

Core Course Preview

Introduction to Model Checking

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

Clemens Grabmayer

<https://clegra.github.io>

Emilio Tuosto

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

Department of Computer Science



December 11, 2025

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.*

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**)

Model Checking

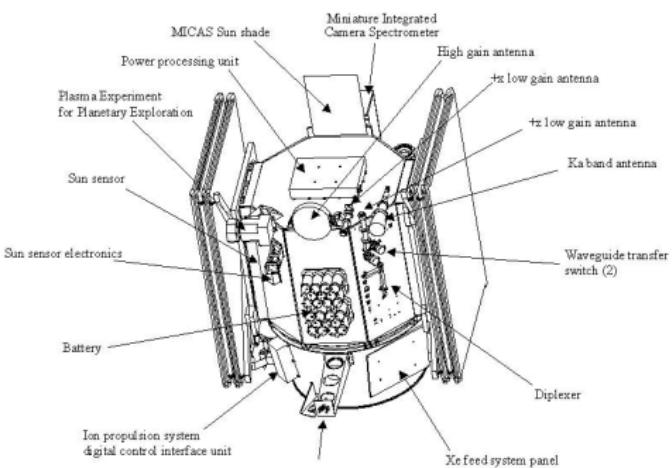
... is an effective automatable technique:

- ▶ *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: use the **Maude** system for examples

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 true
    do
      if x < 200
        then x := x + 1
      fi
    od
```

```
proc Dec
  while true
    do
      if x > 0
        then x := x - 1
      fi
    od
```

```
proc Reset
  while true
    do
      if x = 200
        then x := 0
      fi
    od
```

Example (program concurrency/non-determinism)

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

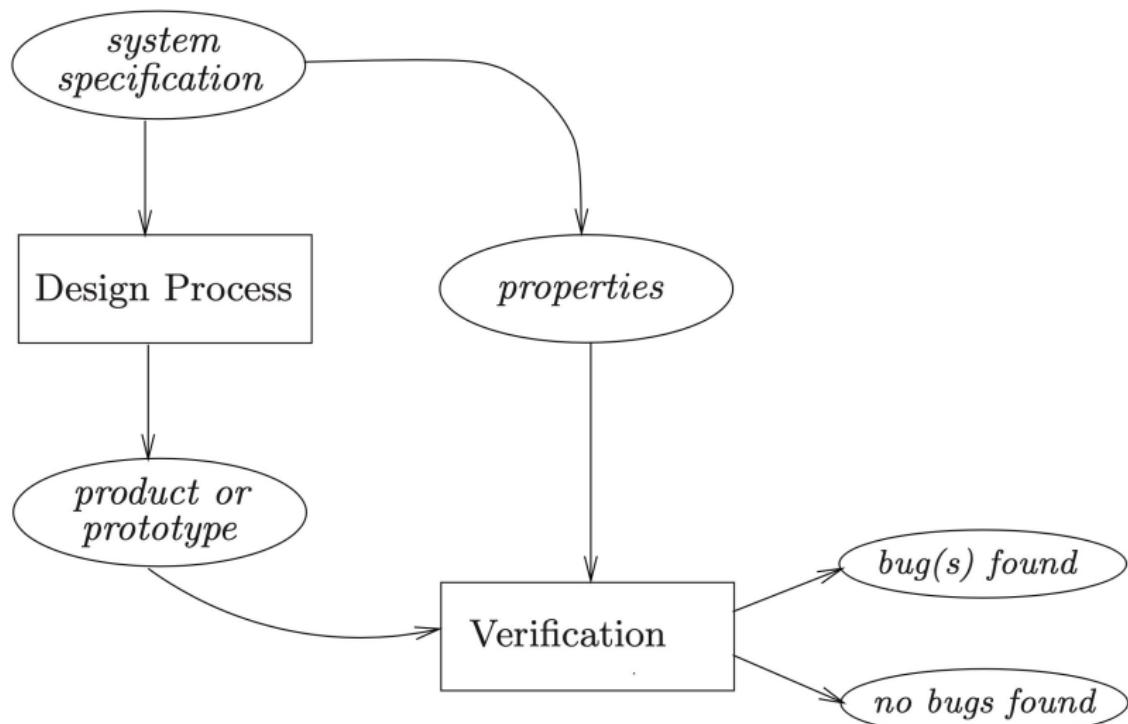
```
proc Inc
  while true
    do
      if x < 200
        then x := x + 1
      fi
    od
```

```
proc Dec
  while true
    do
      if x > 0
        then x := x - 1
      fi
    od
```

