

# Introduction to Model Checking

<https://clegra.github.io/mc.html>

## Lecture 1

Clemens Grabmayer

<https://clegra.github.io>

Emilio Tuosto

<https://cs.gssi.it/emilio.tuosto/>

Department of Computer Science



January 26, 2026

# Organization

## Lectures (Clemens 4 / Emilio 3)

- ▶ this week:
  - ▶ Monday, 10.30–12.30, room B, Aurora
  - ▶ Tuesday, 14.30–16.30, P-2.6, Zenith
  - ▶ Thursday and Friday, 14.30–16.30, conference room -1, Zenith
- ▶ next week:
  - ▶ Monday, Tuesday, Thursday, 14.30–16.30, conference room -1, Zenith
- ▶ presentations (mainly) on blackboard
- ▶ notes after the lecture (notes 2024/25 already available)

## Webpage

- ▶ <https://clegra.github.io/mc.html>  
stable for later: <https://clegra.github.io/mc/25-26/mc.html>

## Exam (more later)

- ▶ options:
  - ▶ small verification project (of an algorithm, e.g. in Maude)
  - ▶ presentation about a paper
  - ▶ written exam?

# Topics of the course

- ▶ modeling systems by **labeled transition systems (LTSs)**
- ▶ **safety**, **liveness**, and **fairness** properties
- ▶ **Linear Temporal Logic (LTL)**
  - ▶ model checking formulas
    - ▶ express properties by **Büchi automata**
    - ▶ model check LTSs and properties via **product automata**
- ▶ **Computation Tree Logic (CTL)**
- ▶ **Maude** examples **model-checker**
- ▶ (**partial** model checking)
  - ▶ (partially known systems (state properties/states/transitions))

# Model Checking

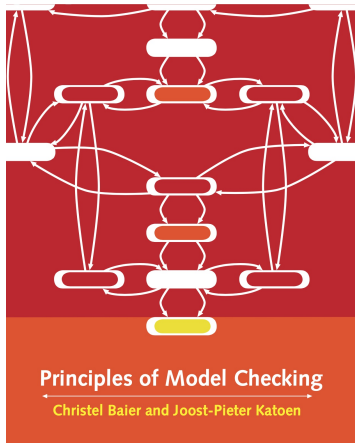
... is an effective automatable technique:

- ▶ *to expose potential software design errors;*
- ▶ *that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for that model.*
- ▶ widely applied in industry
  - ▶ embedded systems, software engin., hardware design, explainable AI
- ▶ supports partial verification (of system parts)
- ▶ provides diagnostic information for debugging
- ▶ has sound mathematical underpinning (logic and process theory)

**Course Goals** are introduction to:

- ▶ **Theory:**
  - ▶ modeling systems by labeled transition systems,
  - ▶ expressing properties by temporal-logic formulas
  - ▶ model-checking algorithms
- ▶ **Practice related:** see Maude examples

# Book



- ▶ pdf online available (see [1])
- ▶ we study chapters 1–3, 5, 6

# Lectures

1. Introduction (preview extended, LTSs)
2. Modeling by labeled transition systems
  - ▶ executions; traces; non-determinism; examples
3. Linear-Time Behaviour and Properties
  - ▶ invariant, safety, and liveness (and fairness) properties
4. Linear Temporal Logic (LTL)
  - ▶ syntax and semantics; interpretation of LTSs; examples
5. LTL (continued)
  - ▶ model checking of LTL formulas, and fairness in LTL
  - ▶ Maude examples
6. Computation Tree Logic (CTL)
  - ▶ syntax and semantics, examples
7. Extensions of CTL, and Outlook
  - ▶ expressibility differences with LTL
  - ▶ model checking formulas in CTL
  - ▶ ( $\mu$ -calculus | partial model-checking | Maude examples)

# Software correctness

- ▶ Software is ubiquitous  $\implies$  software is valuable

*"It is fair to state that in this digital era correct systems for information processing are more valuable than gold." (H.P. Barendregt, 'The quest for correctness', in Images of SMC Research, 1996)."*

