Timed automata – Decidability issues

Patricia Bouyer-Decitre

LSV, CNRS & ENS Cachan, France
An example of a timed automaton

```
x := 0

y := 0

15 \leq x \leq 16

delayed, y := 0

repair, x \leq 15

repair, y := 0

2 \leq y \land x \leq 56

done, 22 \leq y \leq 25

problem, x := 0

This run reads the timed word

(problem, 23)(delayed, 38.6)(repair, 40.9),

(done, 63).
```
An example of a timed automaton

This run reads the timed word

\((\text{problem}, 23)(\text{delayed}, 38.6)(\text{repair}, 40.9), (\text{done}, 63)\).

<table>
<thead>
<tr>
<th>safe</th>
<th>problem, x:=0</th>
<th>alarm, y:=0</th>
<th>repairing, x:=15</th>
<th>done, 22≤y≤25</th>
<th>repair, 2≤y∧x≤56</th>
<th>delayed, y:=0</th>
<th>failsafe</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An example of a timed automaton

This run reads the timed word

(problem, 23)(delayed, 38.

(repair, 40.

(done, 63).

\[ x := 0 \]
\[ y := 0 \]

\[ 9 \leq y \leq 25 \]

\[ 15 \leq x \leq 16 \]

\[ 2 \leq y \land x \leq 56 \]

\[ y := 0 \]

\[ y := 0 \]
An example of a timed automaton

This run reads the timed word

\((\text{problem}, 23)(\text{delayed}, 38.6)(\text{repair}, 40.9), (\text{done}, 63)\).

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>Next State</th>
<th>Value 1</th>
<th>Value 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>safe</td>
<td>23</td>
<td>safe</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>23</td>
<td>y</td>
<td>23</td>
<td>23</td>
</tr>
</tbody>
</table>
An example of a timed automaton
An example of a timed automaton

This run reads the timed word

\((\text{problem}, 23)(\text{delayed}, 38.6)(\text{repair}, 40.9), (\text{done}, 63)\).
An example of a timed automaton

This run reads the timed word $(\text{problem}, 23)(\text{delayed}, 38.6)(\text{repair}, 40.9)$, $(\text{done}, 63).

\[
\begin{array}{c|c|c|c|c|c}
\text{state} & x & y & \text{action} & \text{time} \\
\hline
\text{safe} & 0 & 0 & \text{repair, } x = 0 & 0 \\
\text{alarm} & 0 & 23 & \text{repair, } x \leq 15 & 15.6 \\
\text{failsafe} & 0 & 0 & \text{delayed, } y = 0 & 15.6 \\
\end{array}
\]
An example of a timed automaton

This run reads the timed word

\[
\text{problem, } x := 0 \quad \rightarrow \quad \text{repair, } x \leq 15 \quad \rightarrow \quad \text{delayed, } y := 0 \quad \rightarrow \quad \text{failsafe}.
\]

\[
\text{done, } 22 \leq y \leq 25 \quad \rightarrow \quad \text{repair, } 2 \leq y \land x \leq 56 \quad \rightarrow \quad \text{repair}.
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\text{safe} & 23 & \text{safe} & \text{problem} & \text{alarm} & 15.6 & \text{alarm} & \text{delayed} & \text{failsafe} \\
X & 0 & 23 & 0 & 15.6 & 15.6 & \ldots \\
y & 0 & 23 & 23 & 38.6 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\text{failsafe} & 2.3 & \text{failsafe} & \text{repair} & \text{repairing} \\
\ldots & 15.6 & 17.9 & 17.9 \\
y & 0 & 2.3 & 0 \\
\end{array}
\]
An example of a timed automaton

This run reads the timed word \((\text{problem, } x:=0, 23\rightarrow \text{safe})\), \((\text{delayed, } y:=0, 22\leq y \leq 25\rightarrow \text{failsafe})\), and \((\text{repair, } y:=0, 2 \leq y \wedge x \leq 56\rightarrow \text{repairing})\).
An example of a timed automaton

