Formal Methods in Software Engineering

Marsha Chechik
University of Toronto
Department of Computer Science
June 17, 1997
http://www.cs.toronto.edu/~chechik

Motivation

"You want to know what this system does? Run it!"

"Real programmers don't write specifications. Users should consider themselves lucky to get any programs at all and take what they get."

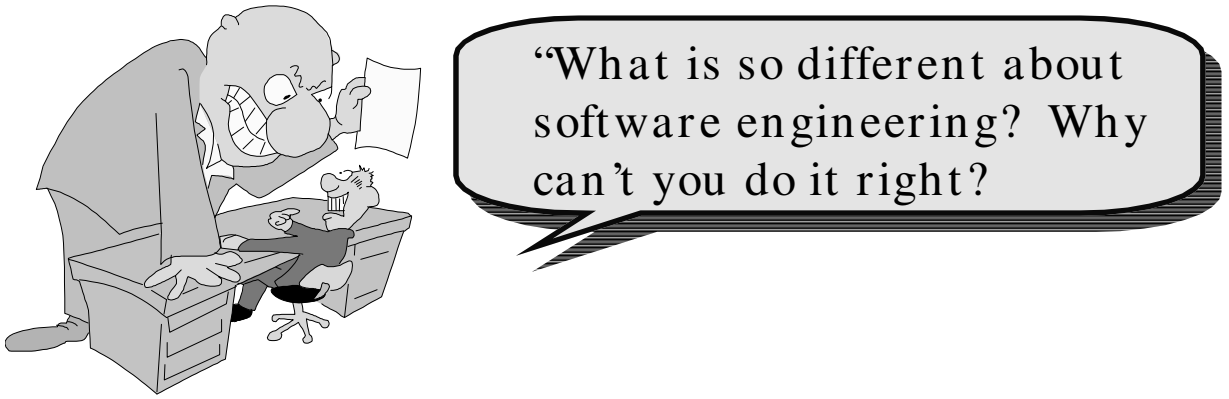
Common wisdom, 1983

"To err is human but to really foul things up requires a computer"

Farmer's Almanac for 1978
"Capsules of Wisdom"

Problems with Software and Its Assurance

- Software is usually late, expensive and wrong



- Traditional disciplines are founded on science and mathematics and are able to model and predict properties of their designs but SE is more a craft

But software IS more complex!

Complexity and Design Faults

- Software provides much of functionality of modern systems and thus expresses the scale and complexity of these systems.

  Complexity ⟷ Design faults

- Failure to anticipate certain interactions among the components of the system or between system and environment
- Software is mutable - so it is most likely to change. And have design faults.
- "Defensive programming" - cure?

Ways to Achieve “Correctness”

- Quality control
- Quality assurance
  - System always does the right thing
    - » Testing (“always” is hard to address)
    - » Fault injection
  - System never does the bad thing
    - » Fault-tree analysis [Leveson95]

Group “essentially similar” behaviors together so that fewer cases need to be considered.

But complex systems usually exhibit “essentially different” behaviors!

Overview of the talk

- Formal methods in software development
  - what are they?
  - where are they useful?
  - why are they useful?
  - industrial case-studies
  - problems with formal methods
- Formal methods *light*
- Formal methods research in Univ. of Toronto
  - big picture
  - successes
  - challenges
- Conclusion

Formal Methods

"The use of mathematical modeling, calculation and prediction in the specification, design, analysis, construction and assurance of computer systems and software."

Engineering - partial differential equations to model variations in physical quantities over time or space.

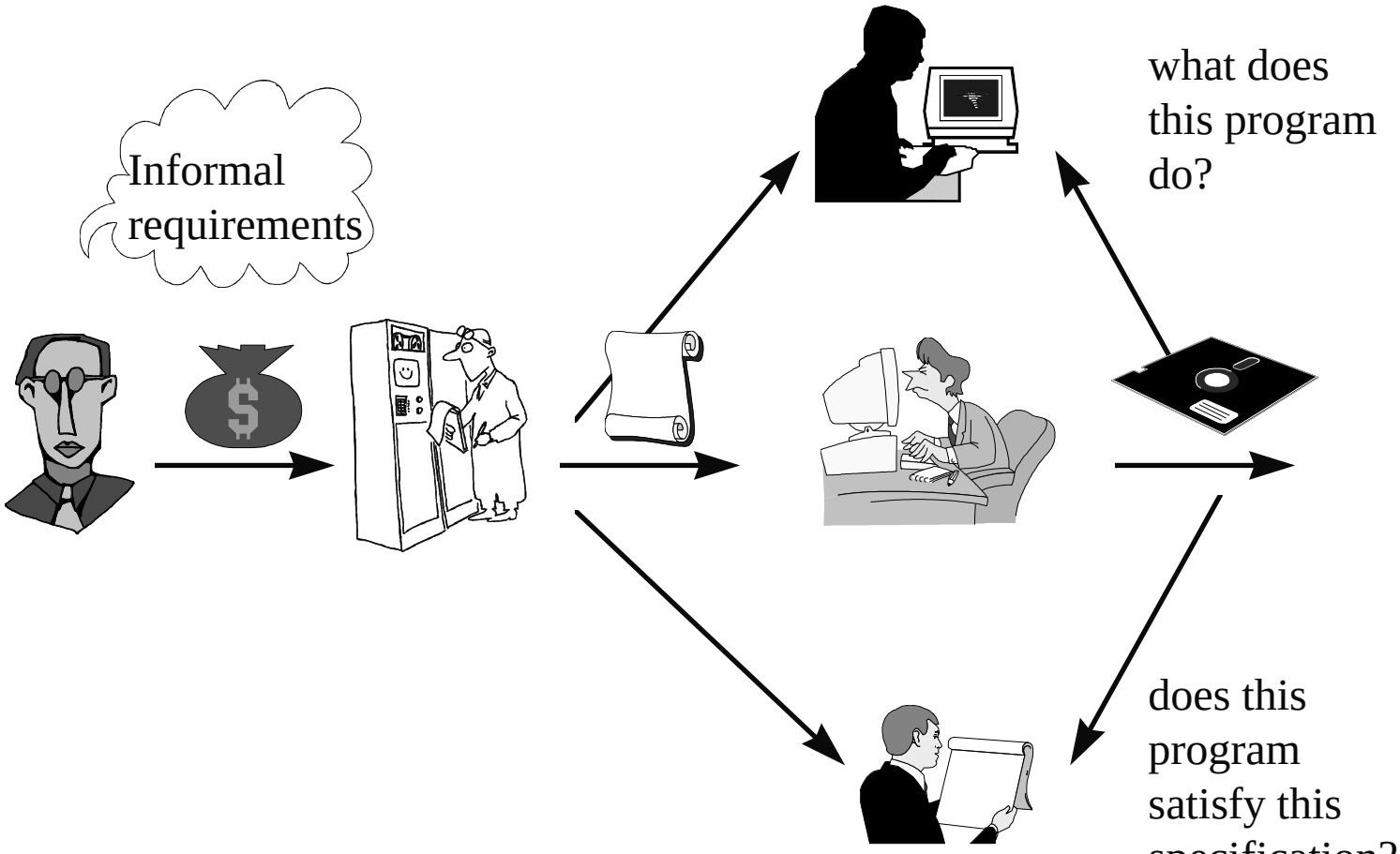
Software - model discrete quantities. Need concepts like sets, graphs, partial-orders, finite-state machines