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

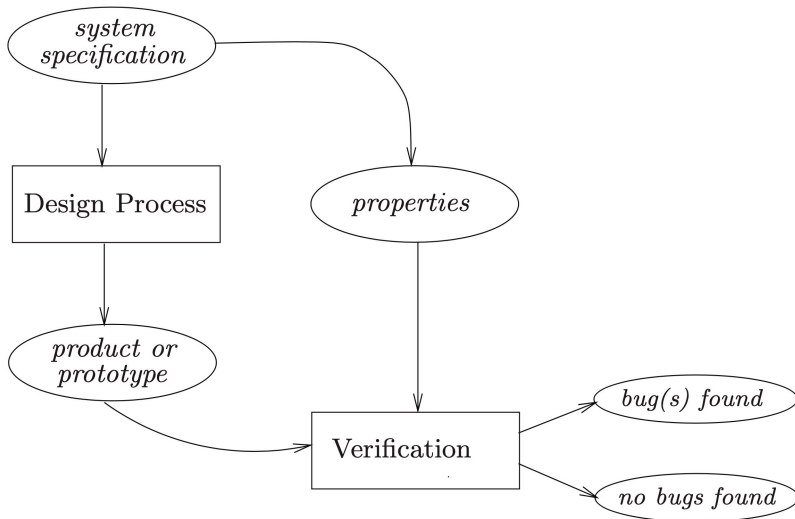
- ▶ Simulation / testing

- $\left\{ \begin{array}{l} + \text{ concrete implemented systems are checked} \\ + \text{ "simple"} \\ - \text{ partial (when do we stop?)} \end{array} \right.$

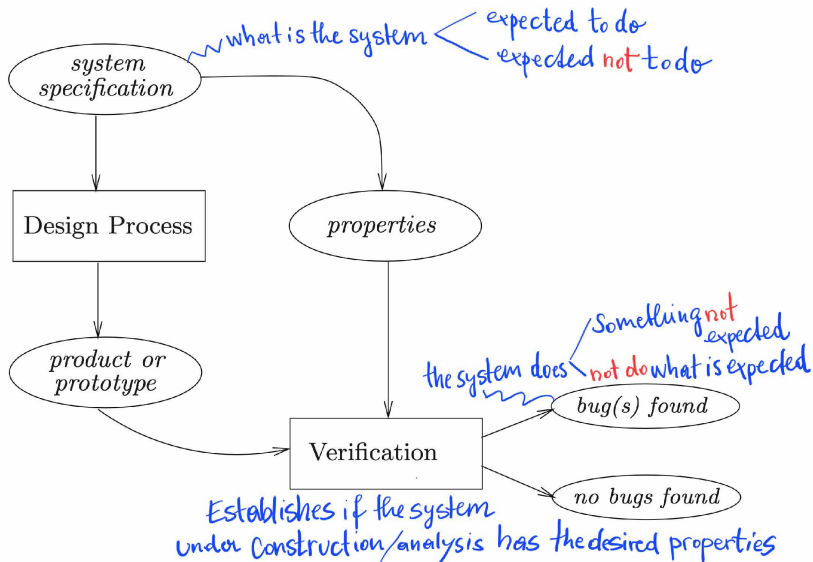
- ▶ Deductive reasoning

- $\left\{ \begin{array}{l} + \text{ also possible for infinite-state system} \\ - \text{ hard and time-consuming} \\ - \text{ interactive} \end{array} \right.$