This run reads the timed word:

\[(\text{problem}, 23) (\text{delayed}, 38.6) (\text{repair}, 40.9) \rightarrow \text{done}, 63)\]
An example of a timed automaton

This run reads the timed word
(problem, 23)(delayed, 38.6)(repair, 40.9), (done, 63).
Outline

1. Decidability of basic properties

2. Equivalence (or preorder) checking

3. Some extensions of timed automata
Verification

Emptiness problem
Is the language accepted by a timed automaton empty?
- basic reachability/safety properties (final states)
- basic liveness properties ($\omega$-regular conditions)
Verification

Emptiness problem

Is the language accepted by a timed automaton empty?

- **Problem:** the set of configurations is infinite
  \[\Rightarrow\] classical methods for finite-state systems cannot be applied

Theorem [AD90, AD94]
The emptiness problem for timed automata is decidable and PSPACE-complete.

Method: construct a finite abstraction
Verification

Emptiness problem
Is the language accepted by a timed automaton empty?

- Problem: the set of configurations is infinite
  \[\leadsto\] classical methods for finite-state systems cannot be applied

- Positive key point: variables (clocks) increase at the same speed

Theorem [AD90, AD94]
The emptiness problem for timed automata is decidable and PSPACE-complete.

Method: construct a finite abstraction
Verification

Emptiness problem
Is the language accepted by a timed automaton empty?

- **Problem:** the set of configurations is infinite
  \[\rightarrow\] classical methods for finite-state systems cannot be applied

- **Positive key point:** variables (clocks) increase at the same speed

**Theorem** [AD90,AD94]
The emptiness problem for timed automata is decidable and PSPACE-complete.

---

[AD90] Alur, Dill. Automata for modeling real-time systems (*ICALP’90*).
Verification

Emptiness problem

Is the language accepted by a timed automaton empty?

- **Problem:** the set of configurations is infinite
  \[ \leadsto \] classical methods for finite-state systems cannot be applied

- **Positive key point:** variables (clocks) increase at the same speed

**Theorem [AD90,AD94]**

The emptiness problem for timed automata is decidable and PSPACE-complete.

**Method:** construct a finite abstraction

The region abstraction

```
only constraints:
x \sim c \text{ with } c \in \{0, 1, 2\}
y \sim c \text{ with } c \in \{0, 1, 2\}
```

The path \( x = 1 \), \( y = 1 \) - can be fired from

- cannot be fired from

"compatibility" between regions and constraints

\( \Rightarrow \) an equivalence of finite index

\( \Rightarrow \) time-abstract bisimulation
The region abstraction

only constraints: $x \sim c$ with $c \in \{0, 1, 2\}$
$y \sim c$ with $c \in \{0, 1, 2\}$

“compatibility” between regions and constraints
The region abstraction

```
<table>
<thead>
<tr>
<th>clock y</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- `x = 1` can be fired from
- `y = 1` cannot be fired from

"compatibility" between regions and constraints
"compatibility" between regions and time elapsing
The region abstraction

```
Decidability of basic properties

The region abstraction

```

![Diagram showing the region abstraction with clocks x and y, and constraints x ~ c and y ~ c where c ∈ {0, 1, 2}. The path x=1 y=1 can be fired from, while the path cannot be fired from.]

- "compatibility" between regions and constraints
- "compatibility" between regions and time elapsing
The region abstraction

- "compatibility" between regions and constraints
- "compatibility" between regions and time elapsing

⇝ an equivalence of finite index
The region abstraction

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
```

- “compatibility” between regions and constraints
- “compatibility” between regions and time elapsing

\[\leadsto\] an equivalence of finite index

a time-abstract bisimulation
Time-abstract bisimulation

This is a relation between ● and ● such that:

∀d > 0 ∃d′ > 0 / u₁D6FF(d)

... and vice-versa (swap ● and ●).
Time-abstract bisimulation

This is a relation between \( \bullet \) and \( \bullet \) such that:

\[
\forall \quad \exists \quad a
\]