```
proc Reset
  while true
    do
      if x = 200
        then x := 0
      fi
    od
```

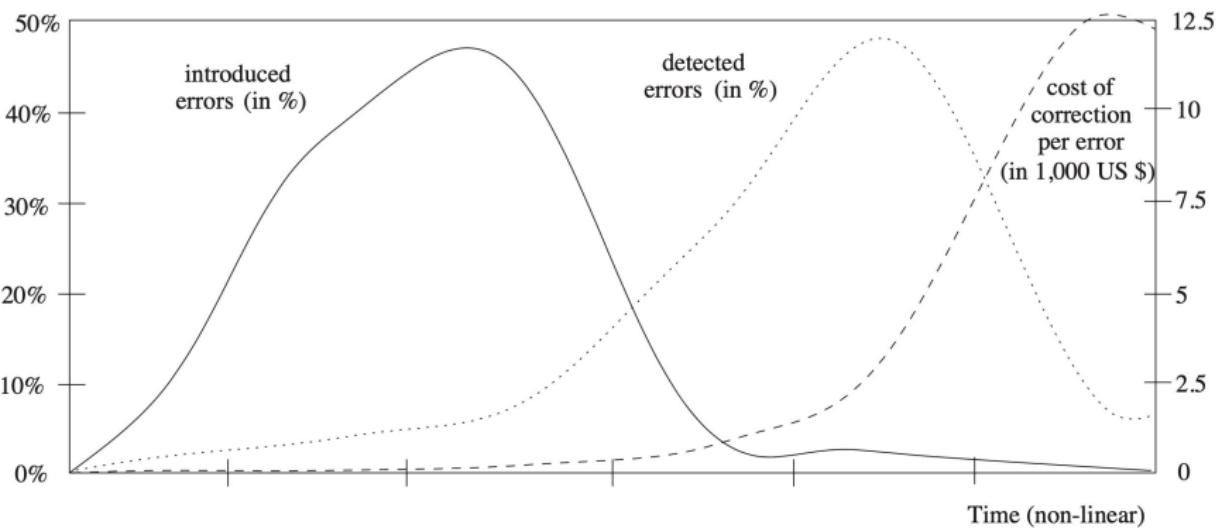
Question: Is $0 \leq x \leq 200$ always guaranteed?

Hard-/Software Verification (traditionally)

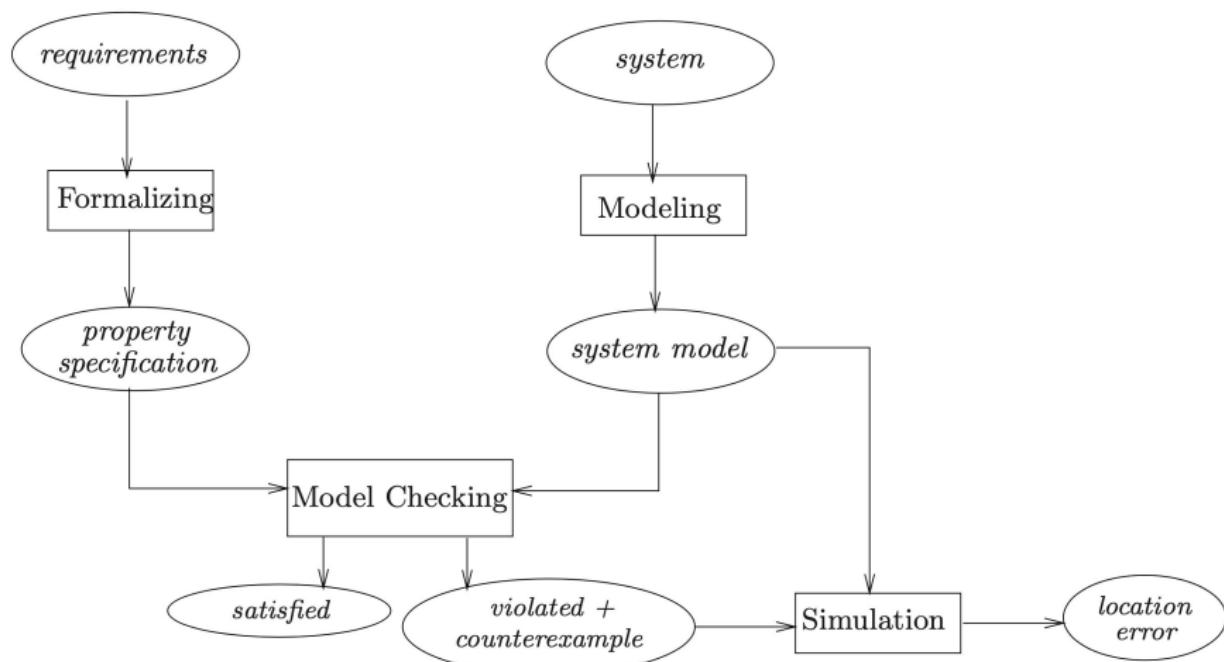


Error introduction, detection, and repair costs

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



Model checking



Modeling (by program graphs)

proc Inc

while true

do

if $x < 200$

then $x := x + 1$

fi

od

proc Dec

while true

do

if $x > 0$

then $x := x - 1$

fi

od

proc Reset

while true

do

if $x = 200$

then $x := 0$

fi

od

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

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
```