# Hard-/Software Verification (traditionally)



# Hard-/Software Verification (traditionally)



# Software verification methods

## ► Peer review

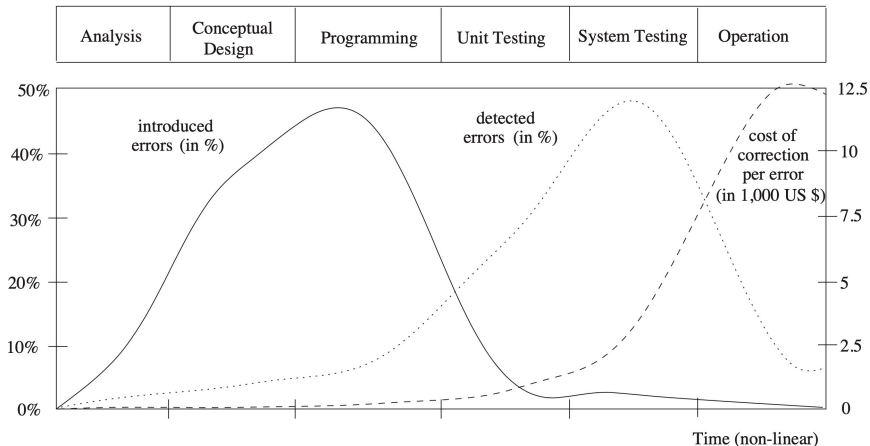
- $\left\{ \begin{array}{l} + \text{ rather useful: between 31\% and 93\% (median 60\%)} \\ - \text{ hard to catch subtle errors (concurrency and algorithmic defects)} \end{array} \right.$

## ► Software testing

- $\left\{ \begin{array}{l} + \text{ applicable to all sorts of software} \\ + \text{ test generation can (partly) be automated} \\ - \text{ exhaustive testing of all execution paths is infeasible} \\ - \text{ When do we stop?} \end{array} \right.$

## ► Catching software errors: [the sooner the better](#).

# Software lifecycle, error introduction/detection, repair costs

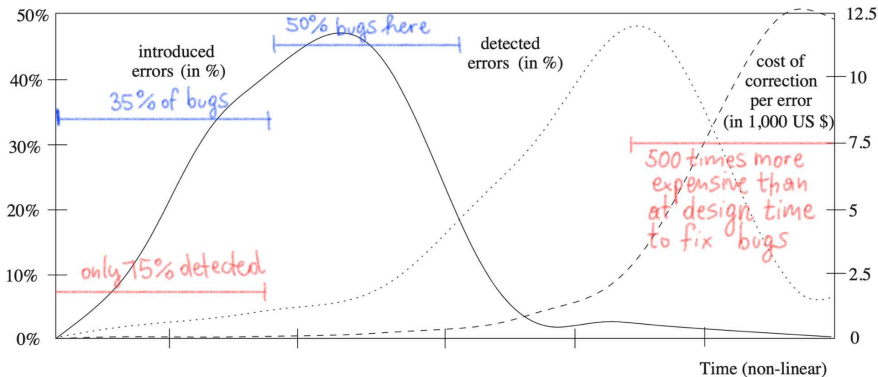


Liggesmeyer et al: "Qualitätssicherung technischer Systeme ..." [4, 1998]

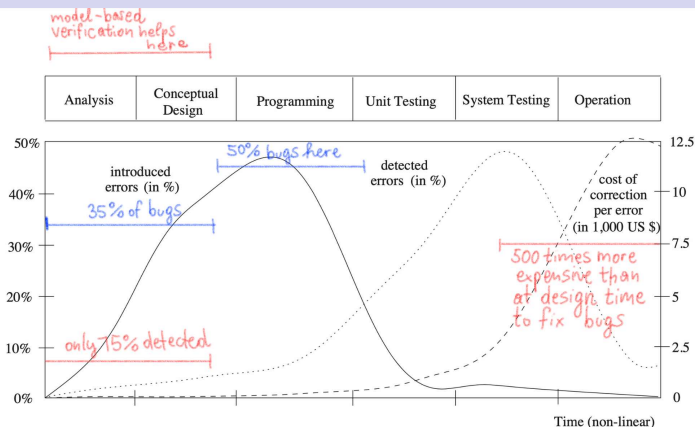
# Software lifecycle, error introduction/detection, repair costs

model-based  
verification helps  
here

Analysis	Conceptual Design	Programming	Unit Testing	System Testing	Operation
----------	-------------------	-------------	--------------	----------------	-----------

