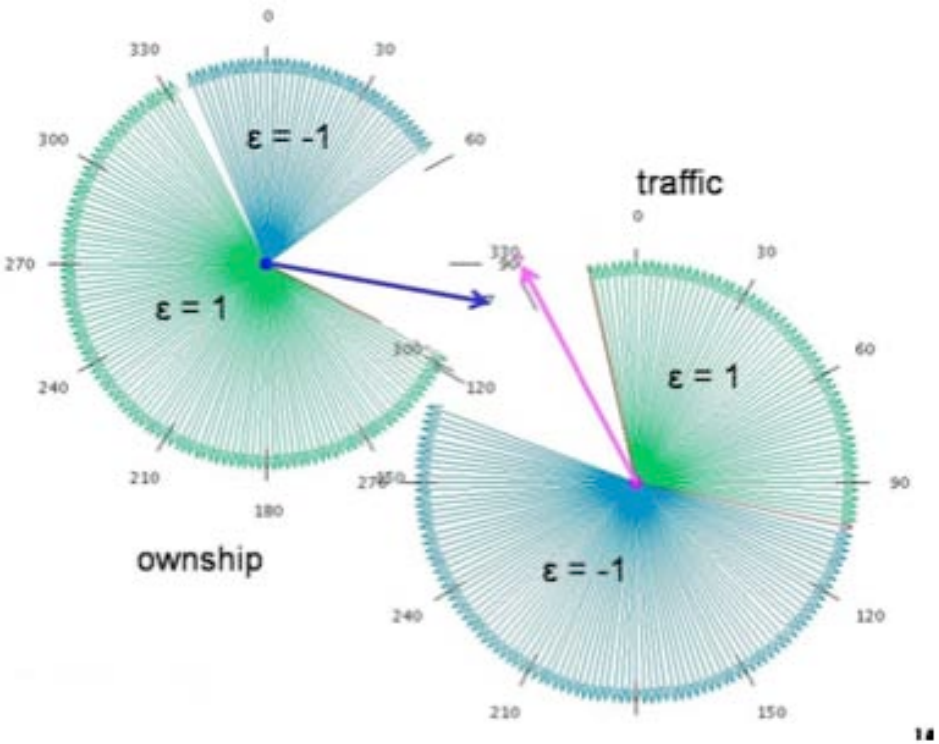
- The project *An Evidential Tool Bus for Flight-Critical Systems* (ETB4FCS) aims to develop a unified semantic framework for the end-to-end
  - Model-based methodologies for the development of flight-critical systems.
  - Analytic capabilities for building dependable flight-critical systems.
  - Public libraries of definitions, decision procedures, and theorems used in formal analysis.
  - Early and compositional analysis tools for flight-critical software-intensive systems.

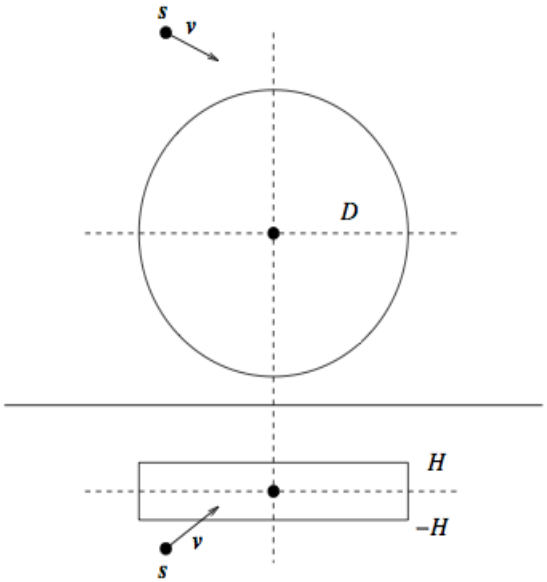
Background

- Software is increasingly critical to avionics and air-traffic control systems.
- Certifying the reliability of flight-critical software systems is a major challenge.
- The ETB4FCS projects addresses the model-based certification of such systems building on
  - SRI's Prototype Verification System (PVS), which has already been used in several landmark verification efforts
  - Honeywell's HiLiTE analyzer for Simulink/Stateflow models that has been used in DO-178B certification of commercial aircraft avionics
  - The Evidential Tool Bus (ETB) for integrating diverse tools and composing end-to-end assurance cases.

