Introduction to Formal Methods
Chapter 01: Formal Methods

Roberto Sebastiani

DISI, Università di Trento, Italy – roberto.sebastiani@unitn.it
URL: http://disi.unitn.it/rseba/DIDATTICA/fm2020/
Teaching assistant: Enrico Magnago – enrico.magnago@unitn.it

CDLM in Informatica, academic year 2019-2020

last update: Monday 18th May, 2020, 14:48

Copyright notice: *some material (text, figures) displayed in these slides is courtesy of R. Alur, M. Benerecetti, A. Cimatti, M. Di Natale, P. Pandya, M. Pistore, M. Roveri, and S.Tonetta, who detain its copyright. Some exampes displayed in these slides are taken from [Clarke, Grunberg & Peled, "Model Checking", MIT Press], and their copyright is detained by the authors. All the other material is copyrighted by Roberto Sebastiani. Every commercial use of this material is strictly forbidden by the copyright laws without the authorization of the authors. No copy of these slides can be displayed in public without containing this copyright notice.*

Outline

Outline

Problems in Developing Industrial Systems

- Functionality Issues: Growing Size & Complexity
- Requirements issues: availability, reliability, safety, security
- Application Domain Issues: Safety-Critical, Mission-Critical or Business-Critical Systems
- Market Issues: Time-to-delivery, Costs
- Maintenance Issues: Requirements change over time

Growing size and complexity

- increasing dependability
  - everything important depends on computers
    (industrial production, banking, stock market, transport,...)
    $\Longrightarrow$ quality is essential
- systems increasingly complex
  - Moore law: exponential growth
    ($\approx 10^{30}$ transistors/processor , multi million LOC's/OS)
    $\Longrightarrow$ cost for testing is exploding

Desired properties of systems

- Availability: a system must be working and able to provide its services
- Reliability: a system must correctly provide its functionalities, as expected by users
- Safety: the system should do nothing very undesirable (causing damages to people,...)
- Security: the system should resist to intruders

Critical Systems

- Safety-critical: systems whose failure can cause life losses or serious environmental damage
  (e.g., trains & planes control, nuclear plants control, ...)
- Mission-critical: systems whose failure can cause the failure of the goals of important missions
  (e.g., space craft navigation)
- Business-critical: systems whose failure can cause the loss of big or huge amounts of money
  (e.g., bank management software, operating systems)

Time-to-delivery, Costs

- time-to-market affects potential revenue dramatically:
  - 1 week delay for a microprocessor
    $\Longrightarrow$ loss of more than 20.000.000 US$ (year 2004)

The quest for correctness

"It is fair to state, that in this digital era correct systems for information processing are more valuable than gold."

[H. Barendregt. The quest for correctness. 1996.]

- Reliability increasingly depends on hard- and software integrity
- Defects can be fatal and/or extremely costly

Outline

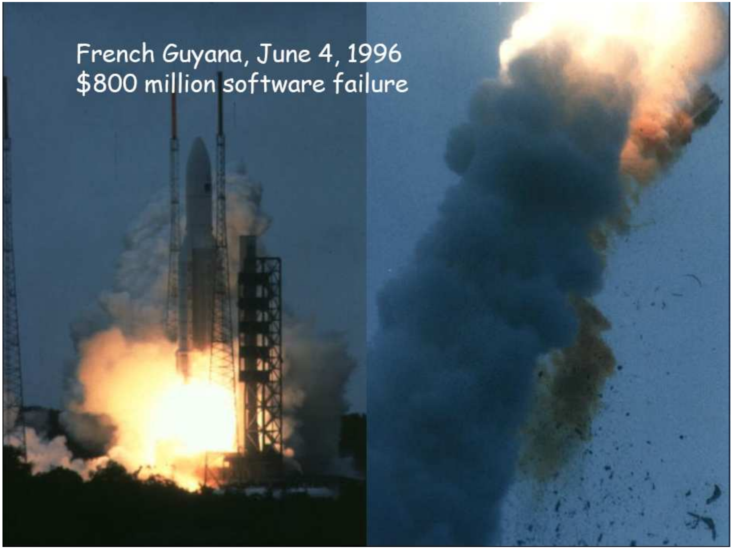
The Therac-25 Case

- Canada-USA 1985-1987: 4 people killed, 2 seriously injured for the wrong behaviour of an anti-tumor irradiating machine (Therac-25)
- Cause: wrong behaviour of its control software (wrong interaction among components)

