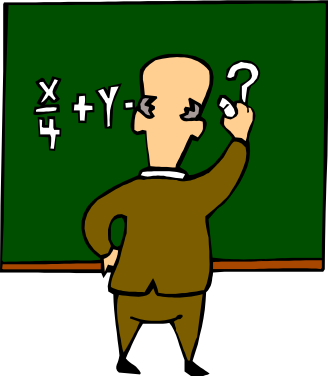


Test de Systèmes Informatiques

Burkhart Wolff
(basé sur Marie-Claude Gaudel)
LRI, Univ Paris-Sud & CNRS



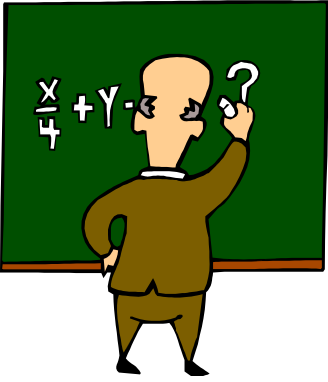
Software Testing can be formal too



- *« We know less about the theory of testing, which we do often, than about the theory of program proving, which we do seldom »*

Goodenough J. B., Gerhart S.,
IEEE Transactions on
Software Engineering, 1975

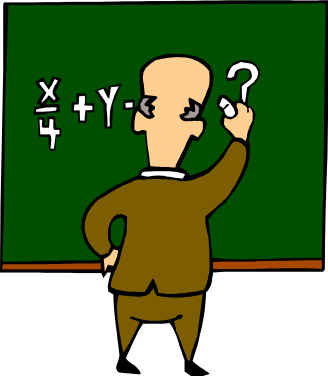




Outline of the course



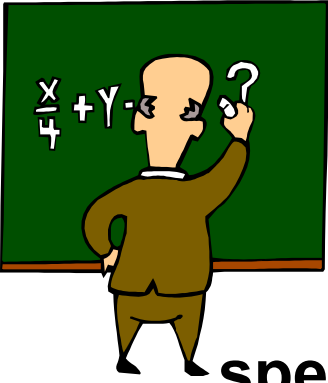
- Introduction
 - Formal specifications
 - Black-box testing
 - Putting them together
- Application to *Algebraic Specifications*
- Application to *Finite State Machines*
- Application to *Labelled Transition Systems with inputs and outputs*



Formal Specification?



- There is a notation
 - Pieces of text
 - Algebraic Spec (CASL), Z, VDM, B, CSP, Lotos, ...
 - Annotated diagrams
 - Finite State Machines, Petri Nets, « Automata-derived » diagrams (LTS, Kripke structures), Statecharts, ...
- **There is a formal semantics**
 - Algebras, Predicate transformers, Sets, Traces and Failures...
- There is a **formal system** (proofs) or a **verification method** (model-checking), or both.



Example 1: CASL

spec CONTAINER = NAT, BOOL

then

generated type *Container* ::= [] | _::__(Nat ; Container)

