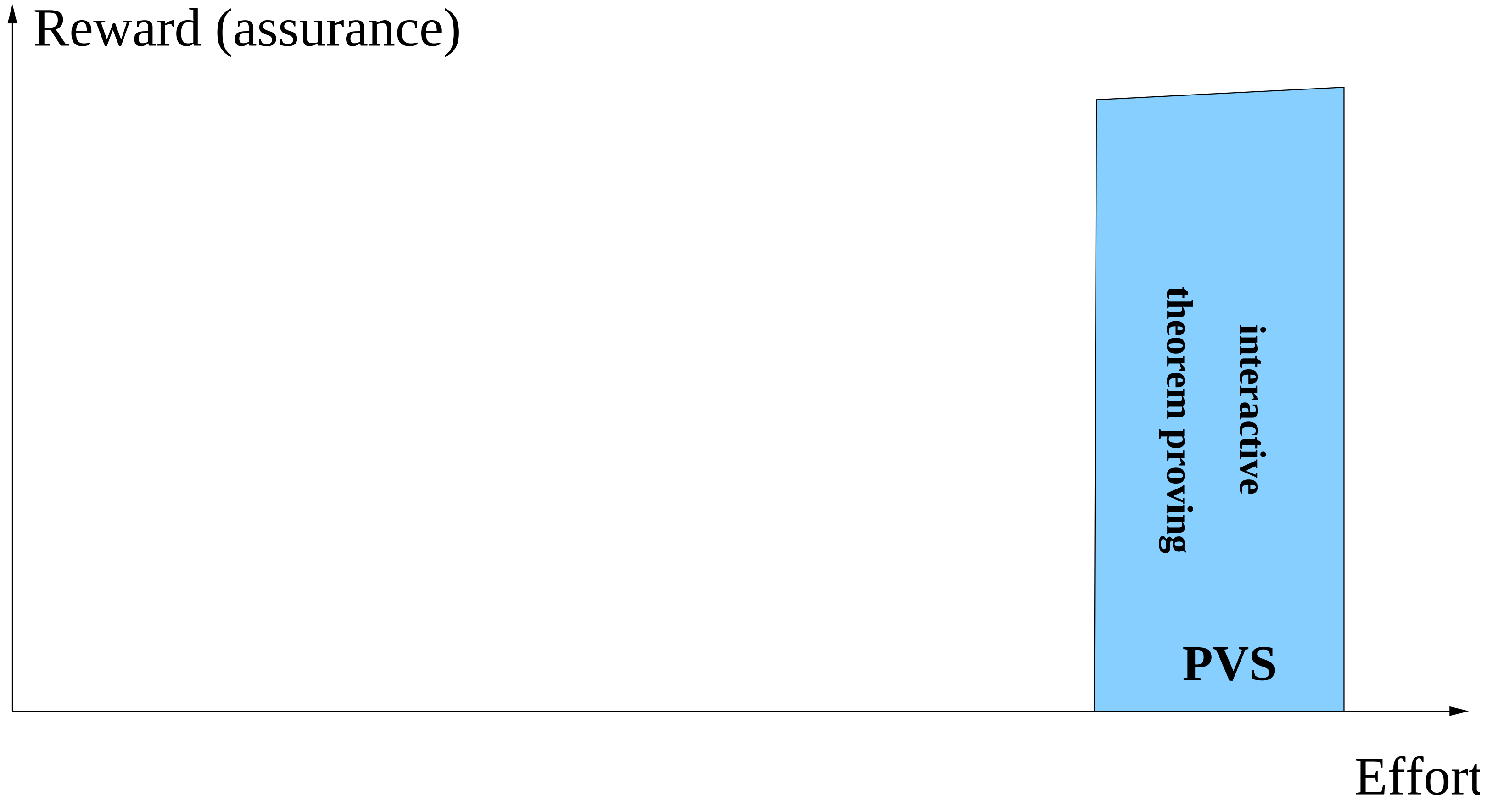
IFIP WG 10.4 Winter Meeting, Rincon PR 30 Jan 2005

