“Model-Checking” Software with VeriSoft

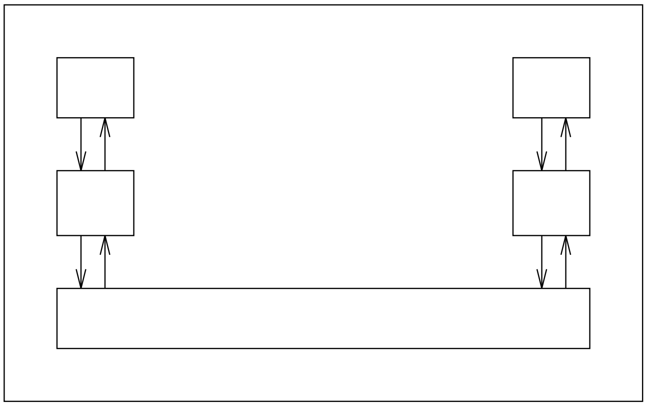
Patrice Godefroid

Bell Laboratories, Lucent Technologies

Overview:

1. What is VeriSoft?
2. How does it work?
3. Industrial applications.
4. Summary + comparison with related work.
5. Future work – challenges.

1. What is VeriSoft? Concurrent Reactive System Analysis



Each component is viewed as a "reactive" system, i.e., a system that continuously interacts with its environment.

Precisely, we assume:

- finite set of *processes* executing aribitrary code (e.g., C, C++, Java, Tcl, ...);
- finite set of *communication objects* (e.g., message queues, semaphores, shared memory, TCP connections, UDP packets,...).

Problem:
Developing concurrent reactive systems is hard!
(many possible interactions)

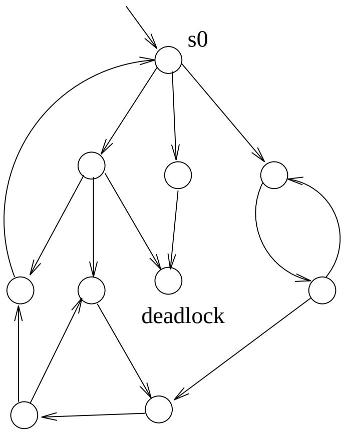
Traditional testing is of limited help! (poor coverage)

Scenarios leading to errors are hard to reproduce!

Alternative: *Systematic State-Space Exploration*

State Space (Dynamic Semantics)

- Processes communicate by executing *operations* on communication objects.
- operations on communication objects are *visible*, other operations are *invisible*;
- only executions of visible operations may be *blocking*;
- the system is said to be in a *global state* when the next operation to be executed by *every* process is *visible*;
- a move from one global state to another global state is a *transition*;
- *state space* = set of global states + transitions.



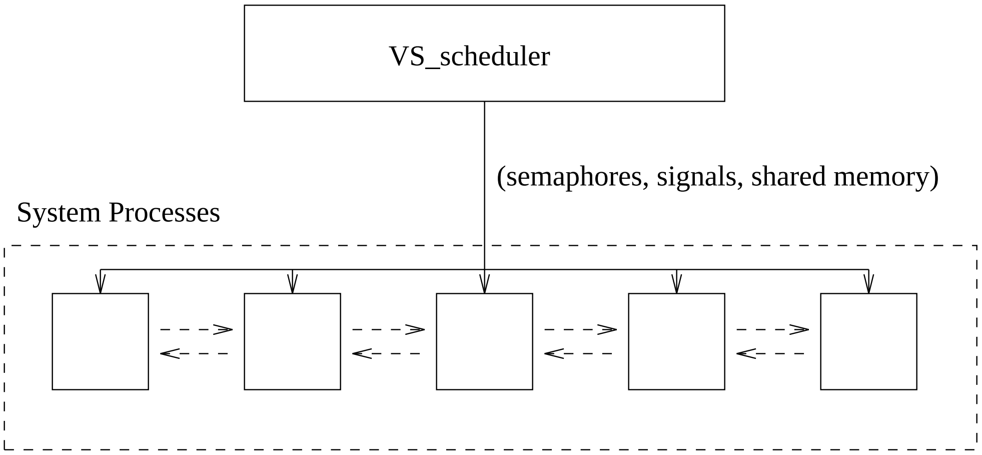
Theorem: Deadlocks and assertion violations are preserved in the "state space" as defined above.

