An Introduction to Timed Automata

Time is of the essence

Gaëtan Staquet
gaetan.staquet@ec-nantes.fr
https://www.gaetanstaquet.com/research/schools.html

École Centrale de Nantes – LS2N

8th September 2026

Contents

General context

Some systems are **critical**.

- A plane ought to not crash.
- Two trains cannot be in the same place at the same time.
- *Ingenuity* (*Perseverance*'s helicopeter) should be able to fly at least 30 seconds every week.

General context

Some systems are **critical**.

- A plane ought to not crash.
- Two trains cannot be in the same place at the same time.
- *Ingenuity* (*Perseverance*'s helicopeter) should be able to fly at least 30 seconds every week.

Manually testing such systems is unfeasible.

$\rightsquigarrow$ **Testing** (unit, integration, and so on).

General context

Some systems are **critical**.

- A plane ought to not crash.
- Two trains cannot be in the same place at the same time.
- *Ingenuity* (*Perseverance*'s helicopeter) should be able to fly at least 30 seconds every week.

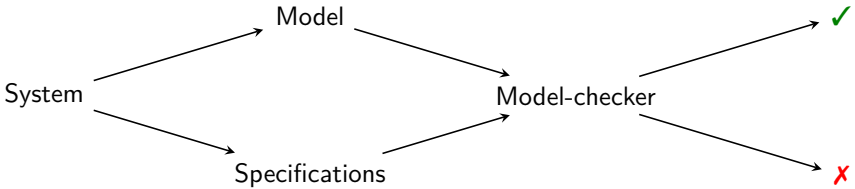
Manually testing such systems is unfeasible.

$\rightsquigarrow$ **Testing** (unit, integration, and so on).

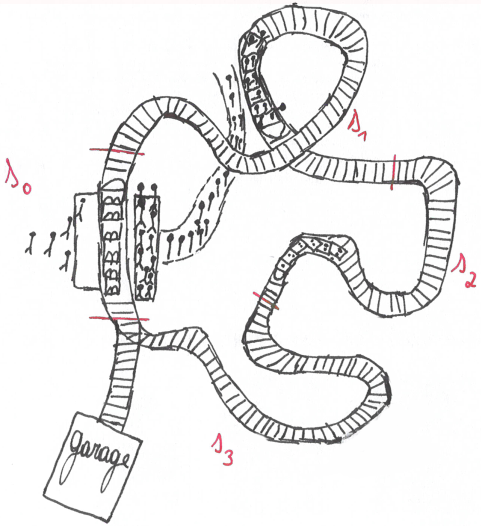
But we may miss some cases...

$\rightsquigarrow$ **Model-checking!**

Model-checking in a nutshell

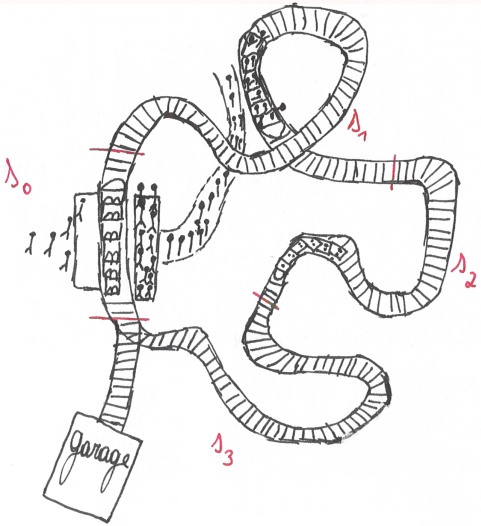


Our running-example: an attraction



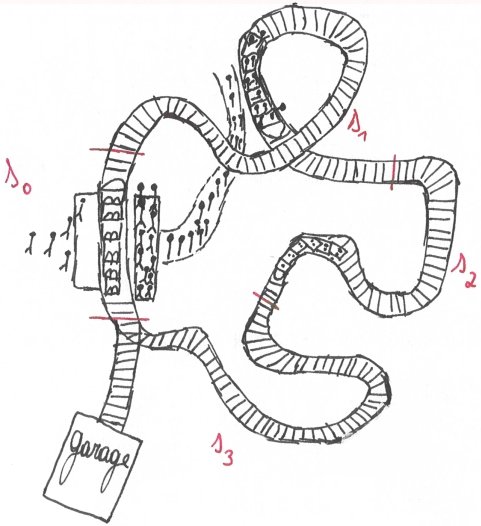
- Two components of the system: the **station** and the **trains**.

Our running-example: an attraction



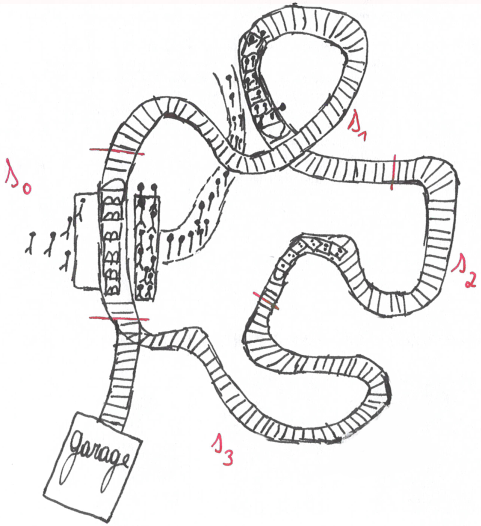
- Two components of the system: the **station** and the **trains**.
- The track has 4 sections: s_0 (in which the station lies), s_1 (the looping), s_2 (a drop), and s_3 (cooling down before the station). There is also a garage with three trains.

Our running-example: an attraction



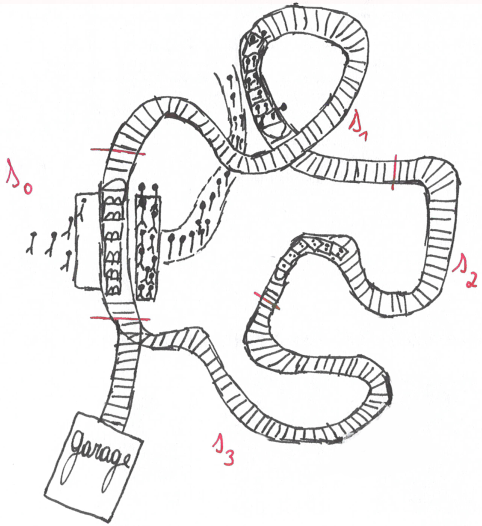
- Two components of the system: the **station** and the **trains**.
- The track has 4 sections: s_0 (in which the station lies), s_1 (the looping), s_2 (a drop), and s_3 (cooling down before the station). There is also a garage with three trains.
- The **station** manages when a train leaves the garage, and when a train leaves the station.

Our running-example: an attraction



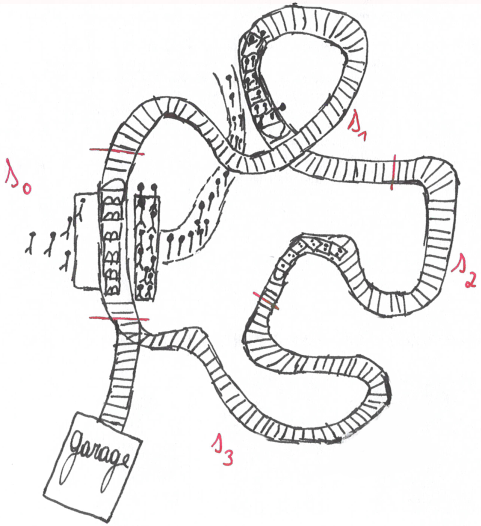
- Two components of the system: the **station** and the **trains**.
- The track has 4 sections: s_0 (in which the station lies), s_1 (the looping), s_2 (a drop), and s_3 (cooling down before the station). There is also a garage with three trains.
- The **station** manages when a train leaves the garage, and when a train leaves the station.
- All trains share the exact same behavior.

Our running-example: an attraction



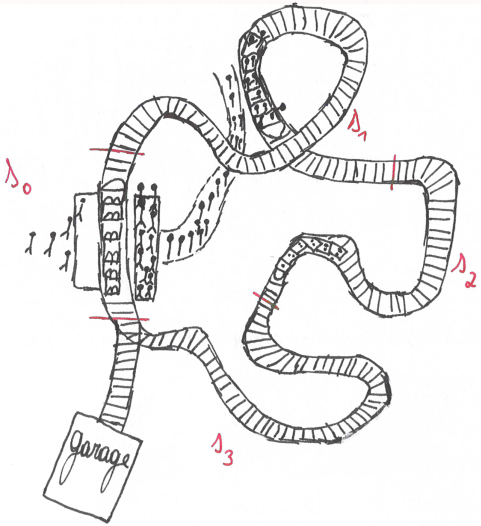
- Two components of the system: the **station** and the **trains**.
- The track has 4 sections: s_0 (in which the station lies), s_1 (the looping), s_2 (a drop), and s_3 (cooling down before the station). There is also a garage with three trains.
- The **station** manages when a train leaves the garage, and when a train leaves the station.
- All trains share the exact same behavior.
- Timing constraints:
  - Time for a train in s_0: in $(3, 4]$.
  - Time in s_1: 2.
  - Time in s_2: 3.
  - Time in s_3: in $(2, 5]$.

Our running example: properties



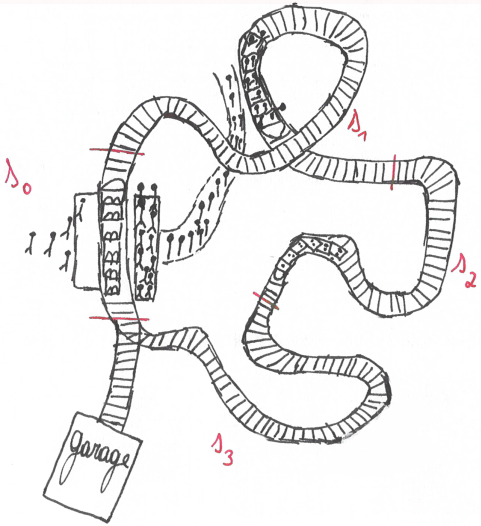
- Two trains can never be in the same section at the same time.