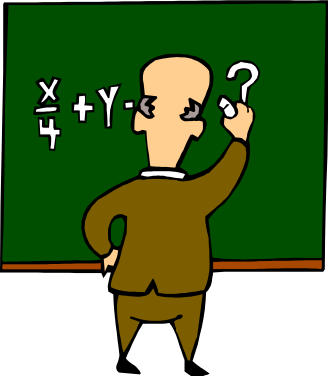
op *isin* : Nat × Container → Bool

op *remove* : Nat × Container → Container

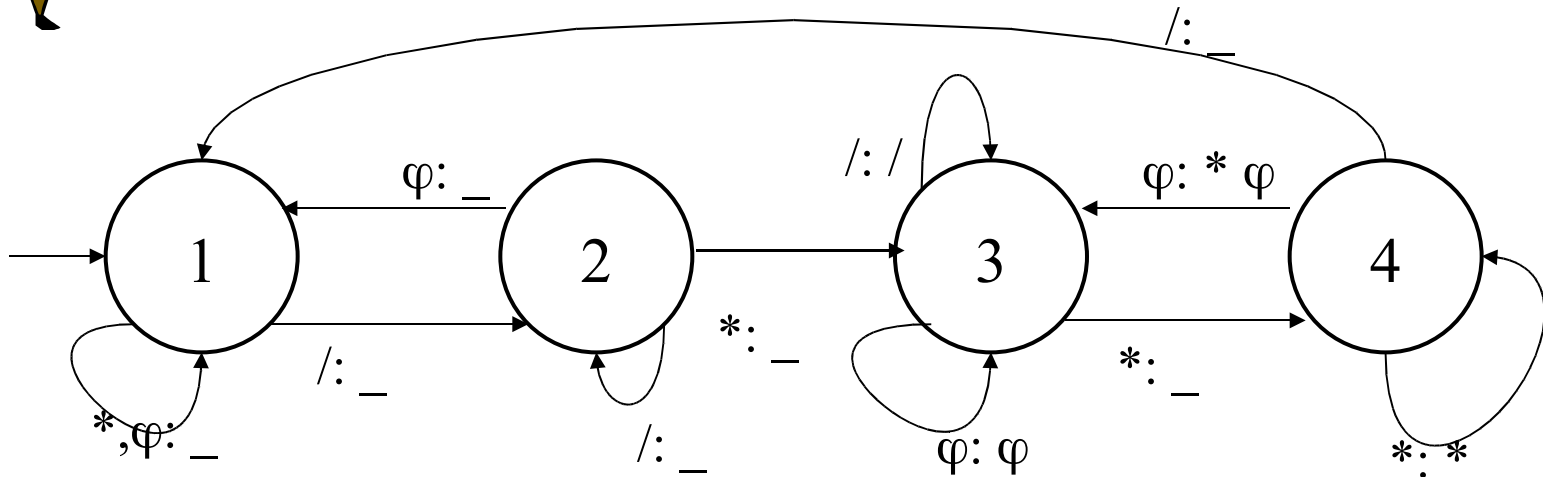
∀ *x, y*:Nat; *c*:Container

- *isin*(*x*, []) = false
- *eq*(*x*, *y*) = true ⇒ *isin*(*x*, *y*::*c*) = true
- *eq*(*x*, *y*) = false ⇒ *isin*(*x*, *y*::*c*) = *isin*(*x*, *c*)
- *remove*(*x*, []) = []
- *eq*(*x*, *y*) = true ⇒ *remove*(*x*, *y*::*c*) = *c*
- *eq*(*x*, *y*) = false ⇒ *remove*(*x*, *y*::*c*) = *y*::*remove*(*x*, *c*)

end



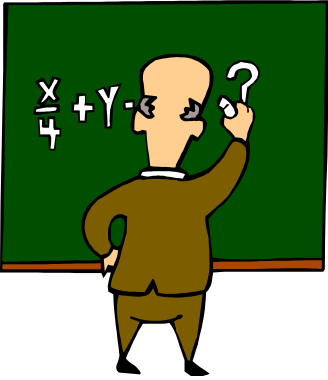
Example 2: FSM



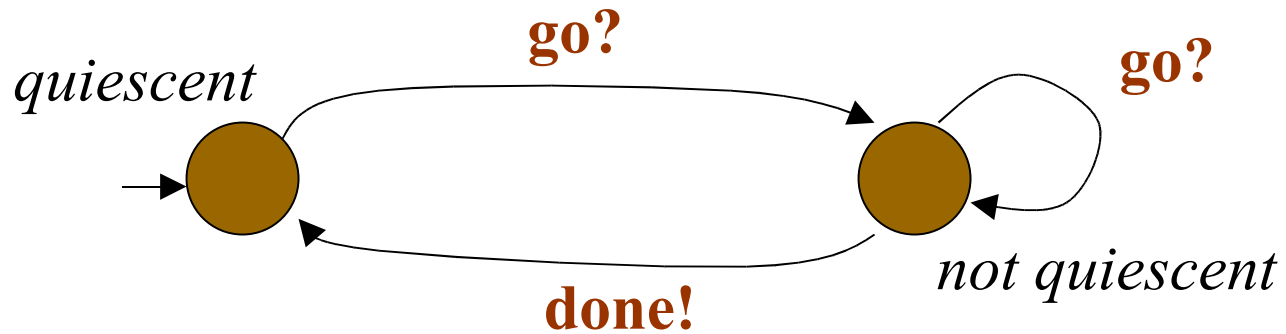
φ = any character but * and /

This is not a comment /* **all that** /*
is **** a comment** */ this is no more a comment.