Systematic State-Space Exploration

VeriSoft can systematically explore the state space of a concurrent reactive system.

Interceptions of all visible operations:

- control of all the processes;
- complete control over nondeterminism (i.e., concurrency + VS_toss(n));
- observation of visible operations and global states.



VeriSoft

VeriSoft searches state spaces for:

- deadlocks,
- assertion violations,
- livelocks (no enabled transition for a process during x successive transitions),
- divergences (a process does not communicate with the rest of the system during more than x seconds).

When an error is detected, VeriSoft reports a *scenario* leading to that error.

An interactive graphical simulator/debugger is also available.

2. How does VeriSoft work?

VeriSoft looks simple! Why did we have to wait for so long (15 years) to have it?

Existing state-space exploration tools are restricted to the analysis of *models* (i.e., abstract descriptions) of software systems.

Each state is represented by a *unique identifier*.

During state-space exploration, visited states are saved in memory (hash-table, BDD,...).

With *programming languages*, states are much more complex!

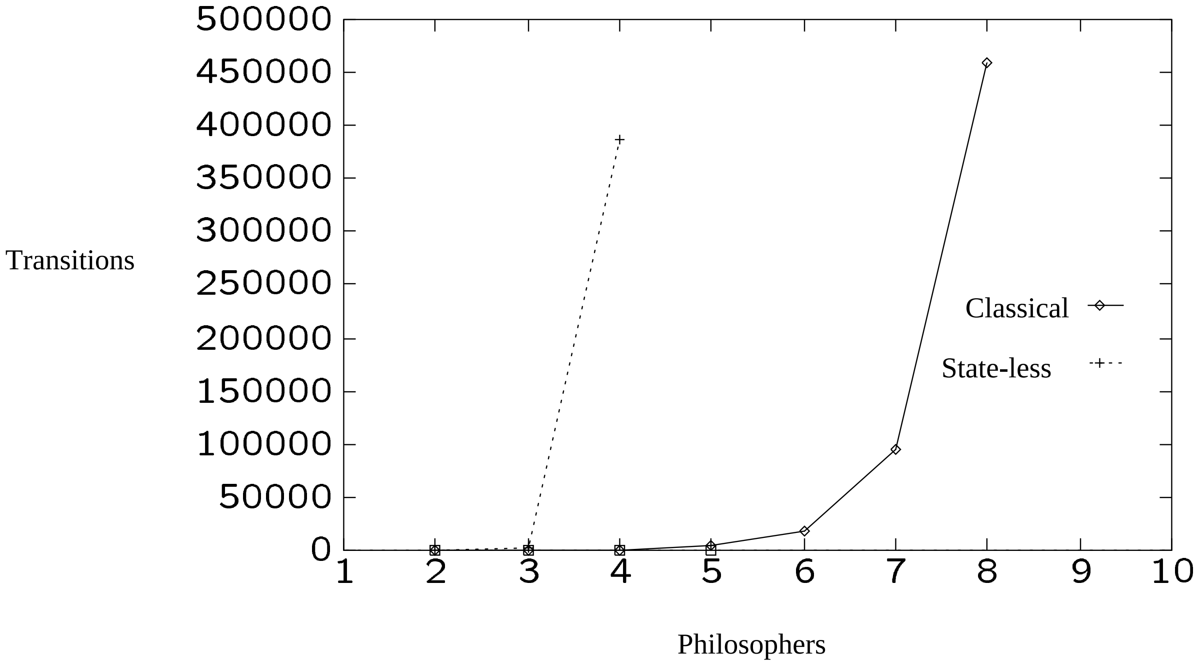
Computing and storing a “unique identifier” for each state is unrealisitc!

State-Less Search

Idea: perform a *state-less search*!
(still terminate when state space is acyclic)

Equivalent to *"state-space caching"* with an empty cache: this search technique is terribly inefficient! [H85,JJ91]

Example: dining philosophers (toy example)



For 4 philosophers, a state-less search explores 386,816 transitions, instead of 708.

Every transition is executed on average 546 times!