Our running example: properties



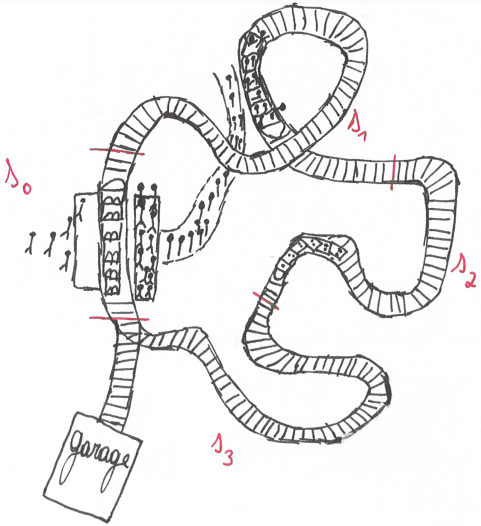
- Two trains can never be in the same section at the same time.
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward.

Our running example: properties



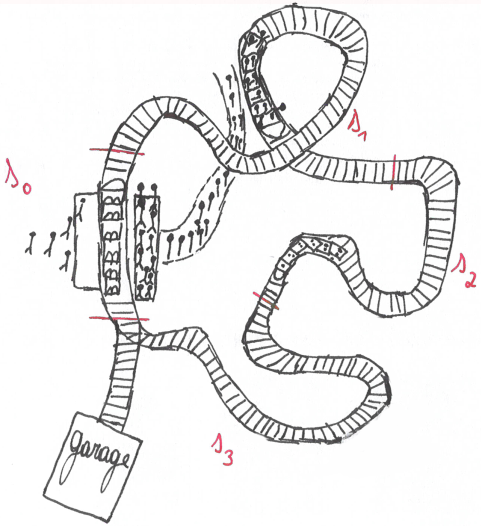
- Two trains can never be in the same section at the same time.
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward.
- It must be possible to have one train running.

Our running example: properties



- Two trains can never be in the same section at the same time.
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward.
- It must be possible to have one train running.
- It must be possible to have two trains running.

Our running example: properties



- Two trains can never be in the same section at the same time.
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward.
- It must be possible to have one train running.
- It must be possible to have two trains running.
- It must be possible to have three trains running.

Symbols, alphabets, and words

- A **symbol** can be anything.
  - $a, b, c, 0, 1, \dagger$ are all symbols.

Symbols, alphabets, and words

- A **symbol** can be anything.
- An **alphabet** is a finite, non-empty set of symbols.

- $a, b, c, 0, 1, \dagger$ are all symbols.
- $\{a, b, 0\}$ and $\{c, 1\}$ are two alphabets.

Symbols, alphabets, and words

- A **symbol** can be anything.
- An **alphabet** is a finite, non-empty set of symbols.
- A **word** over an alphabet Σ is a finite sequence of symbols from Σ.

- $a, b, c, 0, 1, \dagger$ are all symbols.
- $\{a, b, 0\}$ and $\{c, 1\}$ are two alphabets.
- $aab0b$ is a word over $\{a, b, 0\}$; $1cc1$ is a word over $\{c, 1\}$.

Symbols, alphabets, and words

- A **symbol** can be anything.
- An **alphabet** is a finite, non-empty set of symbols.
- A **word** over an alphabet Σ is a finite sequence of symbols from Σ.
- The **empty word** is written ε.

- $a, b, c, 0, 1, \dagger$ are all symbols.
- $\{a, b, 0\}$ and $\{c, 1\}$ are two alphabets.
- $aab0b$ is a word over $\{a, b, 0\}$; $1cc1$ is a word over $\{c, 1\}$.

Symbols, alphabets, and words

- A **symbol** can be anything.
- An **alphabet** is a finite, non-empty set of symbols.
- A **word** over an alphabet Σ is a finite sequence of symbols from Σ.
- The **empty word** is written ε.
- For an alphabet Σ, we write Σ^* for the set of all words over Σ.

- $a, b, c, 0, 1, \dagger$ are all symbols.
- $\{a, b, 0\}$ and $\{c, 1\}$ are two alphabets.
- $aab0b$ is a word over $\{a, b, 0\}$; $1cc1$ is a word over $\{c, 1\}$.
- $\{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \dots\}$.

A first family of models: finite automata

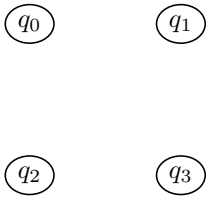
Definition 1 (Finite automata). A **finite automaton** (**FA**) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ with

- Σ an **alphabet**;

A first family of models: finite automata

Definition 1 (Finite automata). A **finite automaton** (**FA**) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ with

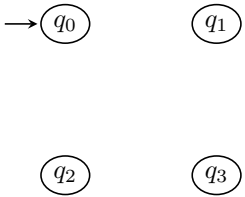
- Σ an **alphabet**;
- Q a non-empty finite set of **states**;



A first family of models: finite automata

Definition 1 (Finite automata). A **finite automaton** (**FA**) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ with

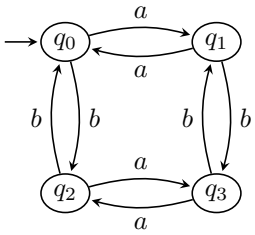
- Σ an **alphabet**;
- Q a non-empty finite set of **states**;
- $q_0 \in Q$ the **initial state**;



A first family of models: finite automata

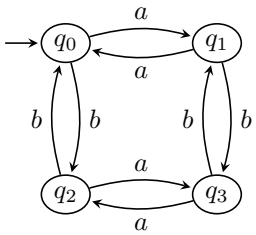
Definition 1 (Finite automata). A **finite automaton** (**FA**) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ with

- Σ an **alphabet**;
- Q a non-empty finite set of **states**;
- $q_0 \in Q$ the **initial state**;
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ the **transition function** (or transition relation).



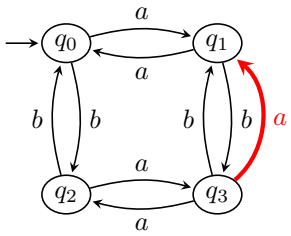
Finite automata and runs

- $q_0 \xrightarrow{a} q_1$ is a **transition** of the FA because $\delta(q_0, a) \ni \{q_1\}$
- A **run** is a finite sequence of transitions: $q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_3 \xrightarrow{a} \xrightarrow{a} q_3$ is a run.
- We write $q_0 \xrightarrow{abaa} q_3$ as a shorthand.



Nondeterministic transitions

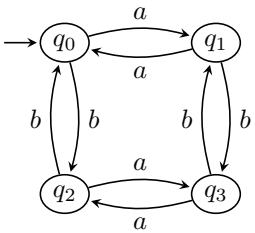
- From q_3, there are two choices when reading a: $q_3 \xrightarrow{a} q_2$ or $q_3 \xrightarrow{a} q_1$.
- So, $q_0 \xrightarrow{abaa} q_3$ or $q_0 \xrightarrow{abaa} q_0$ are two valid runs, reading the same word.
- We say that the transition and the FA are **nondeterministic**.
- Examples and exercices here will be over deterministic automata, but keep in mind that nondeterminism can appear.



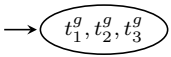
Finite automata and properties

Giving properties over the system $\Leftrightarrow$ giving properties over the runs.

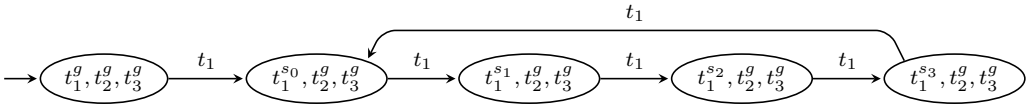
- For every run over a word w, the run ends in q_1 if and only if the number of a in w is odd and the number of b is even.
- Every run can be continued, *i.e.*, we do not have a **deadlock**.