Automated Test Generation with `sal-atg`

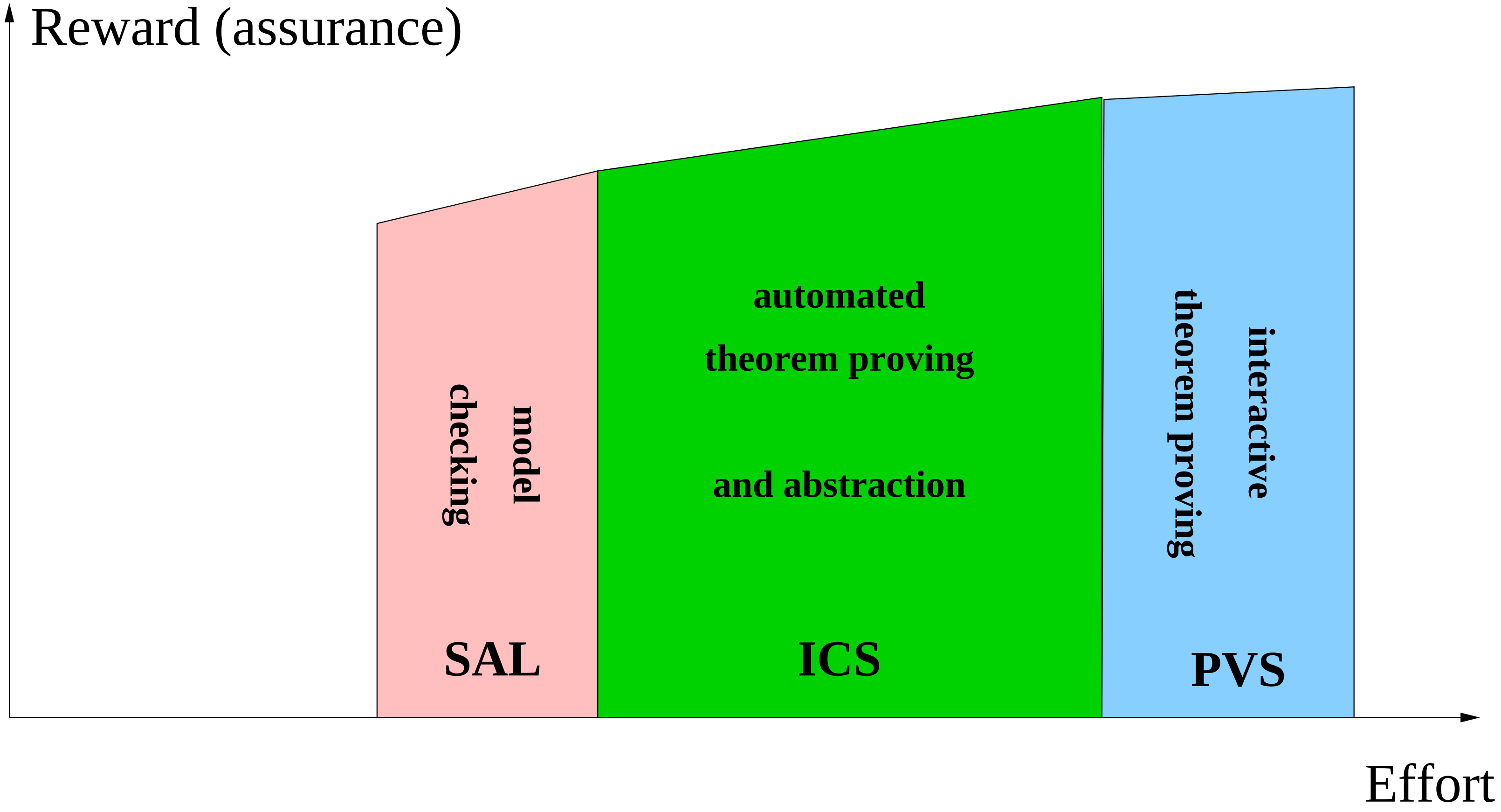
John Rushby
with Grégoire Hamon and Leonardo de Moura