An Efficient State-Less Search

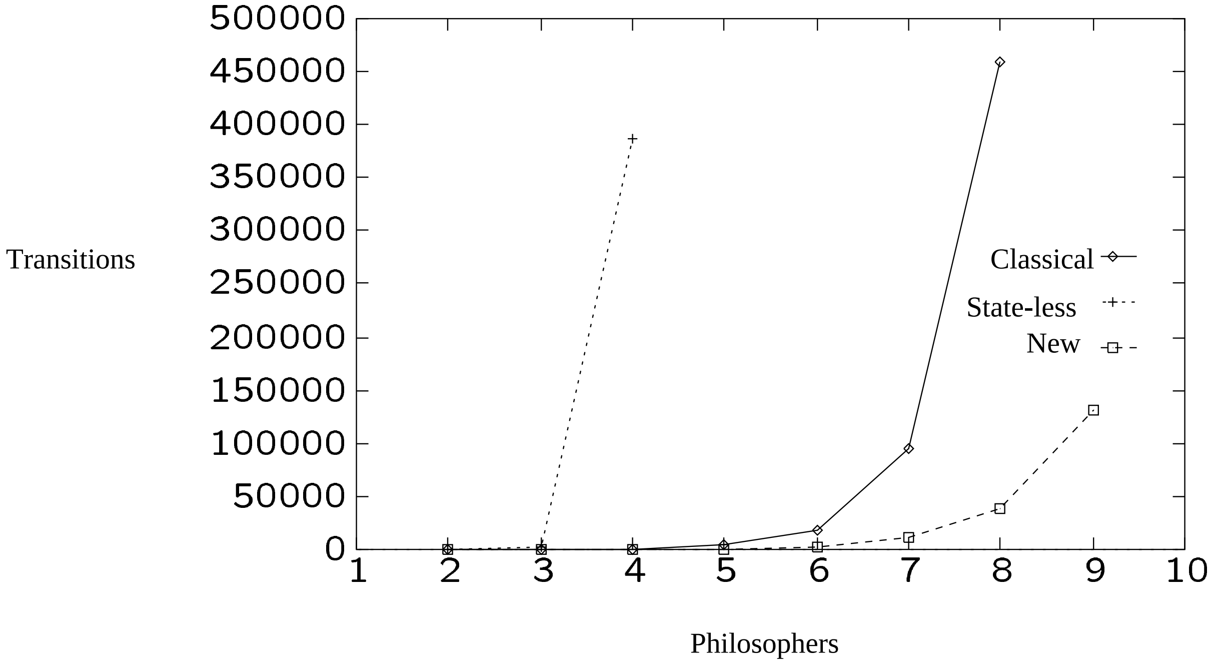
[GHP92]: Redundant explorations due to state-space caching can be strongly reduced by using *Sleep Sets* [G90], and *"partial-order methods"* in general [G96].

VeriSoft: original algorithm combining

- state-less search,
- sleep sets [G90,GW93],
- conditional stubborn sets [V90,GP93,G96].

Theorem: For finite acyclic state spaces, the above algorithm can be used for the detection of deadlocks and assertion violations without incurring the risk of any incompleteness in the verification results.

Observation:
when using this algorithm, most of the states are visited *only once* during the search.
⇝ Not necessary to store them!



VeriSoft – Summary

VeriSoft is the first tool for systematically exploring the state spaces of systems composed of several concurrent processes executing arbitrary (e.g., C or C++) code.

Originality: framework, search, tool [POPL'97].

The key to make this approach tractable is to use *smart* state-space exploration algorithms!

In practice, the search is typically incomplete.

From a given initial state, VeriSoft can always guarantee a complete coverage of the state space up to some depth.

3. Industrial Applications

Examples of Applications:
(within Lucent Technologies)

3.1 4ESS Heart-Beat Monitor analysis
(debugging, reverse-engineering).

