Further Topics in Real-time Model Checking

Monday Oct 05, 2015

Philipp Rümmer
Uppsala University
Philipp.Ruemmer@it.uu.se

Plan

- Today:
  - Some further techniques and important aspects of *timed automata*
- Wednesday:
  - Towards verification of *software* programs

Exam

- October 21st, 8:00 – 13:00
- **If you want to participate, register now!**

- (You also need to bring
  ID card/passport to the exam)

Recap: Timed automata (TA)

- A *Timed Automaton* is a finite graph with
  - finitely many vertices/locations V, labeled with
    - ⋆ location invariants,
  - initial location $l_0 \in V$,
  - finitely many edges $E \subseteq V \times V$, labeled with
    - ⋆ guards,
    - ⋆ actions,
    - ⋆ assignments.
- *Guards*: Clock constraint or predicate over data variables
  - Clock constraint: $g ::= x \leqslant n \mid x \geqslant n \mid x < n \mid x > n \mid g \wedge g$
  - Data predicate: "any logical expression" you would write in C
- *Actions*: On channels for synchronization
  - `a?` or `a!`
- *Assignments*: Clock reset or data variable assignment
  - `x:=0`, `i:=i+1`
- *Location Invariants:* Clock constraints, like guards

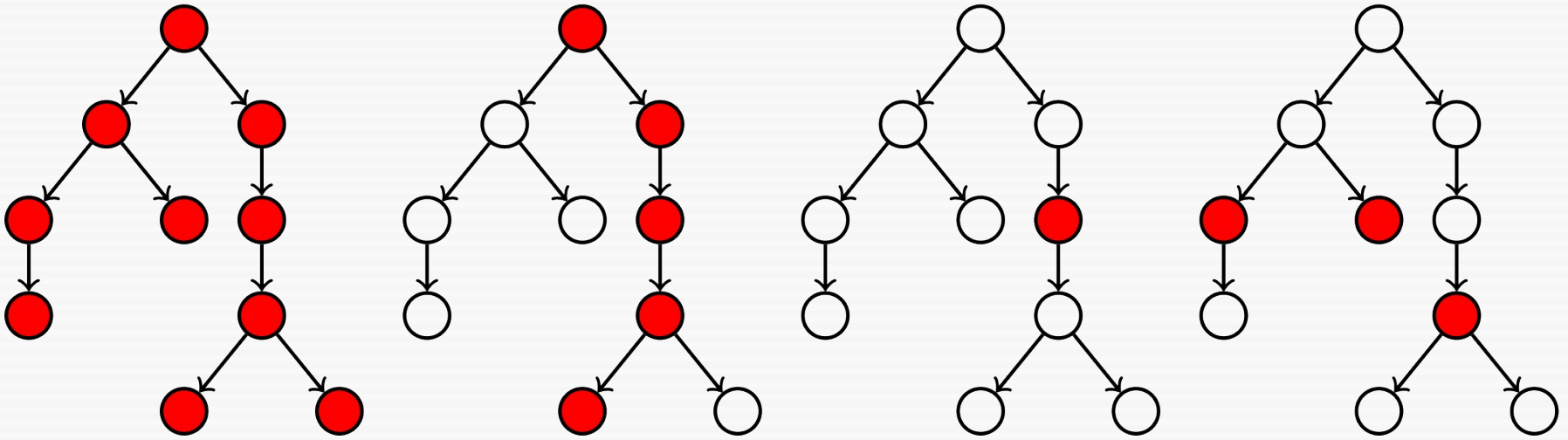
Recap: Properties

- *Propositional* state properties:

```
p ::= A.n | gc | gd | not p | p or p | p and p
                   | p imply p
```

`A.n`: Automaton A is in location n
`gc`: Clock constraint, e.g., $x \geqslant 2$
`gd`: Data predicate, e.g., $i == 100$
Rest: Logical connectives (boolean)