The Ariane 5 Case

- 4 June 1996: the first flight of the Ariane 5 failed. After 40 seconds the rocket changed trajectory and exploded
- The SW of the Inertial Reference Systems ceased to work after 36 seconds.
- 800 Million US$ lost
- Cause: a variable overflow!

The Ariane 5 Case (cont.)

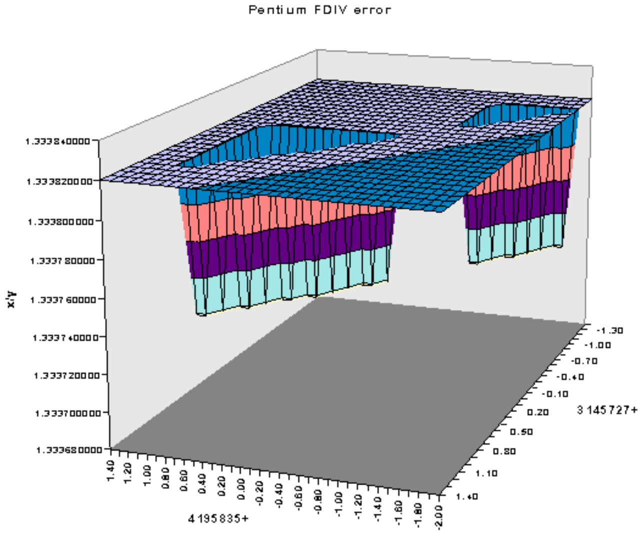


Courtesy of Pao-Ann Hsiung, National Chung Cheng University

The PENTIUM Bug

- Professor Thomas Nicely from Lynchburg College in Virginia discovered incorrect behaviors in the Pentium chip.
- Cause: a design error in the floating point division algorithm in the ALU.
- The chip was withdrawn and substituted by Intel.
- 450 US$ millions lost!
- Since 1994, Intel adopts formal methods!

The PENTIUM Bug (cont.)



Courtesy of Pao-Ann Hsiung, National Chung Cheng University

The Denver Airport Case

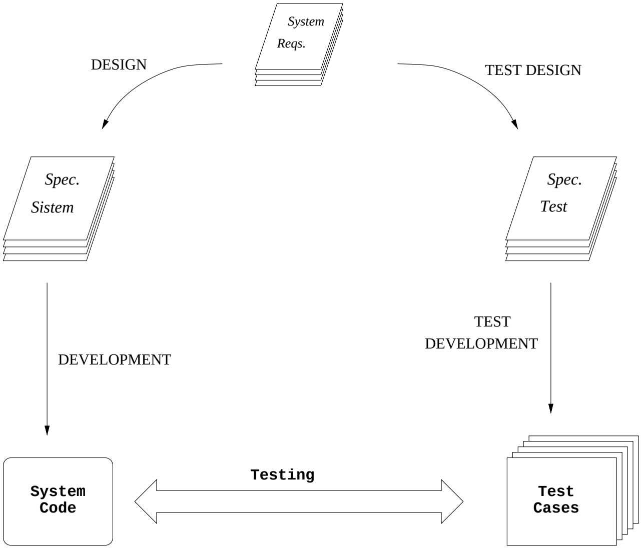
- Denver Airport: designed to be a state-of-the-art airport
- State-of-the-art baggage-delivering computerized system, 5.300 miles optic-fiber cables
- the system turned out to be completely unreliable, huge amounts of luggages were lost, erroneously delivered or even damaged
- the airport was inaugurated with 16-month delay with a manual baggage-delivering system
- 3.2 US$ billions lost!