"Calculation" is in terms of formal logic rather than numerical computation.

Formal Methods in Computer Systems

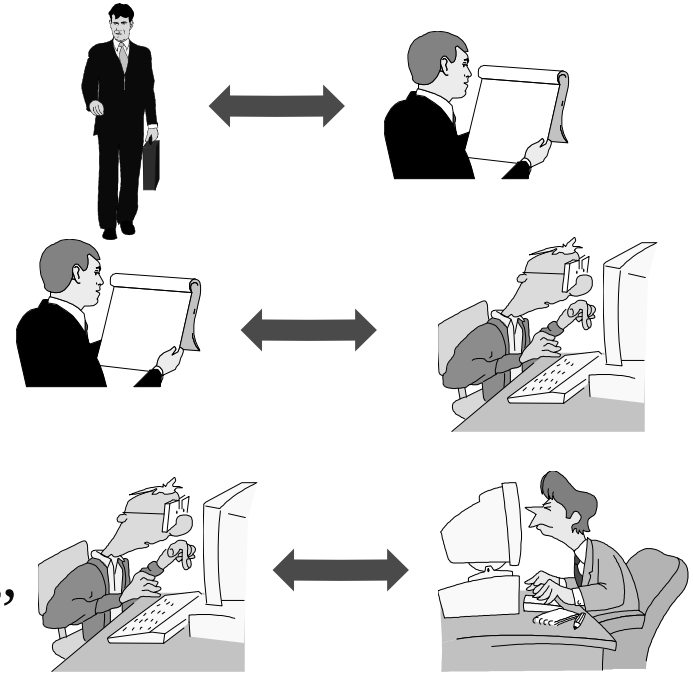
- Formal methods provide mathematically-based techniques that describe system properties
- As such, they present a framework for systematically specifying, developing, and verifying systems
- A formal method should possess a set of guidelines or a “style sheet” :
  - when can the method be applied
  - how can the method be applied most effectively
- Tangible output - formal requirements specification

Users of formal specs



System Documentation

- specification is a description alternative to system implementation
- serves as a communication medium between
  - a client and specifier
  - an implementor and specifier
  - between implementors
- capture “what” rather than “how”



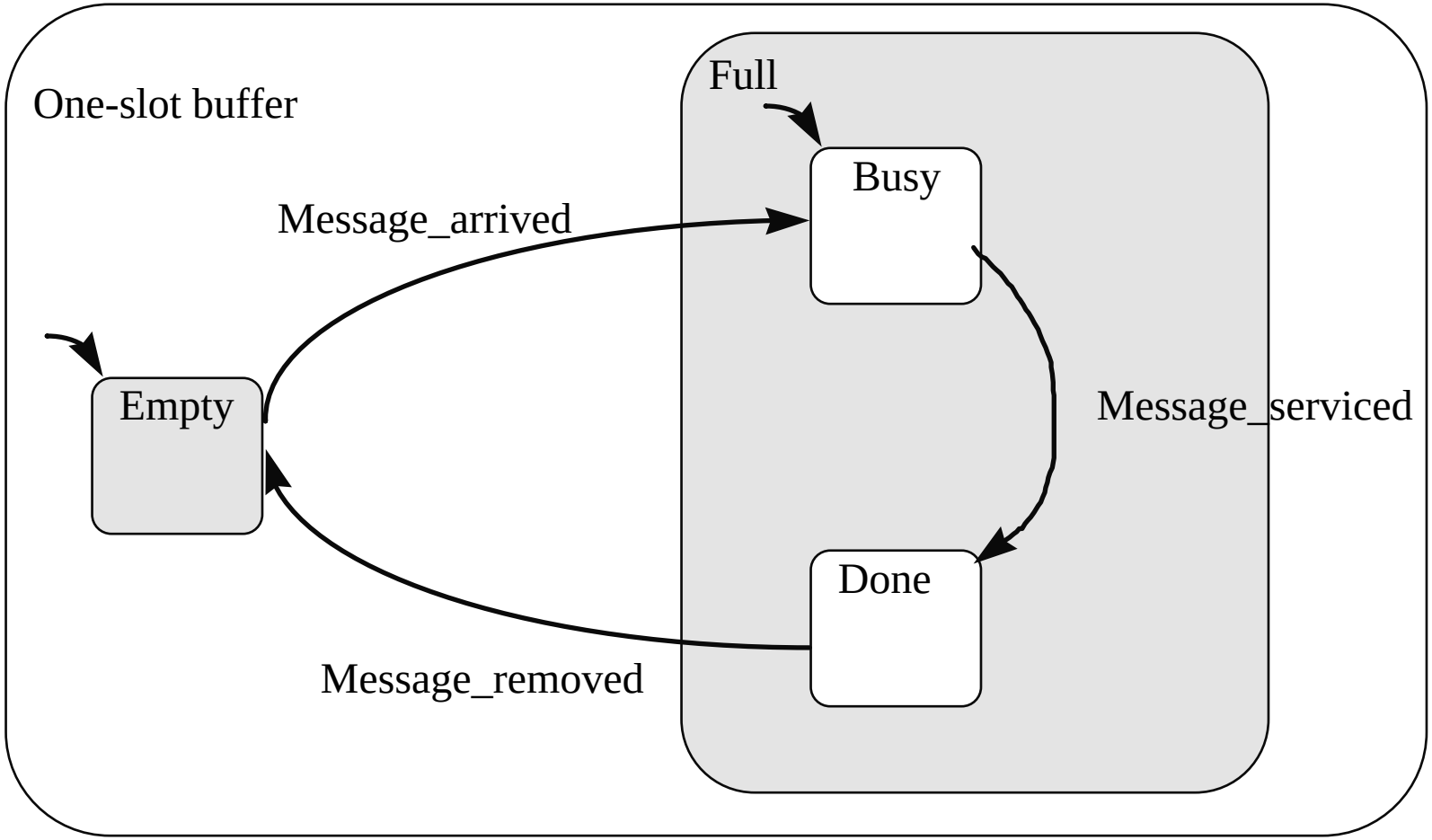
What is a Specification Language?

