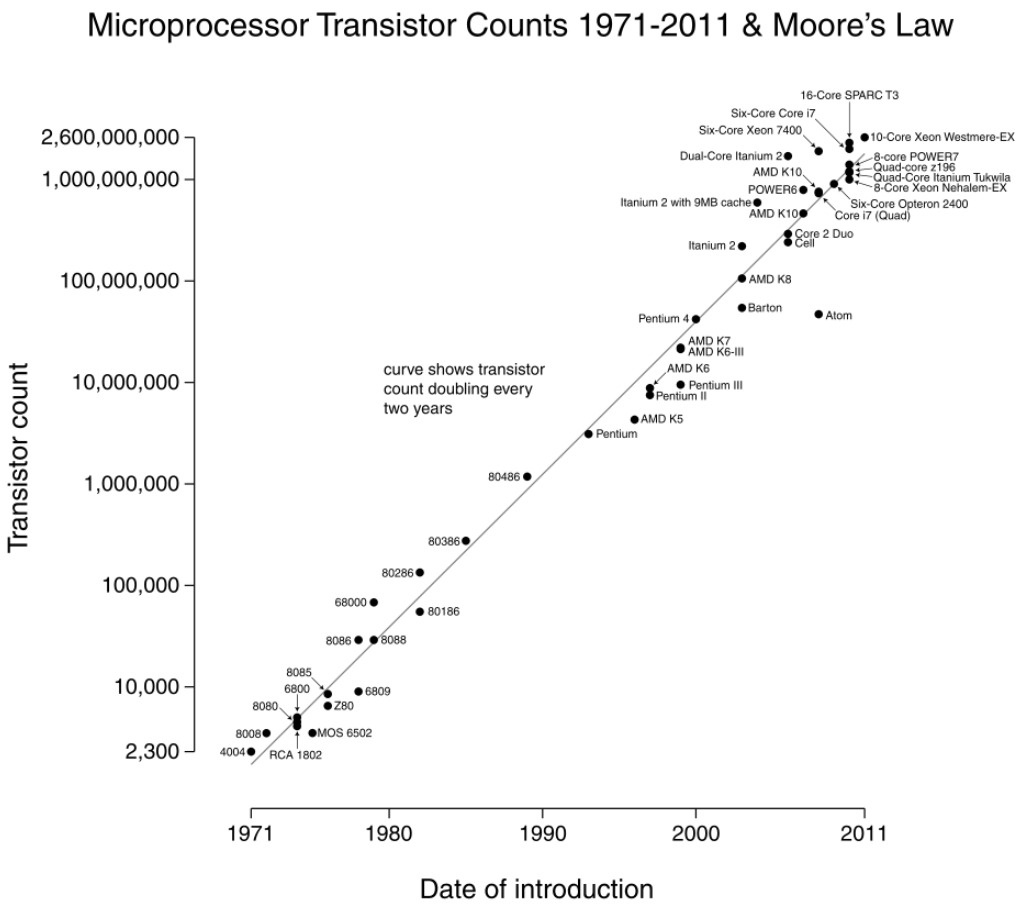
Industrial Verification Using the KIND Model Checker

Lucas Wagner
Jedidiah McClurg
{lgwagner,jrmcclur}@rockwellcollins.com

Rockwell Collins

Software Complexity is Becoming Overwhelming

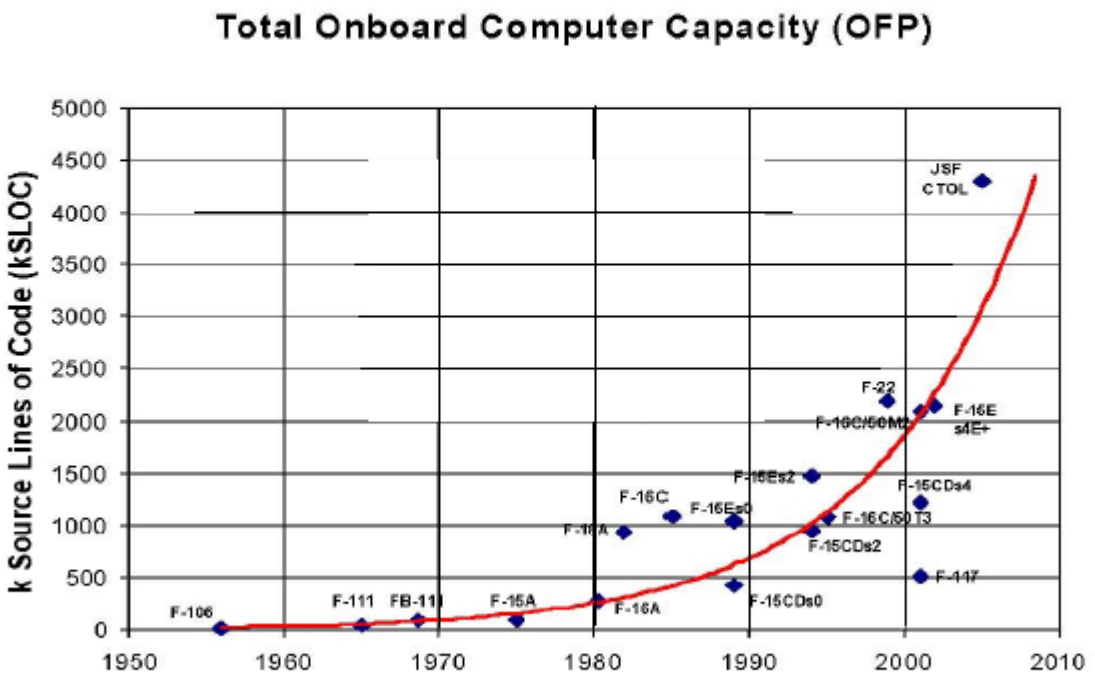
Advancements in computing technology have increased the capability of systems by enabling more complex software to be utilized.