Computer Science Laboratory
SRI International
Menlo Park CA USA

Full Formal Verification is a Hard Sell: The Wall

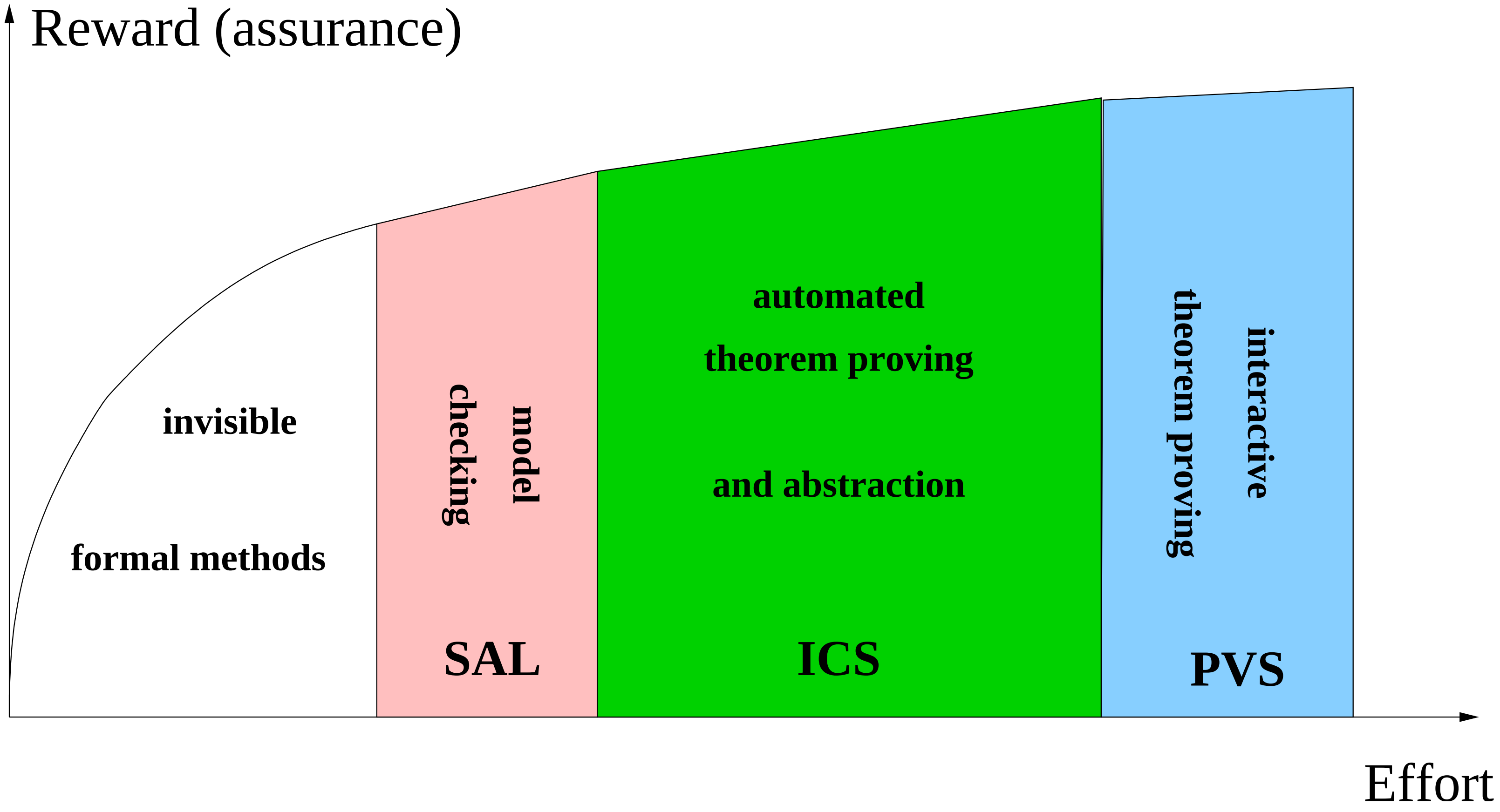


Newer Technologies Improve the Value Proposition



But only by a little

The Unserved Area Is An Interesting Opportunity



Conjecture: reward/effort climbs steeply in the invisible region

Invisible Formal Methods

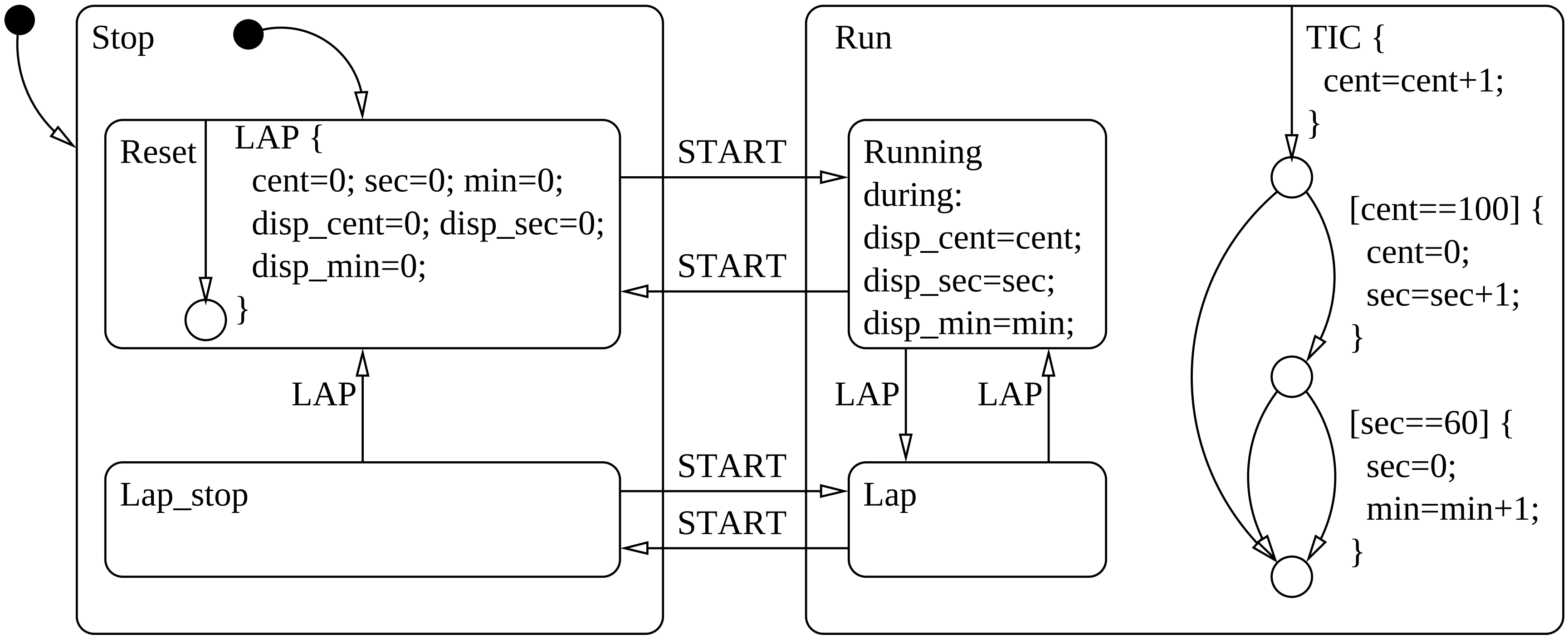
- Use the technology of formal methods
    - Theorem proving, constraint satisfaction, model checking, abstraction, symbolic evaluation
- To augment traditional methods and tools
    - Compilers, debuggers
- To automate traditional processes
    - Testing, reviews, debugging
- Or to create new capabilities
    - Strong static analyzers, autocode by constraint solving
- To do this, we must unobtrusively (i.e., invisibly) extract
    - A formal specification
    - A collection of properties
- And deliver a useful result in a familiar form