... and vice-versa (swap \( \bullet \) and \( \bullet \)).
Time-abstract bisimulation

This is a relation between ● and ○ such that:

∀ (d) → ∃ (d') > 0 (d') → ∀ (d') > 0 (d) → ... and vice-versa (swap ● and ○).
Time-abstract bisimulation

This is a relation between \(\bullet\) and \(\bullet\) such that:

\[
\begin{align*}
\forall \quad & \quad \exists \quad a \\
\exists \quad & \quad \forall \quad a
\end{align*}
\]

\[
\begin{align*}
\forall d > 0 \quad & \quad \exists \quad \delta(d)
\end{align*}
\]
Time-abstract bisimulation

This is a relation between $\bullet$ and $\bullet$ such that:

\[
\forall \exists a / u1D6FF (d)\]

\[
\forall d > 0 \exists d' > 0 / u1D6FF (d')\]

... and vice-versa (swap $\bullet$ and $\bullet$).
Time-abstract bisimulation

This is a relation between $\bullet$ and $\bullet$ such that:

$$\forall \exists a \quad \forall \exists d > 0 \quad \exists d' > 0$$

... and vice-versa (swap $\bullet$ and $\bullet$).
The region abstraction (2)

- region $R$ defined by:
  
  \[
  \begin{cases}
  0 < x < 1 \\
  0 < y < 1 \\
  y < x
  \end{cases}
  \]
The region abstraction (2)

- region $R$ defined by:
\[
\begin{align*}
0 &< x < 1 \\
0 &< y < 1 \\
y &< x
\end{align*}
\]

- time successors of $R$
The region abstraction (2)

- region $R$ defined by:
  \[
  \begin{cases} 
  0 < x < 1 \\
  0 < y < 1 \\
  y < x
  \end{cases}
  \]

- time successors of $R$

- image of $R$ when resetting clock $x$
The construction of the region graph

It “mimicks” the behaviours of the clocks.
Region automaton $\equiv$ finite bisimulation quotient

\[
\begin{align*}
\ell_1 & \quad y < 1, a, x := 0 \\
\ell_2 &
\end{align*}
\]

timed automaton

region graph
Region automaton \equiv finite bisimulation quotient

timed automaton

region graph

region automaton

\[ y < 1, a, x := 0 \]
Region automaton \equiv finite bisimulation quotient

\[
\begin{array}{c}
\ell_1 \xrightarrow{y<1,a,x:=0} \ell_2 \quad \otimes \\
\text{timed automaton}
\end{array}
\]

\[
\begin{array}{c}
\text{region automaton}
\end{array}
\]

\[
\mathcal{L}(\text{reg. aut.}) = \text{UNTIME}(\mathcal{L}(\text{timed aut.}))
\]
An example [AD94]
An example [AD94]

Decidability of basic properties
An example [AD94]
Decidability of basic properties

A timed automaton can be transformed into a large (but finite) automaton through a process involving finite bisimulation. The number of regions in the quotient is given by:

$$Y \times (2^M + 2) \cdot \left| X \right|! \cdot 2^\left| X \right|$$

This can be used to check for:
- Reachability/safety properties
- Liveness properties (like B"uchi properties)
- Problems with Zeno behaviours (infinitely many actions in bounded time)

NB: Standard problem in timed automata...
Decidability of basic properties

- **large**: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is:

\[
\prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|}
\]
Decidability of basic properties

- **large**: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is:

\[ \prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|} \]

- It can be used to check for:

  - reachability/safety properties
  - liveness properties (like Büchi properties)
  - problems with Zeno behaviours?