- SL provides…
  - a notation (its syntactic domain)
  - a universe of objects (its semantic domain)
  - a precise rule defining which objects satisfy each spec
- A specification is a sentence written in terms of the elements of the syntactic domain.
- Progr. lang. (PL) ⊃ SL
  - Specifications do not have to be executable or restricted to expressing only computable functions
  - All programs are formal objects and can be compiled and executed

Syntactic and Semantic Domains

- What do specifications consist of?
  - a set of symbols (constraints, variables, logic connectives)
  - a set of grammatical rules for combining them
  - can be graphics, but with formal semantics
- What can be specified? A lot of things!
  - ADT SLs - for algebras, theories and programs
  - Concurrent and distributed systems SLs - for state sequences, event sequences, streams, state machines…
  - PLs - for functions from input to output, computations, relations, etc.

Example Specification - StateCharts



Where to use formal methods?

- Can be applied in all phases of system development
- Greatest benefit - from process of formalizing!
- One end - just specify requirements
- Other end - use the abstract model to predict the behavior of the system via automatic deduction

Formal Methods in Requirements

- consistency analysis - is the specification sufficiently well-formed to be a description of something?

  shouldn't say *a* and *not a*

- completeness analysis - has everything you want to specify been specified?
- animation - "test" formal specification by "running" it
- formal challenges - pose and prove putative theorems - "if this specification says what it should, then the following ought to follow"

  sorting a sequence

  Challenge: *sort(sort(x)) = sort(x)*

Formal Methods in Verification

- predict behavior and verify refinement

  can the system deadlock or survive a particular fault or give response within certain time?

  – "try all possible behaviors by brute force" - model-checking

  – good for hardware and distributed algorithms like protocols

  Problem: system gets too large

  – need to abstract (e.g., just look for Null/non-Null pointers or aggregate several variables together)

  – but how do we report meaningful errors?

Formal Methods in Verification (Cont'd)

- verify desired properties
  - use theorem-provers to show that property is implied by the requirements (or design or algorithm)
  - need to have a formal requirement of this artifact
  - requires skill and experience but has been effectively used in practice
  - PVS is being used at Hydro right now!
- verify design refinement
  - is design on one level correctly implemented on a more detailed level?

Formal Methods in Testing and Debugging

- can generate test cases for black-box testing from specs
- specs can be used to explicitly state assumptions on use of module for boundary conditions
- specs + implementation can be used for path testing, unit testing, integration testing.

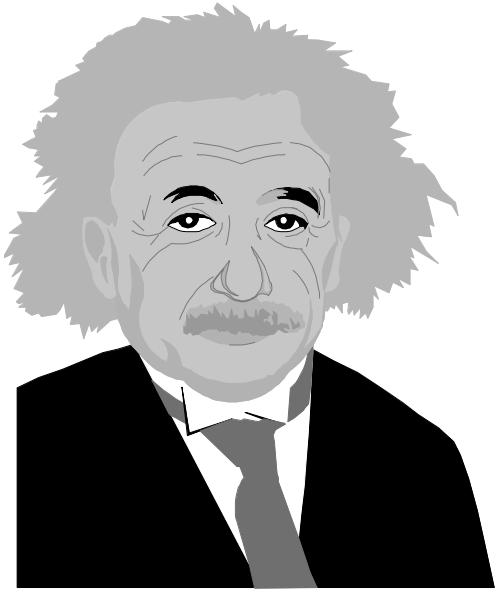
  (can show that there are no paths missing!)

System Analysis and Evaluation

Much recent work in the application of FMs has been in specifying a system already built, running and used

- Revealed bugs in published algorithms and circuit designs
- Revealed unstated assumptions, inconsistencies, and unintentional incompleteness in the system

Formal Methods. Why not?

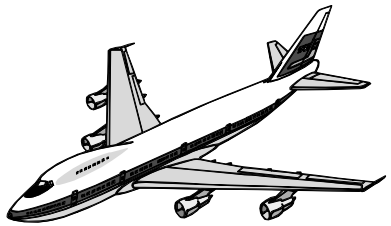


"Why does this magnificent applied science which saves work and makes life easier bring us so little happiness? The simple answer runs: because we have not yet learned to make sensible use of it."