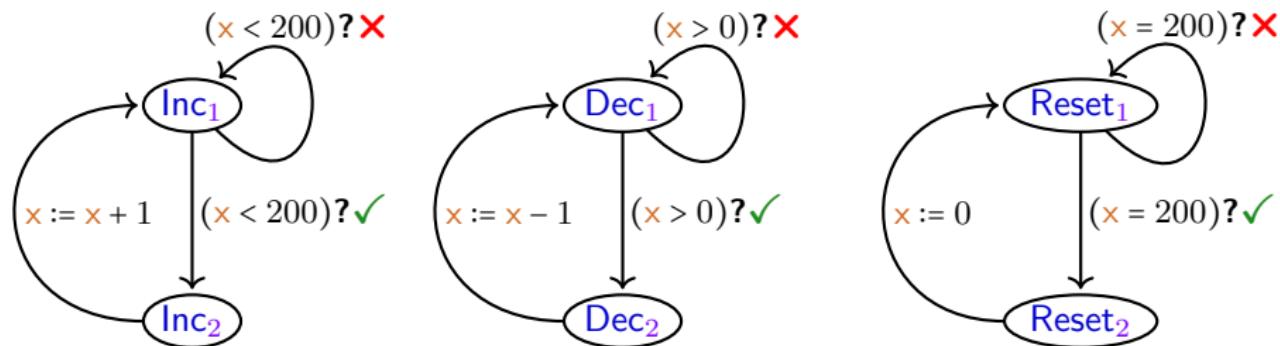
```
    do
```

```
1:      if  $x = 200$ 
```

```
2:      then  $x := 0$ 
```

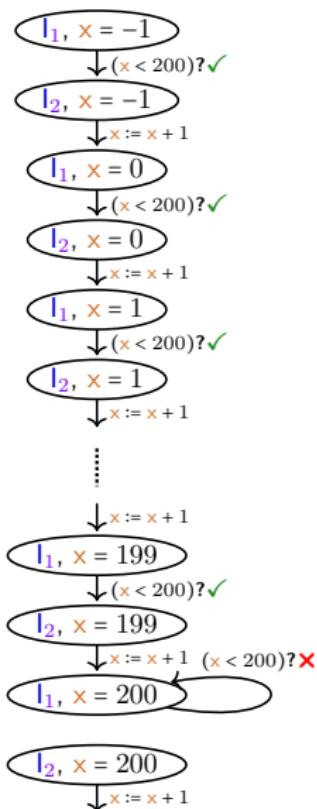
```
    fi
```

```
  od
```

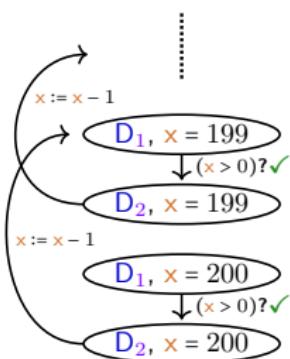
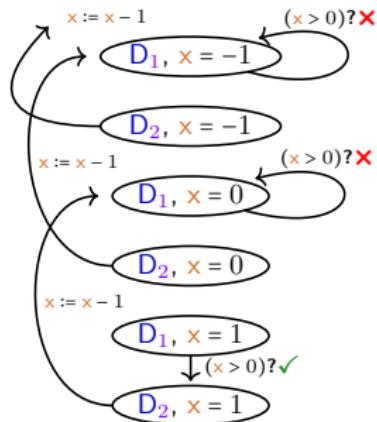


Program graphs (PG)

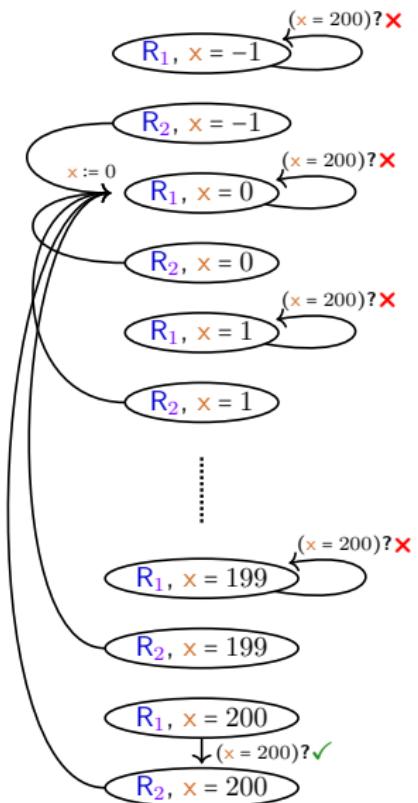
Modeling (by labeled transition systems, with state space explosion)