Software in Defense Systems

U.S. AIR FORCE

DoD software is growing in size and complexity

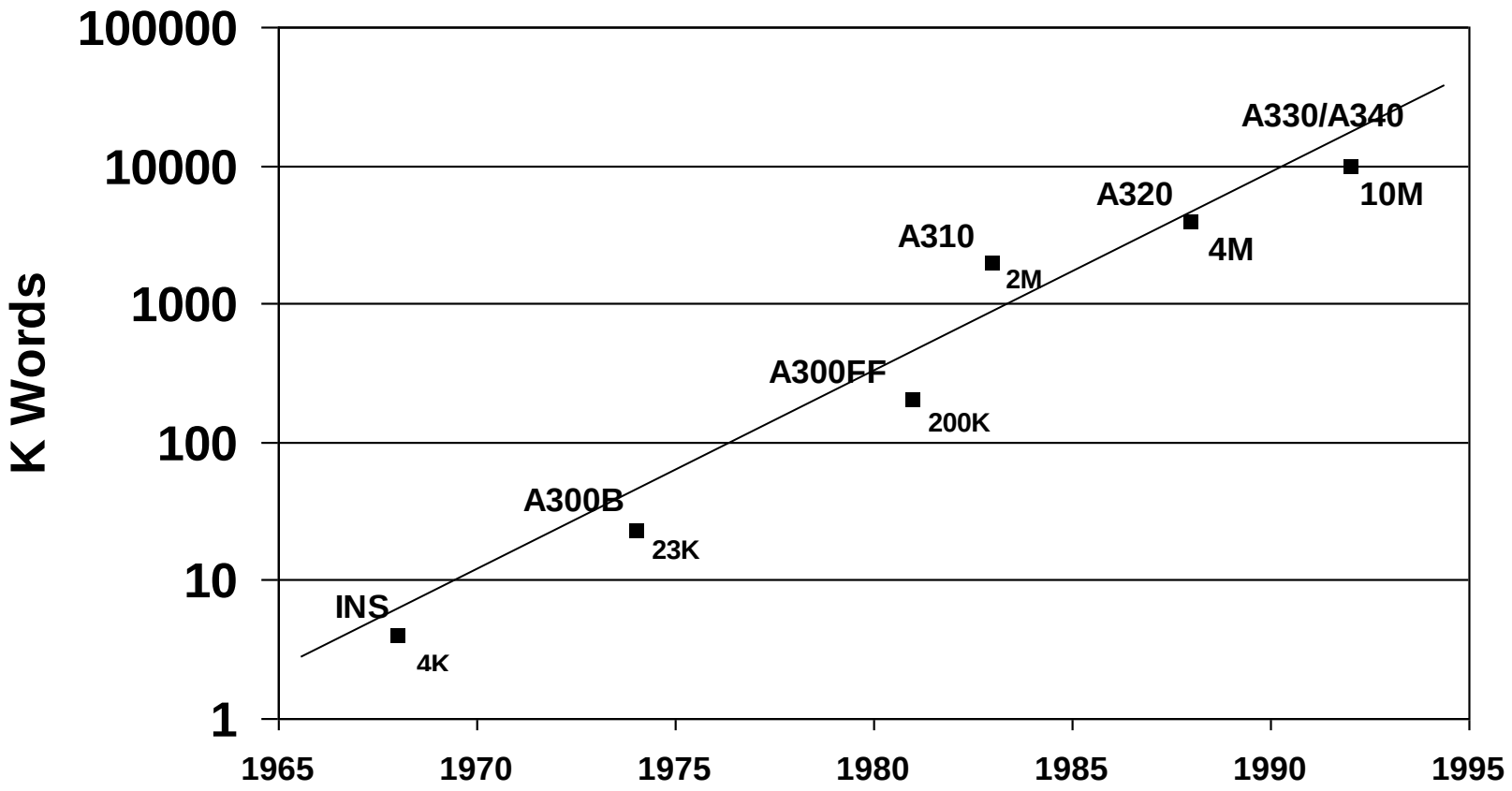


Source: "Avionics Acquisition, Production, and Sustainment: Lessons Learned -- The Hard Way", NDIA Systems Engineering Conference, Mr. D. Gary Van Oss, October 2002.

Robert Gold, OSD

Airborne Software Complexity

Doubles Every Two Years!

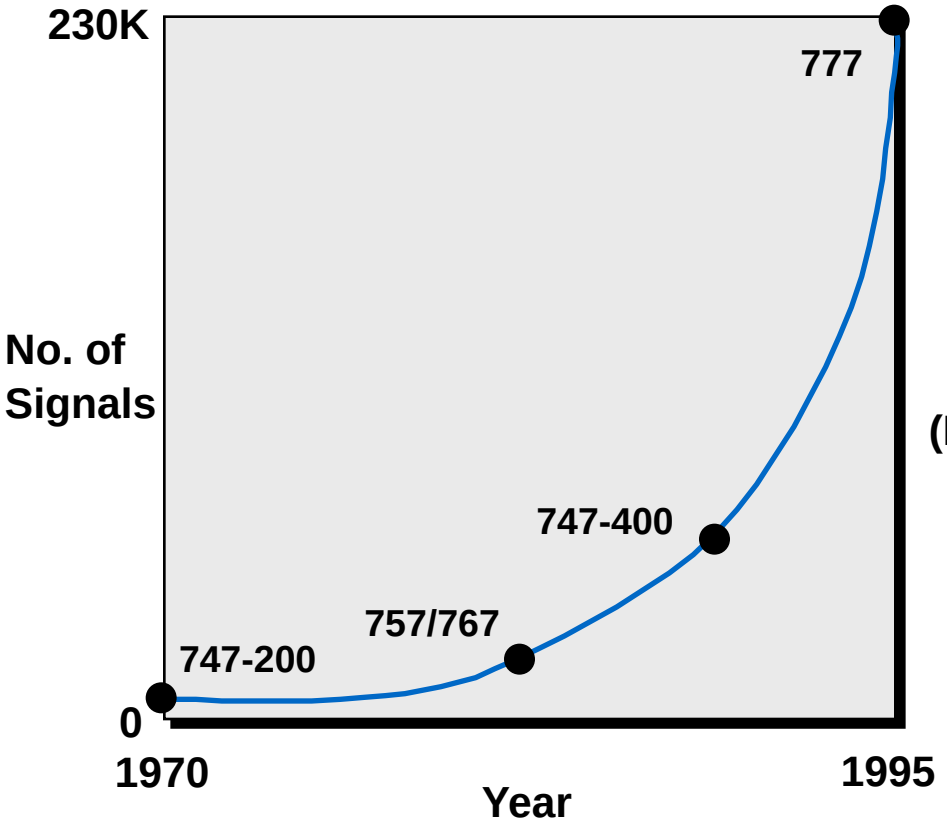


J.P. Potocki De Montalk, Computer Software in Civil Aircraft, Sixth Annual Conference on Computer Assurance (COMPASS '91), Gaithersberg, MD, June 24-27, 1991.