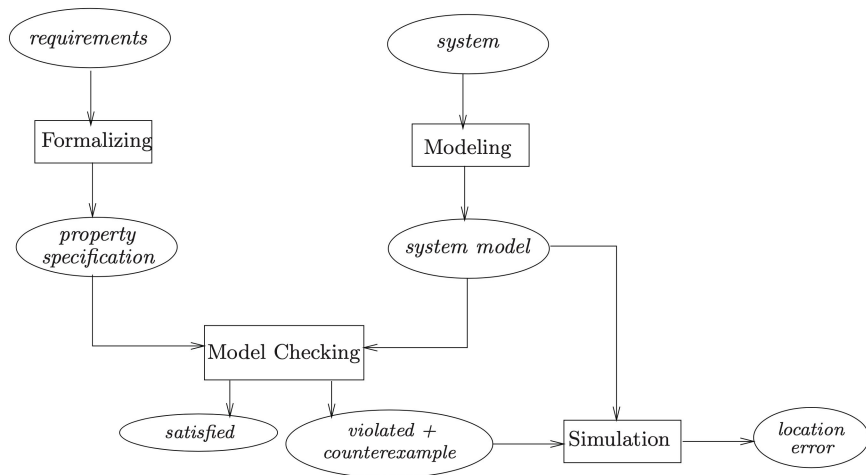


# Software lifecycle, error introduction/detection, repair costs



“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



# Glancing at temporal logic

## Temporal logic

- ▶ stems from philosophy: modal logic to reason about time  
(in an idealized natural language)
- ▶ makes it possible to reason about concurrent events
- ▶ events are ordered in time
- ▶ time is considered to be discrete
- ▶ but moments of time are only referred to relatively

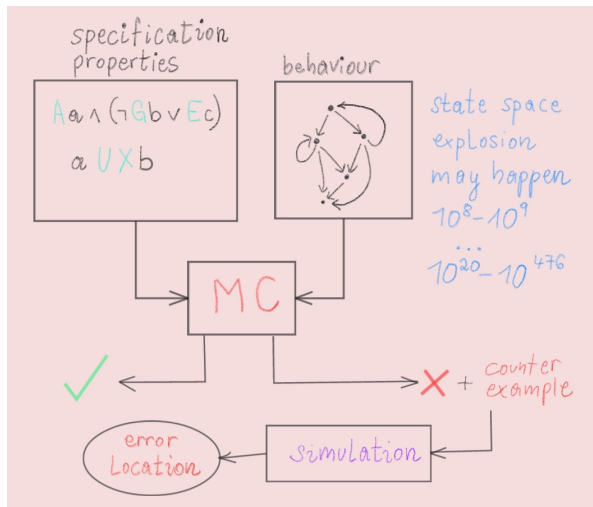
## Example

Modality  $\Box\phi \hat{=}$  for all time moments,  $\phi$  holds.

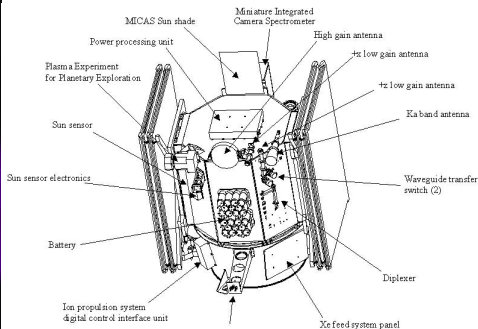
Then we have:

$$\begin{aligned} \Box(\neg(a \wedge b)) &\hat{=} \text{for all time moments,} \\ &\quad \text{events } a \text{ and } b \text{ do not occur at the same time,} \\ &= \text{it will never happen that} \\ &\quad \text{events } a \text{ and } b \text{ occur at the same time.} \end{aligned}$$

# Glancing at temporal logic (its use for model checking)



# Deep Space 1 (NASA)



- ▶ Flyby of asteroid 9969 Braille (1999)
- ▶ Entered the coma of Comet Borrelly (2001)
- ▶ Model checking discovered **5 concurrency errors**