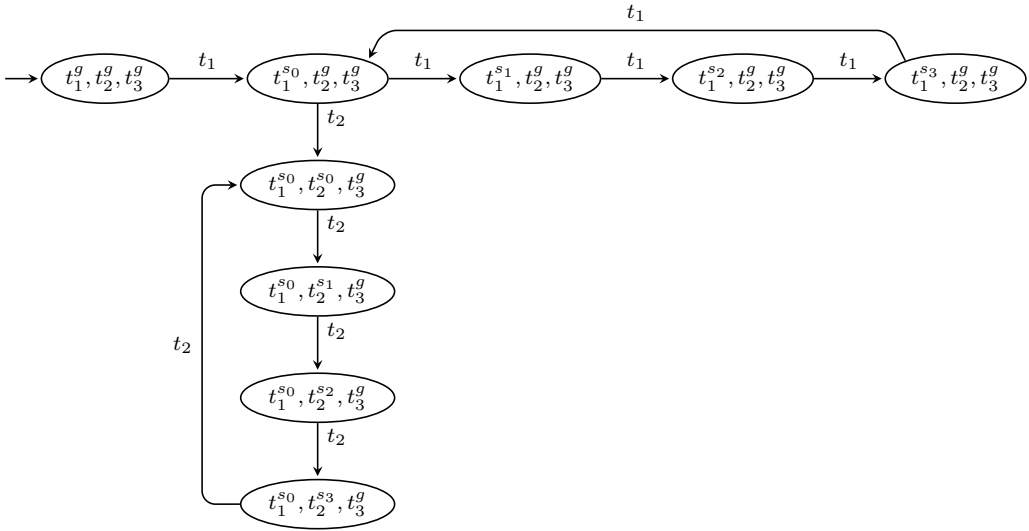
A first model of our attraction



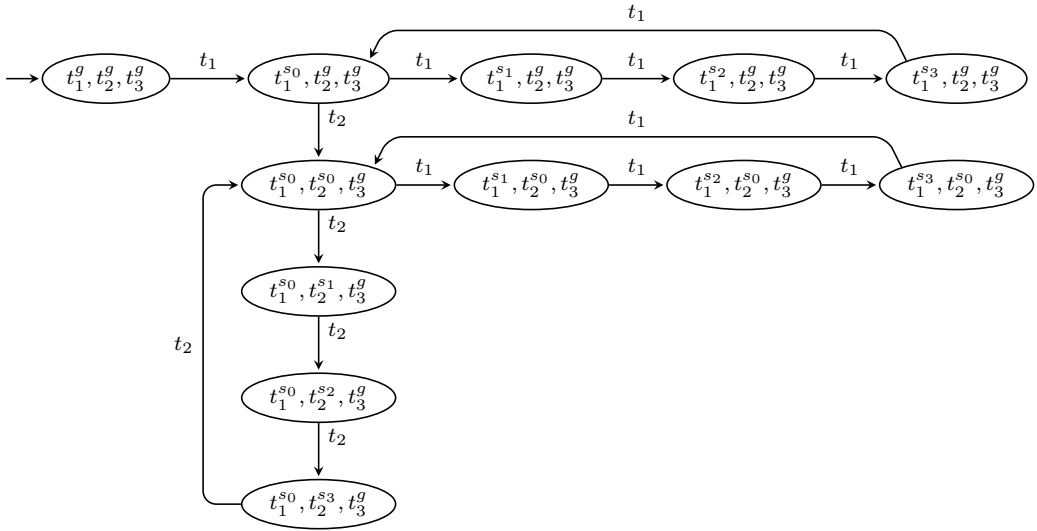
A first model of our attraction



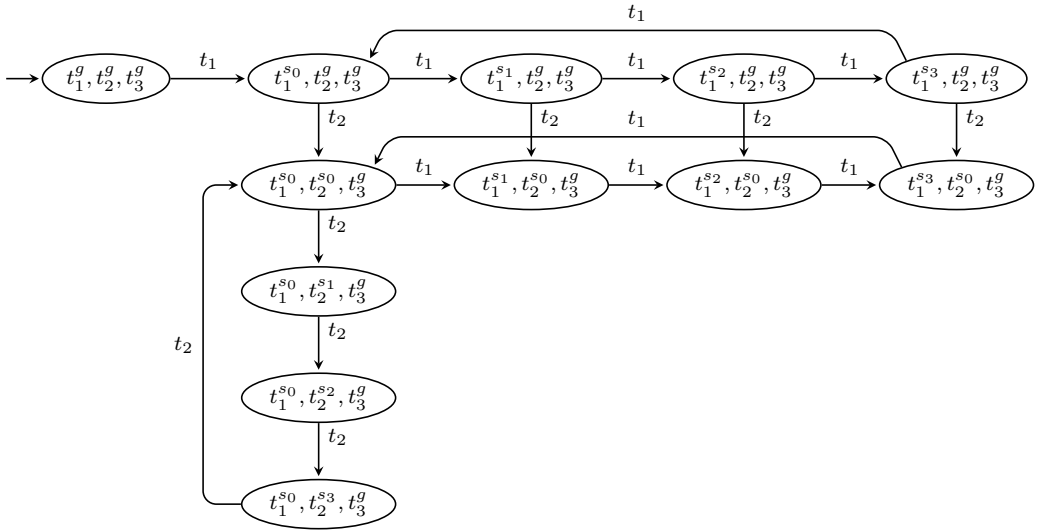
A first model of our attraction



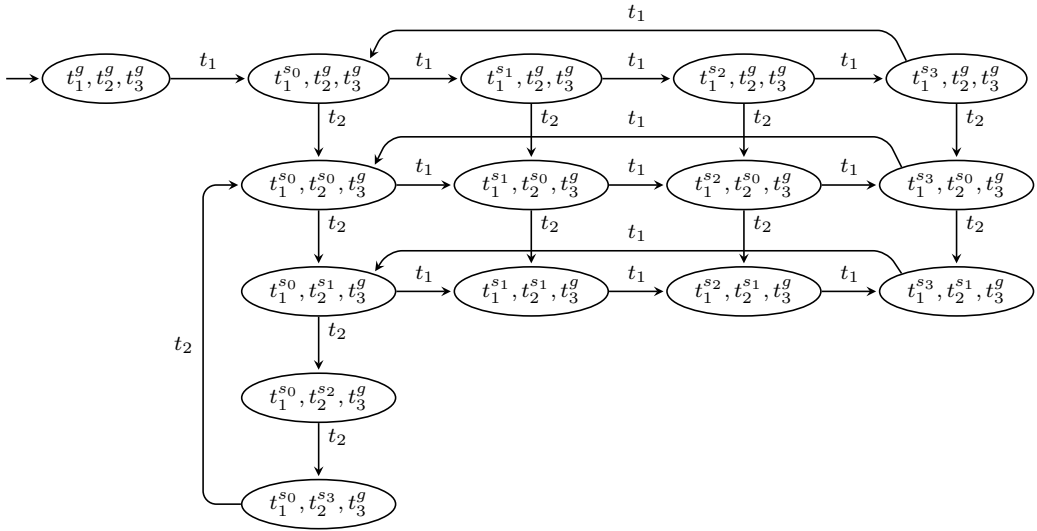
A first model of our attraction



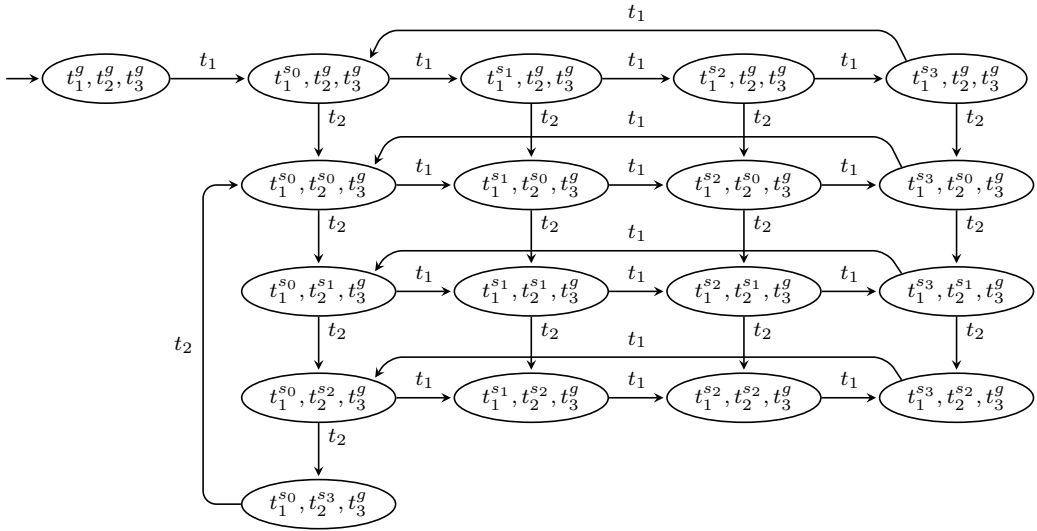
A first model of our attraction



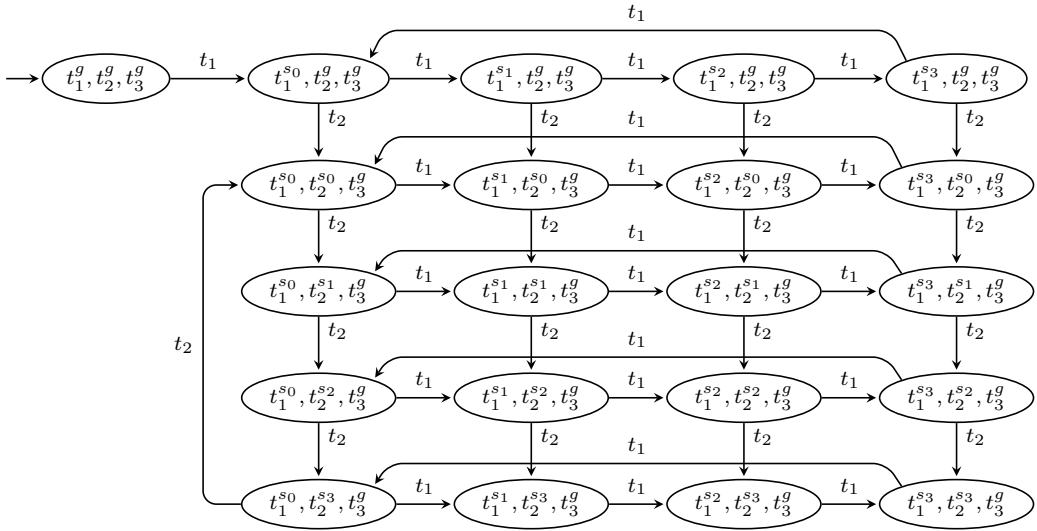
A first model of our attraction



A first model of our attraction



A first model of our attraction



Way too messy...

STOP

Way too messy...

STOP

- It is getting incomprehensible...

Way too messy...

STOP

- It is getting incomprehensible...
- And we are still missing transitions. In particular, t_3 cannot move yet.

Way too messy...

STOP

- It is getting incomprehensible...
- And we are still missing transitions. In particular, t_3 cannot move yet.
- And we only focused on the trains; we still have to consider the station.

Way too messy...

STOP

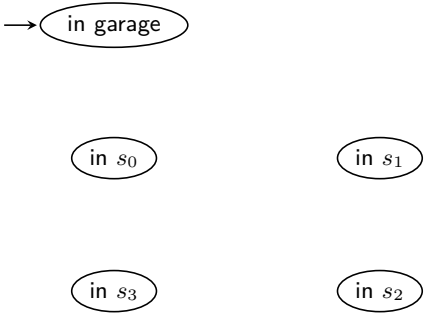
- It is getting incomprehensible...
- And we are still missing transitions. In particular, t_3 cannot move yet.
- And we only focused on the trains; we still have to consider the station.

Creating a single big automaton for the whole system is not a good idea.

Let's exploit **compositionality**!

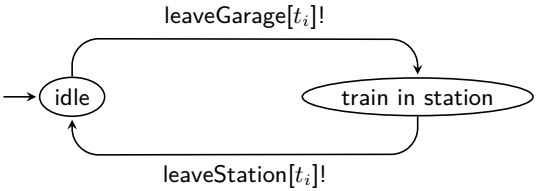
Station and trains

Model for the **station**.

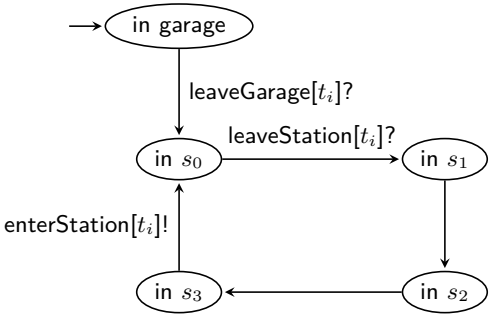


Model for the **train** t_i.

