

These 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

INTRODUCTION

SW is nowadays ubiquitous \Rightarrow SW correctness is valuable

always relative to specs!

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)

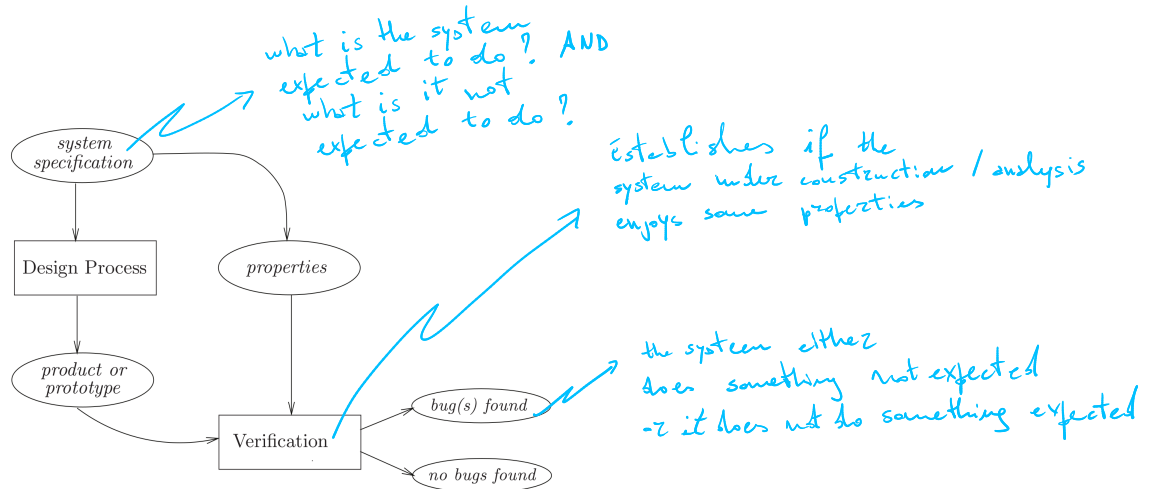
- SW bugs = loss of $\left. \begin{array}{l} \text{lives} \\ \text{money} \\ \text{reputation} \end{array} \right\}$

Examples
 Therac-25 \gg 6 deaths between 1985-1987
 Ariane-5 exploded 36s after launch
 Baggage handling system @ Denver airport ($\$ 1.1 \cdot 10^6 \times \text{day} \times 3 \text{ months}$)
 Pentium bug $\$ 495 \cdot 10^6$

<https://www.reuters.com/business/autos-transportation/us-probing-fatal-tesla-crash-that-killed-pedestrian-2021-09-03>
<https://www.tesladeaths.com/>

- Simulation / testing $\left\{ \begin{array}{l} + \text{concrete artefacts are checked} \\ + \text{"simple"} \\ - \text{partial (when should we stop?)}$
- Deductive reasoning $\left\{ \begin{array}{l} + \text{infinite state systems} \\ - \text{"hard" \& time consuming} \\ - \text{interactive}$

Borrowed from [1]



Exercise. Consider the 3 python functions essentially implementing 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 $\varphi = \text{"counter never stops"}$