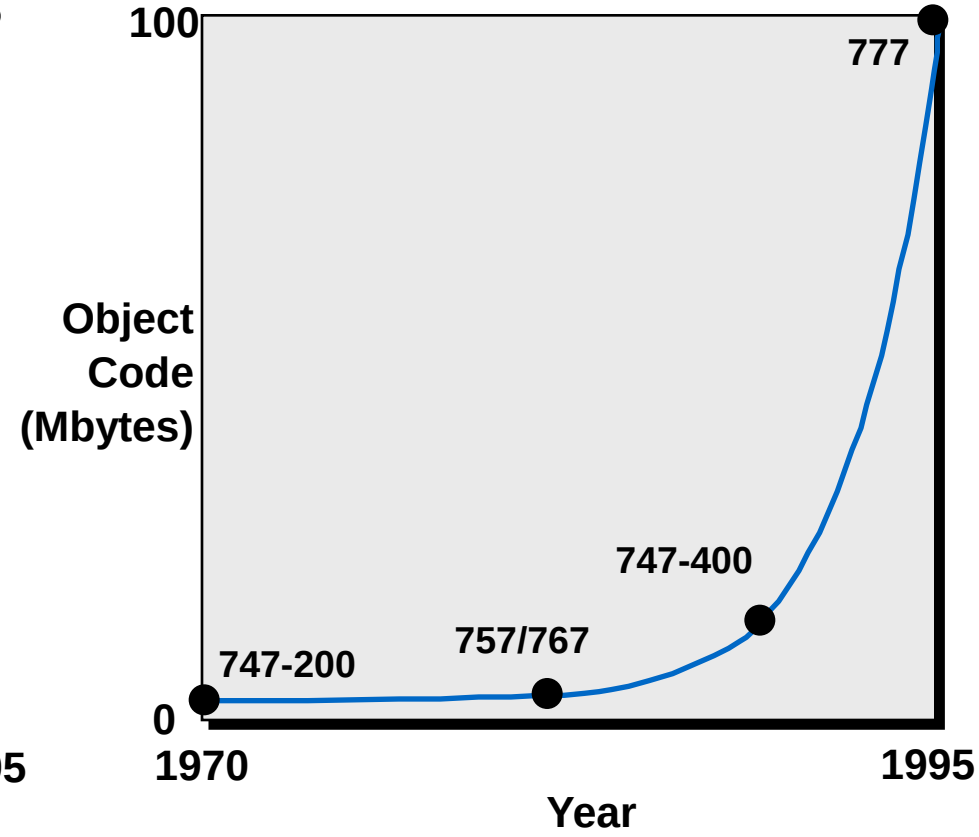
Airborne Software Complexity

Similar Growth Has Been Seen by Boeing

Complexity



Size



Why is complexity a problem?

- **Complex modern software is becoming prohibitively expensive to validate and verify using traditional methods.**
  - **Validation and Verification is usually a level of effort activity. One can always write more tests, but how much is enough?**
  - **Exhaustive methods are necessary to prove the absence of errors, but are impractical (and impossible) using traditional methods on modern systems.**
- **Verification and Validation using traditional methods is costly.**
  - **The traditional waterfall development paradigm can lead to costly iterative fixes when errors are discovered late in the development cycle.**
  - **Errors missed by gaps in traditional testing might be discovered in service leading to very costly.**

A Partial Solution

What is needed?

- **An iterative V&V approach that can detect errors early in the development cycle, preferably at the design stage.**
- **Provides a high level of assurance, preferably exhaustive.**
- **Can be learned rather quickly (weeks to months) and does not require years to become an expert.**

Software model-checking is a partial solution to this problem.

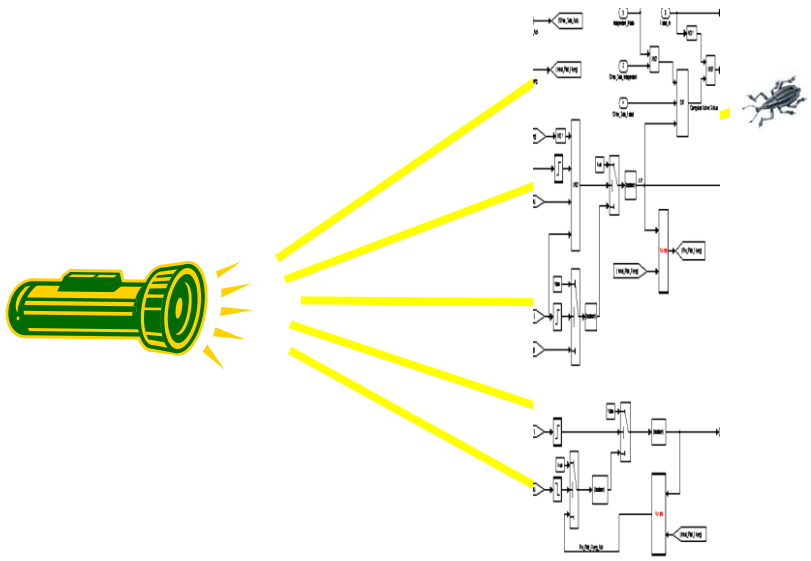
Over the last decade, Rockwell Collins has developed and refined a formal translation framework to enable model-checking of software developed in the Model-Based Development paradigm.

- **What is Model-Based Development (MBD)?**
  - **A set of domain specific graphical notations used to develop software. Using MBD enables early simulation and debugging of models.**
  - **Demonstrably correct models can be used to generate tests cases and source code for use on targeted processors.**
  - **MATLAB Simulink®, Esterel Technologies SCADE Suite™**
- **What is Model-Checking?**
  - **Exhaustively proves user specified properties about a model**
  - **Fully automated**
  - **Generates a counterexample if a property is false**

Reduce Costs and Improve Quality by
Using Analysis to Find Errors During Early Design

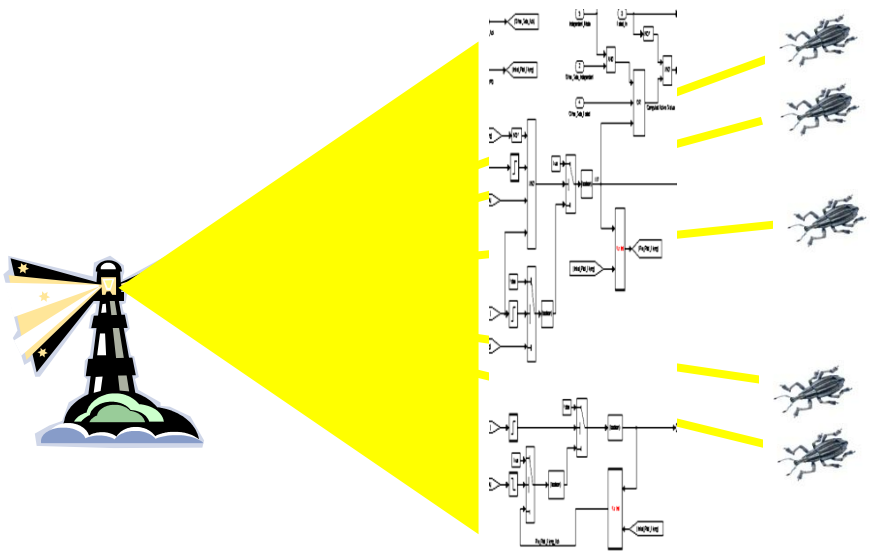
Traditional Verification vs. Model Checking