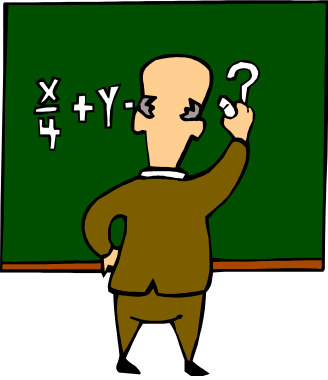
This FSM removes from the input text all that is not a comment



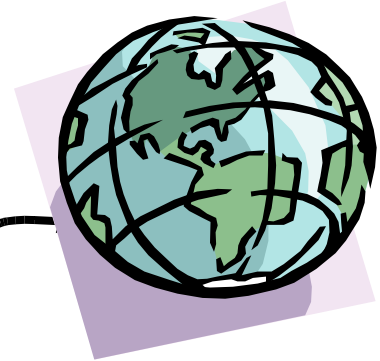
Example 3: IOTS



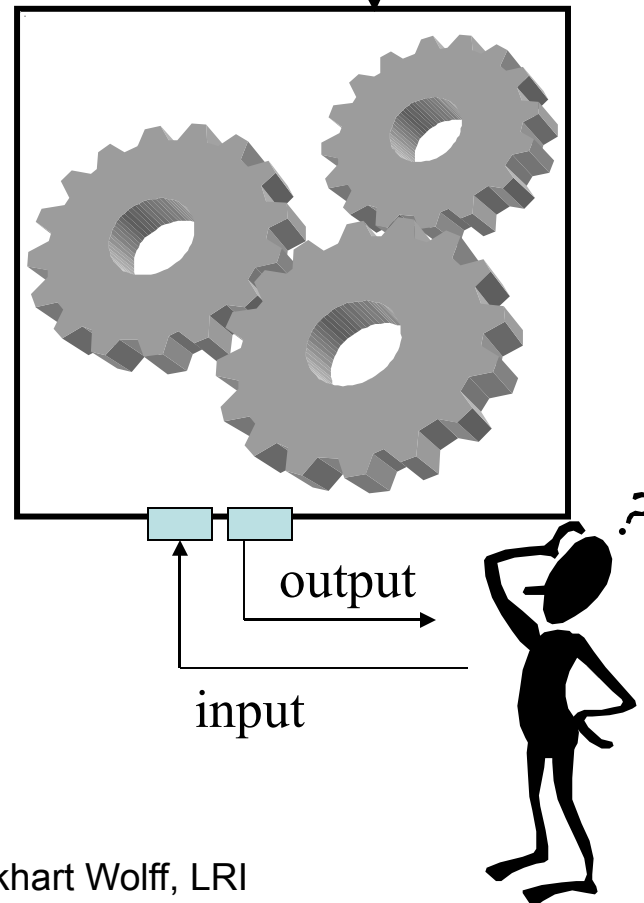
- In its initial state, this IOTS is idle until it receives *go*
- In any state, it accepts *go* (« input-enabledness »); after *go*, it may emit *done*.

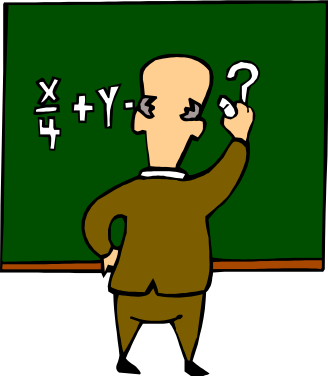


Testing Systems



- A system is a dynamic entity, embedded in the physical world
- It is *observable* via some limited interface/procedure
- It is not always *controllable*
- Quite different from a piece of text (formula, program) or a diagram