The AT&T Case

- a 9-hour blackout in all AT&T long-distance calls caused by software errors
- the worst blackout in the story of American telecoms
- Cause: one single wrong line of code!

Outline

Standard Development Process (over-simplified!)



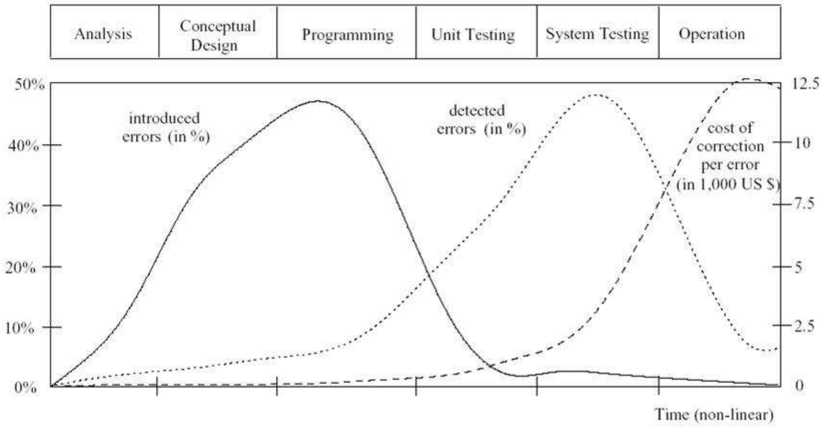
Difficulties with Traditional Methodologies

- Ambiguous Specifications (Requirements, Analysis, Design)
- Errors in specifications/design refinements
- Limited coverage by tests

$$\Downarrow$$

- Expensive errors in the early design phase
- Low software quality (hard to maintain)
- Infeasibility of achieving (ultra-high) reliability requirements

Error introduction & detection, and relative costs



Current verification techniques

- Reviewing, Testing & Simulation (currently mostly used)
- Formal verification methods (increasingly used)

Reviewing, Testing and Simulation

- Peer reviewing (SW): manual code inspection
- Testing: The implemented system is executed on sets of inputs and external events
- Simulation: the behaviour of an abstract model is simulated (included input data, external events)

Peer reviewing: Limitations and Disadvantages

- time-consuming, expensive, boring,
- subtle errors (e.g., concurrency, algorithmic, etc.) hard to catch

Testing: Limitations and Disadvantages

- Not all input configurations can be given to the system (limited coverage)
- Each run cannot last forever, or be run infinitely often
- No guarantee that bad behaviors are covered
- The verification occurs too late in the process
- Very difficult, in particular for concurrent systems

Testing: Limitations and Disadvantages (cont.)

Current figures: in industrial SW and HW development, $\geq$ 50% of the effort is devoted to testing, and is increasing
$\Longrightarrow$ *testing/verification has become the bottleneck of the development processes*

Simulation: Limitations and Disadvantages

- Much slower than the system simulated
- Each run cannot last forever, or be run infinitely often
- Very expensive
- Not all behaviors are simulated (limited coverage)
- No guarantee that bad behaviors are covered

Testing & Simulation: Key Disadvantage

"Both testing and Simulation can detect bugs, but they cannot guarantee the absence of bugs" (Dijkstra '70).
$\Longrightarrow$ Need for something different: Formal Methods

Outline

Brief History of Formal Verification

Use of logical assertions

- First ideas [Floyd 1967], [Dijkstra].
- Axiomatic Verification of sequential programs [Hoare 1969]. Extended to Concurrent Programs in 70s and 80s. Compositionality.
  - Very powerful. Manual proofs (some times machine checked).
  - Acceptance has been low.
- Formal Methods are making impact.
  - Reactive and concurrent systems
  - Model checking (algorithmic verification)