`A[] p` `E[] p` `E<> p` `A<> p`



Some further Uppaal features

- Urgent states
- Committed states
- Broadcast channels
- Datatypes (arrays, int, bounded int)
- `select` expressions on transitions
- Leads-to query operator -->

Urgent/committed

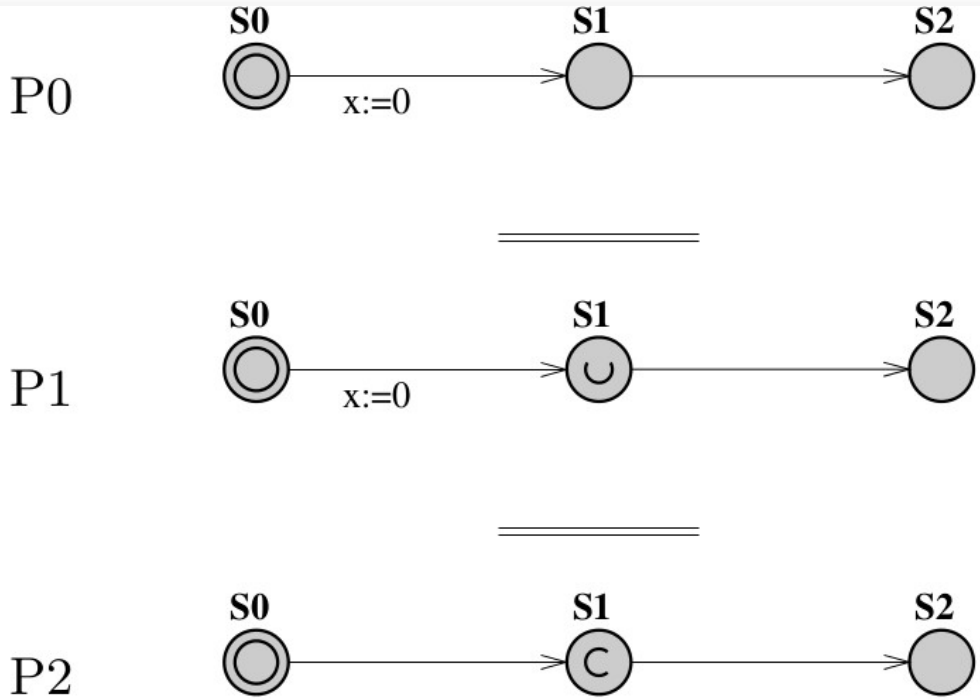


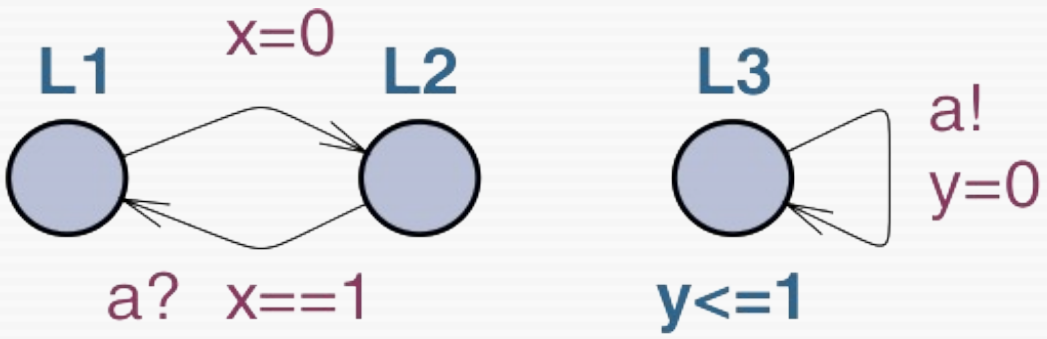
Fig. 8. Automata in parallel with normal, urgent and commit states. The clocks are local, i.e., `P0.x` and `P1.x` are two different clocks.

Anomalies in timed automata

- Not all TA are well-formed;
  There are a few classes of typical **modelling flaws**
- TA with such flaws can have very surprising behaviour
  → To be avoided in models!