  NB: standard problem in timed automata...
Decidability of basic properties

- **large**: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is:

\[ \prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|} \]

- It can be used to check for:
  - reachability/safety properties
  - liveness properties (like Büchi properties)
  - problems with Zeno behaviours? (infinitely many actions in bounded time)

NB: standard problem in timed automata...
Decidability of basic properties

- **large**: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is:

\[ \prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|} \]

- It can be used to check for:
  - reachability/safety properties
  - liveness properties (like Büchi properties)
Decidability of basic properties

- **large**: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is:

\[
\prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|}
\]

- It can be used to check for:
  - reachability/safety properties
  - liveness properties (like Büchi properties)

→ problems with Zeno behaviours?
  (infinitely many actions in bounded time)
Decidability of basic properties

- **large**: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is:

\[
\prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|}
\]

- It can be used to check for:
  - reachability/safety properties
  - liveness properties (like Büchi properties)

\[\Rightarrow \text{problems with Zeno behaviours?} \]

(Infinitely many actions in bounded time)

**NB**: standard problem in timed automata...
Back to the example

Decidability of basic properties
Back to the example

0
1
2
3

\( x > 0, a \)

\( y = 1, b \)

\( x < 1, c \)

\( x < 1, c \)

\( y < 1, a, y := 0 \)

\( x > 1, d \)

\( y < 1, a, y := 0 \)
Back to the example
Back to the example

Decidability of basic properties

Zeno cycles
Back to the example

Decidability of basic properties
Outline

1. Decidability of basic properties

2. Equivalence (or preorder) checking

3. Some extensions of timed automata
Strong timed (bi)simulation

This is a relation between ● and ● such that:

∀d > 0 ∃/u1D6FF(d)...

... and vice-versa (swap ● and ●) for the bisimulation relation.

Theorem

Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete. (see later for a simple proof of the upper bound)
Strong timed (bi)simulation

This is a relation between ● and ● such that:

∀ ● → ●

Theorem
Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete.
(see later for a simple proof of the upper bound)
Strong timed (bi)simulation

This is a relation between $\bullet$ and $\bullet$ such that:

$$\forall \exists \ a$$

... and vice-versa (swap $\bullet$ and $\bullet$) for the bisimulation relation.

Theorem

Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete.

(see later for a simple proof of the upper bound)
Strong timed (bi)simulation

This is a relation between \( \bullet \) and \( \bullet \) such that:

\[
\forall a \quad \exists \quad a
\]

\[
\forall d > 0 \quad \exists \quad \delta(d)
\]
Strong timed (bi)simulation

This is a relation between $\bullet$ and $\bullet$ such that:

\[
\forall \quad \exists \quad \forall d > 0 \quad \exists
\]

\[
\exists \quad \forall \quad \exists \quad \exists
\]

... and vice-versa (swap $\bullet$ and $\bullet$) for the bisimulation relation.

Theorem

Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete.

(see later for a simple proof of the upper bound)
Strong timed (bi)simulation

This is a relation between $\circ$ and $\bullet$ such that:

$\forall a \exists (\d)$

$\forall d > 0 \exists \delta(d)$

... and vice-versa (swap $\circ$ and $\bullet$) for the bisimulation relation.

Theorem
Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete.
(see later for a simple proof of the upper bound)
Strong timed (bi)simulation

This is a relation between $\bullet$ and $\bullet$ such that:

$$\forall a \exists a (d) \quad \forall d > 0 \exists \delta(d)$$

... and vice-versa (swap $\bullet$ and $\bullet$) for the bisimulation relation.

**Theorem**

Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete.

(see later for a simple proof of the upper bound)
Language (or trace) equivalence and inclusion

Question

Given two timed automata $A$ and $B$, is $L(A) = L(B)$ (resp. $L(A) \subseteq L(B)$)?

Theorem

Language equivalence and language inclusion are undecidable in timed automata.

... as a special case of the universality problem (are all timed words accepted by the automaton?).

Proof by reduction from the recurring problem of a two-counter machine.
Language (or trace) equivalence and inclusion

Question
Given two timed automata $\mathcal{A}$ and $\mathcal{B}$, is $L(\mathcal{A}) = L(\mathcal{B})$ (resp. $L(\mathcal{A}) \subseteq L(\mathcal{B})$)?

Theorem [AD90, AD94]
Language equivalence and language inclusion are undecidable in timed automata.

... as a special case of the universality problem (are all timed words accepted by the automaton?).