- Albert Einstein

Industrial uses of Formal Methods

- Lockheed C130J (1995, Lockheed and Prexis)
  - Analyzed code for avionics software supplied to US Air Force, coded in SPARK-annotated subset of Ada
  - Specifications written in CORE (based on SCR)
  - Software was “correct by construction” and passed all test cases
  - No rework was necessary - added quality decreased the overall project cost



Industrial-size Specifications (Cont'd)

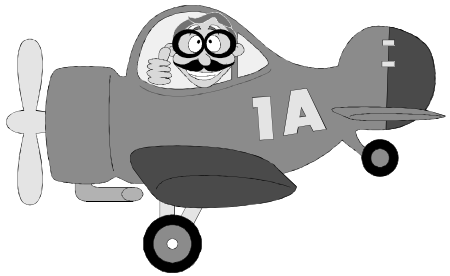
- CICS (Oxford University and IBM Hursley Laboratories)
  - 1980s, using Z
  - not a safety-critical system!
  - formalized part of IBM's Customer Information Control System
  - 100,000 lines of code, 1000 pages of documentation
  - improvement in quality and 9% decrease in total development cost

Queen's Award for Technological Achievement

Industrial-size Specifications (Cont'd)

- CDIS (1992, Praxis for UK Civil Aviation)
  - used VDM and CCS to specify Display Information System
  - formal methods were used as an integral part of development process
  - productivity - same or better than comparable projects
  - defect rate 0.75 faults/KLOC

    2 - 10 times improvement!



Industrial-size Specifications (Cont'd)

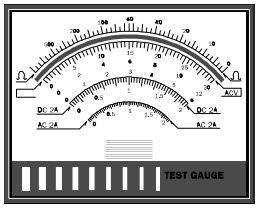
- TCAS (early 1990s, Irvine - U. Washington)
  - Used RSML to specify requirements for TCAS
  - The system is used on all commercial aircrafts flying in US airspace
  - RSML is based on StateCharts
  - TCAS II specification has been automatically checked for completeness and consistency
  - This case-study showed that one can build a formal model of a system that is readable and reviewable by application experts without special training

Formal Methods in Commercial and Safety-Critical Systems

Databases

- HP Medical Instruments real-time database for storing patient monitoring information (1991)

Devices



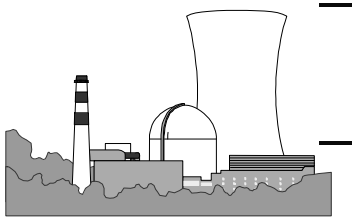
- Tektronix family of oscilloscopes (1990)
- Schlunberger line of household electricity meters (1996)

Medical

- Clinical Neutron Therapy System at U. of Wash. (1995)

Formal Methods in Commercial and Safety-Critical Systems (Cont'd)

Nuclear



- Reactor Safety System for the Experimental Breeder Reactor-II. Argonne National Laboratories (1987-89)
- Shutdown system of the Darlington Nuclear Generating System, Canada (1990)

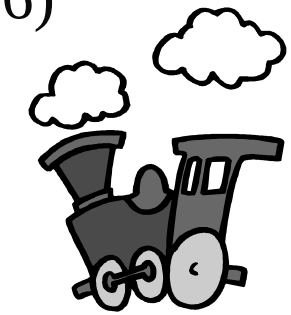
Security

- Security policy model for the NATO Air Command and Control System (1995)
- Token-based Access Control System for US NIST (1990)
- Secure transmission of datagrams in the Multinet Gateway System (1984)

Formal Methods in Commercial and Safety-Critical systems (Cont'd)

Telephony

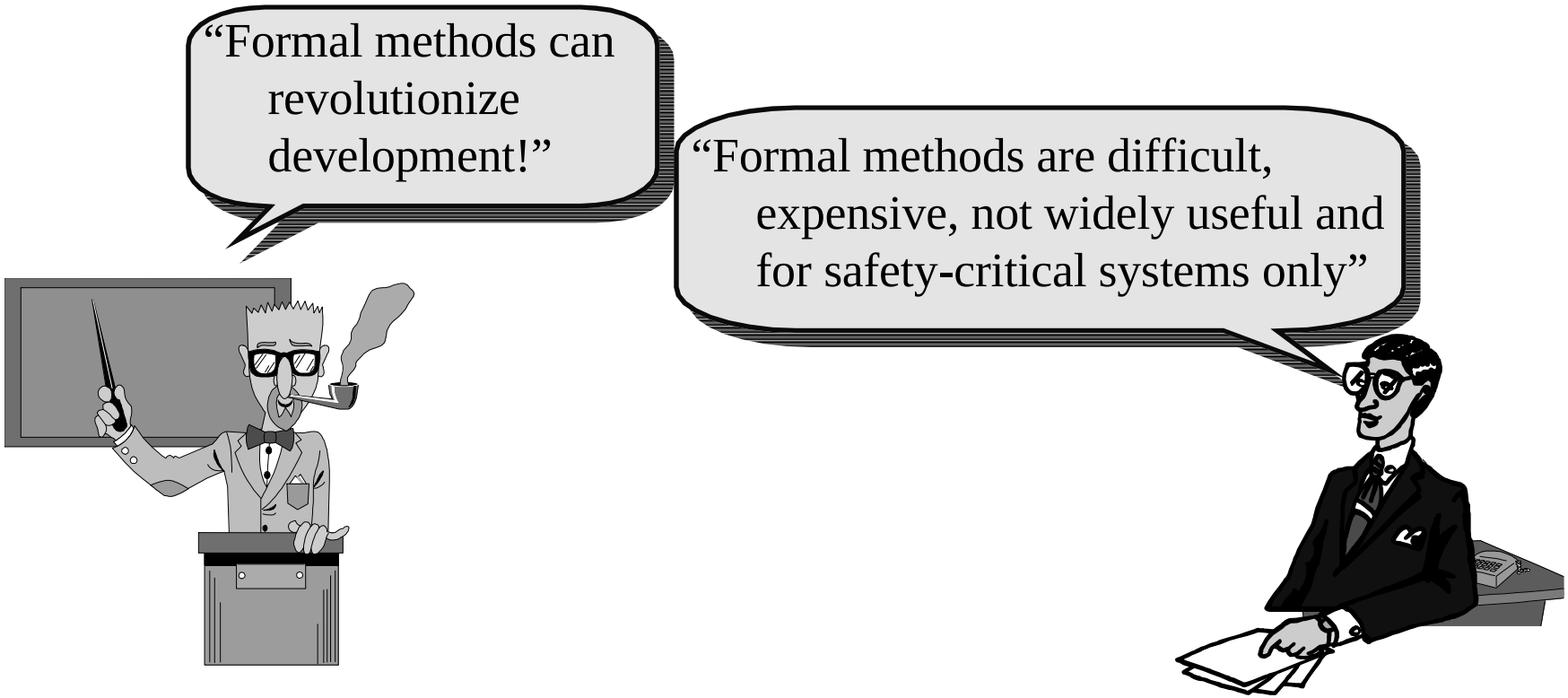
- Various features of AT&T 5ESS switching system using Estrel (1996) and combinations of Z and CSP (1995-96)
- Customizable telephone services and features of Deutsche Telekom (1996)



Transportation

- Automatic train protection system for the Paris Metro (1992)
- British Rail's signaling rules (1994)
- On-board avionics software for Levy aircraft (1992)

Why aren’t FMs used more?