- **Timelocks**
- **Zeno-behaviour**

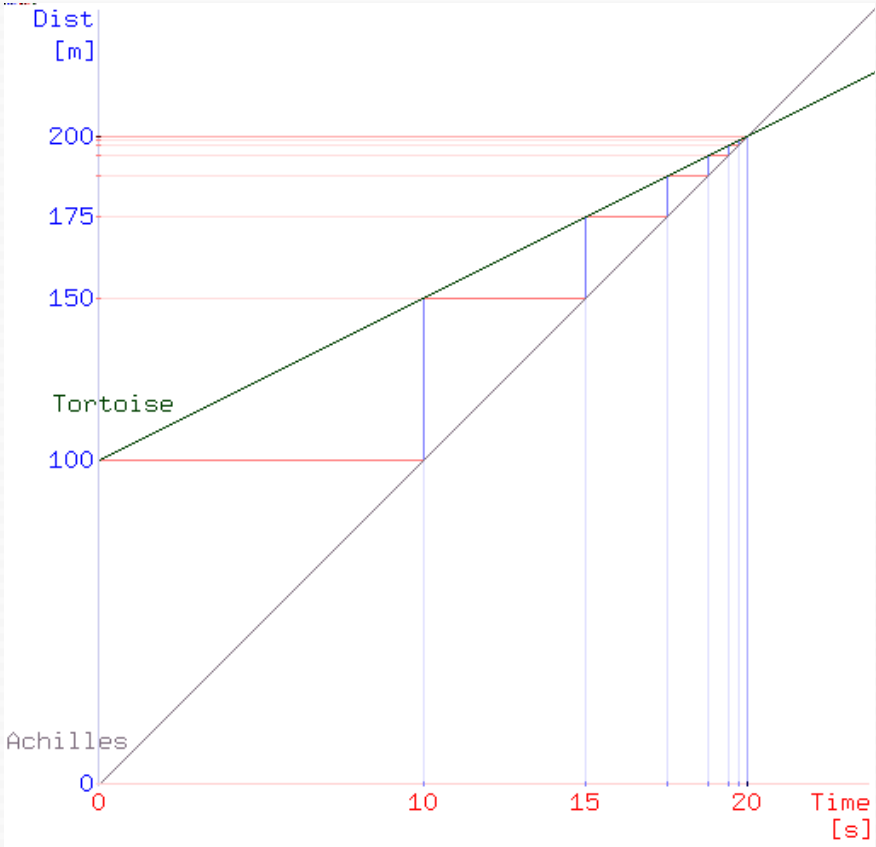
Timelocks



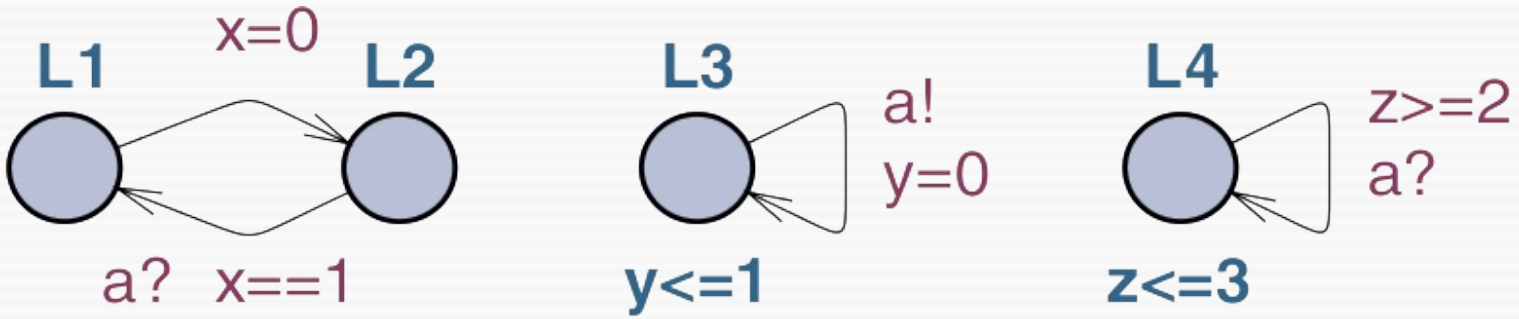
- A state is reachable in which time can no longer pass
  (if transition L1 → L2 occurs too late, synchronisation with L3 not possible)
- TA in which timelocks are reachable are **ill-formed**