# Example (program concurrency/non-determinism)

Programs **Inc**, **Dec**, and **Reset** cooperate, and use a shared variable  $x$ :

**proc Inc**

```
while  $0 \leq x \leq 200$ 
do
  if  $x < 200$ 
  then  $x := x + 1$ 
  fi
od
```

**proc Dec**

```
while  $0 \leq x \leq 200$ 
do
  if  $x > 0$ 
  then  $x := x - 1$ 
  fi
od
```

**proc Reset**

```
while  $0 \leq x \leq 200$ 
do
  if  $x = 200$ 
  then  $x := 0$ 
  fi
od
```

**Question:** When started on  $x = 0$ , do the counter programs run forever?  
Is  $0 \leq x \leq 200$  always guaranteed?

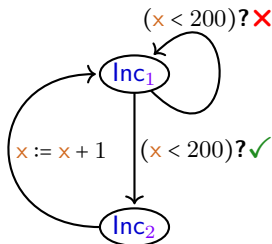
# Modeling (by program graphs)

**proc Inc**

**while true**

**do**

```
1:  if  $x < 200$ 
2:  then  $x := x + 1$ 
    fi
od
```

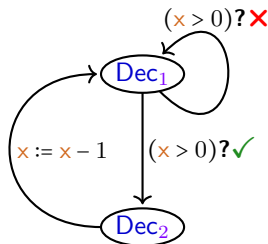


**proc Dec**

**while true**

**do**

```
1:  if  $x > 0$ 
2:  then  $x := x - 1$ 
    fi
od
```

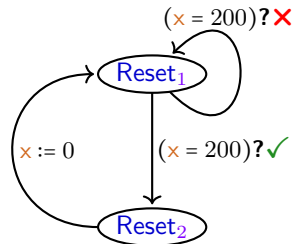


**proc Reset**

**while true**

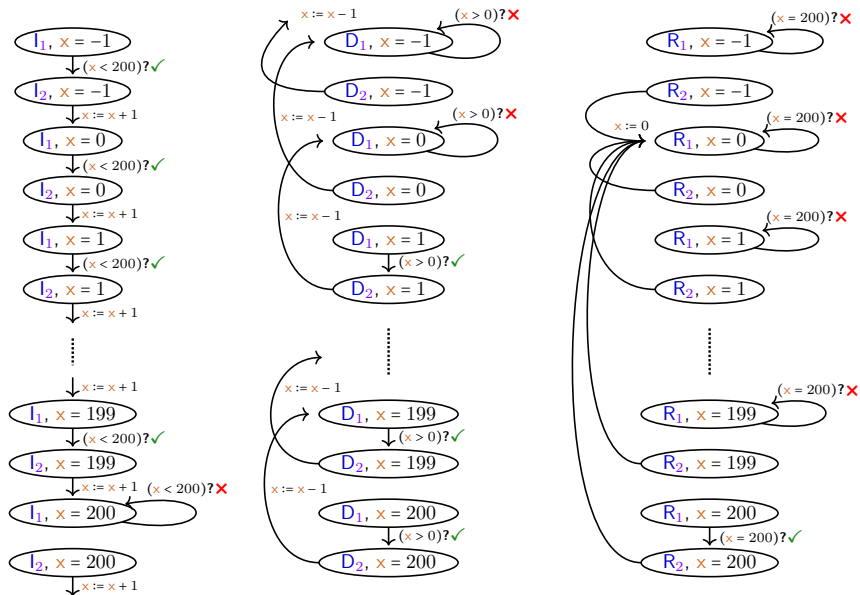
**do**

```
1:  if  $x = 200$ 
2:  then  $x := 0$ 
    fi
od
```

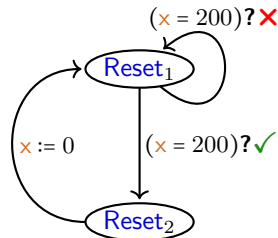
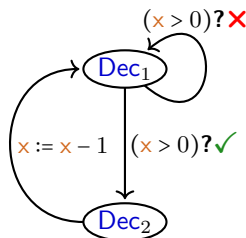
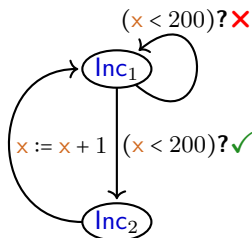


Program graphs (PG)

# Modeling (by labeled transition systems, with state space explosion)



# Formalizing properties (in temporal logic)



We assume  $x := 0$  initially.