3.2 Wavestar 40G integration testing
(testing).

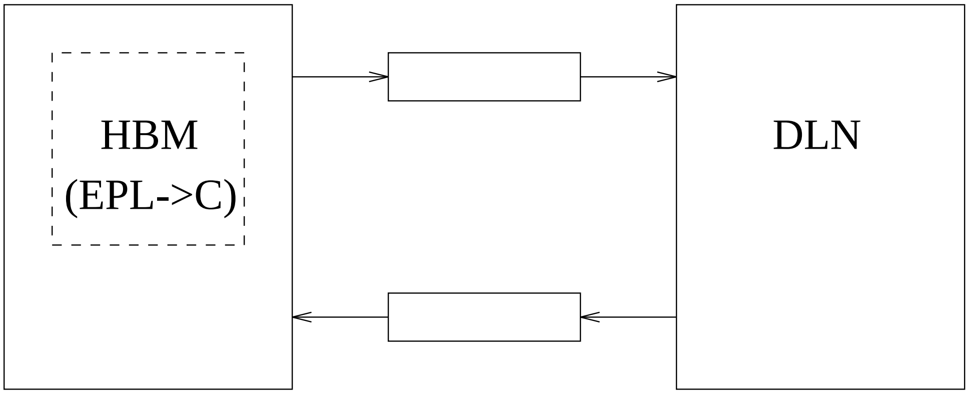
3.3 Automatic Protection Switching analysis
(interoperability protocol testing).

3.1 4ESS Heart-Beat Monitor Analysis

- May affect millions of calls per day.
- Determines status of elements connected to 4ESS switch from propagation delays of messages.
- Plays an important role in routing new calls in 4ESS switch (by triggering “No Trunk Hunt” (NTH) = switch from out-of-band to in-band signalling).
- November 1996: “field incident”…
- June 1997: calls from “field rep.”…
- Code is 7 years old, modified 3 years ago.
- Several hundred lines of EPL (assembly) code.
- How does this code work exactly???