NASA's ACCoRD Framework for Coordination

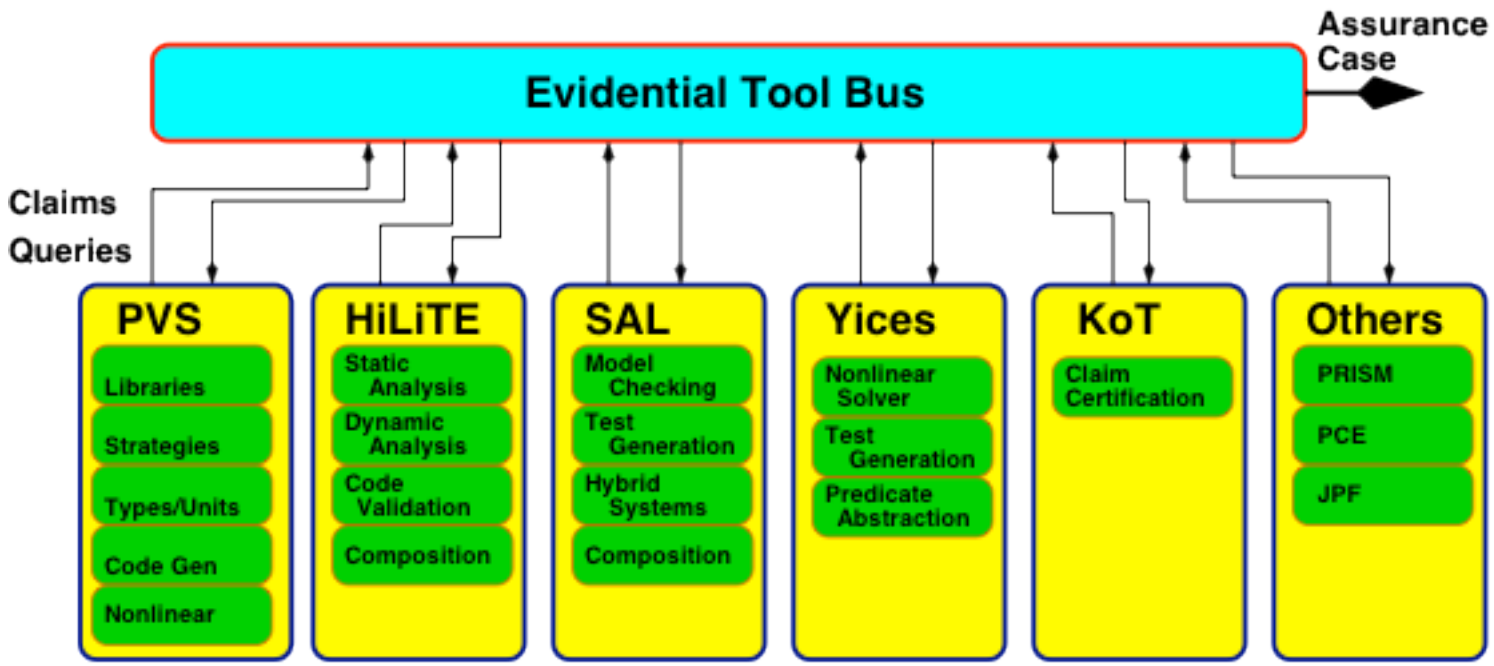
Algorithms for detecting and avoiding impending conflicts
Algorithms for recovering from loss of separation





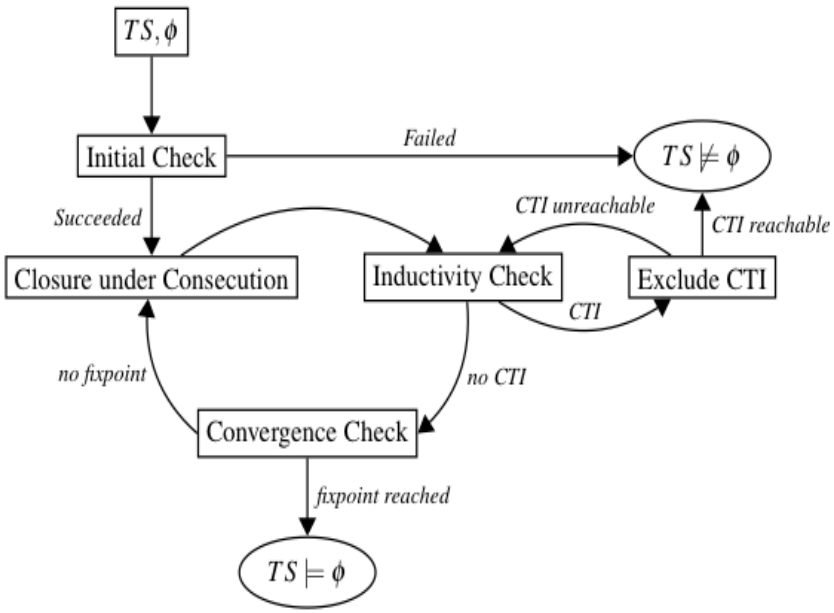
Comprehensive PVS proofs of correctness

Technical Approach



Evidential Tool Bus (ETB)

- Verification tools include SAT/SMT solvers, static/dynamic analyzers, and interactive theorem provers.
- Useful verification functionality, e.g., symbolic simulation or test case generation, is typically built from combinations of these tools.
- The ETB is an operating system for integrating verification tools and composing assurance cases.
- Prototype design using XSB Prolog; Implementation using Python
- Ongoing pilot study of invariant strengthening tool using SAL and Yices for transition systems



PVS Enhancements: Libraries

- Prototype Verification System (PVS) is a widely used theorem proving system combining interactive and automated verification techniques.
- Many complex systems have been verified with PVS.
- PVS has an extensive set of formal libraries for foundational mathematics, mostly developed by many researchers (including those at NASA Langley).
- We add background theories for linear algebra and probability as needed to tackle correctness arguments for cyber-physical systems.

```
det_type: TYPE = {f: (alternating) | FORALL (i: posnat): f(I(i)) = 1}
det_exists: lemma EXISTS (f: det_type): true
det_unique: lemma FORALL (f, g: det_type): f = g

jensens_E_inequality: THEOREM convex?(f) => f(E(X)) <= E(f o X)
```