Station and trains



Model for the **station**.

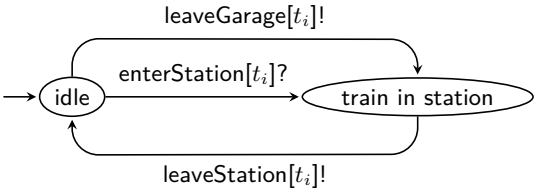


Model for the **train** t_i.

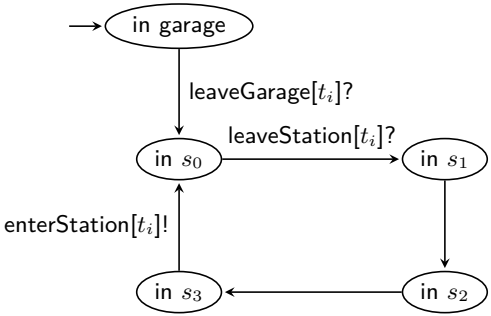
The symbols $s!$ and $s?$ are used to **synchronize** components of the system:

- $s!$ sends the symbol s;
- $s?$ waits for the symbol s.

Station and trains



Model for the **station**.

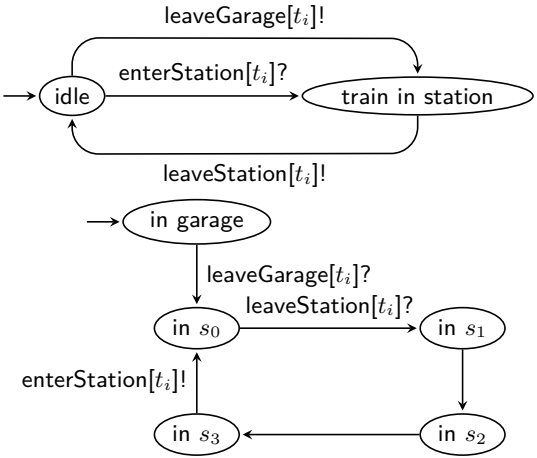


Model for the **train** t_i.

The symbols $s!$ and $s?$ are used to **synchronize** components of the system:

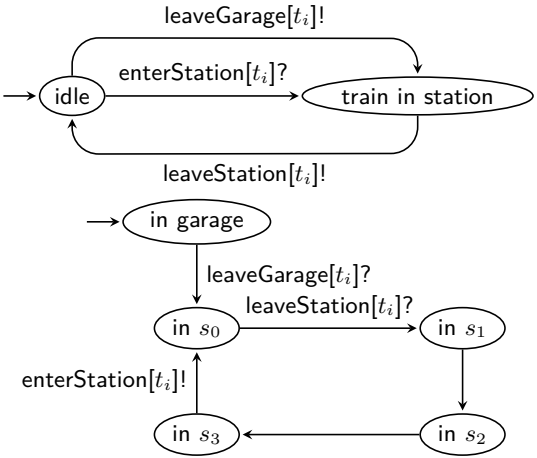
- $s!$ sends the symbol s;
- $s?$ waits for the symbol s.

Station and trains – Properties



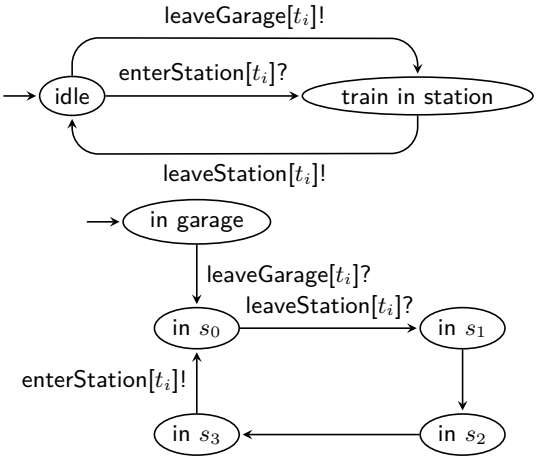
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward.

Station and trains – Properties



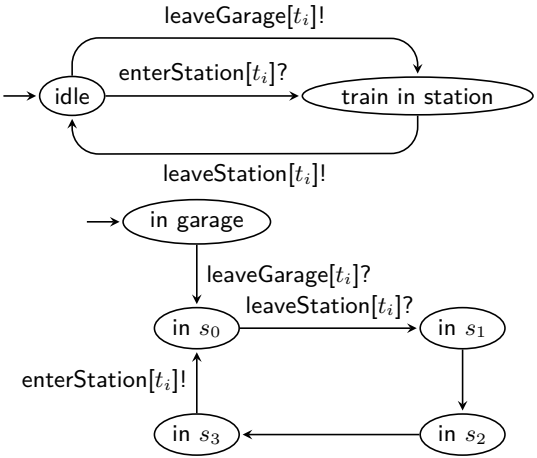
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓

Station and trains – Properties



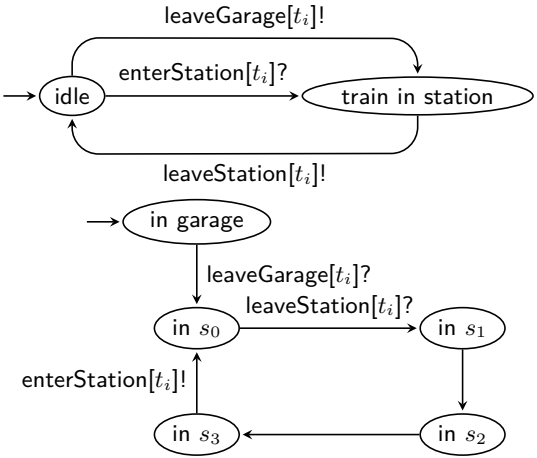
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running.

Station and trains – Properties



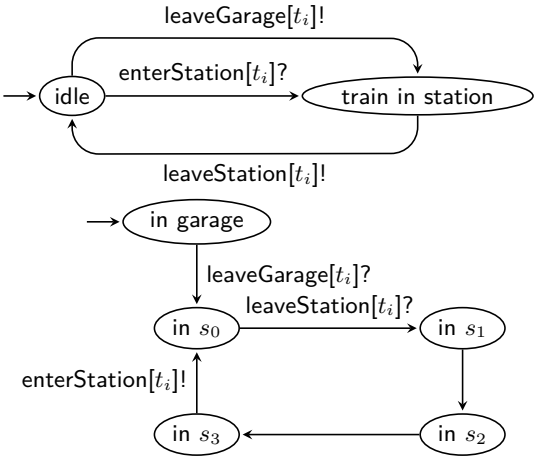
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓

Station and trains – Properties



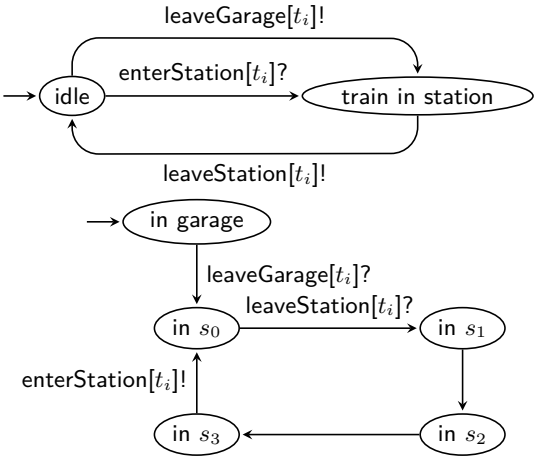
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓
- It must be possible to have two trains running.

Station and trains – Properties



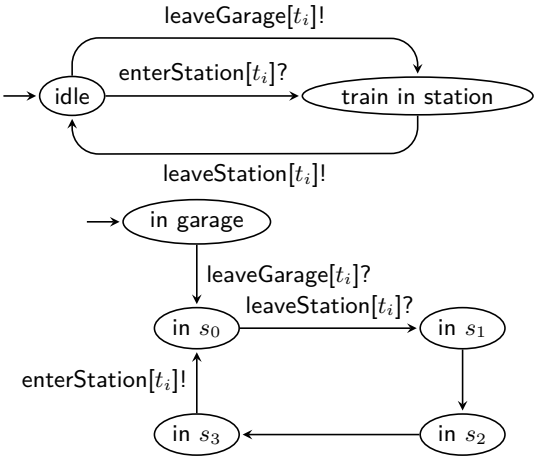
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓
- It must be possible to have two trains running. ✓

Station and trains – Properties



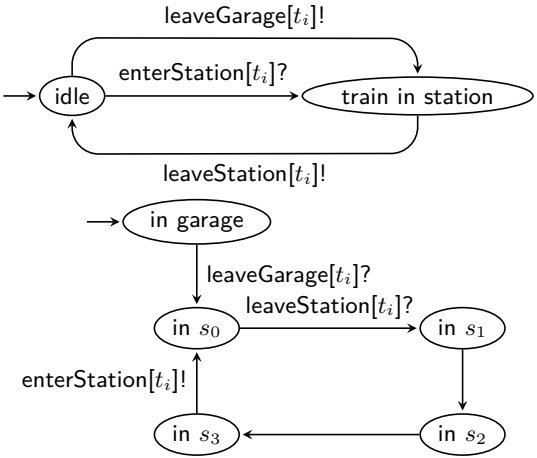
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓
- It must be possible to have two trains running. ✓
- It must be possible to have three trains running.