“Formal Methods are Difficult”

- can achieve a lot without any formal proofs, just by using specifications
- have some nice tool support (although need more research)
- usually need an expert at hand, before local expertise is built-up
- various requirements languages (SCR, RSML, StateCharts) were designed to give FMs familiar flavor (tables, graphs, but not logic formulae)

“FMs Increase Cost of Development”

- allow catching bugs early in the lifecycle
- produce a formal document describing what system is supposed to do, for maintenance purposes
- usually result in cleaner designs
- can specify things on the right level of abstraction

“System development should be as formal as necessary but not more formal”

- almost Einstein

- cannot and should not formally specify the whole system
- conducting formal proofs does require mathematical skills and can increase cost of development

“FMs increase cost of development”

- System development cost is out-sized in system maintenance.
  - need to know what the system is supposed to do
  - what each part is supposed to do
  - what needs to be preserved as the software changes

Challenge:

need to maintain requirements, design and code together!

Formal Methods and Reuse

- in 1983, 15% of new code was unique and specific to individual application. The rest could have been reused!
- with formal methods:
  - specifications and good documentation of components allows for their easy composition
  - formal specifications themselves can be reused since they are (ideally) implementation-independent
  - case-study - start with COBOL to produce formal spec then reengineer into better code!

“FMs are not acceptable to users”

- help users understand what they are getting
- capture what the user want before it is built
- can animate specifications
- can demonstrate consequences of specifications

But

- need to paraphrase the spec in natural language
  - it is usually shorter and better than traditionally developed
  - natural language spec might talk not only about scenarios but about what happens on all executions

“Formal Methods are only useful for safety-critical systems”

- largest practical applications were not for critical projects
- FMs should be used wherever the cost of failure is high:
  - system critical in any way
  - replicated many times
  - fixed into hardware
  - dependent on quality for commercial reason

Almost any serious piece of software qualifies!

Don’t have to use proofs!

“Formal methods are not used on real, large-scale software”

and what about many case-studies?

- from specification to complete verification
- using tools and by hand

“Formal Methods mean forsaking traditional engineering design methods”

“A method is a way in which a process is to be conducted”

- an underlying model of development
- languages
- defined, ordered steps
- guidance for applying these steps in coherent manner

Many formal methods don’t address all of these issues

Structured development methods do:

- “waterfall model” [Royce, 1970]
- “spiral model” [Boehm]

Formal methods shouldn’t replace but rather compliment traditional engineering!

FMs and traditional engineering design methods

- Need to create a true method to support software life cycle but allow
  - formal techniques at specification and design
  - supporting refinement to executable code
  - proof of properties, etc.
- Successes:
  - translation of SSADM into Z
  - integration of Yourdon's modern structured analysis and Z
  - integration of various structured notations with VDM and CSP
  - integrating formal methods into Cleanroom

Active area of research!

What can FMs help you with?

- help in finding errors early on and can nearly eliminate certain classes of errors
- work largely by making you think very hard about the system you propose to build
- useful in almost any application
- based on mathematical specifications; easier to understand than programs
- correctly applied, they decrease cost of development
- can help clients understand what they are buying
- have been successfully used in practice
- can be used in conjunction with traditional engineering design methods

Still...

Issues preventing wide acceptance of formal methods

- Cost-effectiveness
- Unfamiliar, “mathematical” notation
- Problems of scalability - can a “really large” program be specified and verified?

Unbounded Buffer in Temporal Axiom Method [Lamport]

module BUFFER **with subroutines** PUT, GET

state functions:

buffer : **sequence of** message
parg : message **or** *NULL*
gval : message **or** *NULL*

initial conditions:

$|buffer| = 0$

safety properties

1. (a) $at(\text{PUT}) \Rightarrow parg = \text{PUT.PAR}$
   (b) $after(\text{PUT}) \Rightarrow parg = NULL$
2. (a) $at(\text{GET}) \Rightarrow gval = NULL$
   (b) $after(\text{GET}) \Rightarrow \text{GET.PAR} = gval$
3. **allowed changes to** buffer
   parg **when** *in*(PUT)
   gval **when** *in*(GET)

(a) $\alpha[\text{BUFFER}]{:}in(\text{PUT}) \wedge parg \neq NULL \rightarrow parg' = NULL \wedge buffer' = buffer * parg$