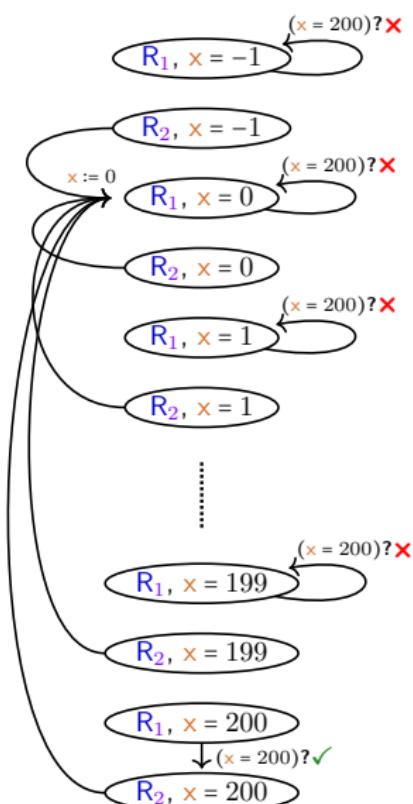
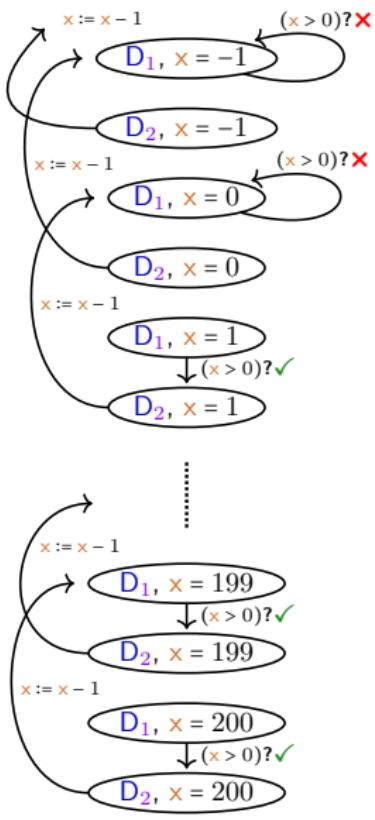
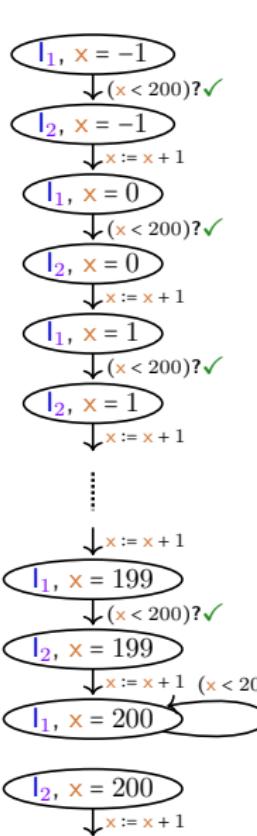
Modeling (by labeled transition systems, with state space explosion)



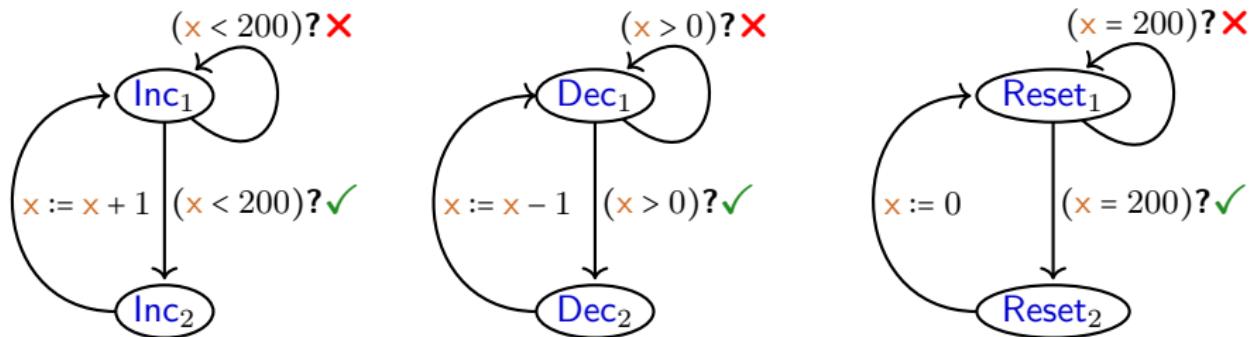
Modeling (by labeled transition systems, with state space explosion)



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



Formalizing properties (in temporal logic)



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

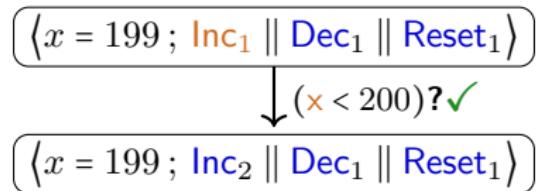
Counterexample (offending execution trace)

$$\langle x = 199 ; \text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \rangle$$

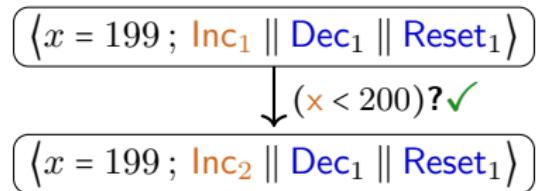
Counterexample (offending execution trace)

$$\langle x = 199 ; \text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \rangle$$

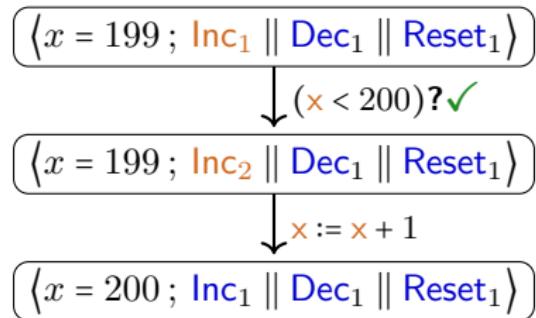
Counterexample (offending execution trace)



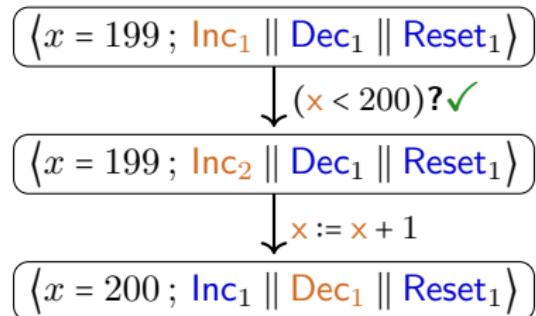
Counterexample (offending execution trace)