Zeno's paradox

In the paradox of Achilles and the Tortoise, Achilles is in a footrace with the tortoise. Achilles allows the tortoise a head start of 100 meters, for example. If we suppose that each racer starts running at some constant speed (one very fast and one very slow), then after some finite time, Achilles will have run 100 meters, bringing him to the tortoise's starting point. During this time, the tortoise has run a much shorter distance, say, 10 meters. It will then take Achilles some further time to run that distance, by which time the tortoise will have advanced farther; and then more time still to reach this third point, while the tortoise moves ahead. Thus, whenever Achilles reaches somewhere the tortoise has been, he still has farther to go. Therefore, because there are an infinite number of points Achilles must reach where the tortoise has already been, he can never overtake the tortoise.
[Zeno of Elea (490 – 430 BC); Wikipedia]



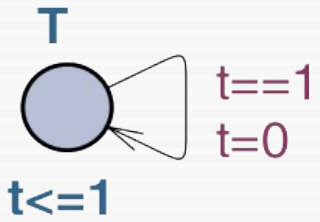
Zeno in timed automata



- Zeno-trace:
  Trace with infinitely many discrete transitions, but only finitely much time passes
  (trace is **time-convergent**)
- Again, in well-formed TA all traces are non-Zeno

How to find anomalies? (In Uppaal)

- Add a new clock `t` and a test automaton to system:



- Check the liveness property t==0 --> t==1
- This property only holds in well-formed models

Further reading

- “Efficient Detection of Zeno Runs in Timed Automata”, Rodolfo Gomez and Howard Bowman, 2007

Safety properties

- How to express
  (in vending machine example):

  “A product is delivered only if some money was inserted beforehand”