Station and trains – Properties



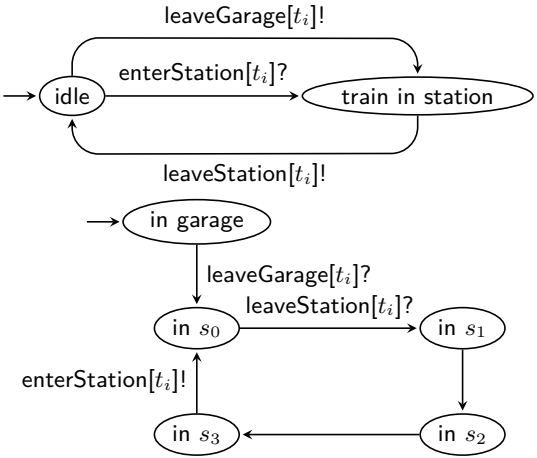
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓
- It must be possible to have two trains running. ✓
- It must be possible to have three trains running. ✓

Station and trains – Properties



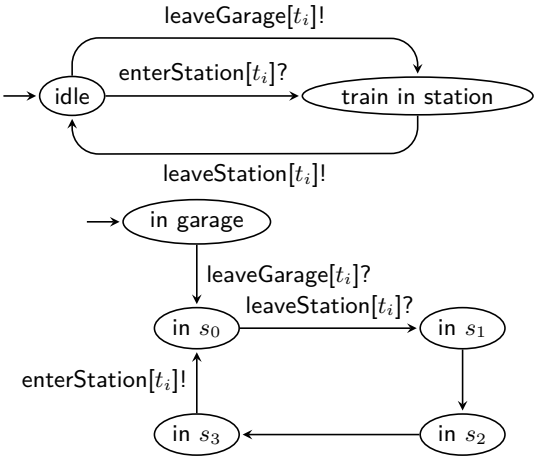
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓
- It must be possible to have two trains running. ✓
- It must be possible to have three trains running. ✓
- Two trains can never be in the same section at the same time.

Station and trains – Properties



- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓
- It must be possible to have two trains running. ✓
- It must be possible to have three trains running. ✓
- Two trains can never be in the same section at the same time. ✗

There is still hope



- There is a part of the system's specifications we ignored so far!
- Timing constraints:
  - Time for a train in s_0: in $(3, 4]$.
  - Time in s_1: 2.
  - Time in s_2: 3.
  - Time in s_3: in $(2, 5]$.
- Finite automata cannot handle time...

Clocks

Let's extend finite automata with a way to take time into account.
Definitions for this section come from Bouyer et al., "Model Checking Real-Time Systems", 2018.

Clocks

Let's extend finite automata with a way to take time into account.
Definitions for this section come from Bouyer et al., "Model Checking Real-Time Systems", 2018.

Definition 2 (Clocks and valuations). A **clock** x is a real-valued variable.
Let C be a set of clocks. A **(clock) valuation** over C is a mapping $\nu : C \to \mathbb{R}^{\geq 0}$, which assigns to each clock a real value.
We write $\mathsf{Val}(C)$ for the set of all valuations over C.
We write $\mathbf{0}_C$ for the valuation assigning 0 to every clock $x \in C$.

Example 3. Let $C = \{x, y\}$.

- $(x = 5, y = 1)$ is a valuation over C.
- $\mathsf{Val}(C) = \{x = k \mid k \in \mathbb{R}^{\geq 0}\} \times \{y = k \mid k \in \mathbb{R}^{\geq 0}\}$.

Clock operations

Operations. A clock x will be used to remember how many **time units elapsed** since the last time x was **set to zero**.

Clock operations

Operations. A clock x will be used to remember how many **time units elapsed** since the last time x was **set to zero**.

Definition 4 (Clock operations). Let C be a set of clocks and $\nu \in \mathrm{Val}(C)$ be a valuation over C.

Elapsing time For every $t \in \mathbb{R}^{\geq 0}$, the valuation $\nu + t$ is defined by

$$\forall x \in C : (\nu + t)(x) = \nu(x) + t.$$

Resetting to zero For every $R \subseteq C$, the valuation $\nu[R]$ is defined by

$$\forall x \in C : \nu[R](x) = \begin{cases} 0 & \text{if } x \in R \\ \nu(x) & \text{if } x \in C \setminus R \end{cases}$$

Clock constraints

Comparisons. We will need to enable or disable behaviors, according to the current clock valuation.

Clock constraints

Comparisons. We will need to enable or disable behaviors, according to the current clock valuation.

Definition 5 (Clock constraints). Let C be a set of clocks. The set $\Phi(C)$ of **clock constraints** over C is defined by

$$\Phi(C) \ni \varphi ::= true \mid x \bowtie k \mid \varphi_1 \wedge \varphi_2 \qquad (x \in C, k \in \mathbb{N}, \bowtie \in \{<, \leq, =, \geq, >\}).$$

Let $\nu \in \mathsf{Val}(C)$ and $\varphi \in \Phi(C)$. We write $\nu \models \varphi$ if

- $\varphi = true$;
- when $\varphi = x \bowtie k$, $\nu(x) \bowtie k$; or
- when $\varphi = \varphi_1 \wedge \varphi_2$, $\nu \models \varphi_1 \wedge \nu \models \varphi_2$.

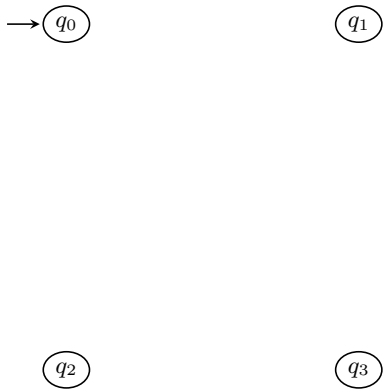
Example 6. Let $C = \{x, y\}$ and $\varphi = (x > 4) \wedge (y \leq 3)$. Then,

- $(x = 4, y = \frac{4}{3}) \not\models \varphi$, as $(x = 4, y = \frac{4}{3}) \not\models (x > 4)$.
- $(x = 4.5, y = \frac{10}{3}) \not\models \varphi$, as $(x = 4.5, y = \frac{10}{3}) \not\models (y \leq 3)$
- $(x = 4.5, y = \frac{4}{3}) \models \varphi$.

Timed automata

Definition 7 (Timed automata). A **timed automaton** (**TA**) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, C, I, \delta)$ with

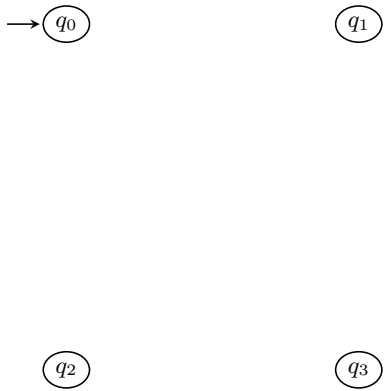
- Σ an **alphabet**;
- Q a non-empty finite set of **states**;
- $q_0 \in Q$ the **initial state**;



Timed automata

Definition 7 (Timed automata). A **timed automaton** (**TA**) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, C, I, \delta)$ with

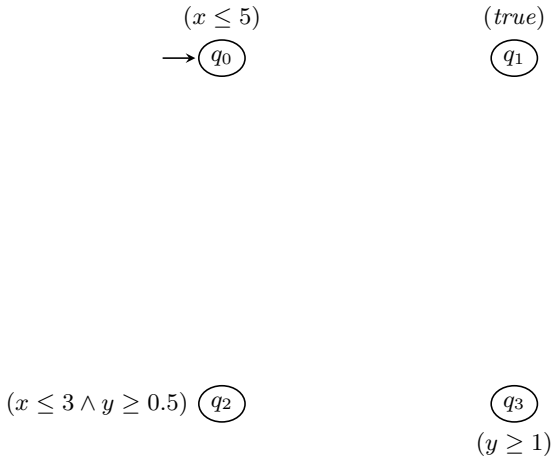
- Σ an **alphabet**;
- Q a non-empty finite set of **states**;
- $q_0 \in Q$ the **initial state**;
- C a set of **clocks**;



Timed automata

Definition 7 (Timed automata). A **timed automaton** (**TA**) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, C, I, \delta)$ with

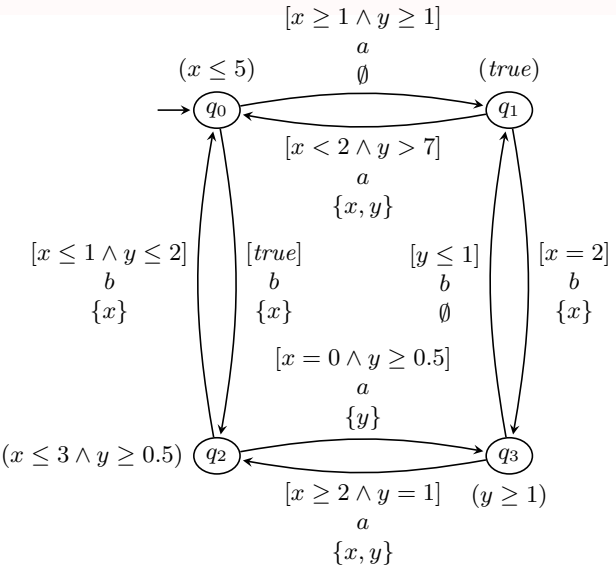
- Σ an **alphabet**;
- Q a non-empty finite set of **states**;
- $q_0 \in Q$ the **initial state**;
- C a set of **clocks**;
- $I : Q \to \Phi(C)$ an **invariant mapping**;



Timed automata

Definition 7 (Timed automata). A **timed automaton (TA)** is a tuple $\mathcal{A} = (\Sigma, Q, q_0, C, I, \delta)$ with

- Σ an **alphabet**;
- Q a non-empty finite set of **states**;
- $q_0 \in Q$ the **initial state**;
- C a set of **clocks**;
- $I : Q \to \Phi(C)$ an **invariant mapping**;
- $\delta : Q \times \Phi(C) \times \Sigma \to \mathcal{P}(Q \times \mathcal{P}(C))$ the **transition function**.



Configurations, transitions, and runs

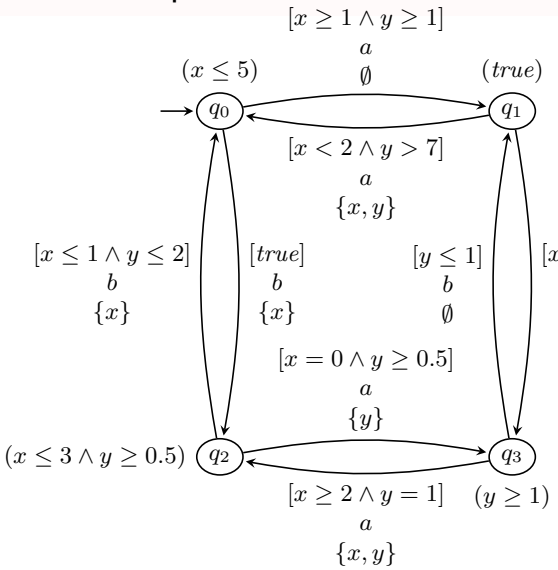
Definition 8 (Configurations and transitions). Let $\mathcal{A} = (\Sigma, Q, q_0, C, I, \delta)$ be a TA.