[AD90] Alur, Dill. Automata for modeling real-time systems (ICALP’90).
Language (or trace) equivalence and inclusion

Question
Given two timed automata $\mathcal{A}$ and $\mathcal{B}$, is $L(\mathcal{A}) = L(\mathcal{B})$ (resp. $L(\mathcal{A}) \subseteq L(\mathcal{B})$)?

Theorem [AD90,AD94]
Language equivalence and language inclusion are undecidable in timed automata.

... as a special case of the universality problem (are all timed words accepted by the automaton?).

$\implies$ Proof by reduction from the recurring problem of a two-counter machine

[AD90] Alur, Dill. Automata for modeling real-time systems (ICALP’90).
Undecidability of universality

Theorem [AD90,AD94]

Universality of timed automata is undecidable.
Undecidability of universality

Theorem [AD90,AD94]

Universality of timed automata is undecidable.

- One configuration is encoded in one time unit.
- Number of $c$'s: Value of counter $c$.
- Number of $d$'s: Value of counter $d$.
- One time unit between two corresponding $c$'s (resp. $d$'s).
Undecidability of universality

Theorem [AD90, AD94]

Universality of timed automata is undecidable.

- One configuration is encoded in one time unit
- Number of $c$'s: value of counter $c$
- Number of $d$'s: value of counter $d$
- One time unit between two corresponding $c$'s (resp. $d$'s)

\[\rightarrow \text{We encode "non-behaviours" of a two-counter machine}\]
Example
Module to check that if instruction $i$ does not decrease counter $c$, then all actions $c$ appearing less than 1 t.u. after $b_i$ has to be followed by an other $c$ 1 t.u. later.
Example

Module to check that if instruction $i$ does not decrease counter $c$, then all actions $c$ appearing less than 1 t.u. after $b_i$ has to be followed by another $c$ 1 t.u. later.

\[ s_0 \xrightarrow{b_i, x := 0} s_1 \xrightarrow{x < 1, c, x := 0} s_2 \xrightarrow{x \neq 1} x = 1, \neg c \]

The union of all small modules is not universal iff
The two-counter machine has a recurring computation
Bad news

- Language inclusion is **undecidable**
  (Bad news for the application to verification)
- Complementability is **undecidable**
- ...

[AD90,AD94] [Tri03,Fin06]

[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata (*FORMATS’03*).
[Fin06] Finkel. Undecidable problems about timed automata (*FORMATS’06*).
Bad news

- Language inclusion is *undecidable*  
  (Bad news for the application to verification)
- Complementability is *undecidable*  
- ...

An example of non-determinizable/non-complementable timed aut.:

![Timed Automaton Diagram]

[AD90,AD94]  
[Tri03,Fin06]  

[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata (*FORMATS’03*).  
[Fin06] Finkel. Undecidable problems about timed automata (*FORMATS’06*).
Bad news

- Language inclusion is undecidable
  (Bad news for the application to verification)
- Complementability is undecidable
- ...

An example of non-determinizable/non-complementable aut.: [AM04]

---

[AD90,AD94] Tripakis. Folk theorems on the determinization and minimization of timed automata (FORMATS’03).
[Fin06] Finkel. Undecidable problems about timed automata (FORMATS’06).
Bad news

- Language inclusion is undecidable  
  (Bad news for the application to verification)
- Complementability is undecidable  
- ...

An example of non-determinizable/non-complementable aut.:  

\[
\begin{align*}
 s_0 &\xrightarrow{a, x := 0} s_1 \\
 s_1 &\xrightarrow{x \neq 1, a, b} s_0
\end{align*}
\]

\[\text{UNTIME } \left( \bar{L} \cap \{(a^*b^*, \tau) \mid \text{all } a's \text{ happen before } 1 \text{ and no two } a's \text{ simultaneously} \} \right) \text{ is not regular (exercise!)}\]

[AD90,AD94] Alur. Deterministic timed automata (FORMATS’90).
[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata (FORMATS’03).
[Fin06] Finkel. Undecidable problems about timed automata (FORMATS’06).
Outline

1. Decidability of basic properties

2. Equivalence (or preorder) checking

3. Some extensions of timed automata
What if we extend the clock constraints?

- Diagonal constraints \((i.e. \ x - y \leq 2)\)
What if we extend the clock constraints?