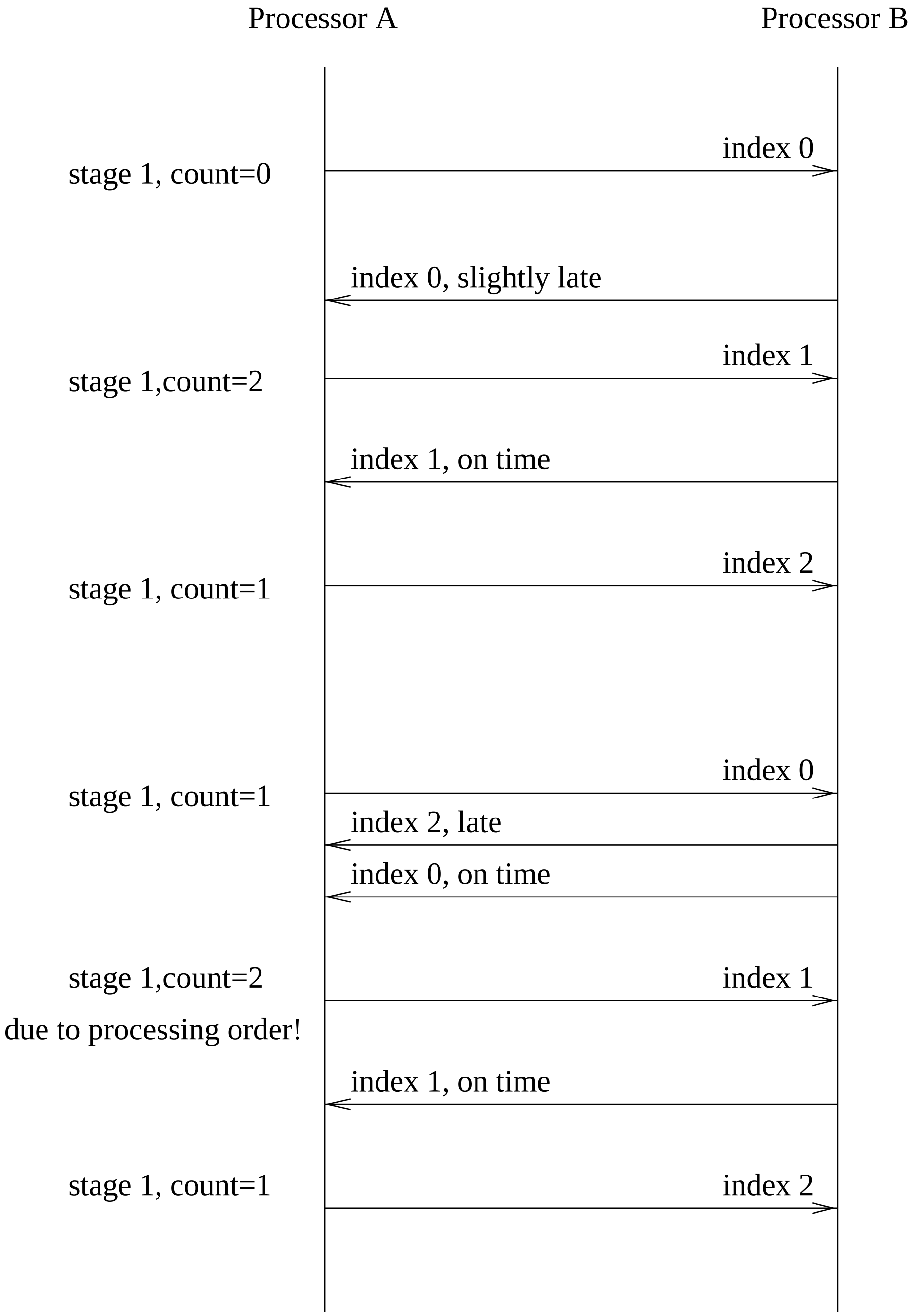
Analysis of 4ESS HBM using VeriSoft

- Translate EPL code to C code (using existing partial compiler).
- Build test harness for HBM C code: simple “wrapper” program (takes only a few hours!).
- Model the environment of the HBM: with “VS_toss(n)” (takes only a few hours!).
- Add “VS_assert(0)” where NTH in HBM code.