Invisible FM Example: Generating Unit Tests

- Necessity and costs of testing well understood
- Automation could be a huge win
- In model based development (MBD), we have an executable model of the system (e.g., in Simulink/Stateflow)
- Generate tests by structural coverage in the model
- Model also provides the oracle
- It is well known that model checkers can be used as test generators

Example: Stopwatch in Stateflow

Inputs: START and LAP buttons, and clock TIC event

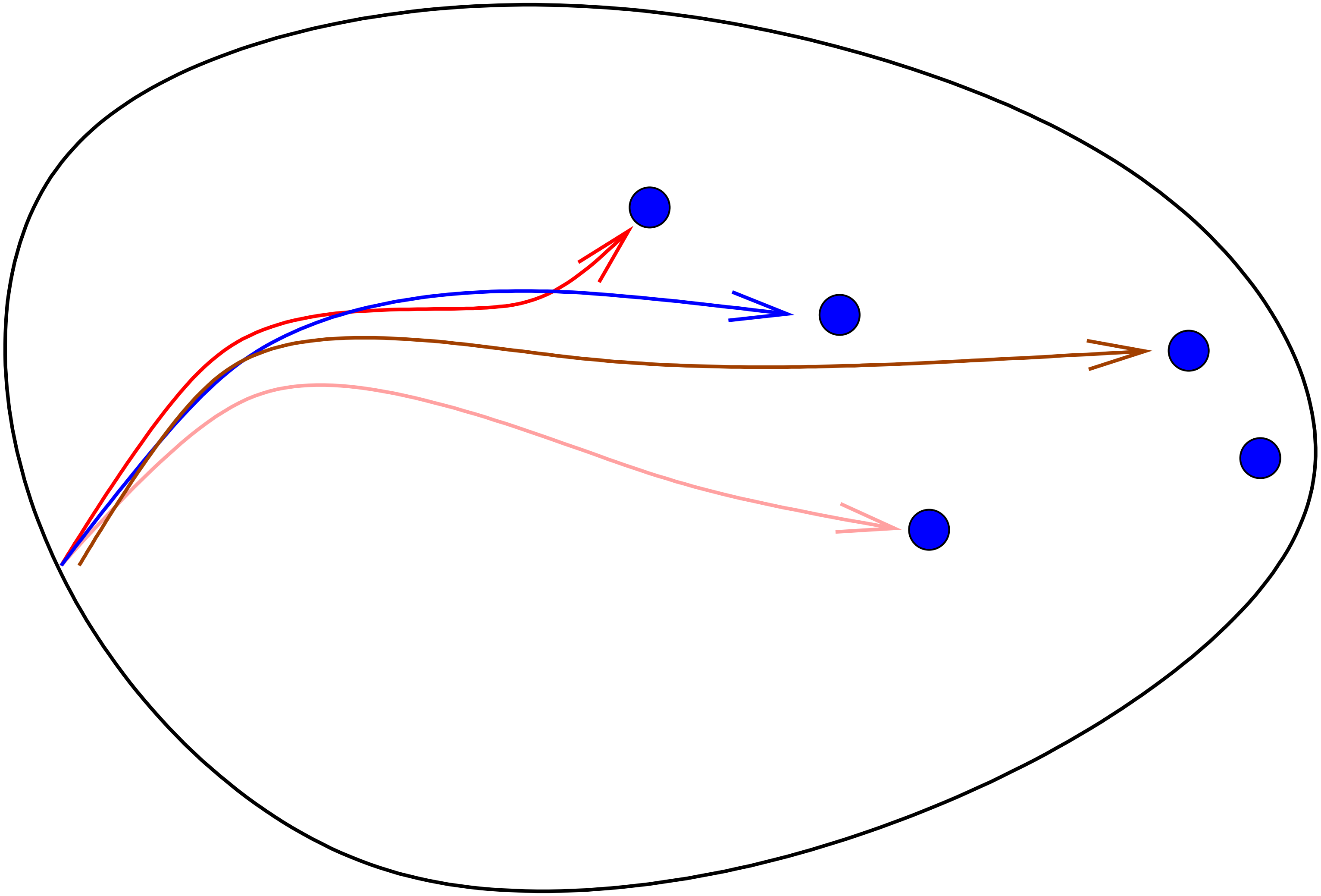


Example test goals: generate input sequences to exercise Lap_stop to Lap transition, or to reach junction at bottom right