- A **configuration** is a pair $(q, \nu) \in Q \times \mathsf{Val}(C)$.
- From $(q, \nu) \in Q \times \mathsf{Val}(C)$, there are two possible types of **transitions**.

  Delay With $t \in \mathbb{R}$, $(q, \nu) \xrightarrow{t} (q, \nu + t)$ if and only if $\forall t' \in [0, t] : v + t' \models I(q)$.

  Discrete With $a \in \Sigma$, $(q, \nu) \xrightarrow{\varphi, a, R} (q', \nu[R])$ if and only if
  - $\delta(q, \varphi, a) \ni (q', R)$; and
  - $\nu \models \varphi$.
- The **initial configuration** is $(q_0, \mathbf{0}_C)$.
- A **run** is a finite sequence of delay and discrete transitions, starting from the initial configuration.

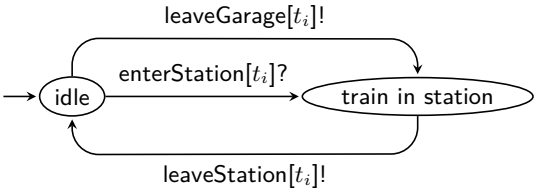
Runs – Example



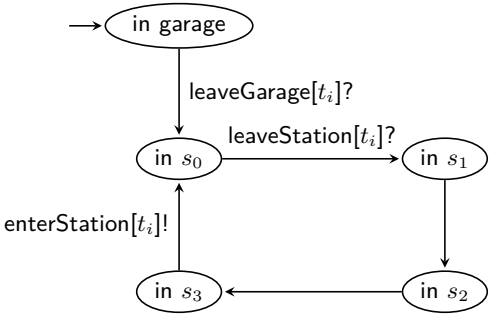
$$(q_0, (x = 0, y = 0)) \xrightarrow{2} (q_0, (x = 2, y = 2))$$
$$\xrightarrow{a} (q_1, (x = 2, y = 2))$$
$$\xrightarrow{b} (q_3, (x = 0, y = 2))$$
$$\xrightarrow{1} (q_3, (x = 1, y = 3))$$

From there, all we can do is wait. Forever...

Previously, in our roller coaster



Model for the **station**.

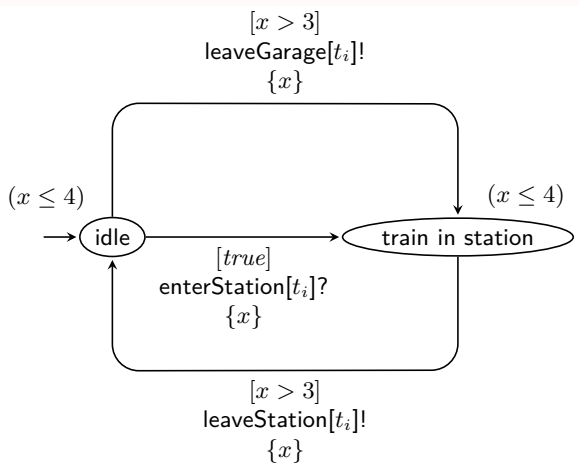


Model for the **train** t_i.

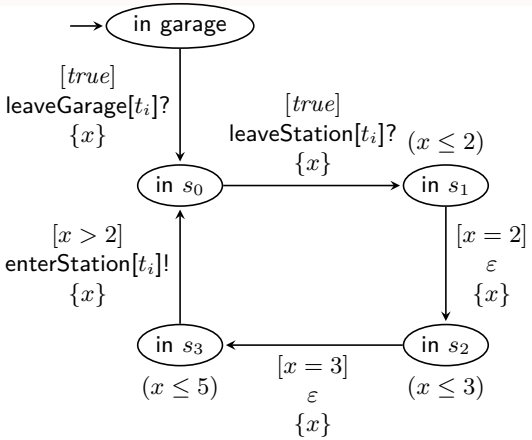
Timing constraints:

- Time for a train in s_0: in $(3, 4]$.
- Time in s_1: 2.
- Time in s_2: 3.
- Time in s_3: in $(2, 5]$.

Timed roller coaster

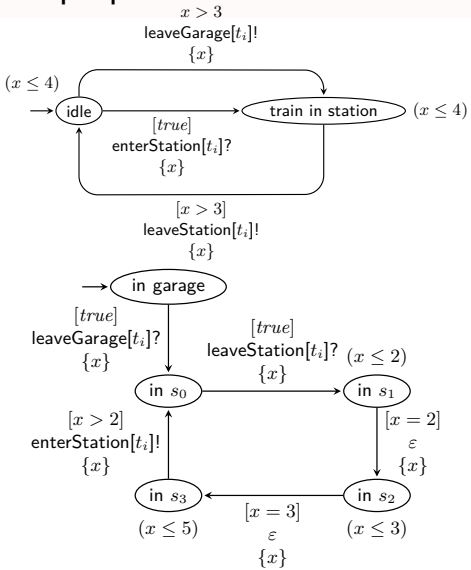


Model for the **station** with clock x.



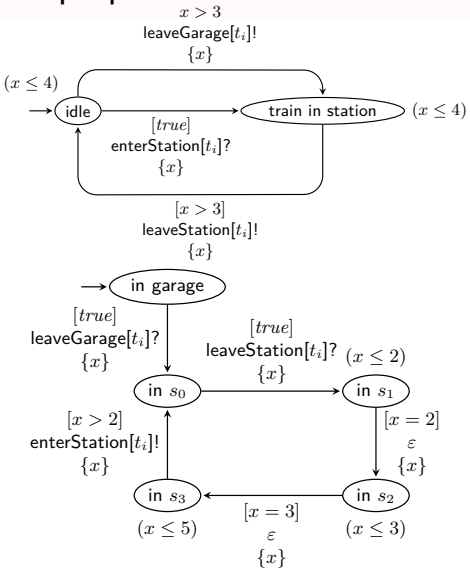
Model for the **train** t_i with clock x.

Which properties hold?



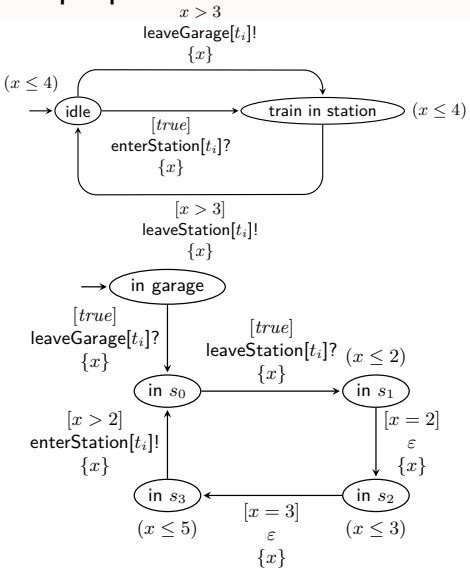
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward.

Which properties hold?



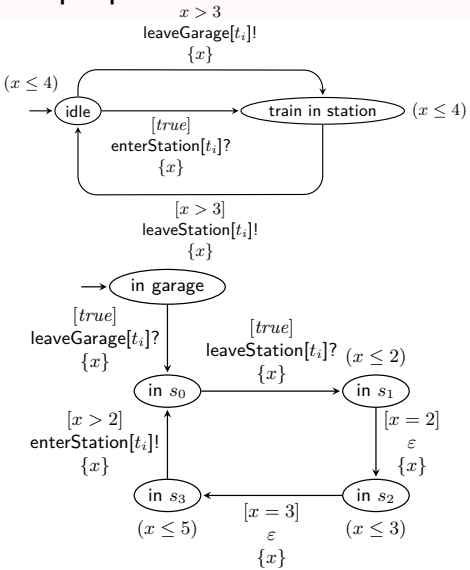
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓

Which properties hold?



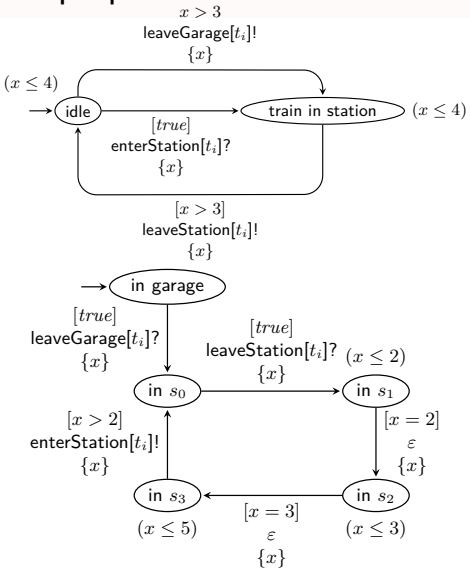
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running.

Which properties hold?



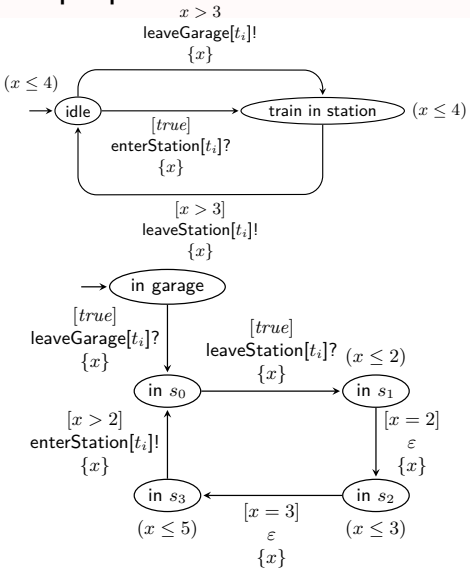
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓

Which properties hold?



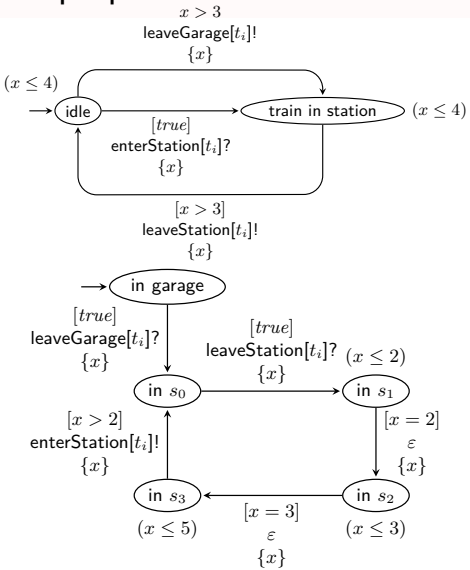
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓
- It must be possible to have two trains running.