Formal Methods: basics

"Applied Mathematics for modeling and analyzing ICT systems"

- Mathematical Models for system behaviors.
- Logical notations for specifying properties of programs.
- Methods for checking that program meets its desired specification.
- Three problems
  - Formal specification
  - Formal verification
  - Formal synthesis

Formal Specification

- Specify system requirements with formal, non ambiguous language.
- Language and tools available (e.g., Z, VHDL, VERILOG, Esterel, SDL, StateCharts, SMV, Promela,...);
- abstracts away unnecessary implementation details
- Benefits:
  - first step for formal verification and synthesis
  - Consistency of formal specification may be checked automatically (e.g., theorem proving): $S \not\models \perp$
  - *the effort of writing requirements in a formal language, alone, may reveal early specification bugs!!!*
  - may produce executable specifications (early debuggable)

Formal Synthesis (hints)

Problem: *given a specification S, synthesize a model M (system/program/circuit) which verifies it:* $M \models S$

- Most important in HW (but increasingly used in SW)
- pure top-down design, incremental refinement steps
- Integrates verification within the development process
- Works, but expensive!
- Approaches: theorem proving, (extended) planning

Formal Verification

Problem: *given a specification S, and a model M (system/program/circuit), check that M verifies S:* $M \models S$

- Most important in HW (but also extensively used in SW)
- Exhaustive verification
- Still expensive, but getting better!
- Approaches: theorem proving, equivalence checking, model checking

Formal Verification in HW

- Fits well in design flow
  - Designs in VHDL, VERILOG
  - Simulation, synthesis, and verification
  - Used as a debugging tool
- Who is using it?
  - Design teams: Intel, AMD, IBM, Lucent, ...
  - CAD tool vendors: Cadence, Synopsis,...
  - Commercial model checkers: FormalCheck,...

A quote

“*... formal verification has now entered the critical path in the process of development of a microprocessor*”
[Bob Bentley, Intel, CAV’2005]

Formal Verification in SW

- Software development process:
  - High-level modeling not common
  - Applications: protocols, telecommunications
  - Languages: ESTEREL, SDL, (UML)
- Recent trend: integrate model checking in programming analysis tools
  - Applied directly to source code
  - Main challenge: extracting model from code
  - Sample projects: SLAM (Microsoft), BLAST (Berkeley), Feaver (Bell Labs)

Benefits

- Find design bugs in early design stages.
- Achieve higher quality standards.
- Shorten time to market reducing manual validation phases.
- Produce maintainable products.

Limitations

- Appropriate for control-intensive applications (not data-intensive ones)
- Decidability and complexity remains an obstacle
- Model, and not system, is verified
- Finding suitable abstractions requires expertise

Outline

Formal Verification Methods: Key Ingredients

- Formal Specifications: *unambiguous* description of the system and of the required properties
  (message sequence charts, temporal logic, automata).
- Formal Validation & Verification: *exhaustive* comparison of the formal description of the system against the formal properties.
- Two main technologies: Theorem Proving & Model Checking

Theorem Proving

- Formal V&V by exhaustive search over the state space.
- System modeled as a set of logical formulae Γ
- Properties expressed as Theorems Ψ
  $\Longrightarrow$ Precise, unambiguous semantics
- Verification via logical reasoning:

$$\models (\Gamma \rightarrow \Psi)$$

  Can Ψ be derived from Γ?
- tools available (e.g., PVS, HOL, Lambda)

Theorem Proving: Limitations and Disadvantages

- Very hard to mechanize (theorem provers are typically interactive)
- The formalization Γ of the system can be very difficult to obtain
- It needs a big expertise to use the theorem prover.
- Most verification problems out of the reach of current theorem provers.

Equivalence Checking (HW)

Two circuits are functionally equivalent if they exhibit the same behavior

- Combinational Circuits: For all possible input values
- Sequential Circuits: For all possible input sequences

Equivalence Checking (cont.)