(b) $\alpha[\text{BUFFER}]{:}in(\text{GET}) \wedge gval = NULL \wedge |buffer| > 0 \rightarrow gval' \neq NULL \wedge buffer = gval' * buffer'$

liveness properties

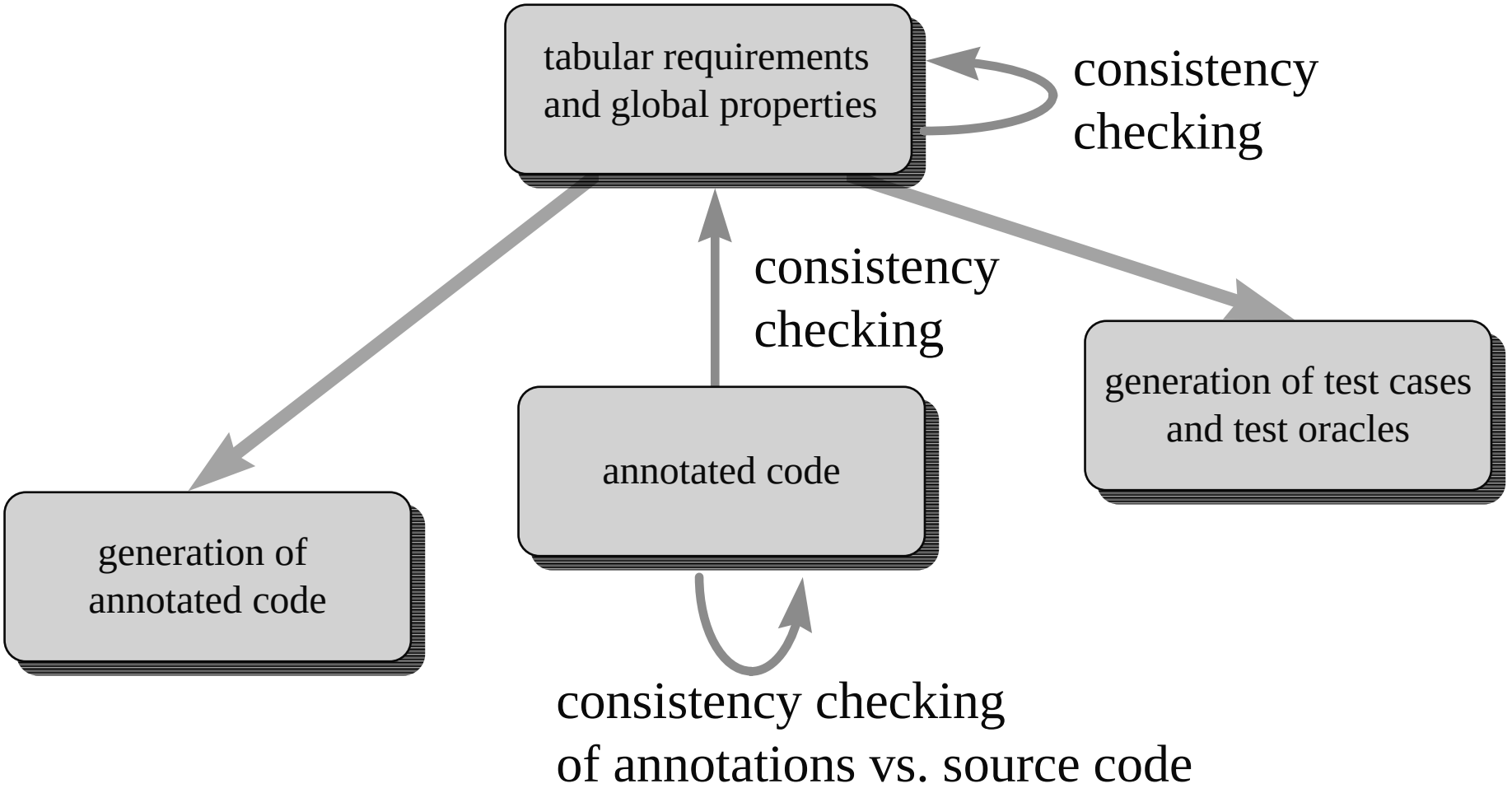
4. $in(\text{PUT}) \wedge |buffer| < \min \rightsquigarrow after(\text{PUT})$
5. $in(\text{GET}) \wedge |buffer| > 0 \rightsquigarrow after(\text{GET})$

Our Approach to Formal Methods

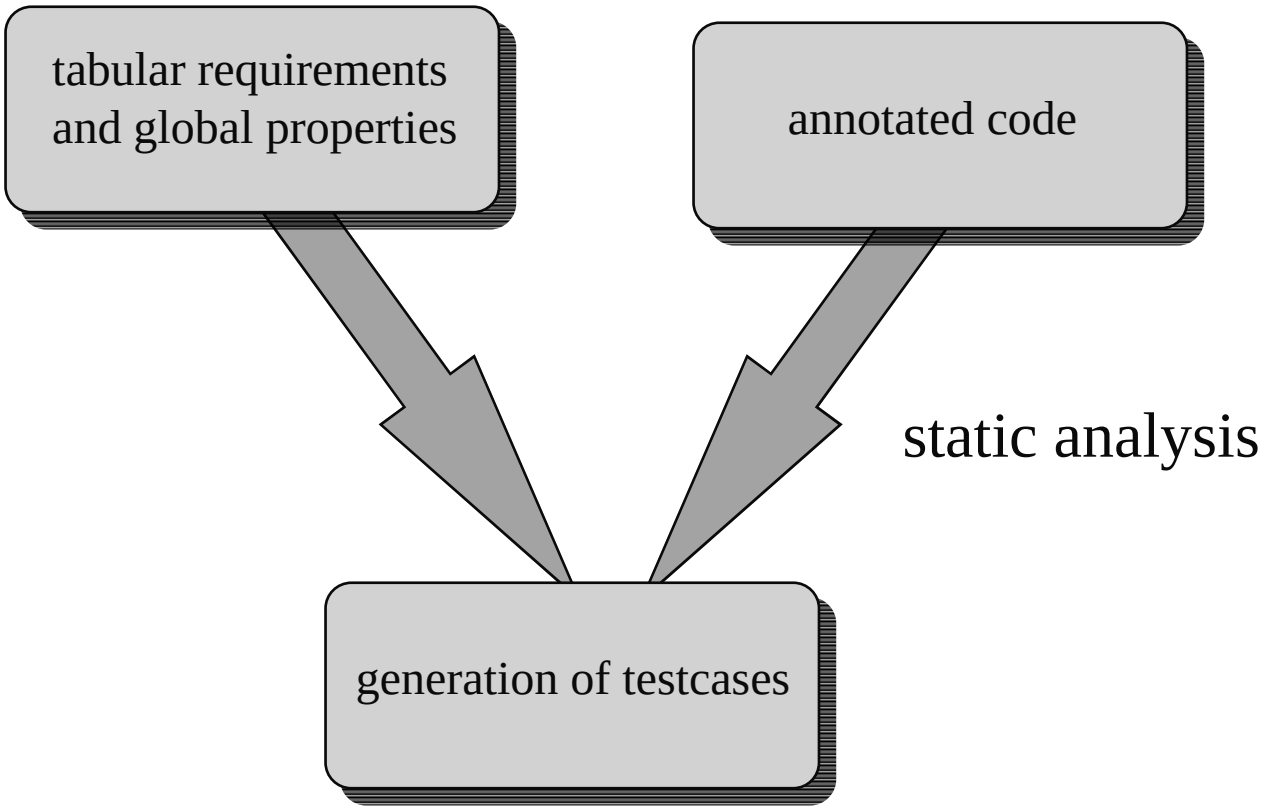
Formal methods “light”