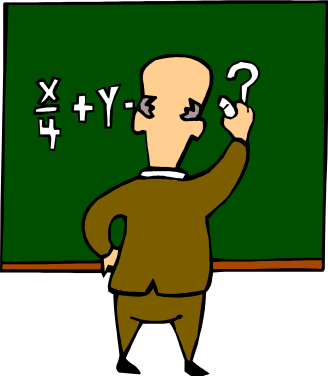


Back to basic concepts

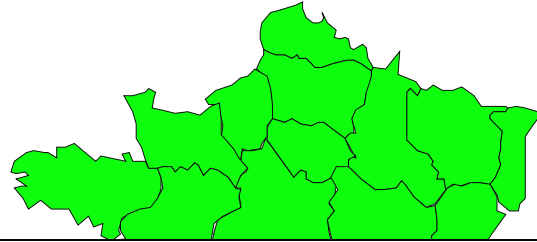


- *One proves formulas*
- *One checks models*
- *One tests systems*

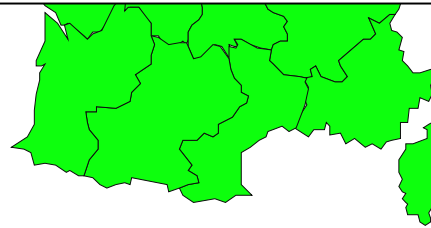
- *A system is not*
 - A formula
 - A formal specification
 - A model
 - A program (considered as a formula when proving programs)



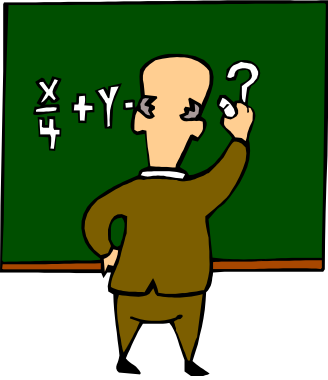
A philosophical interlude



“A map is not the territory”



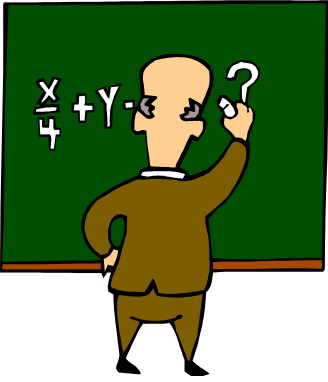
A program text, or a specification text, or a model, is not the system



Black-Box Testing



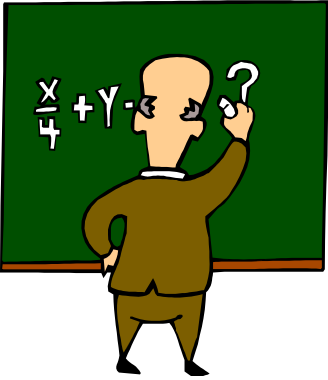
- **Black-Box Testing:**
 - the structure of the SUT (System Under Test) is not known
- **However,**
 - necessity of making explicit the class of “testable implementations” => notion of some **Testability Hypothesis** on the SUT



Testability?



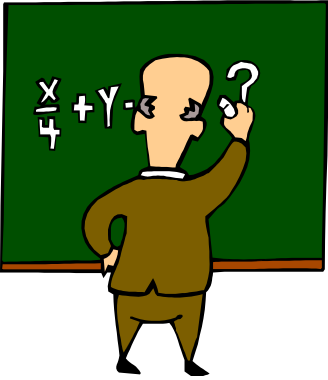
- If the SUT can be *any diabolic system*, there is no sensible way of testing it ☹
- Fortunately, *some basic assumptions are feasible* (example: correct implementation of booleans, determinism, ...)
- Some others can be *verified in another way*: static checks on the program, preliminary tests, ...



BBT + FS



- Importance of an adequate and formal definition of the **Satisfaction Relation** of an implementation w.r.t. a specification
- Careful definition of the class of considered implementations



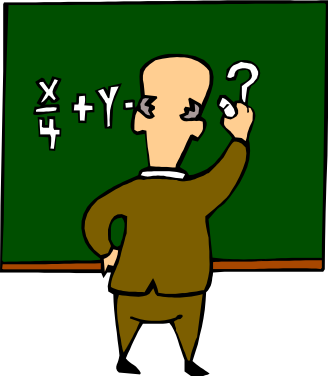
Wanted: a satisfaction relation



?



- Given some “testable” SUT, what does it mean that it satisfies SP?
- What is the correctness reference? Is there an “exhaustive” (or “complete”) set of tests?
- SP is some sort of **model or formula**; SUT is some sort of **system**; how to define “*SUT sat SP*” or “*SUT conf SP*” in such an heterogeneous context?



Content of the course



- The case of Algebraic Specifications
- The case of Finite State Machines
- The case of Input-Output Transition Systems
- NB: other cases in the literature