- Checks if two circuits are equivalent: $\models C_1 \leftrightarrow C_2$
  - Register-Transfer Level (RTL)
  - Gate Level
- Reports differences between the two
- Used incrementally after significant modification/improvements/refinements
- push-button technology
- computationally expensive

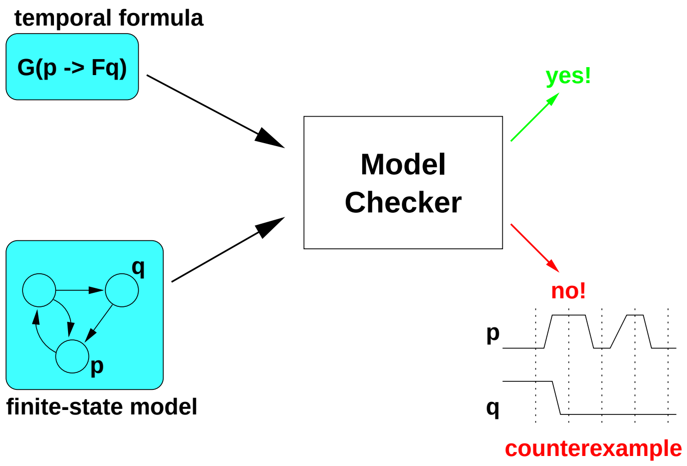
Model Checking

- Formal V&V by exhaustive search over the state space.
- Systems modeled as Finite State Machine M
- Properties expressed with a formal representation Ψ (e.g Temporal Logic, Automata, MSCs, etc.).
  $\Longrightarrow$ Precise, unambiguous semantics
- Verification via logical reasoning:

$$M \models \Psi$$

  Is M a logical model for Ψ?
  - Yes $\Longrightarrow$ the system verifies the property
  - No $\Longrightarrow$ a counter-example is returned (representing an execution leading to a bug).

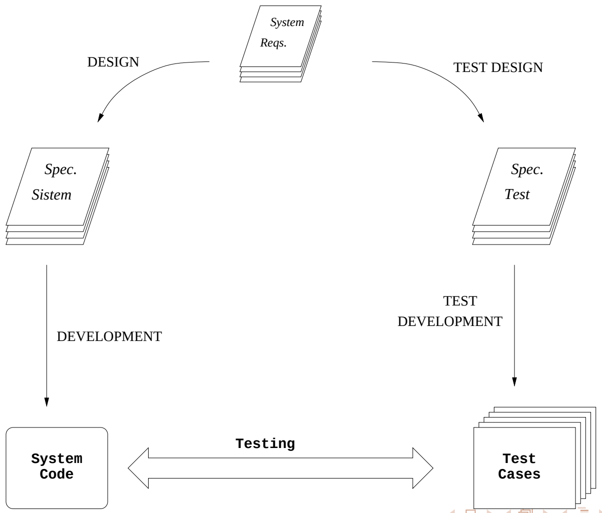
Model Checking (cont.)