Testing Checks Only the Values We Select



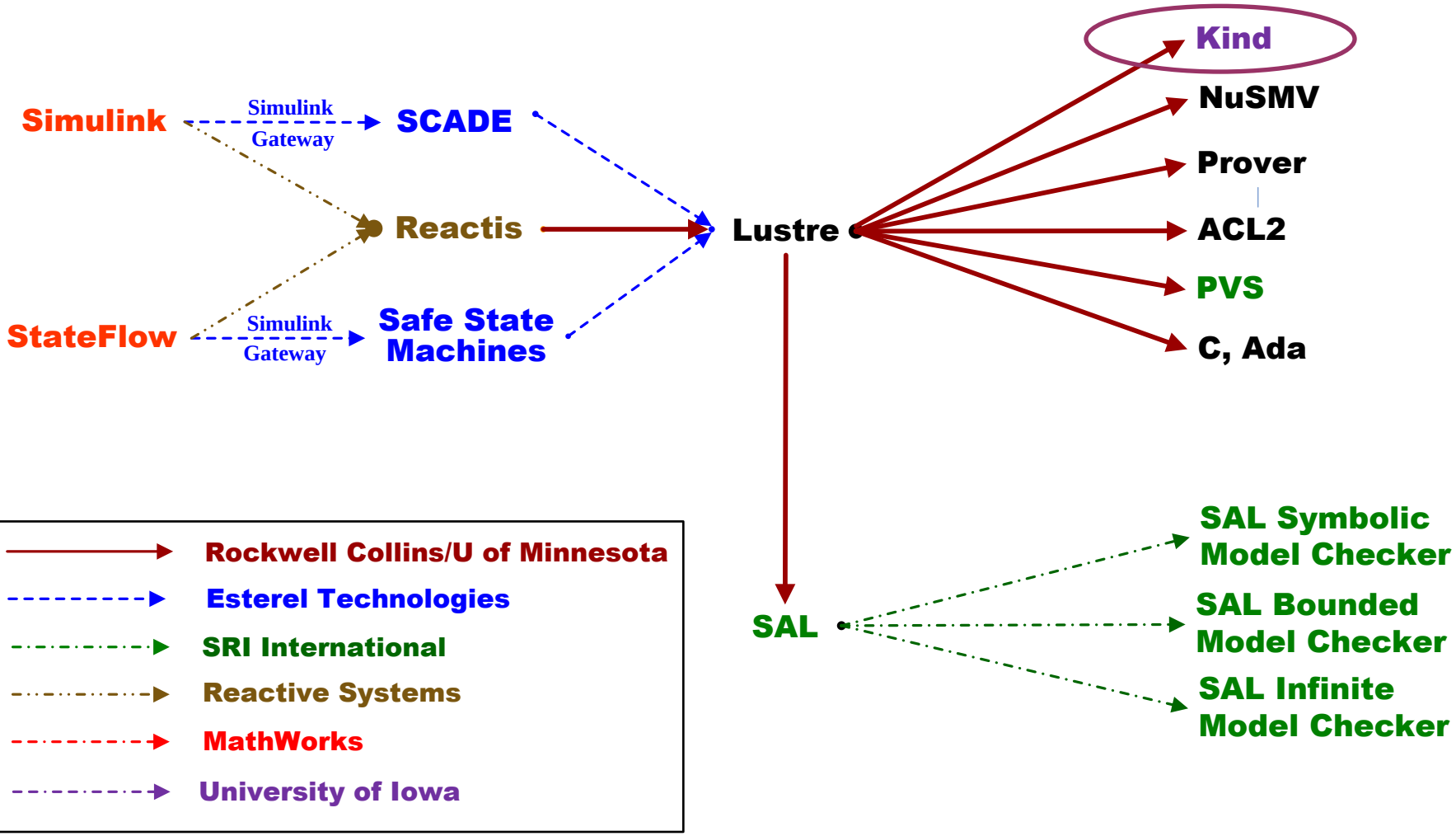
Even Small Systems Have Trillions (of Trillions) of Possible Tests!

Model Checker Tries Every Possible Value!



Finds every exception to the property being checked!

Rockwell Collins Translation Framework



Industrial Application of Model-Checking CerTA FCS Phase I

- **Sponsored by the Air Force Research Labs**
  - **Air Vehicles (RB) Directorate - Wright Patterson**
- **Investigate Roles of Testing and Formal Verification**
  - **Can formal verification complement or replace some testing?**
- **Example Model – Lockheed Martin Adaptive UAV Flight Control System**
  - **Redundancy Management Logic in the Operational Flight Program (OFP)**
  - **Well suited for verification using the NuSMV model-checker**

Lockheed Martin Aero

- Based on Testing
- Enhanced During CerTA FCS
  - Graphical Viewer of Test Cases
  - Support for XML/XSLT Test Cases
  - Added C++ Oracle Framework
- Developed Tests from Requirements
- Executed Tests Cases on Test Rig

Rockwell Collins

- Based on Model-Checking
- Enhanced During CerTA FCS
  - Support for Simulink blocks
  - Support for Stateflow
  - Support for Prover model-checker
- Developed Properties from Requirements
- Proved Properties using Model-Checking

CerTA FCS Phase I – Verification Results

	Model Checking	Testing
Triplex Voter	5	0
Failure Processing	3	0
Reset Manager	4	0
Total	12	0

- Model-Checking Found 12 Errors that Testing Missed
- Spent More Time on Testing than Model-Checking
  - 60% of total on testing vs. 40% on model-checking