$\text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \stackrel{?}{\models} \Box(0 \leq x \wedge x \leq 200)$  (Linear-TL formula)

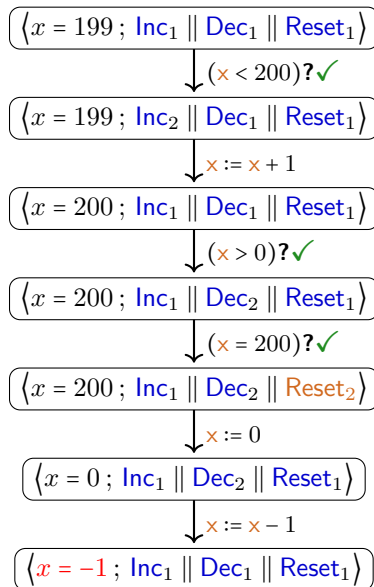
$\text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \models \Diamond(x < 0)$  (LTL formula)

$\text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \not\models \forall \Box(0 \leq x \wedge x \leq 200)$  (Computation-Tree-L formula)

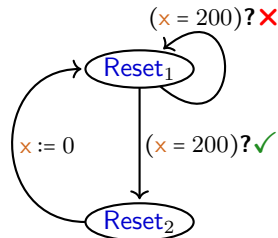
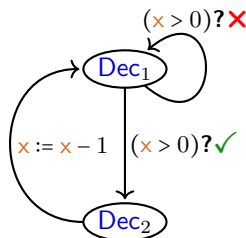
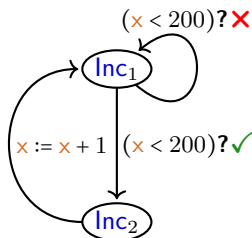
$\text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \models \exists \Box(0 \leq x \wedge x \leq 200)$  (CTL formula)

$\text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \models \forall \Box \exists \Diamond(x < 0)$  (CTL formula)

# Counterexample (offending execution trace)



# Formalizing properties (in temporal logic)



We assume  $x := 0$  initially.

$\text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \stackrel{?}{\models} \Box(0 \leq x \wedge x \leq 200)$  (Linear-TL formula)

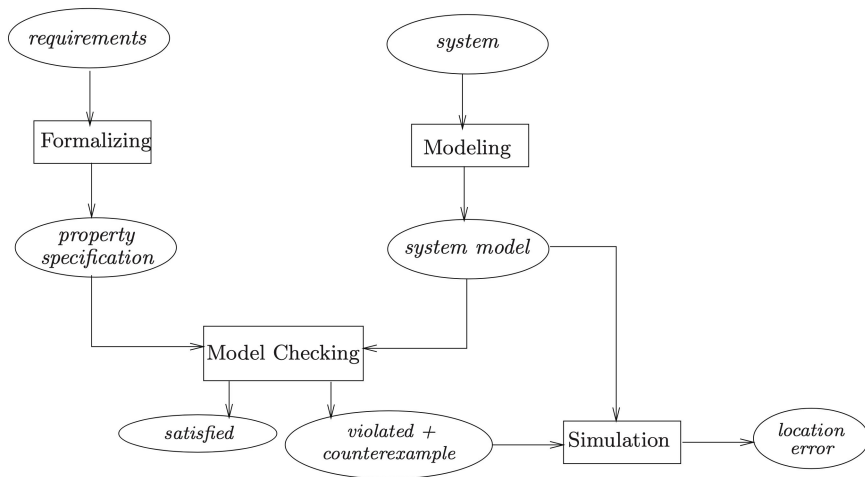
$\text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \models \Diamond(x < 0)$  (LTL formula)