PVS Enhancements: Nonlinear Arithmetic

- When reasoning about physical systems, many proofs require extensive nonlinear arithmetic reasoning.
- Currently, the level of automation for this kind of reasoning is quite low.
- Integration of NASA Field, Manip, PVSio, ProofLite in PVS 5.0 release; integration of NLYices and latest RAHD release; formalization of IEEE 754.
- Collaboration with Edinburgh/Cambridge on nonlinear arithmetic extensions.
- lt_D_t1_lt_t2_1 : LEMMA

```
  FORALL (t,t1,t2,D:real):
        vx*vx + vy*vy /= 0 AND
        t1 < t2 AND
        (FORALL xy: xy*xy >= 0) AND
         (sx+vx*t1)*(sx+vx*t1) + (sy+vy*t1)*(sy+vy*t1) = D AND
       (sx+vx*t2)*(sx+vx*t2) + (sy+vy*t2)*(sy+vy*t2) = D AND
       (sx+vx*t)*(sx+vx*t) + (sy+vy*t)*(sy+vy*t) < D IMPLIES
                t1 < t AND t < t2
```

PVS Enhancements: Physical Dimensions

- In modeling physical systems, variables ranging over numbers represent physical quantities
- Many errors arise from incompatible dimensions and units
- Dimension typing has been added to the PVS type system
- Primitive units are defined below

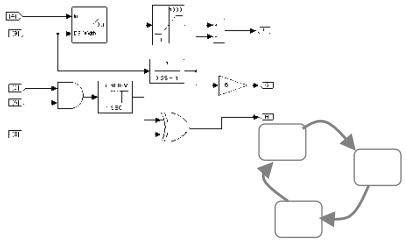
```
units: THEORY
  BEGIN
  metre: _UNIT = metre
  kilogram: _UNIT = kilogram
  second: _UNIT = second
  coulomb: _UNIT = coulomb
  candle: _UNIT = candle
  degree_kelvin: _UNIT = degree_kelvin
  radian: _UNIT = radian
END units
```

PVS and ETB: The Value Proposition

- Safety-critical cyber-physical systems are everywhere
- Cars, planes, ships, spacecraft, robots, power plants and grids, factories, air-traffic control systems, and the human body
- An assurance case for such systems must combine
  - Background mathematical theories
  - Sophisticated constraint solvers for proving and test generation
  - Static and dynamic analyzers
  - Diverse forms of evidence for demonstrating safety, stability, fault tolerance, and robustness
- PVS with libraries, nonlinear arithmetic, dimensions/units is a productive and comprehensive framework for formalizing cyber-physical systems
- ETB supports the scriptable integration of multiple analysis tools for building a verifiable assurance case

Honeywell Integrated Lifecycle Tools and Environment (HiLiTE)

MATLAB Simulink and StateFlow Models



Multiple Block Library sets

- HAM
- Engines Blocks
- B-787 Flight Control

System Dictionaries

I/0, Symbols ranges, type, code ref.

HiLiTE

- **Test Generator**
- **Code Generator (HAM/RTW or HiLiTE)**
- **Model Analysis**

Detection of Design Problems
- Logic/relational conflicts
- untestable conditions / paths
- divide by zero, overflow
- numerical error: ambiguity/stability
- Control function level analysis