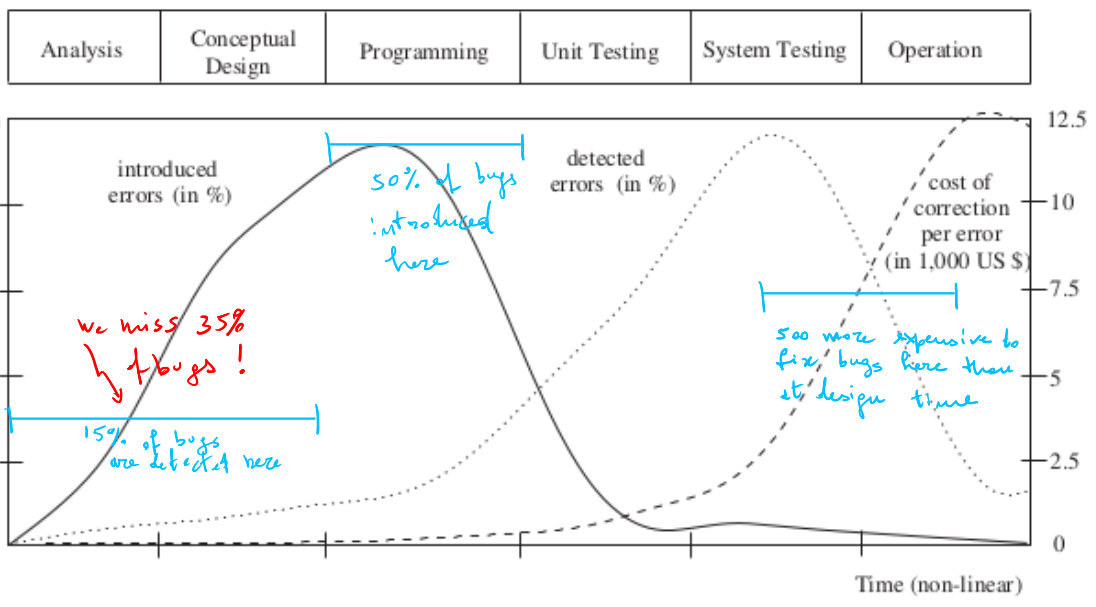
Does φ hold if initially $x=0$, $loop=True$, $bound=200$ & `inc`, `dec`, and `reset` above execute concurrently?

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) and in
 • time (bugs are introduced unevenly in different phase of sw devel.)

The sooner, the better!

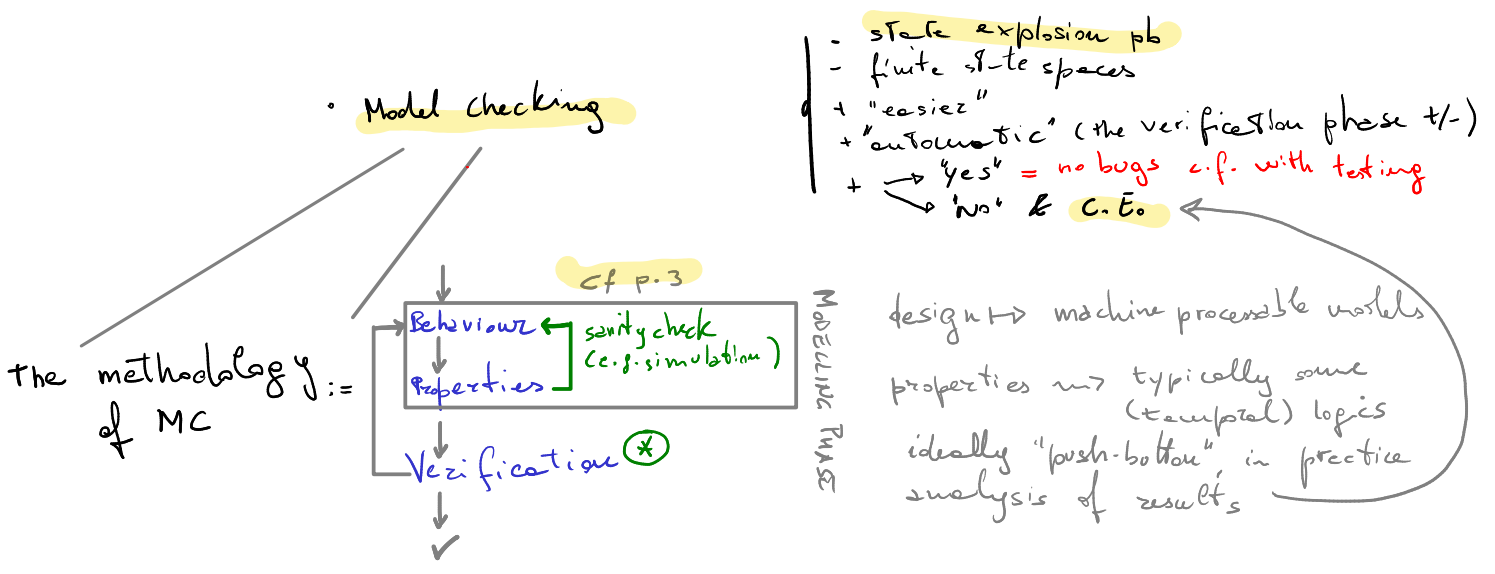
model-based verification helps here



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."