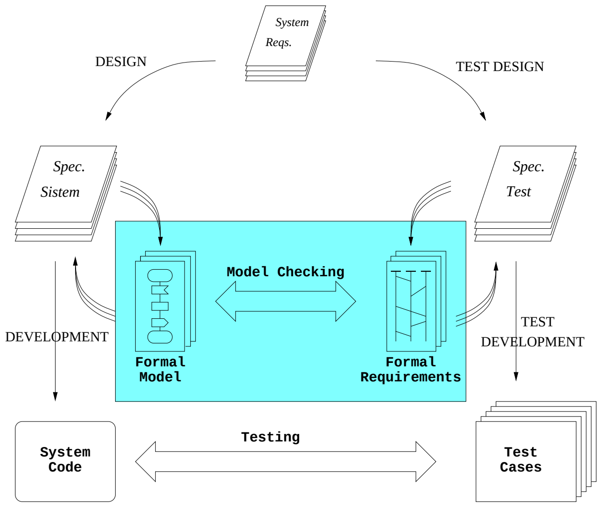
Industrial Success of Model Checking

- Powerful debugging capabilities:
  - helps detecting problems in early stages of the development cycle
  - exhaustive, thus effective
  - provides counterexamples
    (directs the designer to the problem).
- can be integrated within industrial development cycle:
  - compilers for practical design languages
    (e.g., VHDL, VERILOG, Esterel, SDL, StateCharts, SMV, Promela,...);
- Does not require deep training ("push-button" technology).

Extending the traditional development process with M.C.



Extending the traditional development process with M.C.



Model Checking: Limitations and Disadvantages

- Works (mostly) with finite state machines
- Engineers are not much at ease with temporal logic formulas (but encodings can be provided, though)
- The explosion of the state space is a big problem: the size of the F.S.M. grows up to exponentially w.r.t. the number of interacting components.
- The model checking process in practice: $M \models \Psi$
  Is M a logical model for Ψ?
  - Yes $\Longrightarrow$ the system verifies the property
  - No $\Longrightarrow$ a counter-example is returned
  - Timeout/memory overflow $\Longrightarrow$ try a simpler model

$\Longrightarrow$ *It is important to find the right level of details for the model*

Model Checking: 3 main problems [Vardi, '99]

- Scaling
-
-

Model Checking: 3 main problems [Vardi, '99]

- Scaling
- Scaling
-

Model Checking: 3 main problems [Vardi, '99]

- Scaling
- Scaling
- Scaling

Model Checking: State-of-the-art

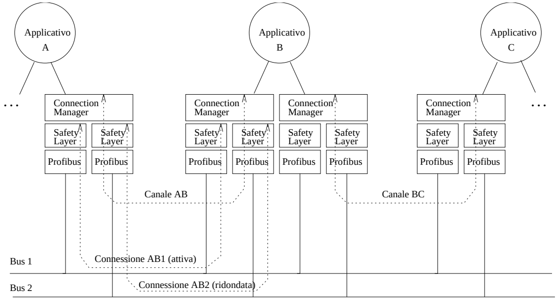
- Well-founded theory and algorithms
- Robust and well-established tools (e.g. VIS, SPIN, COSPAN, NuXMV, Uppaal)
- Very successful for verifying
    - medium-size “isolated” hardware
    - protocols
- increasingly popular in industry

Model Checking: Awards

- Amir Pnueli: *ACM Turing Award 1996*
  "For his seminal work introducing temporal logic into computing science and for outstanding contributions to program and system verification."
- Randal E. Bryant, Edmund M. Clarke, Jr., E. Allen Emerson, and Kenneth L. McMillan *ACM Kanellakis Award 1999*
  "...for their invention of "symbolic model checking," a method of formally checking system designs widely used in the computer hardware industry...".
- Gerard J. Holzmann, Robert P. Kurshan, Moshe Y. Vardi, and Pierre Wolper: *ACM Kanellakis Award 2006*
  "... demonstrated that checking the correctness of reactive systems can be achieved using a mathematical analysis of abstract machines."
- Edmund Clarke, E. Allen Emerson and Joseph Sifakis: *ACM Turing Award 2008*
  "... In recognition of their pioneering work on an automated method for finding design errors in computer hardware and software [Model Checking]"

Outline

Application Ex: Design of a Communication Protocol



- The *Safety Layer*: a high complexity train-to-station communication protocol
- Developed by Ansaldo S.F. and ITC-IRST (1999)
- Safety-critical