- Check properties (reverse-engineering ↔ testing).



⇝ Discovered flaws in documentation and unexpected behaviors in software itself...

Example of Scenario Found



(See paper [BLTJ'98] for details.)

Conclusions of the 4ESS HBM Analysis

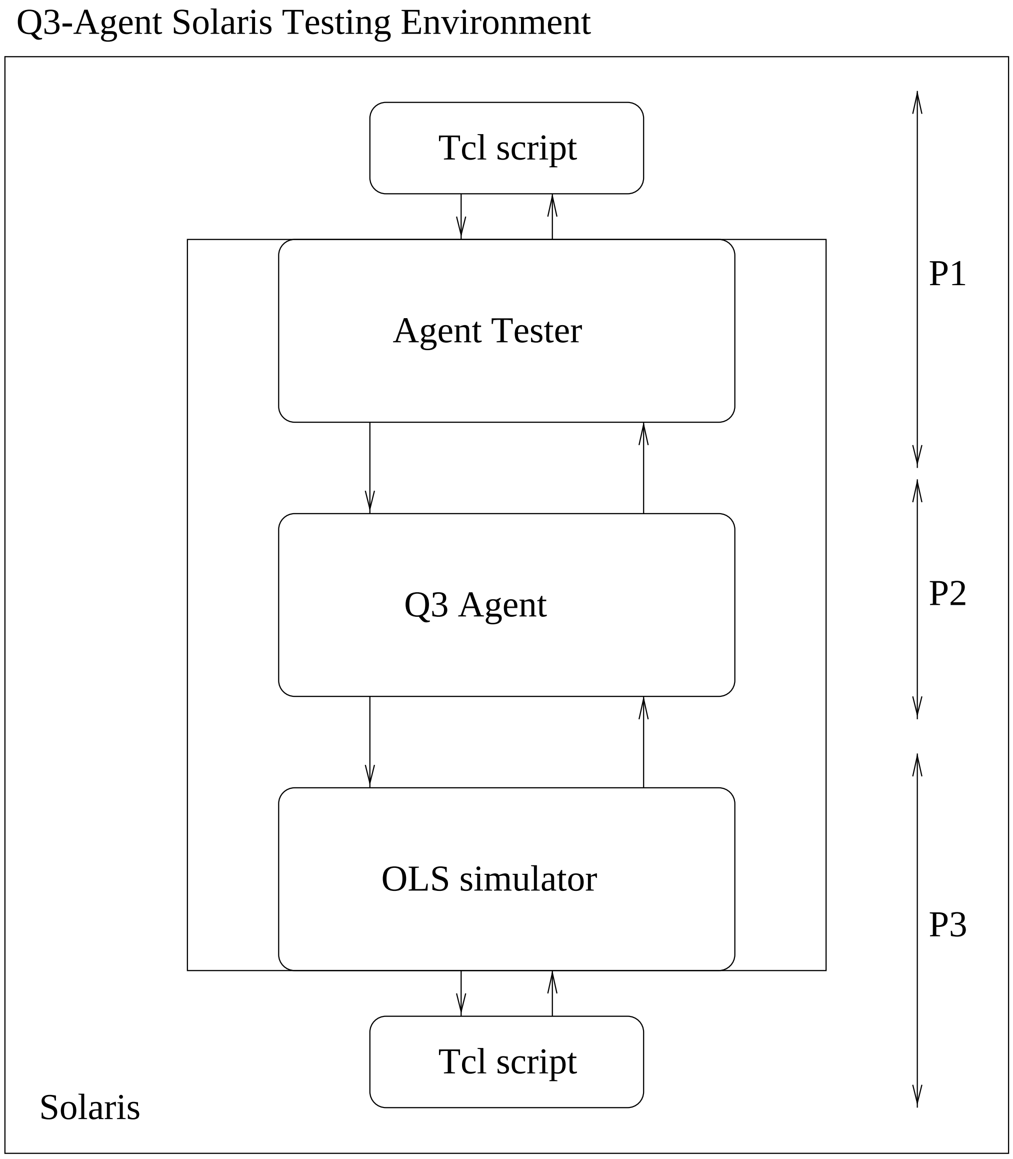
HBM: Analysis revealed flaws in documentation and unexpected behaviors in software itself.