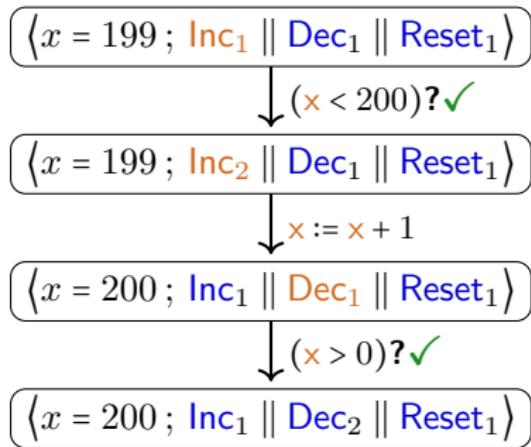
Counterexample (offending execution trace)



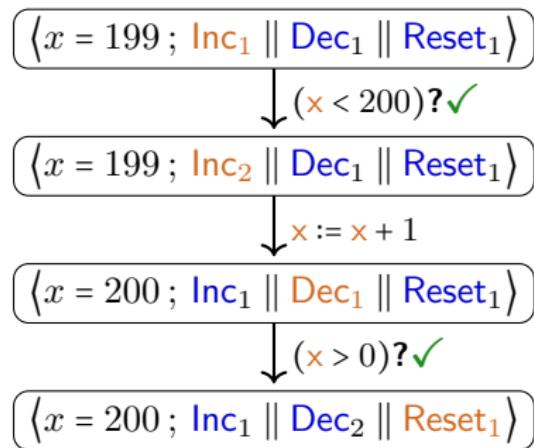
Counterexample (offending execution trace)



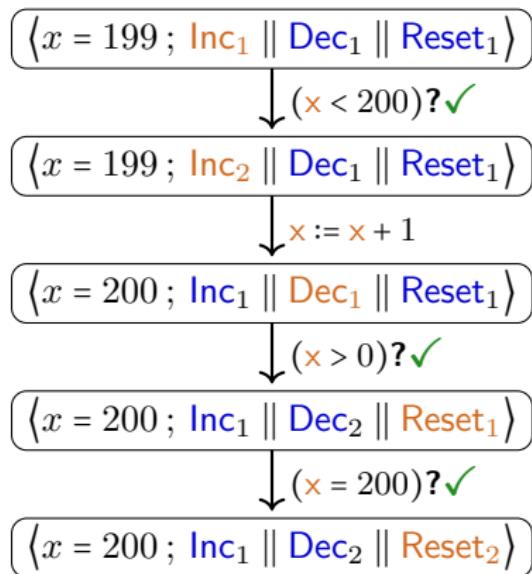
Counterexample (offending execution trace)



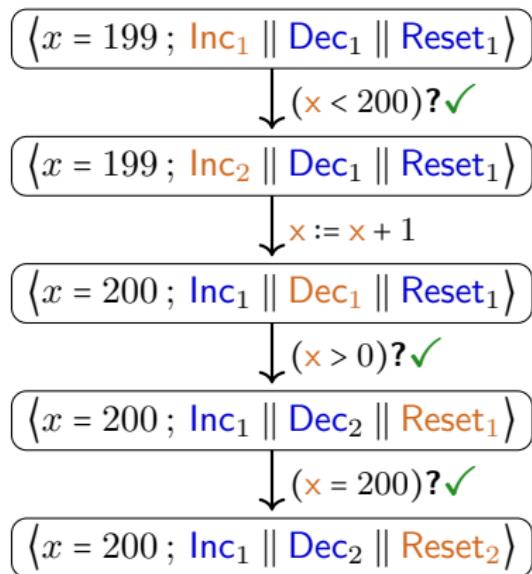
Counterexample (offending execution trace)



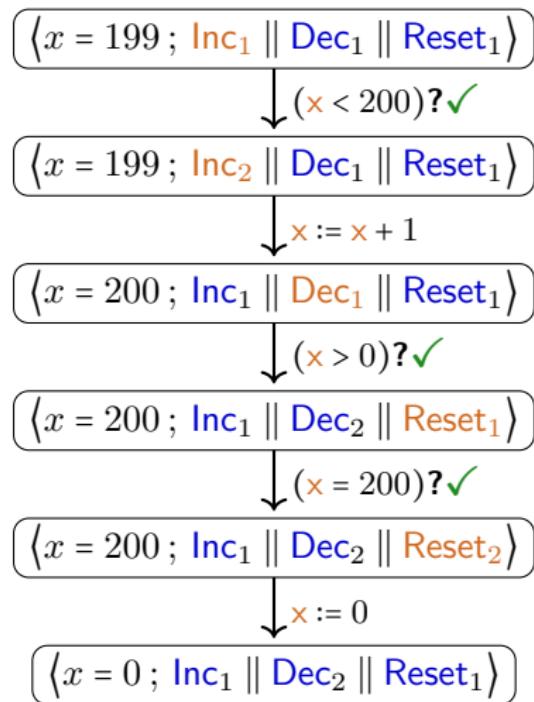
Counterexample (offending execution trace)