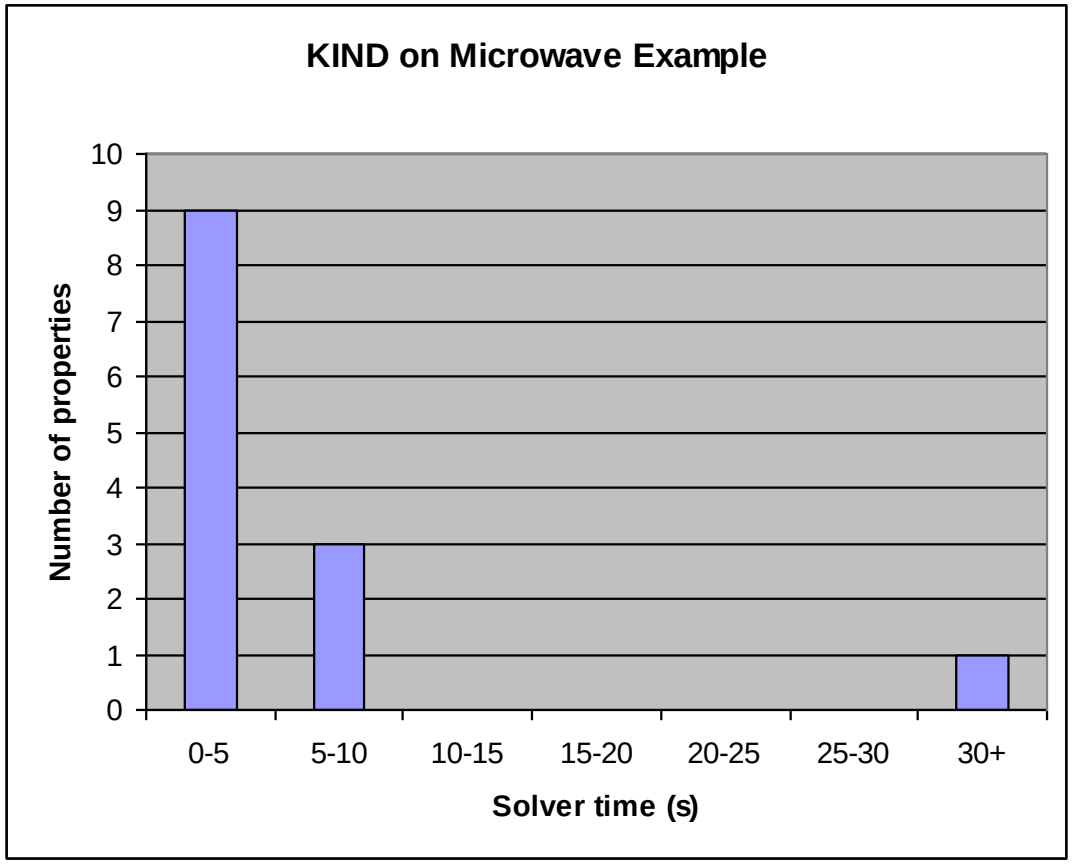
Model-checking was more cost effective than testing at finding design errors.

KIND Model Checker

- **As model-checking technology has improved, Rockwell Collins has been looking for the "next generation" model-checker to expand the set of problems we can solve. Specifically, the following capabilities were considered:**
  - **Improved reasoning over infinite state systems**
  - **Improved reasoning over arithmetically intense systems**
  - **Low or no cost (NuSMV, SAL)**
- **KIND was our choice**
  - **KIND offers *k*-inductive model checking, which is similar to bounded model checking, but works for many- or infinite-state systems.**
  - **KIND has performed well on a large group of our models**
  - **We have made some improvements**
    - Integrated with the Rockwell Collins Translator GUI
    - Intuitive presentation of solver results (counterexamples, etc.)
    - Invariant generation

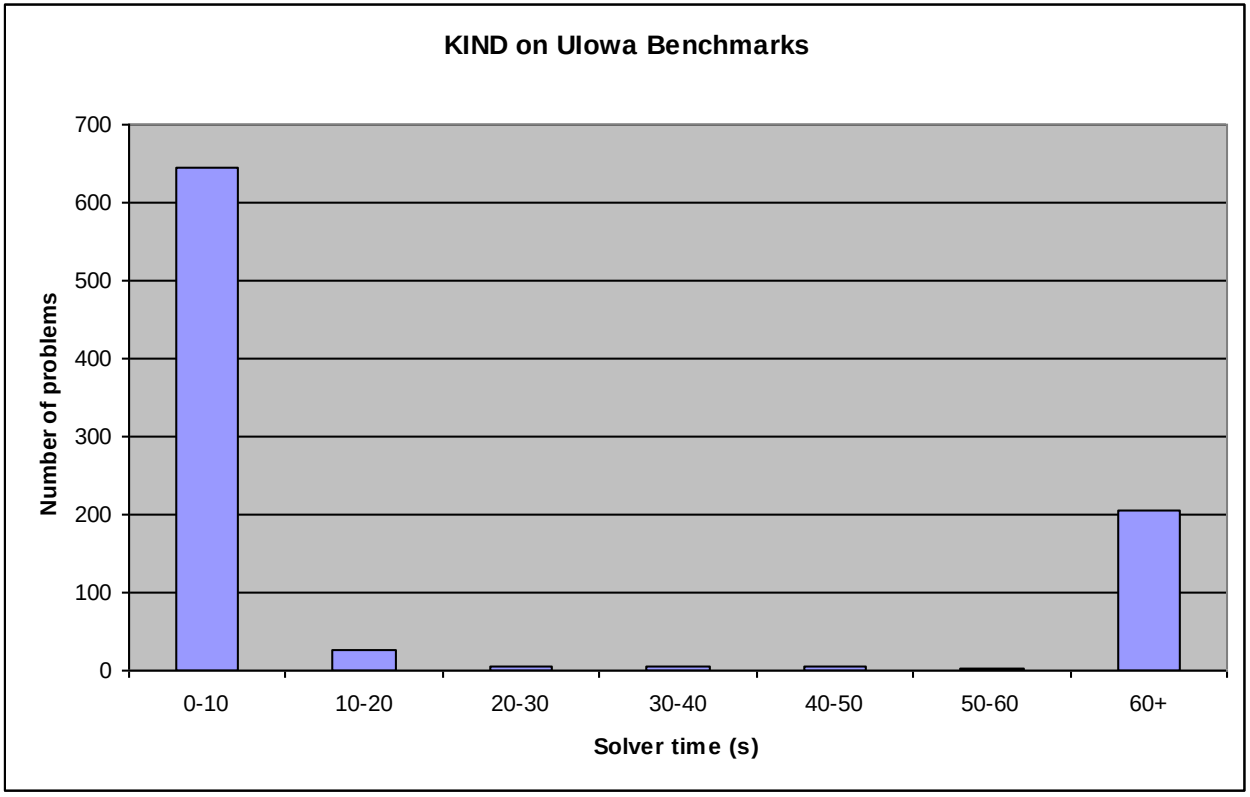
KIND Initial Benchmark Results

- **KIND showed promising results on our first tests**
- **The Microwave Example is a simple model of a microwave, with 13 properties.**