$\text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \not\models \forall \Box(0 \leq x \wedge x \leq 200)$  (Computation-Tree-L formula)

$\text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \models \exists \Box(0 \leq x \wedge x \leq 200)$  (CTL formula)

$\text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \models \forall \Box \exists \Diamond(x < 0)$  (CTL formula)

# Model checking: validation



*Any [such] verification is only as good as the model of the system.*

# Modeling: Verification versus Validation

## VERIFICATION

Are we building the **thing** right?



Does the **design** satisfy the  
expected properties?

## VALIDATION

Are we building the **right thing**?



Is the **design** faithfully capturing  
the requirements?

# Model-checking: strengths and weaknesses

## Strengths:

- ▶ it is a **general** technique
- ▶ supports **partial** verification, checking properties individually
- ▶ bug-detection is **not dependent on chance**
- ▶ provides **diagnostic information** for debugging
- ▶ potential push-button technology
- ▶ widely used in industry
- ▶ can be **integrated** in existing development cycles
- ▶ **sound mathematical underpinning** (graph alg's, data structures, logic)

## Weaknesses:

- ▶ ideal for **control-intensive** applic., **less so** for **data-intensive** applic.
- ▶ verifies **system model**, not the actual system
- ▶ only checks stated requirements (no guarantee for completeness)
- ▶ suffers from **state-explosion** problem
- ▶ modeling requires expertise (finding smaller models, state properties as formulas)
- ▶ model-checker may have defects as well
- ▶ does not permit to directly check generalizations (infinitely many components, parameterized systems)

# Origins of model checking

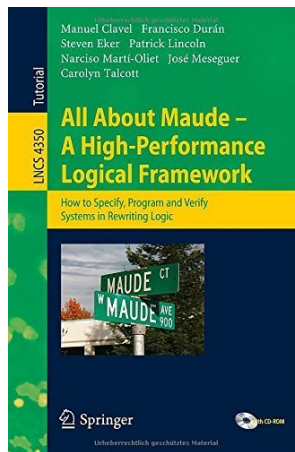
- I. [Clarke and Emerson](#) [2, 1986]: *Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic*.  
They coined the term **model checking**.
- II. [Queille and Sifakis](#) [5, 1982]: *Specification and Verification of Concurrent Systems in CESAR*.
- III. [Hajek](#) (1978), and [West](#) (1978): introduced automated **protocol validation techniques** based on a **brute-force examination of the entire state space**, of which model checking can be viewed to be an extension.  
These earlier techniques were restricted to checking **absence of deadlocks or livelocks**, whereas model checking allows for the investigation of a broader class of properties.

See Baier and Katoen [1] for more references.

# Lectures

1. Introduction (preview extended, LTSs)
2. Modeling by labeled transition systems
  - ▶ executions; traces; non-determinism; examples
3. Linear-Time Behaviour and Properties
  - ▶ invariant, safety, and liveness (and fairness) properties
4. Linear Temporal Logic (LTL)
  - ▶ syntax and semantics; interpretation of LTSs; examples
5. LTL (continued)
  - ▶ model checking of LTL formulas, and fairness in LTL
  - ▶ Maude examples
6. Computation Tree Logic (CTL)
  - ▶ syntax and semantics, examples
7. Extensions of CTL, and Outlook
  - ▶ expressibility differences with LTL
  - ▶ model checking formulas in CTL
  - ▶ ( $\mu$ -calculus | partial model-checking | Maude examples)