- Use easy to read and review requirements notations (StateCharts, RSML, SCR)
- Don’t want to formally specify everything
- Develop partial specifications and analyze them mechanically
- Don’t check for all properties - just important ones
- Sacrifice exactness for scalability
- Apply formal techniques more for debugging than assurance
- Amortize use of formal specs in other parts of the development

Formal Methods Research at UofT



Big Picture of FM research at UofT



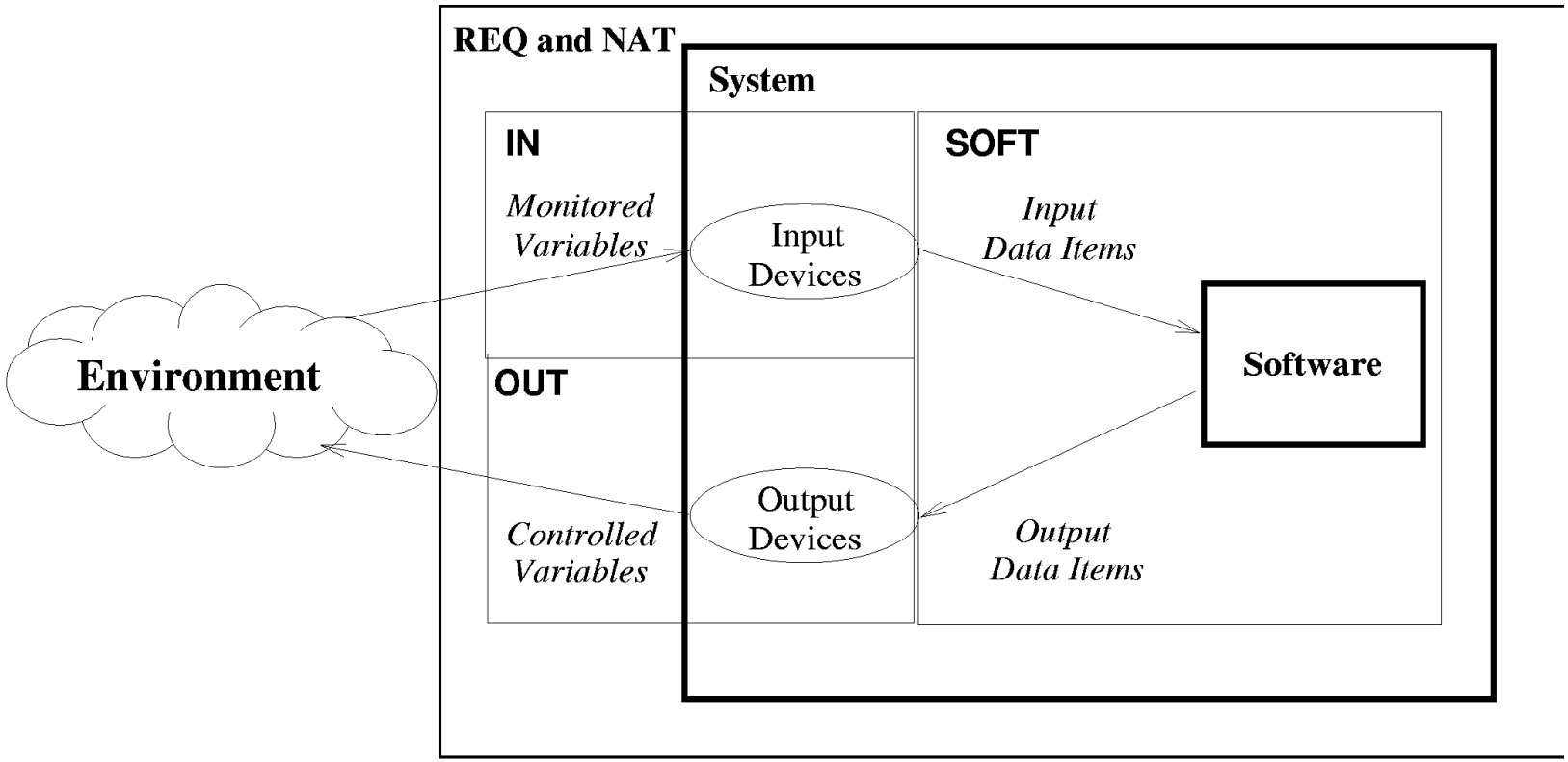
New Process

- Development of specification and its proofs of correctness
- Automatic generation of annotated code
- Hand-improvements of the code
- Checking that the code is annotated correctly
- Checking consistency between requirements and annotated code
- Generation of testcases
- Testing against specification (using specification as an oracle)

In more detail...

- SCR requirements standard
- annotation language
- research interests

SCR and 4-Variable Model



SCR Requirements

- A state is a mapping of variables to values
- A modeclass is a set of system states, called modes, which partition the monitored environment's state space.