Which properties hold?



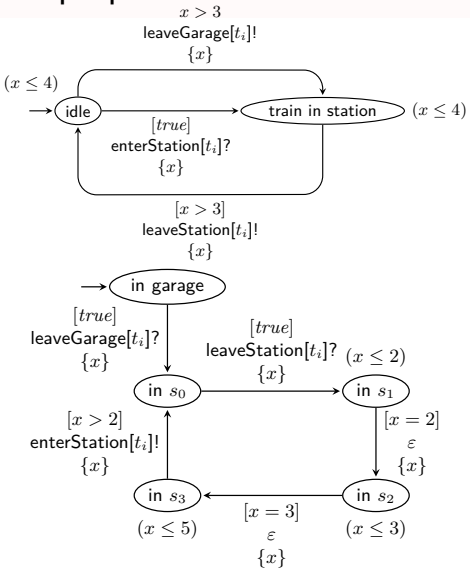
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓
- It must be possible to have two trains running. ??

Which properties hold?



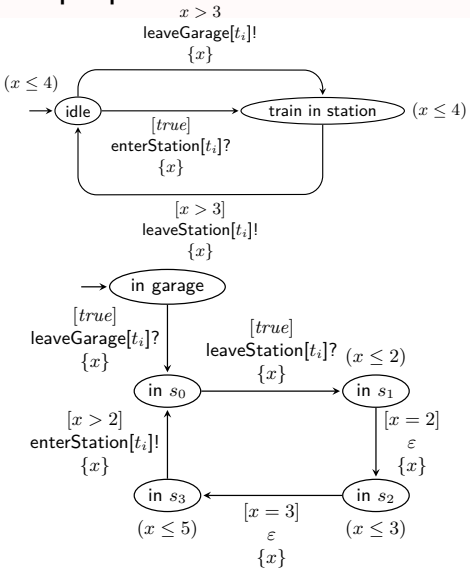
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓
- It must be possible to have two trains running. ??
- It must be possible to have three trains running.

Which properties hold?



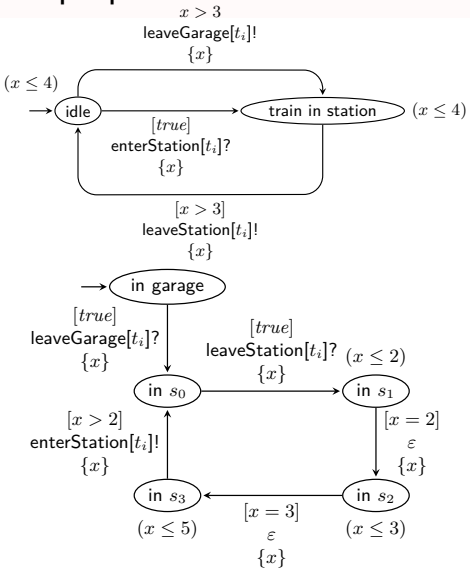
- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓
- It must be possible to have two trains running. ??
- It must be possible to have three trains running. ??

Which properties hold?



- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓
- It must be possible to have two trains running. ??
- It must be possible to have three trains running. ??
- Two trains can never be in the same section at the same time.

Which properties hold?



- The attraction cannot get stuck, *i.e.*, a train must always be able to move forward. ✓
- It must be possible to have one train running. ✓
- It must be possible to have two trains running. ??
- It must be possible to have three trains running. ??
- Two trains can never be in the same section at the same time. ??

Two kinds of properties

In this course and in the exercise session, we focus on two types of properties.

- **Reachability**: is there an execution of the system that reaches a good state?

> *Example 9.* It must be possible to have one train running.

- **Safety**: the system can never reach a bad state.

> *Example 10.* Two trains can never be in the same section at the time.
> A train must always be able to move forward.

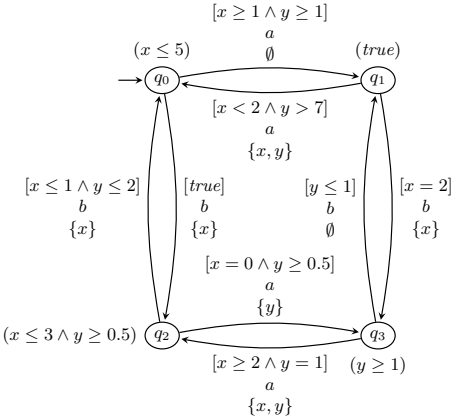
Here, we give the intuition for checking reachability.

Not everything is reachable

Main observation. Given a TA $\mathcal{A}$ and a configuration (q, ν), it is not always possible to construct a run from $(q_0, \mathbf{0}_C)$ to (q, ν).

Not everything is reachable

Main observation. Given a TA $\mathcal{A}$ and a configuration (q, ν), it is not always possible to construct a run from $(q_0, \mathbf{0}_C)$ to (q, ν).

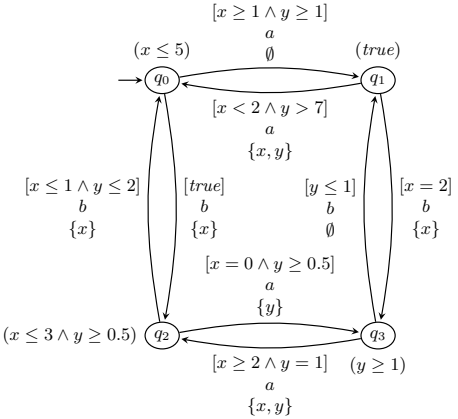


We can reach $(q_1, (x = 1, y = 1))$:

$$(q_0, (x = 0, y = 0)) \xrightarrow{1} (q_0, (x = 1, y = 1)) \xrightarrow{a} (q_1, (x = 1, y = 1))$$

Not everything is reachable

Main observation. Given a TA $\mathcal{A}$ and a configuration (q, ν), it is not always possible to construct a run from $(q_0, \mathbf{0}_C)$ to (q, ν).



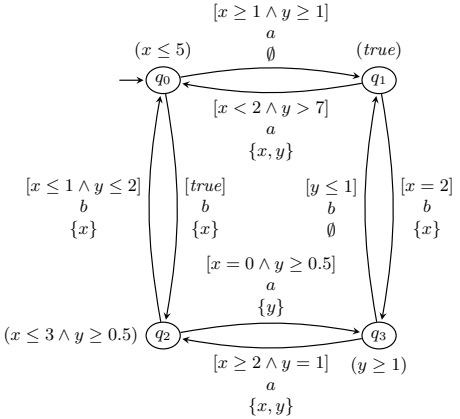
We can reach $(q_1, (x = 1, y = 1))$:

$$(q_0, (x = 0, y = 0)) \xrightarrow{1} (q_0, (x = 1, y = 1)) \xrightarrow{a} (q_1, (x = 1, y = 1))$$

We cannot reach $(q_0, (x = 6, y = 4))$, due to the invariant of q_0.

Not everything is reachable

Main observation. Given a TA $\mathcal{A}$ and a configuration (q, ν), it is not always possible to construct a run from $(q_0, \mathbf{0}_C)$ to (q, ν).



We can reach $(q_1, (x = 1, y = 1))$:

$$(q_0, (x = 0, y = 0)) \xrightarrow{1} (q_0, (x = 1, y = 1)) \xrightarrow{a} (q_1, (x = 1, y = 1))$$

We cannot reach $(q_0, (x = 6, y = 4))$, due to the invariant of q_0.

We cannot reach $(q_0, (x = 0, y = 10))$. That's hard to see...

Constructing the reachable fragment – Rough idea

- We know that we start from $(q_0, \mathbf{0}_C)$.

Constructing the reachable fragment – Rough idea

- We know that we start from $(q_0, \mathbf{0}_C)$.
- We have to satisfy $I(q_0)$.

Constructing the reachable fragment – Rough idea

- We know that we start from $(q_0, \mathbf{0}_C)$.
- We have to satisfy $I(q_0)$.
- If we want to trigger a discrete transition, we must satisfy its guard φ.

Constructing the reachable fragment – Rough idea

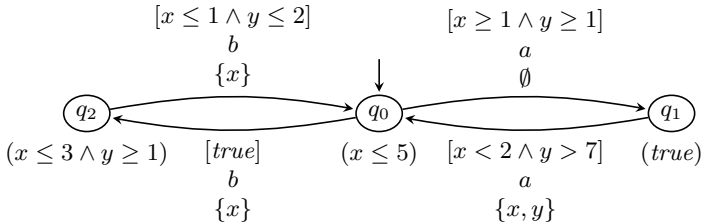
- We know that we start from $(q_0, \mathbf{0}_C)$.
- We have to satisfy $I(q_0)$.
- If we want to trigger a discrete transition, we must satisfy its guard φ.
- We repeat these steps from the newly reached configuration.

Constructing the reachable fragment – Rough idea

- We know that we start from $(q_0, \mathbf{0}_C)$.
- We have to satisfy $I(q_0)$.
- If we want to trigger a discrete transition, we must satisfy its guard φ.
- We repeat these steps from the newly reached configuration.

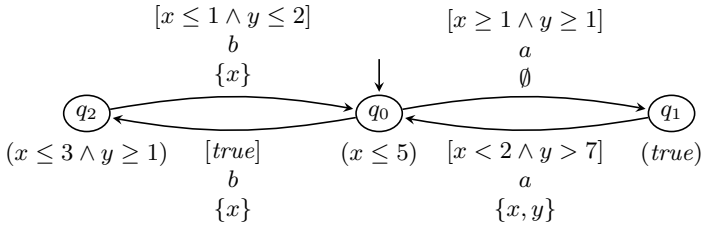
So, along an exploration of the TA, we can accumulate **constraints** over the valuations.