# Book Maude



- Maude documentation page:  
<https://maude.cs.illinois.edu/documentation>

# Maude code (idea)

```

cr1 [Inc1a]      :  Inc1 x => Inc2 x   if x < 200
r1  [Inc2]      :  Inc2 x => Inc1 (x + 1)
cr1 [Inc1b]      :  Inc1 x => Inc1 x   if not(x < 200)

cr1 [Dec1a]      :  Dec1 x => Dec2 x   if 0 < x
r1  [Dec2]      :  Dec2 x => Dec1 (x - 1)
cr1 [Dec1b]      :  Dec1 x => Dec1 x   if not(0 < x)

cr1 [Reset1a]    :  Reset1 x => Reset2 x   if x = 200
r1  [Reset2]     :  Reset2 x => Reset1 0
cr1 [Reset1b]    :  Reset1 x => Reset1 x   if not(x = 200)

eq initial = { Inc1 Dec1 Reset1 0 }

ceq (S1 S2 S3 x |= counterge0) = true   if (0 = x \/ 0 < x)
ceq (S1 S2 S3 x |= counterlt0) = true   if (x < 0)
ceq (S1 S2 S3 x |= counterle200) = true if (x < 200 \/ x = 200)

```

# Maude output (simplified)

```
Maude> red modelCheck(initial, <> counterlt0)
reduce in COUNTERS-CHECK : modelCheck(initial, <> counterlt0)
result ModelCheckResult:
result Bool : true

Maude> red modelCheck(initial, [](counterge0 /\counterle200)
reduce in COUNTERS-CHECK :
      modelCheck(initial, [](counterge0 /\counterle200)
result ModelCheckResult:
counterexample({Inc1 Dec1 Reset1 199}
               {Inc2 Dec1 Reset1 199}
               {Inc1 Dec1 Reset1 200}
               {Inc1 Dec2 Reset1 200}
               {Inc1 Dec2 Reset2 200}
               {Inc1 Dec2 Reset1 0}
               {Inc1 Dec1 Reset1 -1})
```

# Lectures (tentative)

1. Introduction (preview extended)
2. Modeling by labeled transition systems
  - ▶ executions; traces; non-determinism; examples
3. Linear-Time Behaviour and Properties
  - ▶ invariant, safety, and liveness (and fairness) properties
4. Linear Temporal Logic (LTL)
  - ▶ syntax and semantics; interpretation of LTSs; examples
5. LTL (continued)
  - ▶ fairness in LTL
  - ▶ model checking of LTL formulas
6. Computation Tree Logic (CTL)
  - ▶ syntax and semantics, examples
7. Extensions of CTL, and Outlook
  - ▶ expressibility differences with LTL
  - ▶ model checking formulas in CTL
  - ▶ Maude examples | partial model-checking |  $\mu$ -calculus

# Exam options

Together we decide on a topic.

Study an article, or a book chapter.

or

Dive deeper into a proof from the lecture.

or

Model a basic algorithm, and check basic properties.

or

Develop an idea that motivates you.

or

...

You give a 25-minute presentation about what you found.

## Exam examples

Presentations about articles:

- ▶ *Progress, Justness, and Fairness* (2019) by R. van Glabbeek, P. Höfner
- ▶ *Comparison of Model Checking Tools for Information Systems* (2010) by M. Frappier et al. (focusing on [SPIN](#) and [NuSMV](#))

# References I



Christl Baier and Joost-Pieter Katoen.

*Principles of Model Checking.*

MIT Press, 2008.

Available at:

[https://is.ifmo.ru/books/\\_principles\\_of\\_model\\_checking.pdf](https://is.ifmo.ru/books/_principles_of_model_checking.pdf).



Edmund M. Clarke and E. Allen Emerson.

Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic.

In Dexter Kozen, editor, *Logics of Programs*, pages 52–71, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg.

# References II



M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott.

*All About Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic.*

Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007.



Peter Liggesmeyer, Martin Rothfelder, Michael Rettelbach, and Thomas Ackermann.

Qualitätssicherung Software-Basierter Technischer Systeme – Problembereiche und Lösungsansätze.

*Informatik-Spektrum*, (21):249–258, 1998.

# References III



J. P. Queille and J. Sifakis.

Specification and Verification of Concurrent Systems in CESAR.

In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors,  
*International Symposium on Programming*, pages 337–351, Berlin,  
Heidelberg, 1982. Springer Berlin Heidelberg.