Generating Tests Using a Model Checker

- Add trap variables go TRUE when a test goal is satisfied
  - E.g., jabr that goes TRUE when junction at bottom right is reached
  - Trap variables can be inserted automatically during translation from the MBD language to the model checker (Our translator from Stateflow to SAL does this)
- Model check for "always not jabr"
- Counterexample will be desired test case
- Trap variables add negligible overhead ('cos no interactions)
- For finite cases (e.g., numerical variables range over bounded integers) any standard model checker will do
  - Although many pragmatic issues concerning symbolic vs. bounded vs. explicit vs. . . for this application
  - Otherwise need infinite bounded model checker as in SAL

Tests Generated Using a Model Checker



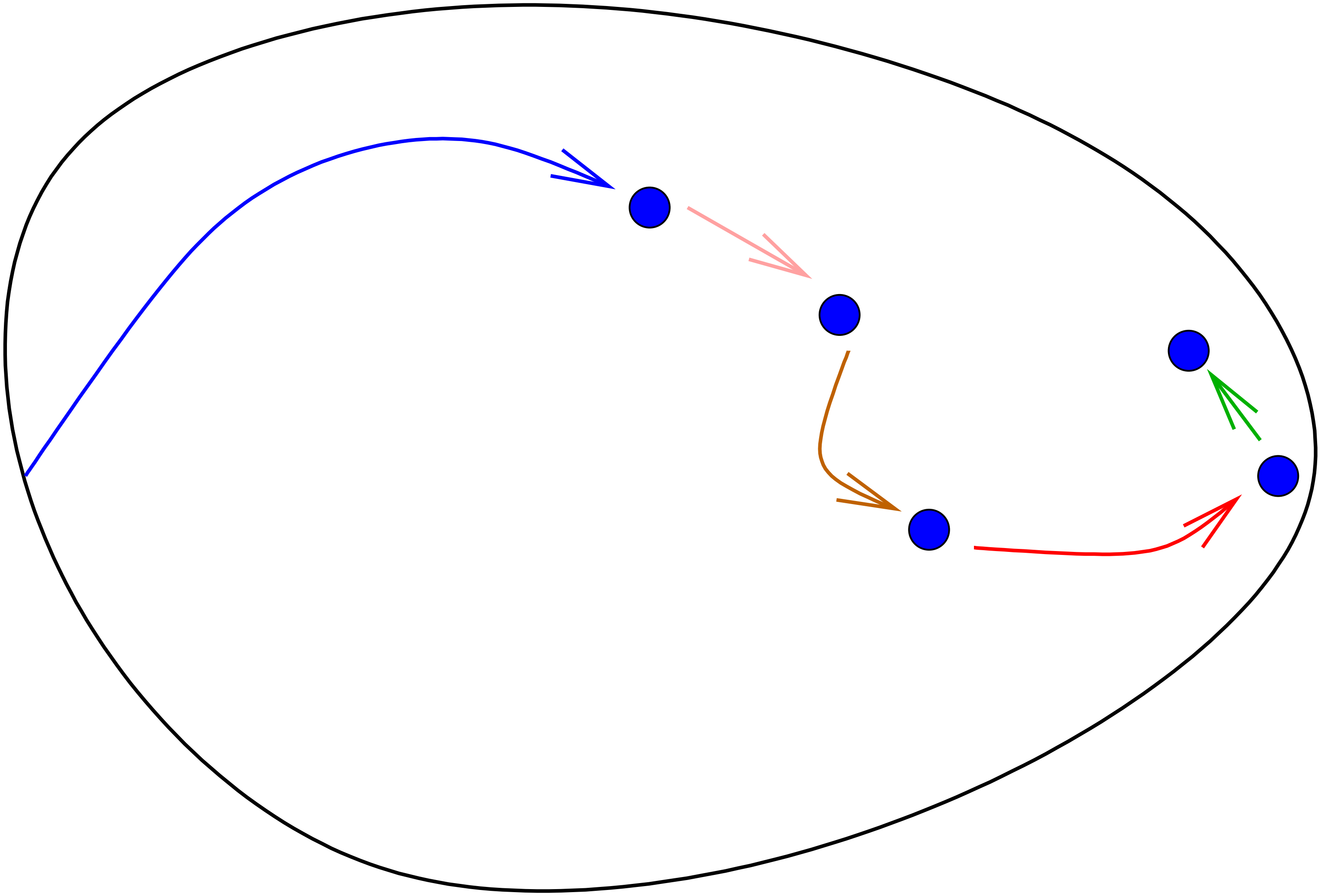
Problems Using OTS Model Checker as Test Generator