Design review automation

Test Cases

Test Driver

Model Impl. Code

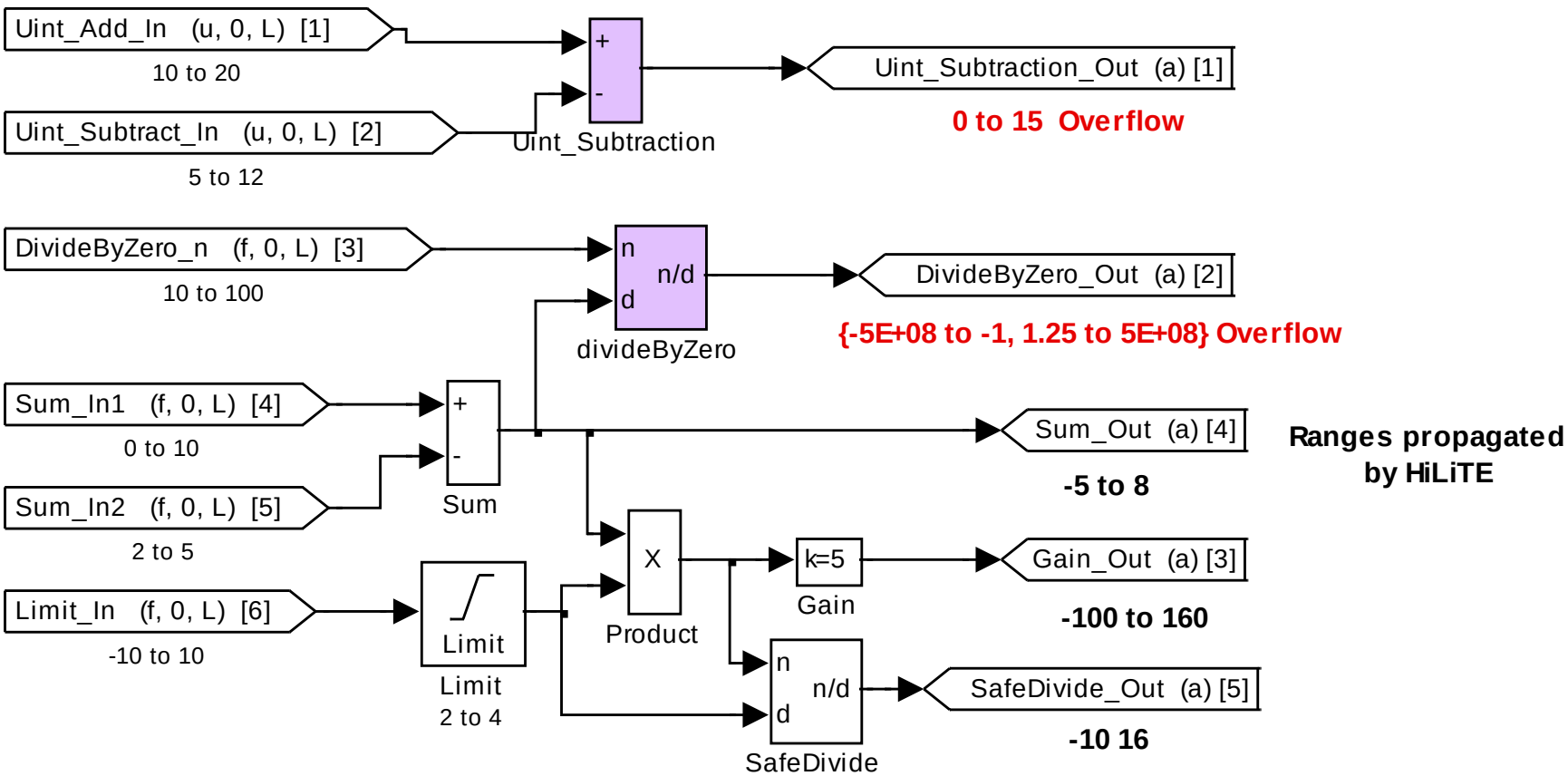
Reuse Libraries

Target Processor

Benefit: Reduced certification cost & cycle time; detect design problems early

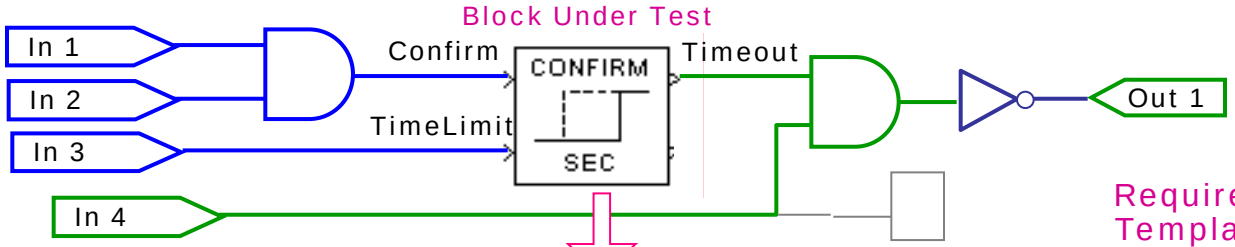
Technology Transfer: HiLiTE Improvement from this NASA program will be used in upcoming commercial aircraft certifications

Simulink Range Analysis Example



– Range at the output of a block is calculated taking into account all possible combinations of input values within their respective ranges over time.

Example Test Vector Generation from Low-Level Requirements



Requirement-based Test Case Template for *ConfirmSec* block

Rate = 50 Hz

Time Steps

No. of Time Steps	Block Under Test			comment
	Confirm	TimeLimit	Timeout	
1	0	Range.mid	0	reset timer
TimeLimit * Rate − 1	1	Range.mid	0	1 cycle before
1	1	Range.mid	1	timer expires
1	1	Range.mid	1	1 cycle after

Generated Test Vector

Time Steps

	In 1	In 2		In 3		In 4	Out 1	comment
Range	0	0	0	0	0	0	0	
	1	1	1	10	1	1	1	
# steps			Confirm	TimeLimit	Timeout			
1	**1**	**0**	**0**	**5**	**0**	**1**	**1**	reset timer
249	**1**	**1**	**1**	**5**	**0**	**1**	**1**	1 cycle before
1	**1**	**1**	**1**	**5**	**1**	**1**	**0**	timer expires
1	**1**	**1**	**1**	**5**	**1**	**1**	**0**	1 cycle after

HiLiTE contains multiple test templates for over 250 different kinds of function blocks including logic, math, integrators, filters, timers, latches, hysteresis