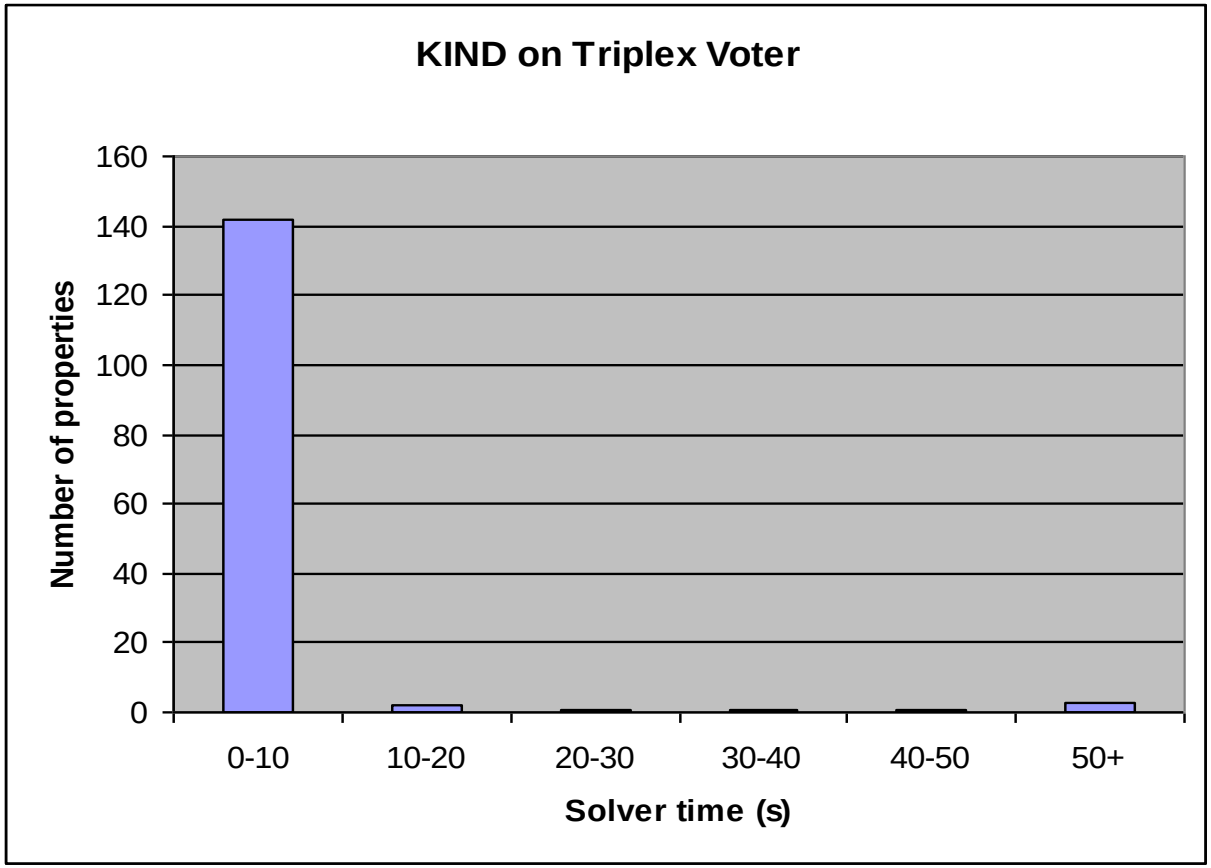
More KIND Benchmark Results

- **The Univ. of Iowa benchmark set contains 896 models from various domains (memory, simulation, counters, etc.), each with a safety property.**
- **Roughly half are valid, and half are falsifiable**



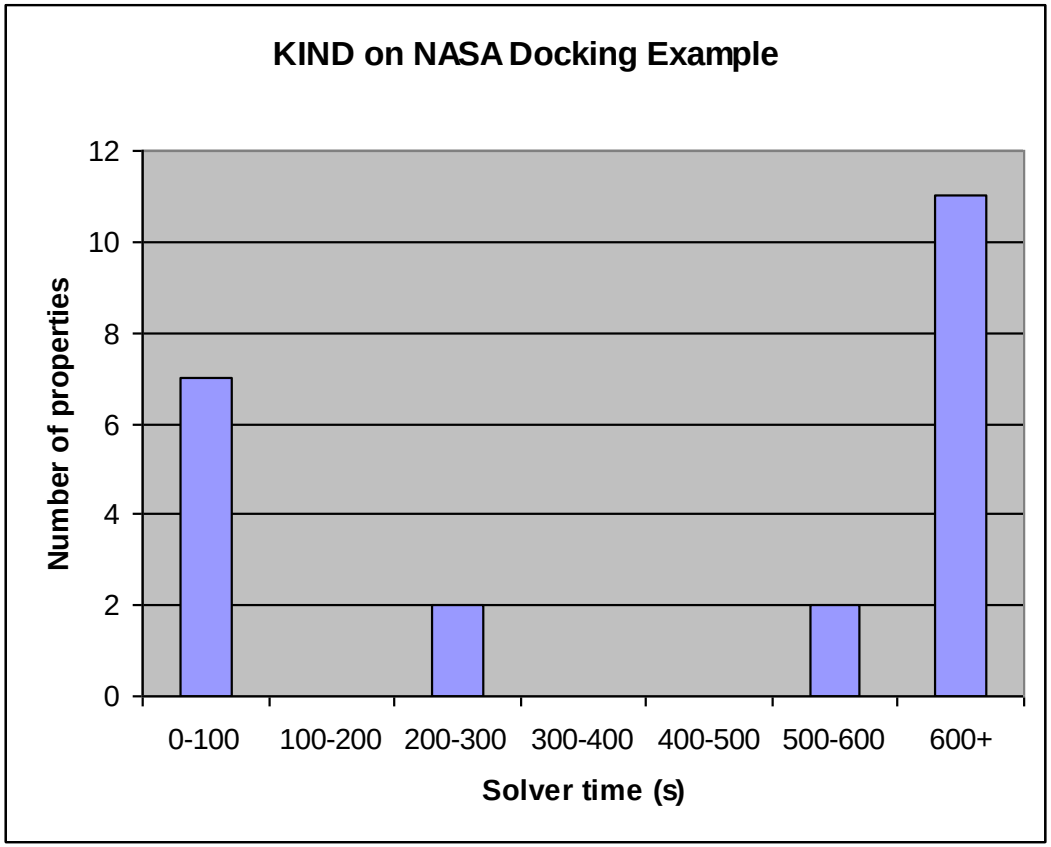
Triplex Voter

- **The triplex voter is a large control system with 150 properties and 23 hand-calculated invariants/assumptions.**