General techniques

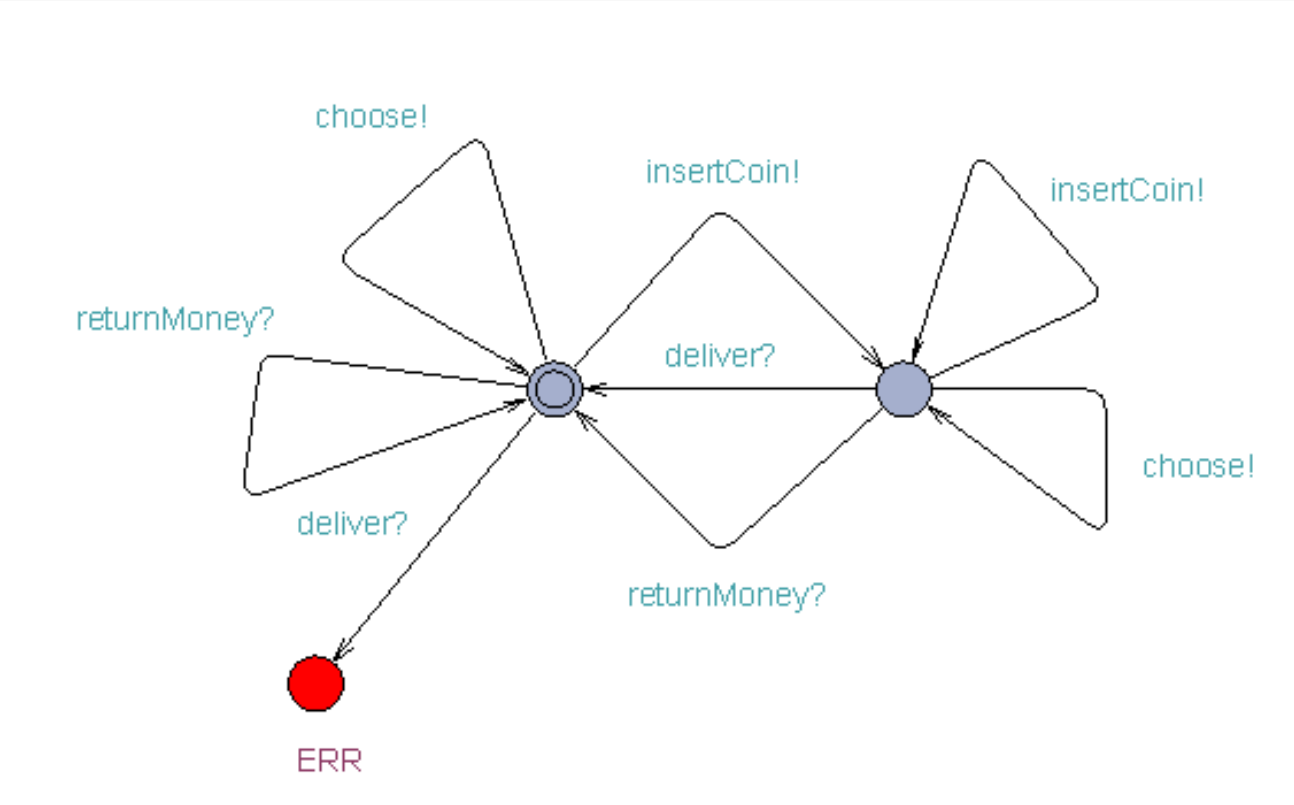
- **Observer/Monitor/Test automaton**
  - TA that checks occurrence of undesired behaviour, by observing a model
- **Instrumentation**
  - Make important information observable
  - Add further variables/updates/actions to a system

Safety in general

- **Not all safety properties can be captured just using `A[]`**
  - `A[] P` describes **invariance:**
    P invariantly holds during execution
  - Special case of safety
- What are safety properties in general?
- How can such properties be checked?

Informally:
“Something bad can never happen”

What happened in vending machine example?



→ A finite automaton over the characters insertCoin!, choose!, deliver?, returnMoney?

Error traces

- Sequences of states that describe scenarios considered as errors
  - E.g., “Product delivered without any money inserted”
  - Applicable also for actions/conditions
- Set of such traces forms a **language** L_{err}
  - Characters = states
  - Words = traces

Safety properties

Safety property =
any property that can be described by a language of **finite** error traces

System *S* is **safe** =
no execution trace of *S* has a prefix in L_{err}

In contrast: Liveness

- “Something good will happen eventually”
- E.g., *“after inserting money and choosing a product, eventually something will be delivered”*