How HILiTE Generates Test Vectors from Test Case Templates

1. Convert formulae in block template to values, using operating ranges for block's ports

2. Search all possible paths & values from block's inputs to *model inputs*

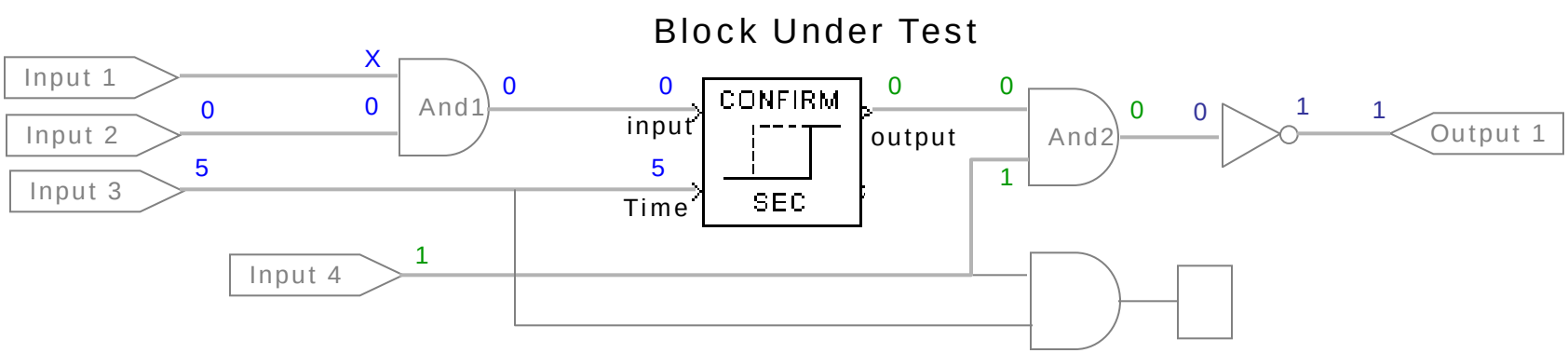
- while computing reverse transformations by each intermediate block

3. Search all possible paths & values from block's outputs to *model outputs* and *set up inputs*

- Determine expected values at model outputs corresponding to expected values at block's outputs while computing transformations by each intermediate block
- Observability is essential for detecting coding errors

4. Repeat steps 2 & 3 for each time step in template

– solve for conflicts of desired value in the same time step



Model Defects Analysis and Test Generation using HiLiTE in Honeywell

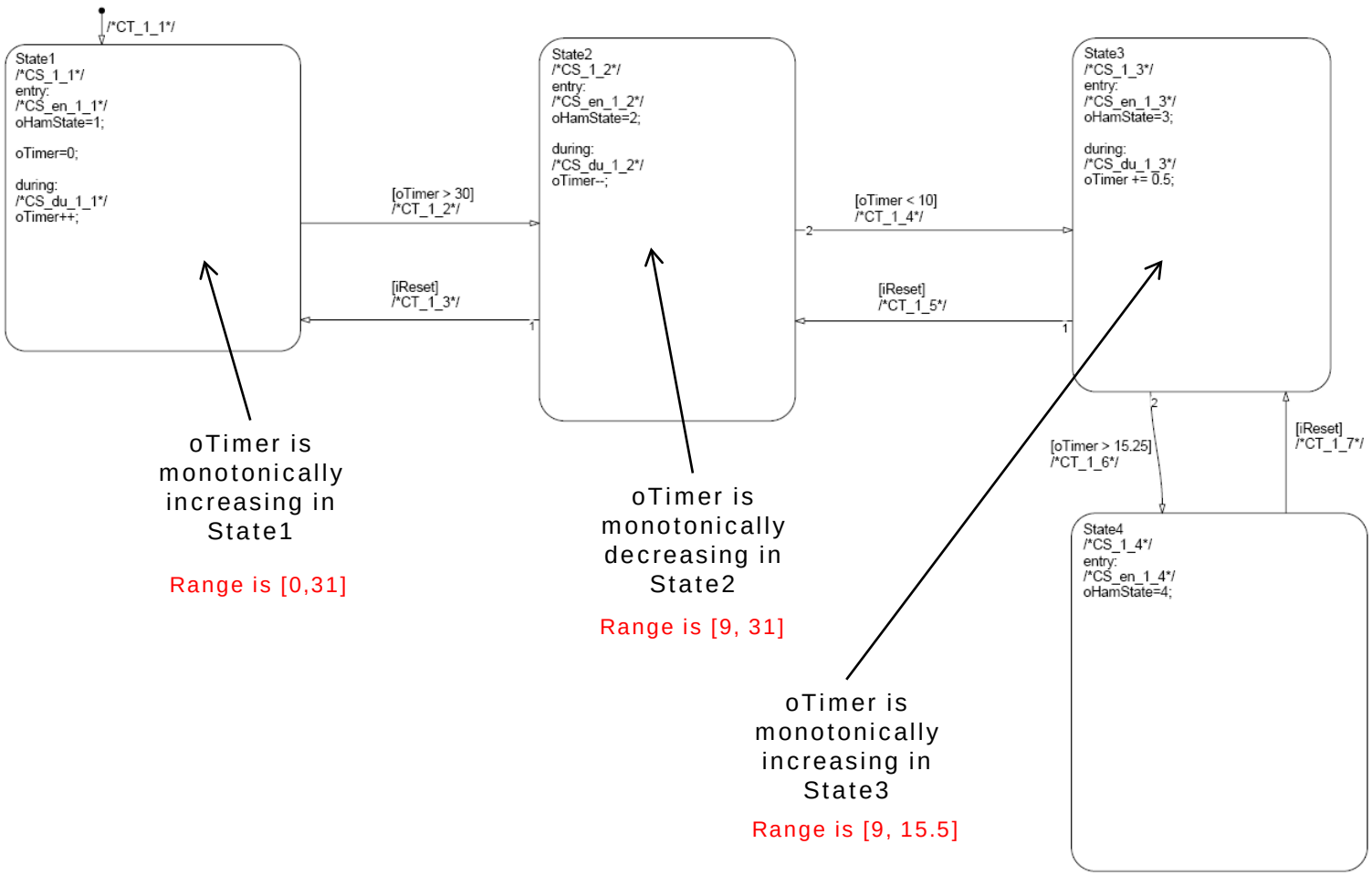
Application Domain	No. of Models (approx order)	Types of Defects of interest in Early analysis	False Alarms	Median Requirements, Robustness Coverage Achieved	Remarks
Domain A	2000	Overflow, underflow, anomalous behaviors, untestable/unreachable constructs, floating point ambiguity	5–10%	> 90%	Protection mechanisms/ constraints elsewhere in the system mitigate some “defects”
Domain B	300	Anomalous behaviors, untestable/unreachable constructs, deactivated code	< 5%	> 90%	Still in the early stage of design. Lot of re-use of sub-models creates deactivated code
Domain C	200	Untestable/unreachable constructs	< 5%	85-95%	Defects were reduced with early analysis and design maturity
Domain D	90	Untestable/unreachable constructs, divergent feedback loops	< 5%	85%	Very specific design patterns