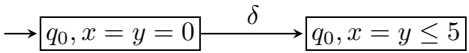
Constructing the reachable fragment – Example



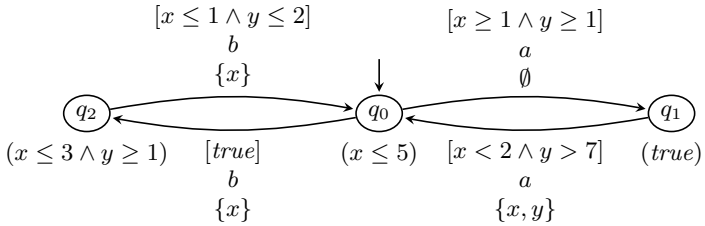
$\longrightarrow \boxed{q_0, x = y = 0}$

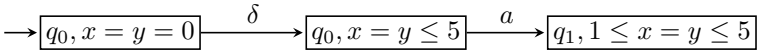
Constructing the reachable fragment – Example



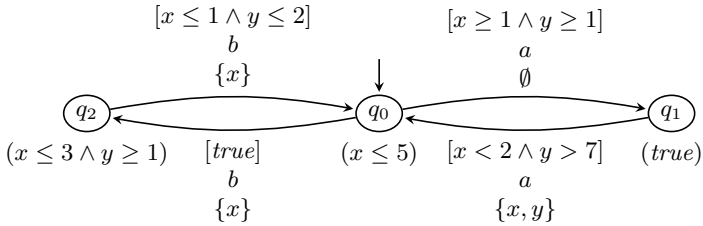


Constructing the reachable fragment – Example



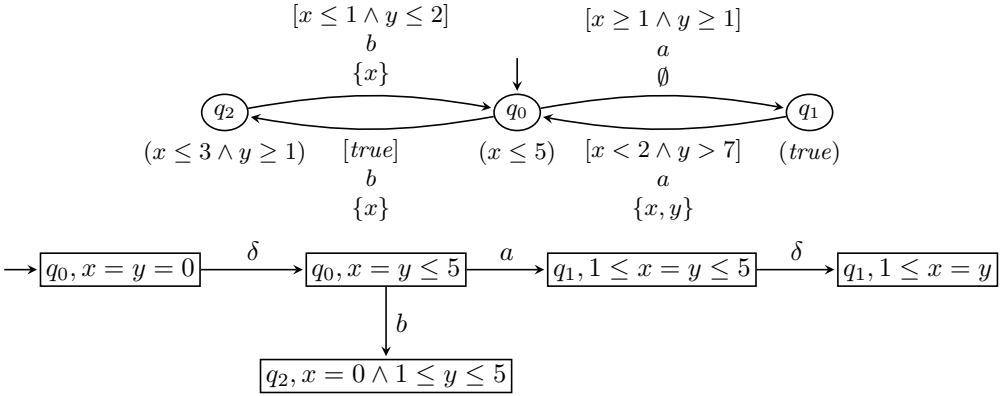


Constructing the reachable fragment – Example

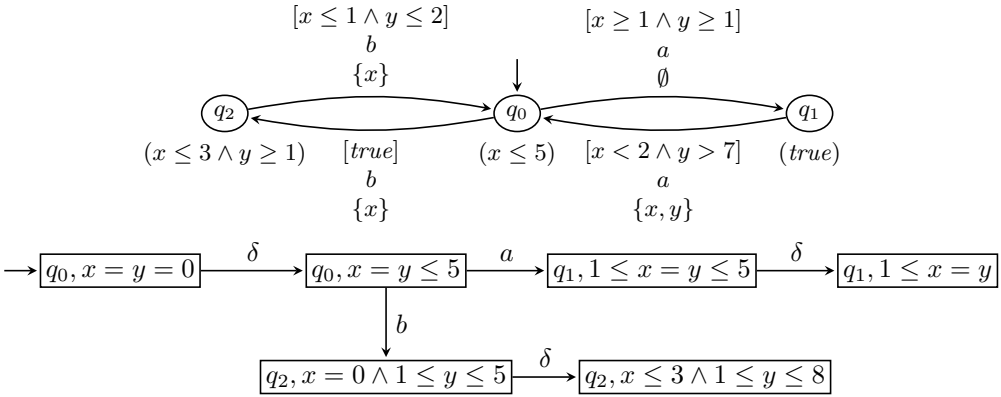


$$\rightarrow \boxed{q_0, x = y = 0} \xrightarrow{\delta} \boxed{q_0, x = y \leq 5} \xrightarrow{a} \boxed{q_1, 1 \leq x = y \leq 5} \xrightarrow{\delta} \boxed{q_1, 1 \leq x = y}$$

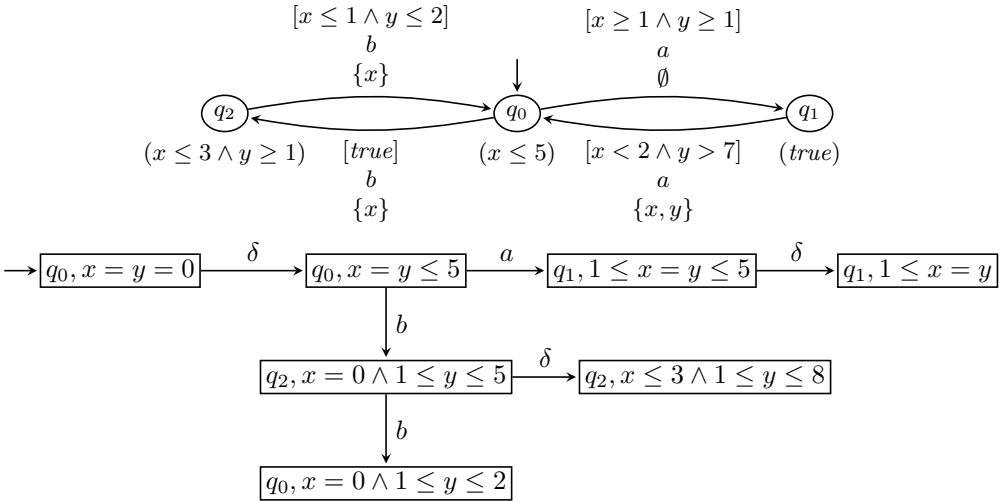
Constructing the reachable fragment – Example



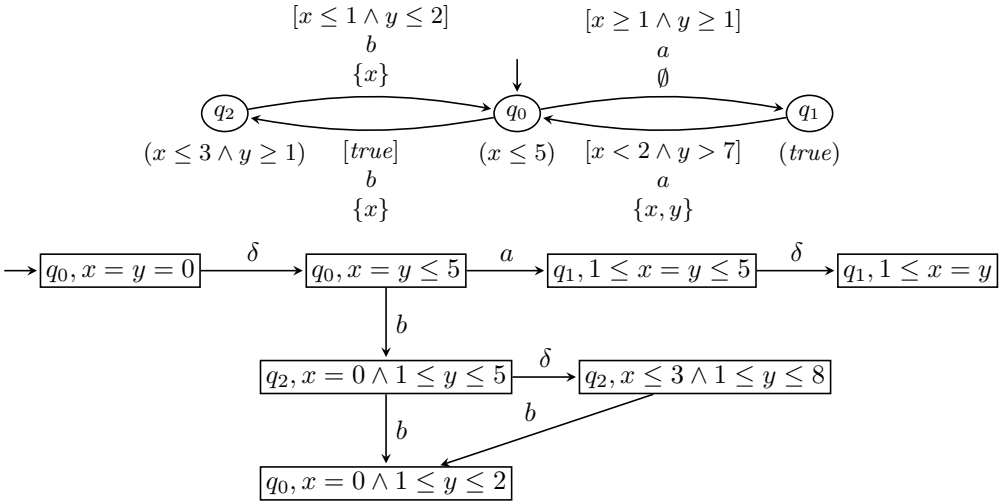
Constructing the reachable fragment – Example



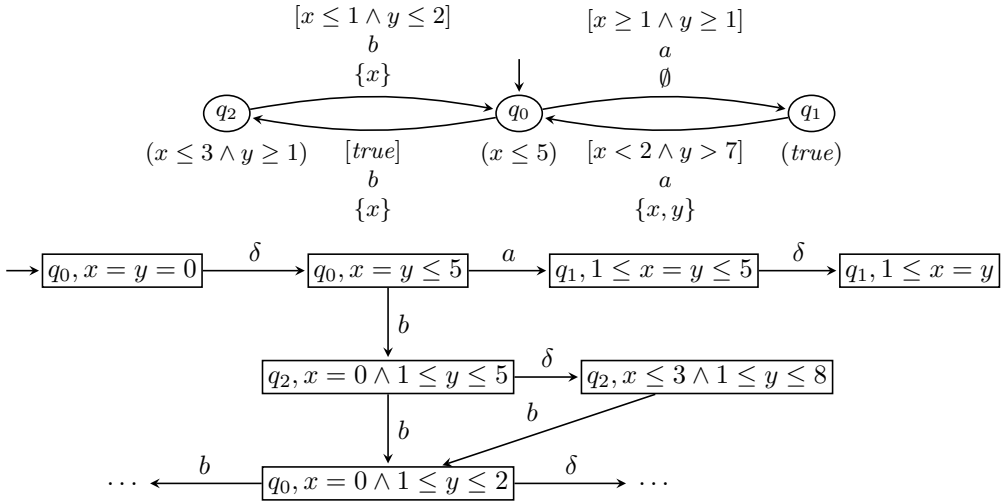
Constructing the reachable fragment – Example



Constructing the reachable fragment – Example



Constructing the reachable fragment – Example



Deciding reachability properties

Theorem 11. Let $\mathcal{A}$ be a TA, and (q, ν) be a configuration. Deciding wether (q, ν) is reachable is PSPACE-complete.

Tools

Luckily, we don't have to verify the properties by hand!

- TChecker: open-source, no graphical interface, timed automata.
  `https://github.com/ticktac-project/tchecker`
- IMITATOR: open-source, no graphical interface, **parametric** timed automata.
  `https://www.imitator.fr/`
- Roméo: open-source, has a graphical interface, timed **Petri nets**.
  `https://romeo.ls2n.fr/`
- **UPPAAL**: closed-source, has a graphical interface, timed automata.
  `https://uppaal.org/`

After the break

In the practical session:

- To get started with **UPPAAL**: the roller coaster example.
- An exercise on modeling a system that shares resources between processes.
- An exercise to write properties for a model of an amusement park.