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

Glancing at Temporal Logics

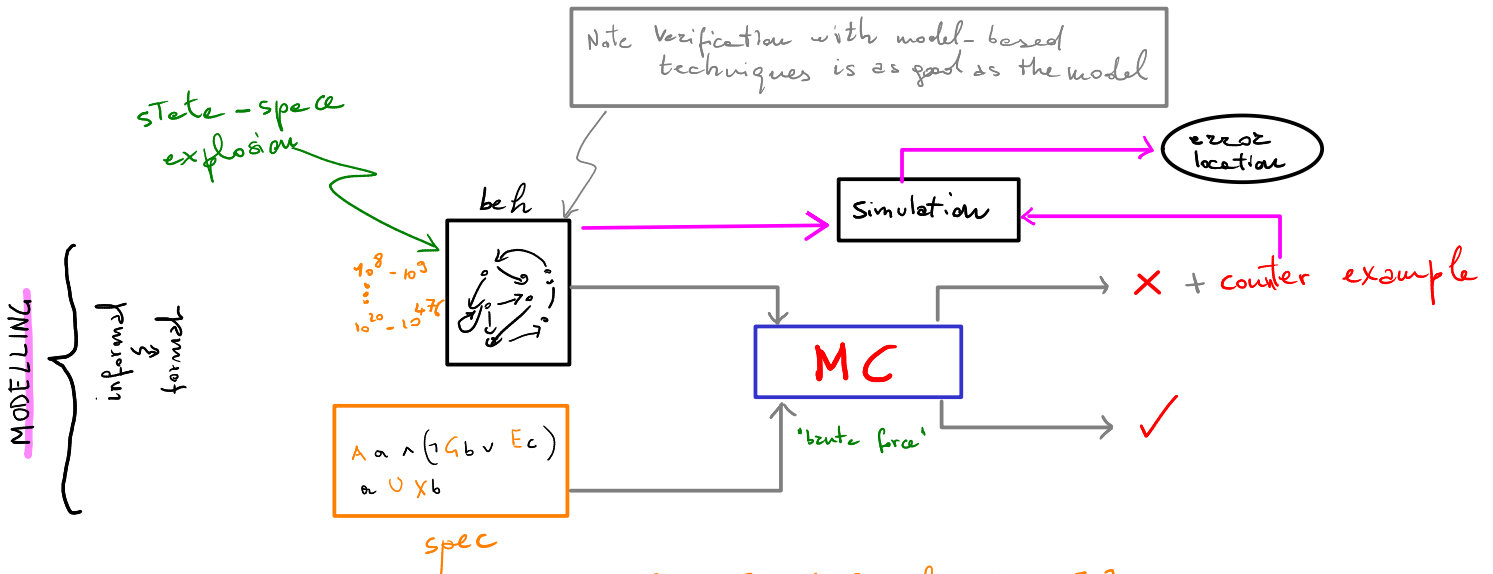
Note Temporal Logics stem from philosophy: modal logics to reason about time in natural language!

- 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".
 = thread a writes x
 = 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 ϕ 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?

Are we building the thing right?

is the design faithfully "capturing" the reqs?

does the design satisfy the expected properties?

- Going formal
 - Right level of abstraction
- ≡ precise, but not "cumbersome"

Reactive Systems
(Mazur, Pouch 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



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 & $\rightarrow(S)$ are finite

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

If $I = \emptyset \Rightarrow$ no behaviour