Extensions under VVFCS TA 4

- Static analysis of flight-critical software models
- Dynamic analysis of flight-critical software models
- Creation of model representation framework

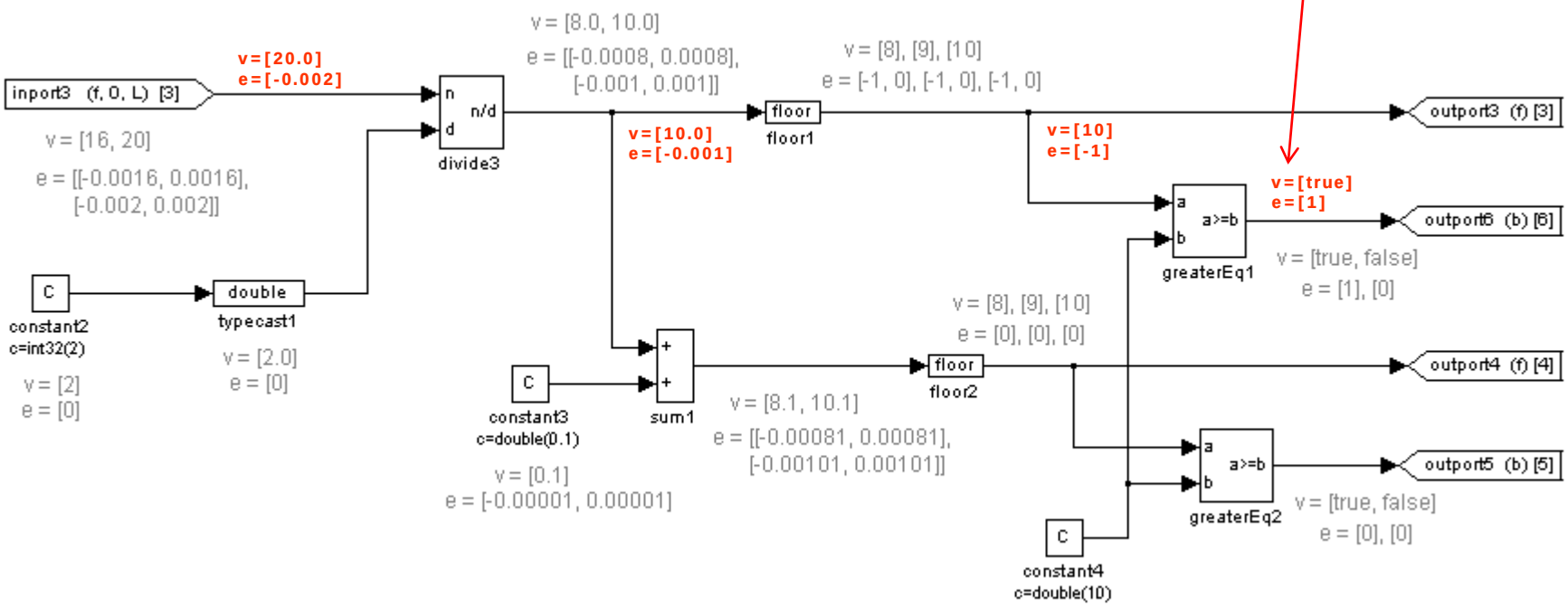
Static analysis of flight-critical software models

- Tighten range propagation results
  - Detect state-based invariants in Stateflow models
    - Monotonically increasing/decreasing variables that guard transitions
  - Cross-signal interactions that are independent of control flow



Static and dynamic analysis of flight-critical software models

- Static analysis of flight-critical software models
  - Propagation of error along with type and range data
  - Detection of anomalies
    - Potential race conditions across sibling parallel states
- Dynamic analysis of flight-critical software models
  - Improved test generation requirements, templates, and coverage results for selected math function blocks