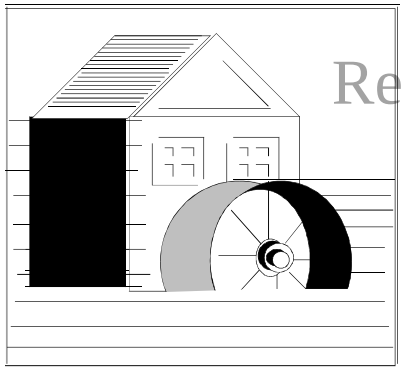
  One mode is designated as the initial mode
- A condition is a predicate on monitored variables

  Running, ButtonPressed
- An event is a point in time when a condition's value changes:

  @T(ButtonPressed)
  @F(Running) WHEN [ButtonPressed]

Water-Level Monitoring System

WLMS monitors and displays the water level in a container, raising visual and audio alarms and shutting off its pump when the level is out of range or when monitoring system fails. Two push buttons, SelfTest and Reset, permit operator to test the system and return it to normal operation.



Requirements:

- 2 mode classes (Normal and Failure) with 7 total modes
- 18 monitored and 7 controlled variables

SCR Table for WLMS

Current Mode	*Inside HysRange*	*Within Limits*	*SlfTst Pressed*	*SltTst Pressed 200*	*Test Interval*	*Reset Interval*	*Shutdown LockTime*	**New Mode**
Standby	true	--	--	--	--	@T	--	Operating
	--	--	--	@T	--	--	--	Test
Operating	--	@F	false	--	--	--	--	Shutdown
	--	--	true	@T	--	--	--	Test
Shutdown	@T	--	false	--	--	--	false	Operating
	--	--	false	--	--	--	@T	Standby
	--	--	--	@T	--	--	--	Test
Test	--	--	--	--	@T	--	--	Standby

Initial: Standby (~SlfTestInterval & ~TestInterval & ~ResetInterval & ~ShutdownLockTime)

Global Properties of WLMS

- "If the system is in mode Operating, the self-test button has not been pressed for 500 ms and either the water level is within limits or the self-test button is being pressed"

inv(Operating -> (~SlfTestPressed500 & (WithinLimits | SlfTestPressed)))

- "If the system is in Shutdown, the water is not inside hysterysis range and the system has been in mode Shutdown for less than 200 ms"

inv(Shutdown -> (~InsideHysRange & ~ShutdownLockTime200))

- "If the system is in Standby, the self-test button has been pressed for 500 ms"

inv(Standby, ~SlfTestPressed500)

- "When the self-test button is pressed for 500 ms, the system should go into self-test mode"

cause(SlfTestInterval, Test)

Requirements analysis case-study

(joint work with J. Gannon, Univ. of Maryland and J. Atlee, Univ. of Waterloo)

- Requirements error:
  - "If Self-Test button is being pressed for less than 50 ms and water goes out of limits, the system is going to miss this event"

Annotation Language

- Goal: document code with events on the requirements level. These events can be used to check consistency with requirements
- Language constructs - KISS principle:
  - Initial: assigns values to variables in starting state
  - Update: assigns values to variables indicating a state change
  - Assert: checks and restricts values of variables indicating a condition

Example - WLMS code

```
while (true) {
   …
   GET_MODE (OperatingClass, OPCM);
   WATER_LEVEL(&LVLCM);
   INLIM=((LVLCM < (float)DPAR_HWL) && (LVLCM < (float)DPAR_LWL));
   GET_TIME(SHUTER, &SHUTTM);
   if (ERR_VAR_F(CLK_ERR) == CLK_OVEFLOW) {
     RESET_TIMER(SHUTER);
     SHUTTM = 0;
   }
   else (MODE_EQ (OPCM, Shutdown) && INLIM && (SHUTTM < DPAR_SLT) &&
     !PRESSED(SlfTst)) {
     SET_MODE (OperatingClass, Operating);
   }
   …
   GET_MODE (OperatingClass, OPCM);
   if ((!MODE_EQ(OPCM, Test))) && (PRESSED(SlfTst))) {
     SET_MODE(OperatingClass, Test);
     …
```

Annotated WLMS Code

```
while (true) {
  …
  if (MODE_EQ (OPCM, Shutdown) && INLIM && (SHUTTM < DPAR_SLT) &&
     !PRESSED(SlfTst)) {
     @@ Assert Normal=Shutdown & WithinLimits = true &
     ShutdownLockTime200=false & SlfTstPressed=false;
      SET_MODE (OperatingClass, Operating);
     @@ Update Normal=Operating;
  }
  …
  GET_MODE (OperatingClass, OPCM);
  if ((!MODE_EQ(OPCM, Test))) && (PRESSED(SlfTst))) {
     @@ Assert ~ Normal=Test &  SlfTstPressed = true;
     SET_MODE(OperatingClass, Test);
     @@ Update Normal=Test;
  }
  …
```

Case-Study (Cont'd)

- Original implementation:
  - 1000+ lines (54 functions) in C with Xlib
  - 49 Update and 60 Assert annotations
- Analyzed annotated implementation
  (time ≈ complication)

Case-Study (Cont'd)

- Errors found:
  - Wrong monitored variable was checked to enable mode transitions
  - Watchdog timers were not set on control buttons
  - No transitions to mode corresponding to complete system failure were implemented!!!

Moral of the story

"Errors are not in the art but in the artificers"

- Sir Isaac Newton

"There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And another way is to make it so complicated that there are no obvious deficiencies"

- C.A.R. Hoare

Graduate Course at Univ. of Toronto

- Automatic tools for verification and specification

  Fall, 1997

  `http://www.cs.toronto.edu/~chechik/courses/2124`

  – Look at formal methods with adequate tool support

    » Model-checking (SMV, Promela/SPIN)

    » Theorem-proving (Larch, PVS)

    » Process algebras (Concurrency Workbench)

    » Z and VDM

    » and more

  – Topics vary each semester

Conclusion

- Formal methods, when properly applied, can significantly improve quality of the product and decrease cost
- In non safety-critical systems, FMs are best used in specifications
- At UofT, we support formal documentation by
  - using familiar notations for specification
  - amortizing cost of formal spec in various analysis activities
  - using/developing automatic tools, requiring minimum user-guidance

Research Interests

- Development of hybrid (formal + informal) methods for specification of software
- Improvement of annotation language and tools
- Industrial-size case-study (so far, only 2-3 K lines)
- Specification-based testing
- Use of existing techniques like model-checking and theorem-proving to aid in analysis of timing properties