Complex Stateflow Model

- **The NASA Docking Example models a spacecraft docking procedure. We examined 22 safety properties with KIND.**
- **A large percentage of these appear to be non-k-inductive!**



XML Property Reports

- **KIND needed a uniform output format**
- **We developed an XML schema for solver outputs:**

```
<?xml version="1.0"?>
<Results xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <File>microwave.kind.lus</File>
  <Property name="s1">
    <Date>2011-03-25</Date>
    <Runtime unit="sec" timeout="false">9.124</Runtime>
    <K>13</K>
    <Answer>valid</Answer>
  </Property>
</Results>
```

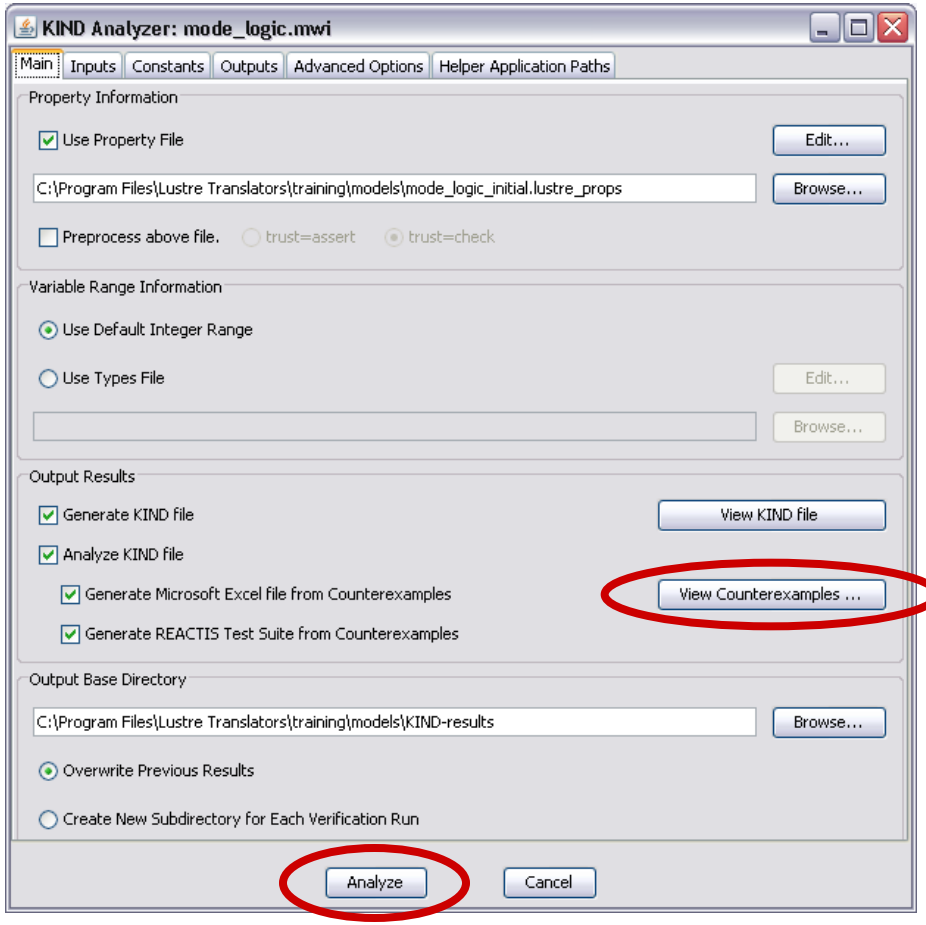
False Property with Counterexample

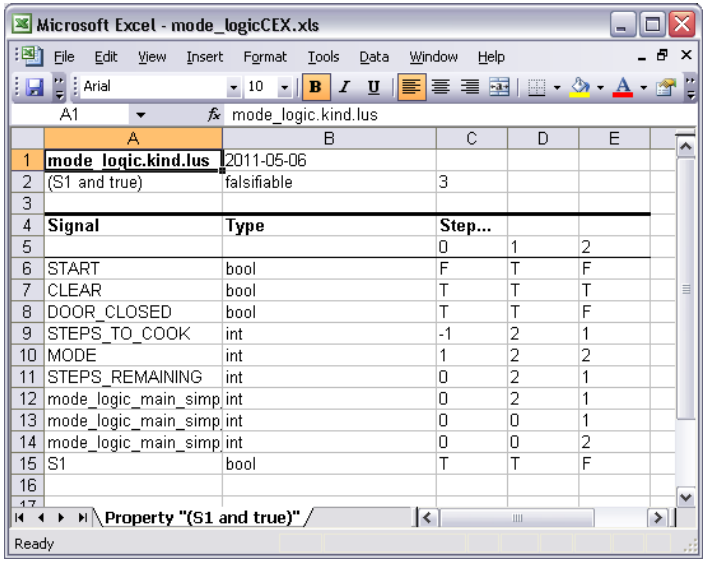
- **Counterexamples are presented in a form that can be parsed and displayed as a spreadsheet, etc.**

```
<?xml version="1.0"?>
<Results xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <File>C:\test\large\car_all_e3_1068_e5_882.kind.lus</File>
  <Property name="OK">
    <Date>2011-05-19</Date>
    <Runtime unit="sec" timeout="false">0.328</Runtime>
    <K>2</K>
    <Answer>falsifiable</Answer>
    <Counterexample>
      <Signal name="OK" node="top" type="bool">
        <Value time="0">1</Value>
        <Value time="1">0</Value>
      </Signal>
      <Signal name="voiture_speed" node="top" type="int">
        <Value time="0">0</Value>
        <Value time="1">-2</Value>
      </Signal>
      <Signal name="voiture_time" node="top" type="int">
        <Value time="0">0</Value>
        <Value time="1">0</Value>
      </Signal>
    </Counterexample>
  </Property>
</Results>
```