- HBM code is very "irregular"
  (very hard to predict behavior).
- Similar analysis performed on previous version:
  - more sensitive, although not strictly;
  - also "irregular".
- Design of a new version:
  - passes these tests;
  - implemented in new release.

VeriSoft:

- Can quickly reveal behaviors virtually impossible to detect using conventional testing techniques
  (due to lack of controllability and observability).
- Strength: no need to model the application!
  - Eliminates this time-consuming and error-prone task required with other state-space exploration tools.
  - VeriSoft is WYSIWYG: great for reverse engineering!

3.2 Wavestar 40G Integration Testing



"Black-box" testing, large processes ($O(10^5 - 10^6)$ lines of C/C++ code).

Wavestar Testing with VeriSoft (work in progress)

- From the testers' point of view, two main new Tcl commands are available with VeriSoft:
  - VS_toss simulates nondeterminism.
  - VS_assert is used to determine whether test passed/failed.

  These commands can be used anywhere
  (any language, any procedure, any process).

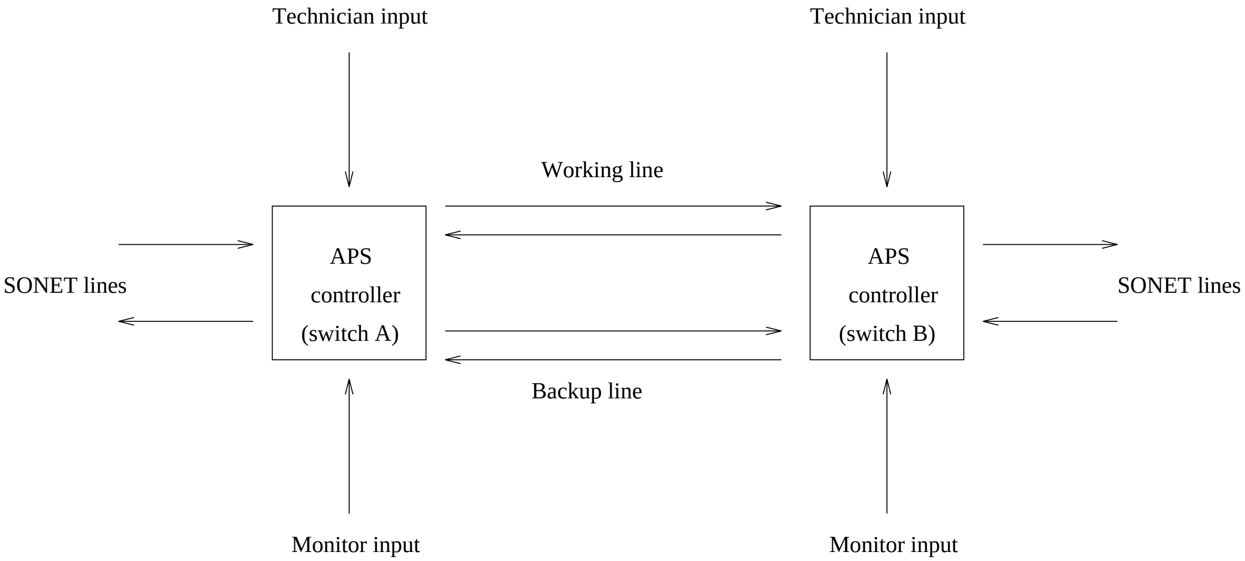
- A single nondeterministic test script can specify a family of thousands of (deterministic) test scripts.

```
[...]
if VS_toss(1) then event = MSG1
            else event = MSG2;
switch(VS_toss(2)) {
    case 0:  param = PARAM1;
        break;
    case 1:  param = PARAM2;
        break;
    case 2:  param = PARAM3;
};
send-to-IUT event param
if (test-failed) then VS_assert(0)
[...]  /* 6 possible combinations */
```

- All these test scripts are automatically generated, executed and evaluated by VeriSoft.

Note: some of these test scripts can also be executed in "target" environment (ok if automatic (and fast))

3.3 Automatic Protection Switching Analysis



- A 5ESS “switch-maintenance” application.
- APS protocol ensures that both switches read data from the same line. (APS is part of SONET/SDH standard.)
- Several thousands lines of C code.
- VeriSoft discovered several incompatibilities between different versions of APS code.

4. Summary: Key Features of VeriSoft

- Tool for analyzing concurrent/reactive software written in any language.
- Automatically generates, executes and evaluates (many) scenarios.
- Complete state-space coverage is guaranteed up to some depth.
- Can quickly reveal behaviors that are virtually impossible to detect using conventional testing techniques (reduce interval, increase quality).
- Applications: testing, debugging, reverse-engineering.
- An interactive graphical simulator/debugger is also available.

Comparison with Related work

Other model-checkers (for software): (e.g., SPIN, VFSMvalid)

- language dependent;
- need a model, or limited to high-level design;
- but analyzing a model is easier.

Specification-based test generation: (e.g., TestMaster)

- language dependent;
- test generation only;
- no support for concurrency (testing through a single interface only).

Static analysis techniques for automatic model extraction (ex of tool ?):

- language dependent + often need additional restrictions;
- abstraction is not a panacea: it always introduces unrealistic behaviors;
- overall, complementary with VeriSoft (e.g., see [PLDI'98]).

$\leadsto$ VeriSoft (the concept) is here to stay...

5. Future Work – Challenges

- Scalability limited by the “state explosion” problem...
- Used naively, poor feedback is likely, but used properly, can be extremely effective.
  $\leadsto$ Training is necessary!
- Help to model the environment...
  “Automatically Closing Open Reactive Programs” [PLDI’98]
- Improve feedback to user...
  coverage information, state-space visualization

“Technology transfer” is starting
inside + outside Lucent Technologies...

See `http://www.bell-labs.com/~god`

Main References

- **[G97]** “Model Checking for Programming Languages using VeriSoft”, P. Godefroid, POPL’97.
- **[G96]** “Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem”, P. Godefroid, LNCS 1032.
- **[GHJ98]** “Model Checking Without a Model: An Analysis of the Heart-Beat Monitor of a Telephone Switch using VeriSoft”, P. Godefroid, B. Hanmer and L. Jagadeesan, ISSTA’98. Journal version in Bell Labs Tech. Journal, 1998.
- **[CGJ98]** “Automatically Closing Open Reactive Programs”, C. Colby, P. Godefroid and L. Jagadeesan, PLDI’98.

See `http://www.bell-labs.com/~god`