Model Checking @ GSSI

30/01/25

this notes are based on
[1] Baier, Katoen: Principles of model checking. MIT Press (chapters 1-3, 5, and 6)
[2] Clarke, Grumberg, Peled: Model checking. MIT Press

We'll upload the material at https://clegra.github.io/mc.html

INTRODUCTION

sw is nowadays ubiquitous ⇒ sw correctness (always relative to specs!) is valuable

> 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', in Images of SMC Research 1996

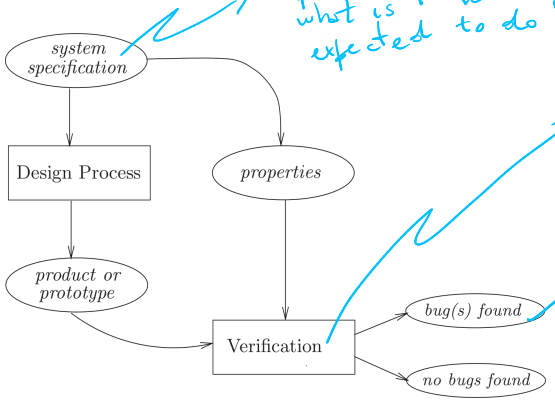
- Sw bugs = loss of { lives, money, reputation }

Examples
- Therac-25 ≥ 6 deaths between 1985-1987
- Ariane-5 exploded 36 s after launch
- Baggage handling system @ Denver airport ($ 1.1·10^6 × day × 9 months)
- Pentium bug $ 475·10^6

https://emwww.github.io/home/swbadness.html

- Simulation / testing { + "concrete" artefacts are checked; + "simple"; − partial (when should we stop?) }
- Deductive reasoning { + infinite state systems; − "hard" & time consuming; − interactive }

Borrowed from [1]



what is the system expected to do? AND what is it not expected to do?

establishes if the system under construction / analysis enjoys some properties

the system either does something not expected or it does not do something expected

Exercise. Consider the 3 python functions implenting Example 1.1 in [1]:

```
def inc():
    while loop:
        if x < bound:
            x += 1
```

```
def dec():
    while loop:
        if x > 0:
            x -= 1
            loop = x ≥ 0
```

```
def reset():
    while loop:
        if x == bound:
            x = 0
```

} counter.py

Take the property

φ = "counter never stops"

Does φ hold if initially x == 0, loop == True, bound == 200 & inc, dec, and reset above execute concurrently?

We'll focus on **Model Checking**, but let's dissect bugs first

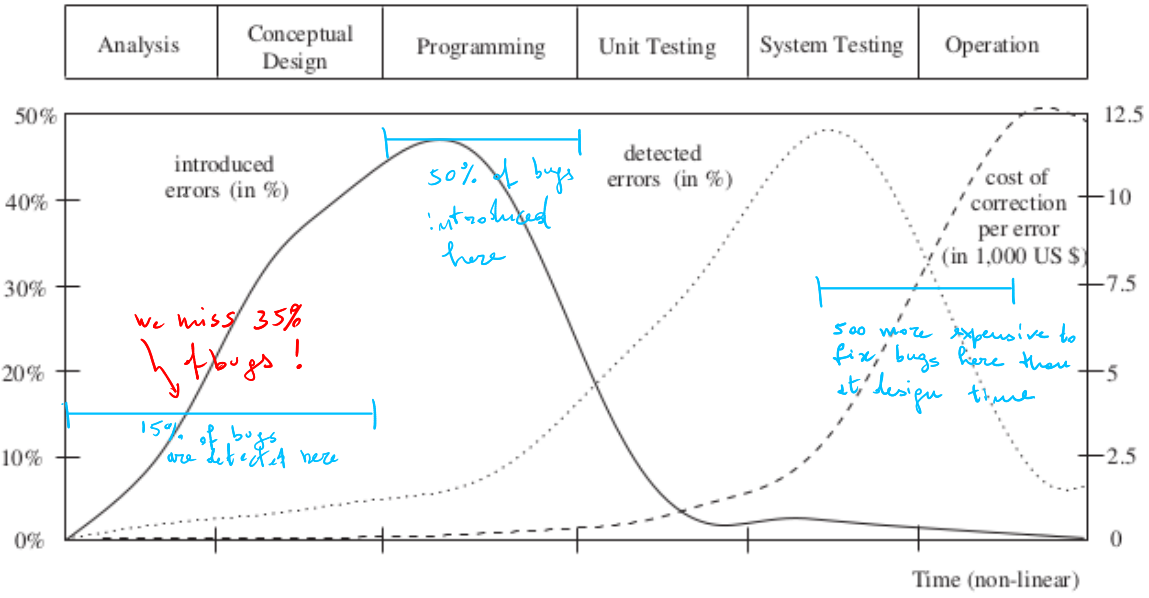
Empirical evidence shows that errors do not distribute evenly

- in space (bugs tend to concentrate in few modules)
- in time (bugs are introduced unevenly in different phases of sw devel.)

The sooner bugs are found, the better!

model-based verification helps here

ref. P. Liggesmeyer and M. Rothfelder and M. Rettelbach and T. Ackermann. Qualitätssicherung Software-basierter technischer Systeme. Informatik Spektrum, 21(5):249–258, 1998.

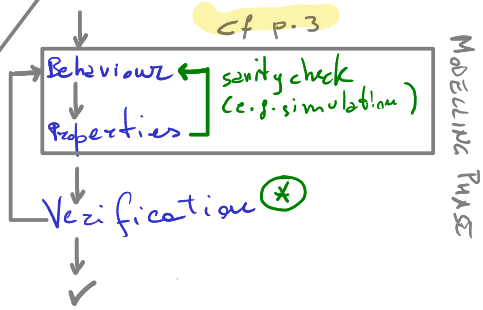


Quoting [1]
"In software and hardware design of complex systems, more time and effort are spent on verification than on construction. Techniques are sought to reduce and ease the verification efforts while increasing their coverage. Formal methods offer a large potential to obtain an early integration of verification in the design process, to provide more effective verification techniques, and to reduce the verification time."