- Each test goal is treated separately: model checker is called repeatedly and performs much redundant work
- Test set has many short tests
  - Each incurs a startup cost during execution
  - Total length is large, so high execution cost
  - Much redundancy among the tests (wasteful)
  - Few long tests (so deep bugs undetected)
- Model checker may be unable to reach deep test goals

A Better Way

- Instead of starting each test from the the start state, we try to extend the test found so far
- Extending tests allows a bounded model checker to reach deep states at low cost
  - 5 searches to depth 4 much easier than 1 to depth 20
- Could get stuck if we tackle the goals in a bad order
- So, simply try to reach any outstanding goal and let the model checker find a good order
  - Can slice the model after each goal is discharged
  - A virtuous circle: the model will get smaller as the remaining goals get harder
- Go back to the start (or another earlier state) when unable to extend current test

An Efficient Test Set



Less redundancy, and longer tests tend to find more bugs

The SAL Automated Test Generator: sal-atg

- SAL is scriptable in Scheme
- `sal-atg` implements the method described in a few hundred lines of Scheme
    - ◦ (Re)starts use either symbolic or bounded model checking
        - ⋆ Parameterized choice and search depth
    - ◦ Extensions use bounded model checking
        - ⋆ Parameterized incremental search depth
    - ◦ Optional slicing after each extension or each restart
    - ◦ Customizable output to drive test harness

Example

- `sal-atg stopwatch clock stopwatch_goals.scm -ed 5 --incremental`
  In 5 seconds, generates single test case of length 17 that covers the states and transitions of the Statechart

- `sal-atg stopwatch clock stopwatch_goals.scm -ed 5 -id 0 --incremental --smcinit`

- Takes 106 seconds to cover flowchart as well: adds test of length 101 for middle junction and one of length 6,001 for jabr