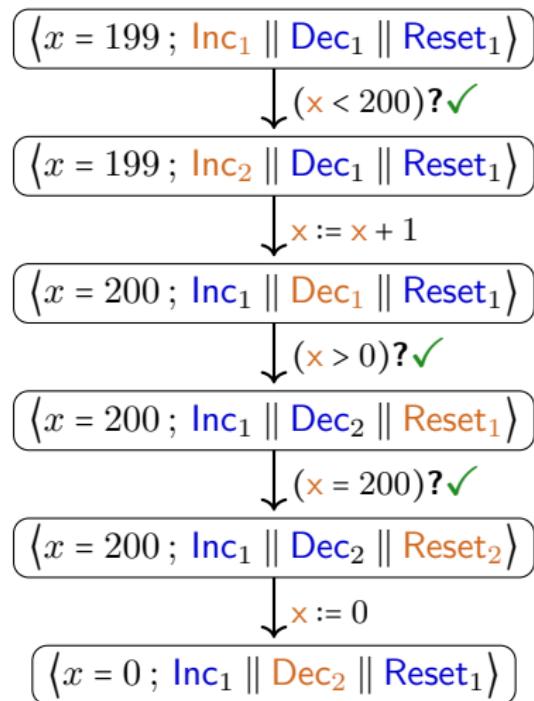
Counterexample (offending execution trace)



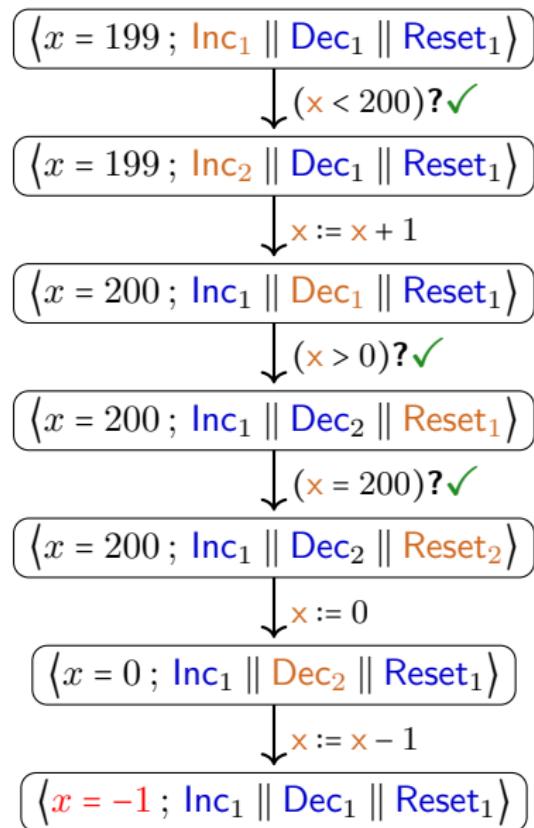
Counterexample (offending execution trace)



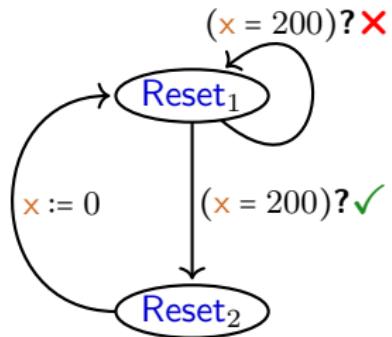
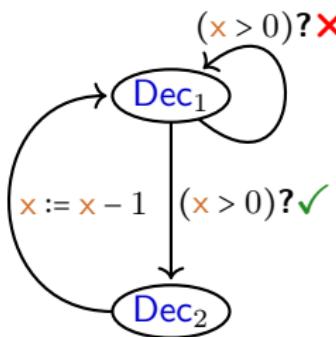
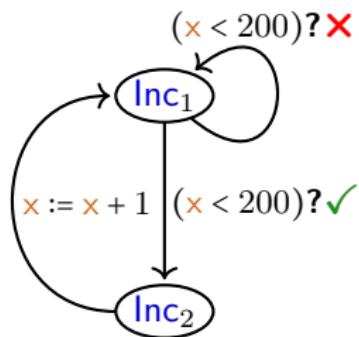
Counterexample (offending execution trace)



Counterexample (offending execution trace)

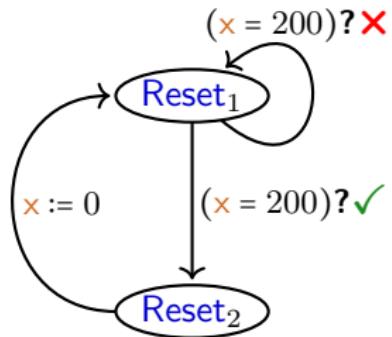
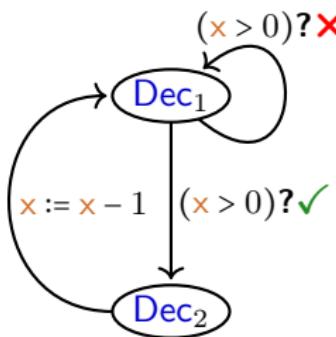
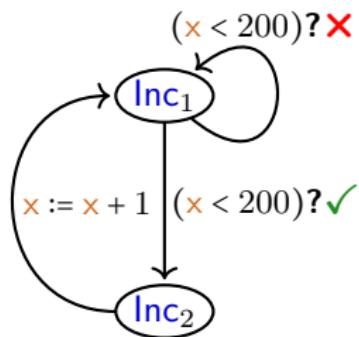