- Traces demonstrating violation of liveness are always **infinite**

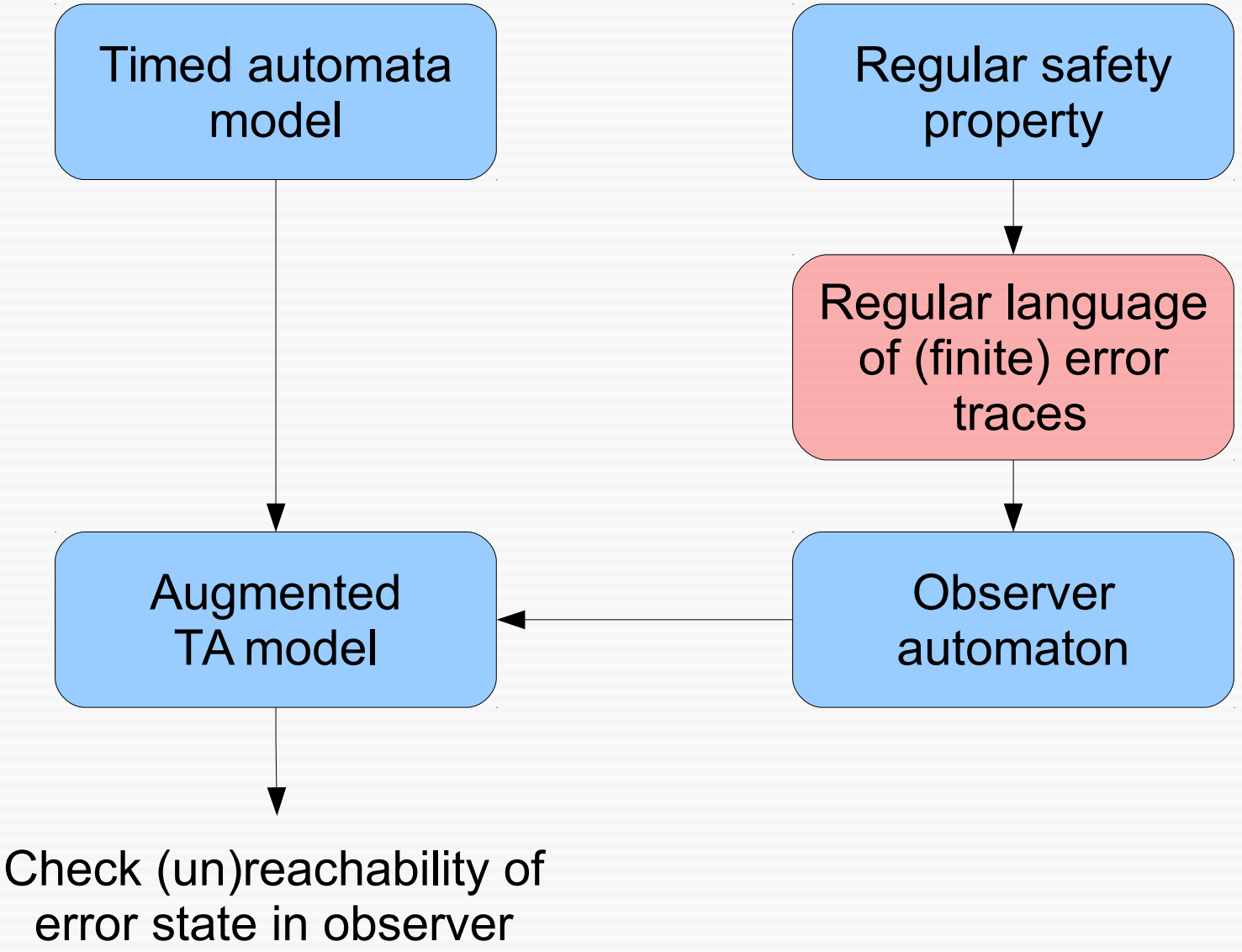
Regular safety

- Important special case:
  Language L_{err} is **regular**
    - Can be described by finite automaton or regular expression
- Finite automaton can be added as observer checking occurrence of errors

  → Reduction of **safety** to **invariance**

  (error location of observer is unreachable)

Big picture


Timed automata model
Regular safety property
Regular language of (finite) error traces
Augmented TA model
Observer automaton
Check (un)reachability of error state in observer

Can this also be done for liveness?

- Yes ...
- But needs a different kind of observer automaton, since error traces for liveness are infinite
    - e.g.,
      Büchi automata over infinite words

Example

- Improved safety property:

  “A product is delivered only after some money has been inserted and a product has been chosen”

Possible system properties

- Uppaal queries:
  - Reachability properties `E<>`
  - Safety/invariance properties `A[]`, `E[]`
  - Liveness properties `A<>`, `-->`
  - (Time-bounded liveness → safety)
- General safety properties
- General temporal logic properties
  - CTL, TCTL, LTL, … *many more*
- Simulation, bisimulation