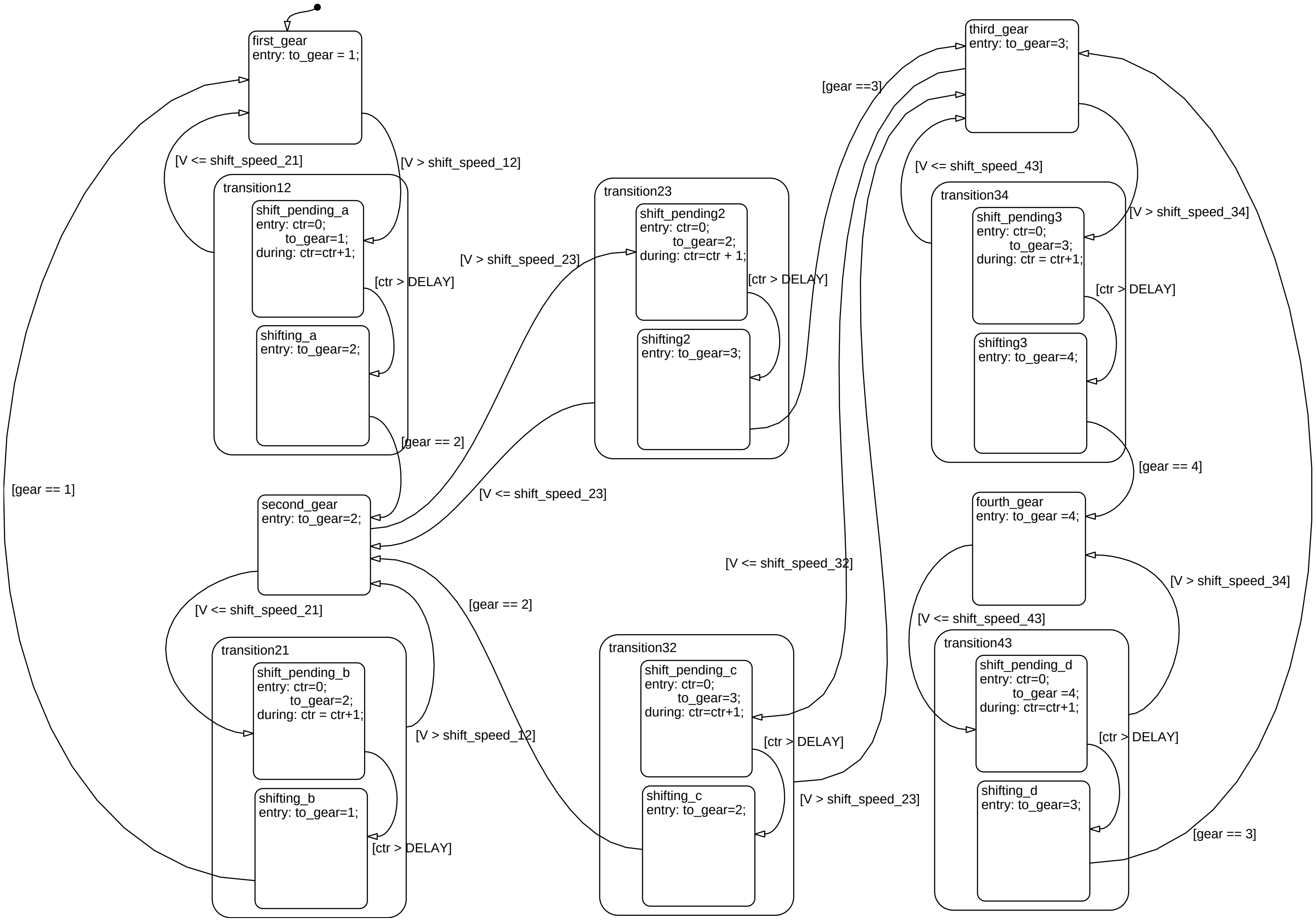
Experimental Results

- Rockwell Collins has developed a series of flight guidance system (FGS) examples for NASA
- SAL translation of largest of these kindly provided by UMN
- Model has 490 variables (576 state bits), 196 reachable control states, and 313 transitions
    - Takes 61 seconds to generate single test case of length 45 that covers all states
    - Takes 98 seconds to generate a single test of length 55 that covers all transitions
- Without extensions, get 73 tests to cover transitions: 1 of length 3, 9 of length 2, and the rest of length 1
    - Poor mutant detection
- We are in the process of testing our tests

Test Engineering with Automation

- Generating tests just to achieve structural coverage is a poor strategy
- Traditional test engineers develop tests to explore interesting cases, requirements, fault hypotheses
- We need to give them a way to do this using automation
- Specify the desired tests rather than constructing them
- Develop an observer module that sets a variable TRUE when a test has achieved some purpose
- Tell `sal-atg` to search for conjunction of each trap variable with the purpose
- In general, `sal-atg` can search for arbitrary conjunctions
  - E.g., product of structural coverage on control states and boundary coverage on some data structure

Example Shift Scheduler



Shift Scheduler

- One input is the gear currently selected by the gearbox
- Tests often change this discontinuously (e.g., 1, 3, 4, 2)
- Can easily establish the test purpose to change only in single steps, and to change at every step

Please Try It Out

- Main FM tools home page: http://fm.csl.sri.com
- SAL home page: http://sal.csl.sri.com
- SAL-atg (next week): http://sal.csl.sri.com/pre-release