HiLiTE Static Analysis vs. Code Analysis Tools

HiLiTE Static Analysis	Polyspace / Klocwork / Purify /Coverity
Works at model design semantics level • Understands the semantics of HAM blocks, modeling constructs, and block combinations • Runtime < 1 minute for many industrial models	Works at the C-Code level • Understands full set of C constructs including pointers/arrays, functions, memory allocation. (note: most such C-constructs are not used in MBD auto code)
Detects model- and block-semantics specific defects e.g. Fader block used on a continuous signal negative input to SQRT, variable timers Multiport switch control input < 1 or > N	These tools have no knowledge of model-level or block level design semantics. Some of the model defects can be masked in the code by protection mechanisms.
Detects untestable block requirements/conditions and corresponding structural coverage holes / deactivated code.	Very limited capability and only at code level, not model or block level (i.e., in terms of block requirements)
Detects signal overflow (e.g., divide-by-zero)	Detects signal overflow (e.g., divide-by-zero)
Performs range propagation within and across models and creates range output data for all signals	Detects range-based code errors only but doesn't create range data output.
Not applicable in MBD auto code. (C-Arrays bounds and buffer sizes are static in MBD code using HAM)	C array bound violation, buffer overflow: noted by Microsoft as the "single" underlying cause of security issues in Windows. (Microsoft uses own, more powerful tools.) Memory leaks, security leaks, etc.
	Can be used on system build including multiple auto code and hand code modules.

Both types of capabilities can be useful to a project
Code Analysis tools are typically useful in a non-MBD usage scenario

Creation of model representation framework

- Definition of XML schema to capture Simulink and Stateflow models

D:\projects\vvfcs\etb4fcs\honeywell\xsd\model.xsd - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