Previous experience

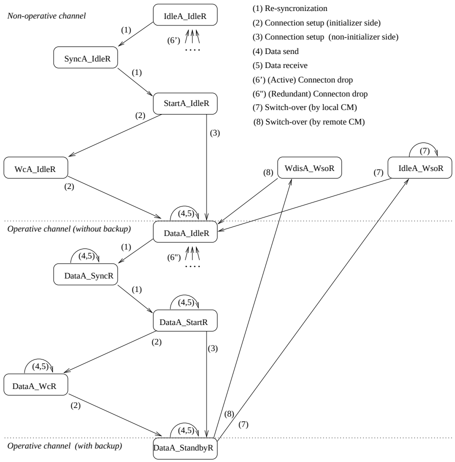
- Incomplete, informal specifications
- Existing implementation, very unsatisfactory
- A history of expensive debugging on-the-field

Application Ex: Design of a Communication Protocol

Approach:

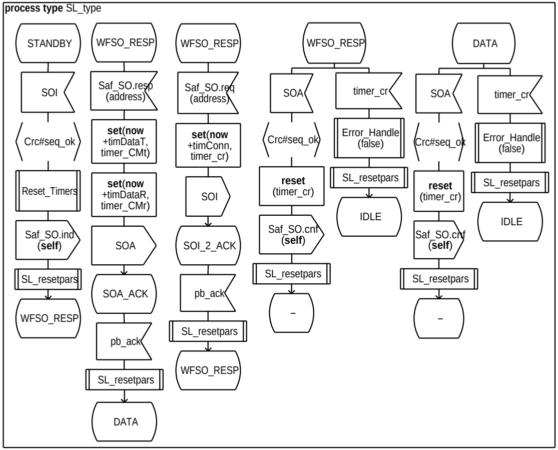
- Formal Specification of Functional Requirements with MSC
- Architectural and Formal Model in SDL
  $\Longrightarrow$ Executable Specification!!
- Formal Validation using Model Checking
  - Subtle bugs detected after exchange of over 200 messages;
  - counter-examples represented as Message Sequence Charts (MSCs)
    $\Longrightarrow$ easy to understand to engineers
- Detailed Informal/Formal specification to code developers.

Specification Abstract FSM diagram: example

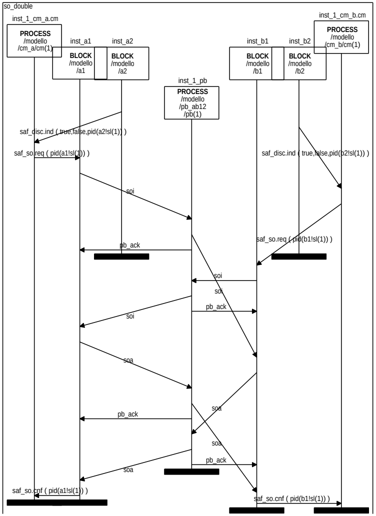


Specification SDL transition diagram: example

Executable specification!



Output MSC counter-examples: example


so_double
inst_1_cm_a.cm
PROCESS /modello /cm_a/cm(1)
inst_a1
BLOCK /modello /a1
inst_a2
BLOCK /modello /a2
inst_1_pb
PROCESS /modello /pb_ab12 /pb(1)
inst_b1
BLOCK /modello /b1
inst_b2
BLOCK /modello /b2
inst_1_cm_b.cm
PROCESS /modello /cm_b/cm(1)
saf_disc.ind (true,false,pid(a2!sl(1)))
saf_so.req (pid(a1!sl(1)))
saf_disc.ind (true,false,pid(b2!sl(1)))
soi
pb_ack
saf_so.req (pid(b1!sl(1)))
soi
soi
pb_ack
soi
soa
pb_ack
soa
soa
pb_ack
soa
saf_so.cnf (pid(a1!sl(1)))
saf_so.cnf (pid(b1!sl(1)))

Results:

- Implementation completed in planned time.
- First implementation passed all tests with 0 errors!
- Considered a methodological milestone for the company.

Reference:
A. Cimatti, P. Pieraccini, R. Sebastiani, P. Traverso, A Villafiorita
"*Formal specification and validation of a vital protocol*".
World Congress on Formal Methods. 1999.