Formalizing properties (in temporal logic)



$\text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \neq \square(0 \leq x \wedge x \leq 200)$ (Linear-TL formula)

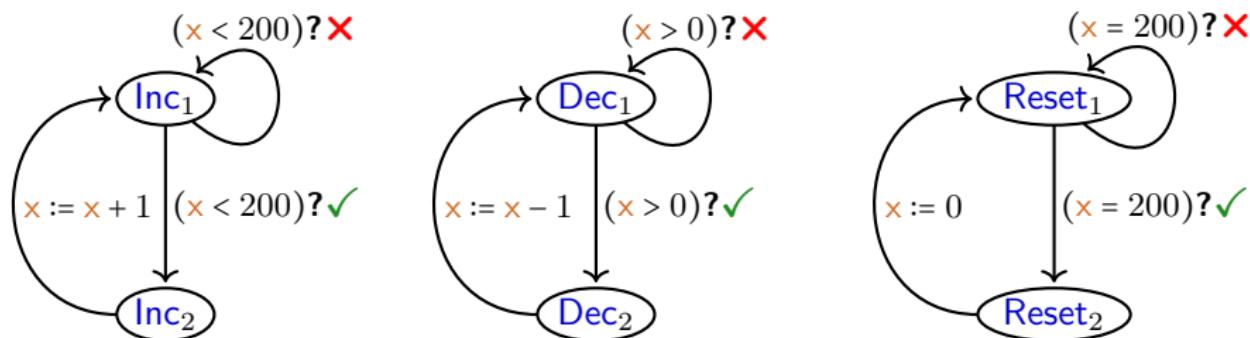
Formalizing properties (in temporal logic)



$\text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \not\models \Box(0 \leq x \wedge x \leq 200)$ (Linear-TL formula)

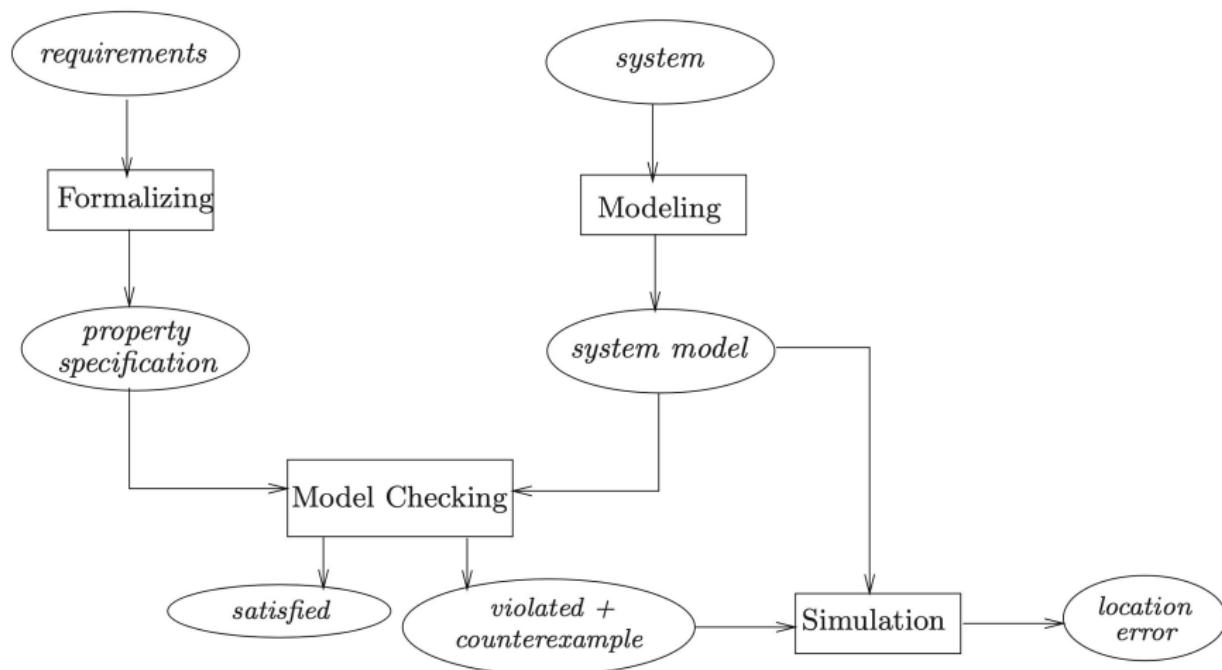
$\text{Inc}_1 \parallel \text{Dec}_1 \parallel \text{Reset}_1 \models \Diamond(x < 0)$ (LTL formula)

Formalizing properties (in temporal logic)



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



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

Maude code (idea)

```
crl [Inc1a]    : Inc1 x => Inc2 x  if x < 200  
rl  [Inc2]     : Inc2 x => Inc1 (x + 1)  
crl [Inc1b]    : Inc1 x => Inc1 x  if not(x < 200)
```