model.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright (c) Honeywell International 2011. All Rights Reserved. -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.honeywell.com"
  xmlns:hon="http://www.honeywell.com">
  <xsd:include schemaLocation="graphicalfunction.xsd"/>
  <xsd:include schemaLocation="matlabfunction.xsd"/>
  <xsd:include schemaLocation="simulink.xsd"/>
  <xsd:include schemaLocation="stateflow.xsd"/>
  <xsd:include schemaLocation="truthtable.xsd"/>
  <xsd:complexType name="ActorType" abstract="true">
    <xsd:attribute name="name" type="xsd:string" use="required">
      <xsd:annotation>
        <xsd:documentation>In Stateflow, name is obtained from the attribute id of the element in
          the mdl file.</xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:element name="Model" type="hon:MoC"/>
  <xsd:complexType name="MoC">
    <xsd:annotation>
      <xsd:documentation>
        Root-level place holder for models.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence minOccurs="1" maxOccurs="1">
      <xsd:element ref="hon:Syntax" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="hon:Semantics" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required">
      <xsd:annotation>
        <xsd:documentation>In Stateflow, model name is obtained from the name attribute of chart
          in the mdl file. In TruthTable, Graphical Function, and Matlab Function, model name is
          obtained from the labelString attribute of corresponding state in the mdl file.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="version" type="xsd:double" use="optional">
      <xsd:annotation>
        <xsd:documentation>In Stateflow, model version is obtained from the sfVersion attribute of
          machine in the mdl file. In TruthTable, Graphical Function, and Matlab Function, this
```

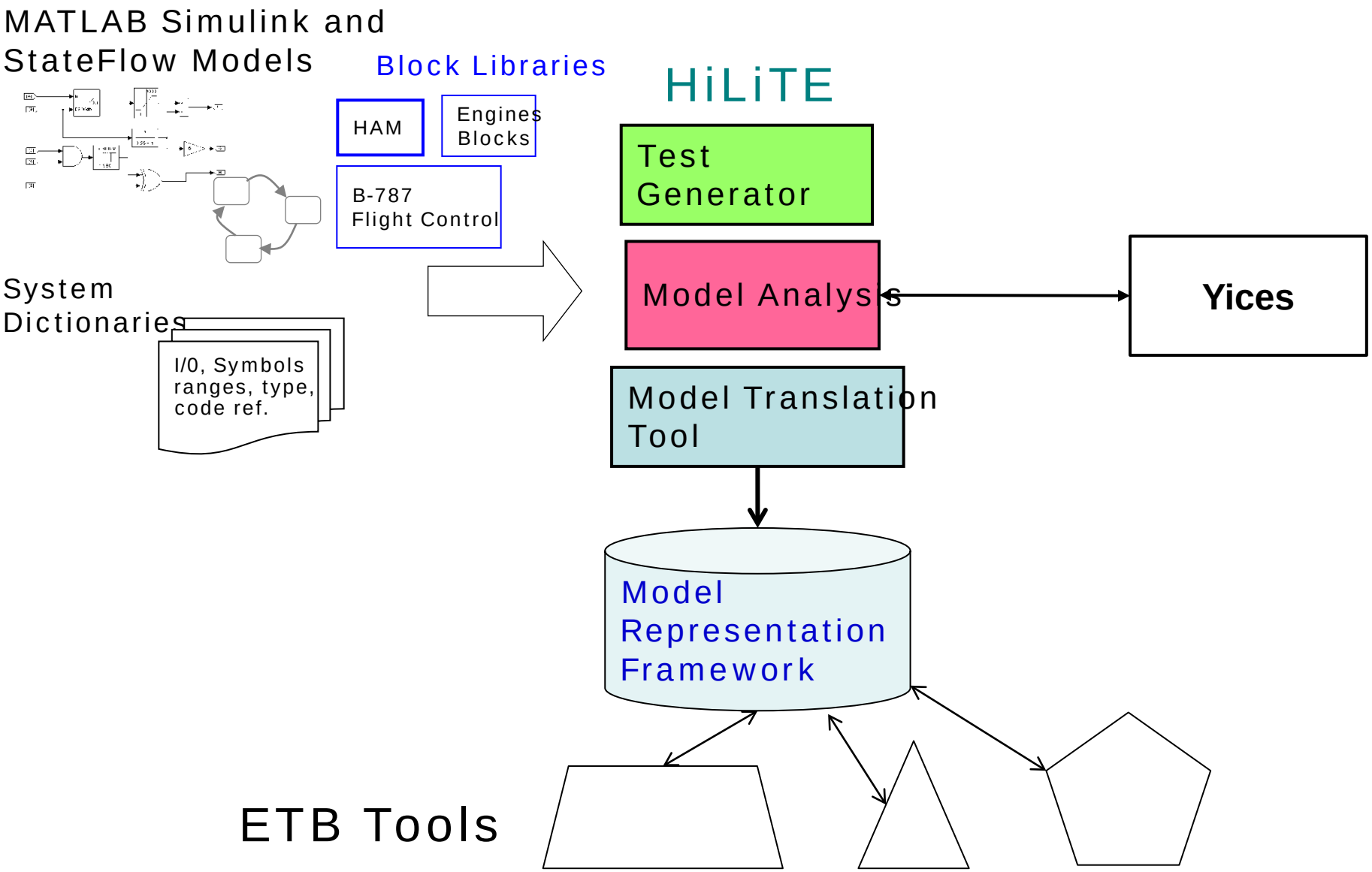
eXtensible Markup Language file | length : 10753 lines : 201 | Ln : 1 Col : 1 Sel : 0 | Dos\Windows | ANSI as UTF-8 | INS

Current Work and Next Steps

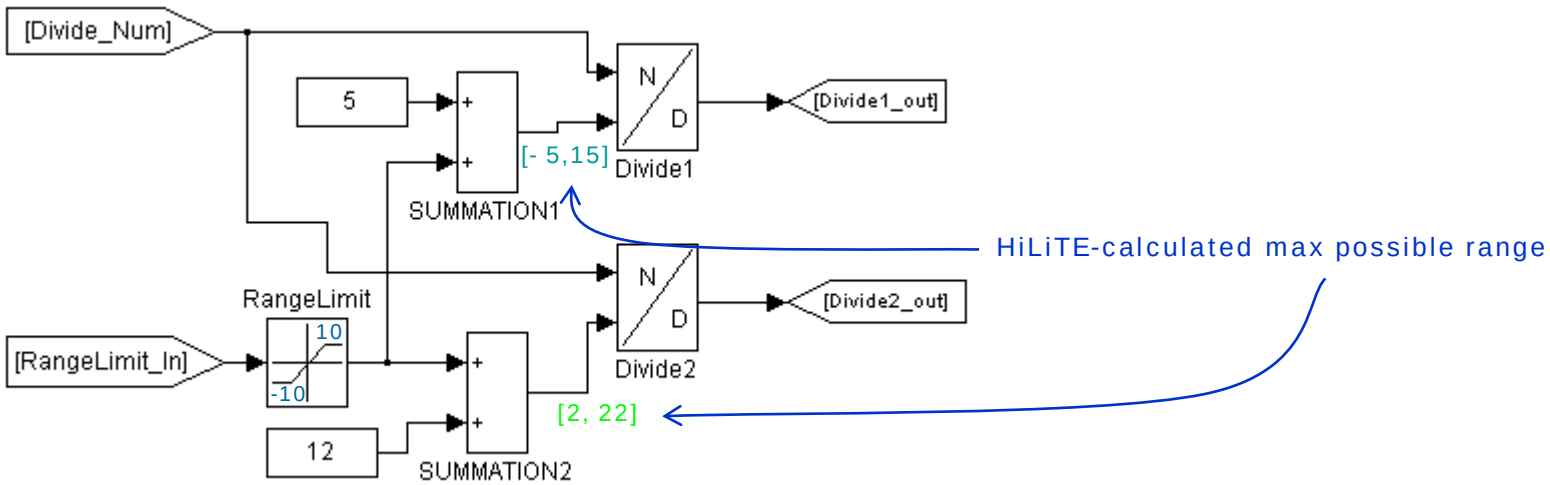
- Static analysis of flight-critical software models
  - Further improve (tighten) range bounds
    - In StateFlow models by taking into account control flow
    - Identify and solve sub-graph constraints in dataflow models – e.g. polynomials
    - Bounds in feedback loops – e.g. counter patterns and other relational abstractions
  - Generalize error propagation to work across Simulink-Stateflow and broader classes of function blocks
  - Detect behavior anomalies based upon range bounds
    - specific to function block's requirements
- Dynamic analysis of flight-critical software models
  - Define common behaviors for classes of mathematical and time-dependent blocks
    - Functional and robustness requirements that need specific testing
    - What should be tested: test values/ranges criteria, equivalence classes
- Creation of model representation framework
  - Generate XML files for Simulink and StateFlow models, representing the contents and interface of the models.

Backup slides

Interfaces to other ETB Tools



Simulink Range Analysis Example: Detecting Divide by Zero Possibility



HiLiTE Output

[WARN] Divide Block Divide1: divide by zero is possible since maximum possible range contains zero
Max Possible Range: [- 5,15] Operating Range: [1,13]

Note: HiLiTE does not warn about Divide2 since its denominator's maximum possible range is [2, 22]

When complex constraints or feedback loops are present, baseline HiLiTE version can result in "loose" range bounds -- that can lead to false alarms