- Model checking

- state explosion pb
- finite state spaces
+ "easier"
+ "automatic" (the verification phase +/-)
+ → "yes" = no bugs c.f. with testing
  → "no" & C.E.

the methodology := of MC



design → machine processable models

properties → typically some (temporal) logics

ideally "push-button" in practice analysis of results

Note Modelling could be partly "automatic" (eg. compiling from design)
Verification is mainly automatic

(*) Question : What should we do when we get an error trace ?

Glancing at temporal logics

> Note Temporal logics stem from philosophy: modal logics to reason about time in natural language!

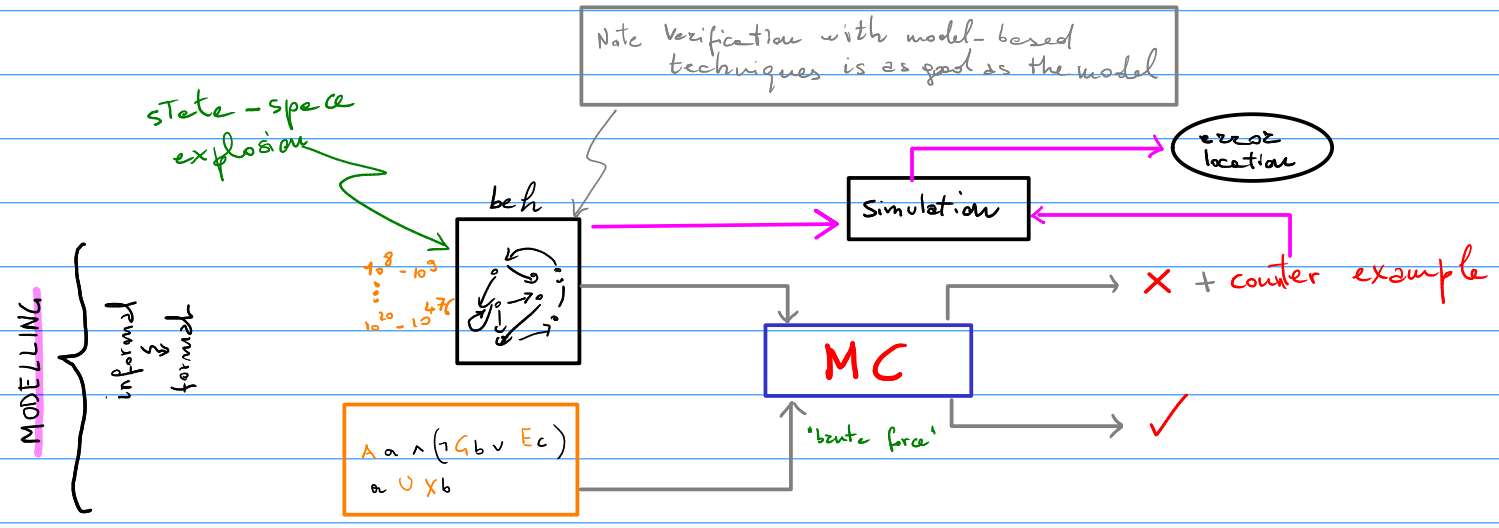
- designed to predicate on concurrent events
- events are ordered in time
- but time is not explicit

modality $\Box\phi$ eg $\Box(\neg(a \wedge b))$ = it will never happen that events (a) and (b) occur "at the same time".

(a) = thread a writes x

(b) = thread b reads x

Schematically cf Fig 1.4 [1]



Exercise. Consider the following python implementation of Example 1.1 in [1]:

```
def inc():
    while loop:
        if x<bound:
            x += 1
```

```
def dec():
    while loop:
        if x>0:
            x -= 1
```

```
def reset():
    while loop:
        if x==bound:
            x = 0
```

How does the (temporal) property

Always $0 \leq x \leq 200$

relate to p in the Exercise on page 1

What we're going to see

- Modelling (concurrent) systems
- Temporal properties
- Fairness conditions
- Temporal logics: LTL, CTL, CTL*
- Partial M.C. ← recent research

What we're not looking at

- Partial order reductions
- μ-calculus
- Abstraction techniques
- Quantitative/performance analysis
- Timed models

Modelling

what?

VALIDATION vs VERIFICATION

Are we building the right thing? ↕ is the design faithfully "capturing" the reqs?

Are we building the thing right? ↕ does the design satisfy the expected properties?

- Going formal
- Right level of abstraction

≡ precise, but not "cumbersome"

Reactive Systems (Manna, Pnueli 1995)

- Concurrent
- Interact with an environment ("open")
- possibly non terminating

} NOT FUNCTIONS!

STATE ↗ snapshot of the system "at a given time"

&

TRANSITION ↖ evolution of the system "in time"

LTS

$$s \xrightarrow{\alpha} s'$$

KRIPKE Structures

Transition system $TS = (S, Act, \rightarrow, I, AP, L)$

Actions can suitably model interactions, synchronisation & communication

where
- S is a set of states
- Act is a set of actions; in Kripke structures Act is a singleton
- $\rightarrow \subseteq S \times Act \times S$ Transition relation
- $I \subseteq S$ are the initial states
- AP is a set of atomic propositions
- $L: S \rightarrow 2^{AP}$

TS is finite if S, Act, and AP are finite

S & $L(S)$ are finite

WLOG we consider transition systems where $I \neq \emptyset$

If $I = \emptyset \Rightarrow$ no behaviour