Maude code (idea)

```
crl [Inc1a]    : Inc1 x => Inc2 x  if x < 200
rl  [Inc2]      : Inc2 x => Inc1 (x + 1)
crl [Inc1b]    : Inc1 x => Inc1 x  if not(x < 200)

crl [Dec1a]    : Dec1 x => Dec2 x  if 0 < x
rl  [Dec2]      : Dec2 x => Dec1 (x - 1)
crl [Dec1b]    : Dec1 x => Dec1 x  if not(0 < x)
```

Maude code (idea)

```
crl [Inc1a]    : Inc1 x => Inc2 x  if x < 200
rl  [Inc2]      : Inc2 x => Inc1 (x + 1)
crl [Inc1b]    : Inc1 x => Inc1 x  if not(x < 200)

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

crl [Reset1a]  : Reset1 x => Reset2 x  if x = 200
rl  [Reset2]    : Reset2 x => Reset1 0
crl [Reset1b]  : Reset1 x => Reset1 x  if not(x = 200)
```

Maude code (idea)

```
crl [Inc1a]    : Inc1 x => Inc2 x  if x < 200
rl  [Inc2]      : Inc2 x => Inc1 (x + 1)
crl [Inc1b]    : Inc1 x => Inc1 x  if not(x < 200)

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

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

eq initial = { Dec1 Inc1 Reset1 199 }
```

Maude code (idea)

```

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

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

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

eq initial = { Dec1 Inc1 Reset1 199 }

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

```

Maude output (simplified)

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

Maude output (simplified)

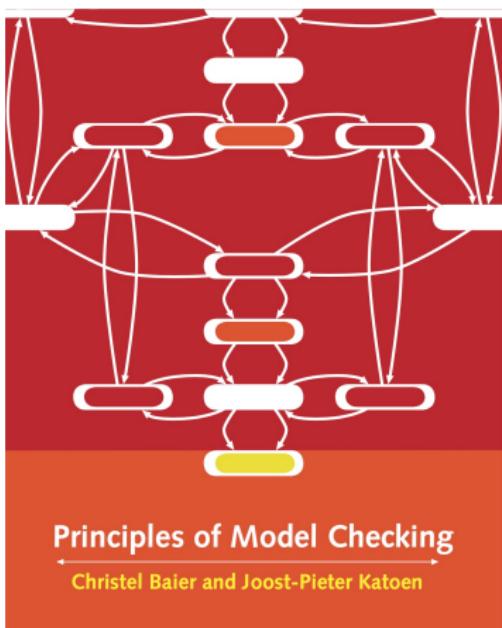
```
Maude> red modelCheck(initial, <> counterlt0)
reduce in COUNTERS-CHECK : modelCheck(initial, <> counterlt0)
result ModelCheckResult:
result Bool : true
```

```
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})
```

Topics of the course

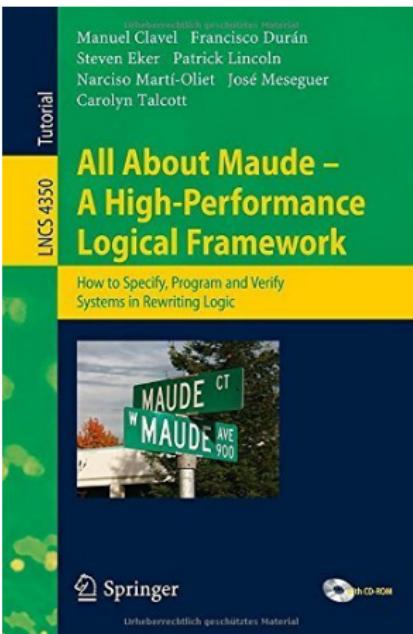
- ▶ modeling systems by labeled transition systems (LTSs)
- ▶ safety, liveness, and fairness properties
- ▶ Linear Temporal Logic (LTL)
 - ▶ model checking formulas
 - ▶ express properties by Büchi automata
 - ▶ model check LTSs and properties via product automata
- ▶ Computation Tree Logic (CTL)
- ▶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

Book



- ▶ pdf available:
https://is.ifmo.ru/books/_principles_of_model_checking.pdf

Book Maude



- ▶ pdf available:

<https://maude.cs.illinois.edu/w/images/0/0d/Maude-book.pdf>

Organization

Lectures (Emilio 3 / Clemens 4)

- ▶ January 19–January 28 (7 meetings)
- ▶ blackboard presentations
- ▶ notes after the lecture (last year's notes available)
- ▶ Maude examples at the end of each lecture

Webpage

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



Organization

Lectures (Emilio 3 / Clemens 4)

- ▶ January 19–January 28 (7 meetings)
- ▶ blackboard presentations
- ▶ notes after the lecture (last year's notes available)
- ▶ Maude examples at the end of each lecture



Webpage

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

Exam

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

Organization

Lectures (Emilio 3 / Clemens 4)

- ▶ January 19–January 28 (7 meetings)
- ▶ blackboard presentations
- ▶ notes after the lecture (last year's notes available)
- ▶ Maude examples at the end of each lecture



Webpage

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

Exam

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

Thank you – we are looking forward to the course!