Graphical User Interface

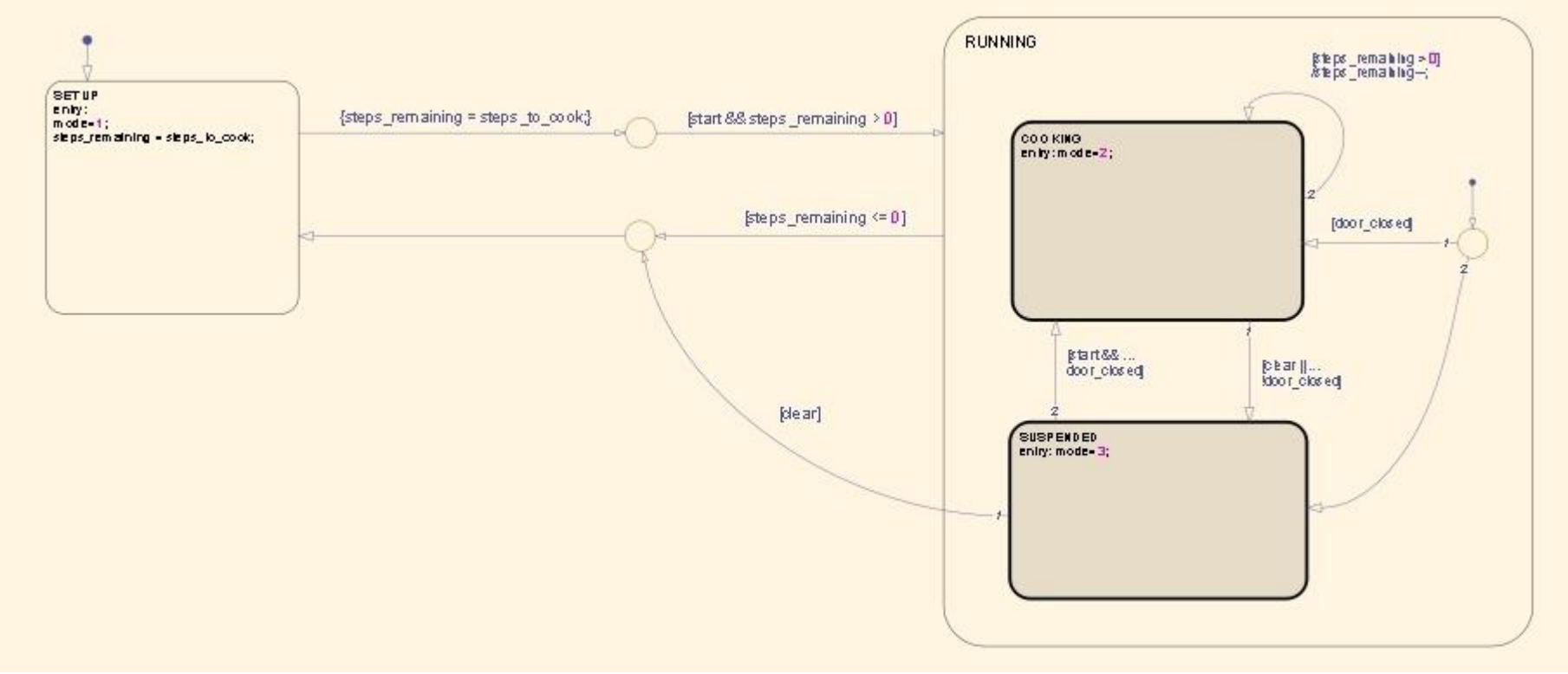
- **The user interface to the tool is important, since many potential users are not formal methods experts.**





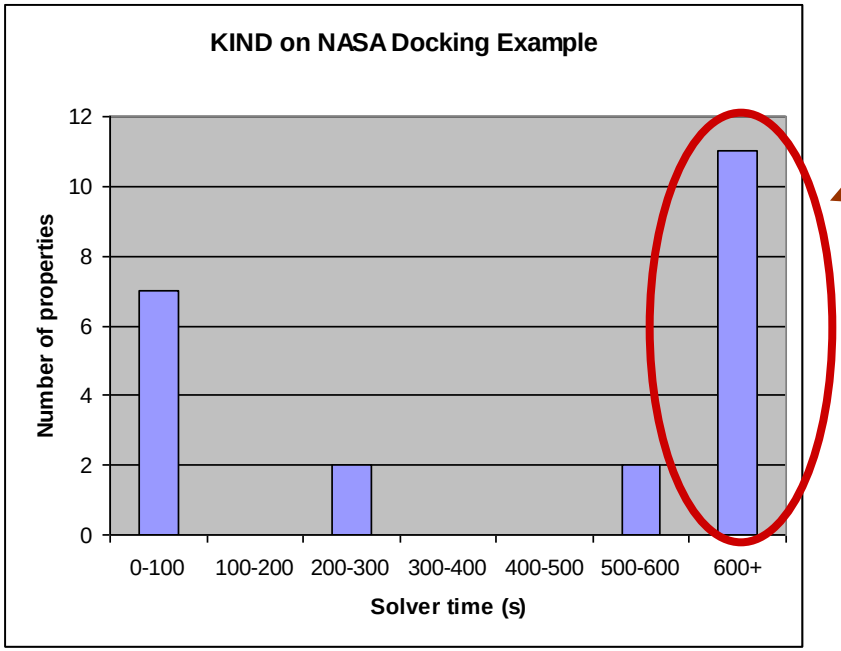
Difficulty - Stateflow Models

- **Stateflow is a complex modeling language with a variety of state diagram-like constructs.**
- **Lustre models derived from Stateflow are hard to verify**
  - **State, counters, variable interdependencies**



Solution - Invariant Generation

- **Invariant Generation via static analysis!**
- **Range invariants can change a property from very difficult (K > 100) to easy (K = 5).**
- **Simple implications between variables ("mode" invariants) can change the property from difficult (K > 50) to easy (K = 2).**



Properties needing additional invariants

Acknowledgements

- **NASA Langley Research Center**
  - **Ricky Butler**
  - **Paul Miner**
  - **George Hagen (formerly of University of Iowa)**
- **Air Force Research Labs**
  - **Dave Homan**
  - **Jon Hoffman**
  - **Brian Hulbert**
  - **Wendy Chou**
- **University of Minnesota**
  - **Prof. Mats P. E. Heimdahl**
  - **Michael Whalen (formerly of Rockwell Collins)**
- **University of Iowa**
  - **Prof. Cesare Tinelli**
  - **Dr. Teme Kahsai**

Questions/Comments?