- **Diagonal constraints** (*i.e.* $x - y \leq 2$)
  - **Decidable** (with the same complexity)
What if we extend the clock constraints?

- **Diagonal constraints** (i.e. $x - y \leq 2$)
  - **decidable** (with the same complexity)

is also a time-abstract bisimulation!
What if we extend the clock constraints?

- **Diagonal constraints** *(i.e. \( x - y \leq 2 \))
  - decidable (with the same complexity)

  is also a time-abstract bisimulation!

- **Linear constraints** *(i.e. \( 2x + 3y \sim 5 \))

The diagram illustrates the decidability of these constraints.
What if we extend the clock constraints?

- **Diagonal constraints** (*i.e.* $x - y \leq 2$)
  - *decidable* (with the same complexity)

  is also a time-abstract bisimulation!

- **Linear constraints** (*i.e.* $2x + 3y \sim 5$)
  - *undecidable* in general
What if we extend the clock constraints?

- **Diagonal constraints** (*i.e.* $x - y \leq 2$)
  - **Decidable** (with the same complexity)

  ![Diagonal constraints graph]

  is also a time-abstract bisimulation!

- **Linear constraints** (*i.e.* $2x + 3y \sim 5$)
  - **Undecidable** in general
  - Only **decidable** in few cases
What if we extend the clock constraints?

- **Diagonal constraints** (i.e. $x - y \leq 2$)
  - decidable (with the same complexity)
  
  is also a time-abstract bisimulation!

- **Linear constraints** (i.e. $2x + 3y \sim 5$)
  - undecidable in general
  - only decidable in few cases

  is a time-abstract bisimulation (when two clocks $x$ and $y$ and constraints $x + y \sim c$)!
What if we allow more operations on clocks?

- that can be transfer operations (i.e. $x := y$), or reinitialization operations (i.e. $x := 4$), or ...

What if we allow more operations on clocks?

- that can be transfer operations (*i.e.*, \(x := y\)), or reinitialization operations (*i.e.*, \(x := 4\)), or ... [BDFP04]

<table>
<thead>
<tr>
<th></th>
<th>simple constraints</th>
<th>+ diagonal constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x := c, x := y)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x := x + 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x := y + c)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x := x - 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x :&lt; c)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x :&gt; c)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x \sim y + c)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(y + c :&lt; x :&lt; y + d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(y + c :&lt; x :&lt; z + d)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[BDFP04] Bouyer, Dufourd, Fleury, Petit. Updatable Timed Automata (*Theoretical Computer Science*).
What if we allow more operations on clocks?

- that can be **transfer operations** \((i.e. \; x := y)\), or **reinitialization operations** \((i.e. \; x := 4)\), or ...

<table>
<thead>
<tr>
<th></th>
<th>simple constraints</th>
<th>+ diagonal constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x := c, ; x := y)</td>
<td>decidable</td>
<td></td>
</tr>
<tr>
<td>(x := x + 1)</td>
<td>decidable</td>
<td></td>
</tr>
<tr>
<td>(x := y + c)</td>
<td>undecidable</td>
<td></td>
</tr>
<tr>
<td>(x := x - 1)</td>
<td>undecidable</td>
<td></td>
</tr>
<tr>
<td>(x :&lt; c)</td>
<td></td>
<td>decidable</td>
</tr>
<tr>
<td>(x :&gt; c)</td>
<td></td>
<td>decidable</td>
</tr>
<tr>
<td>(x :\sim y + c)</td>
<td></td>
<td>undecidable</td>
</tr>
<tr>
<td>(y + c :&lt; x :&lt; y + d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(y + c :&lt; x :&lt; z + d)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


\(\Rightarrow\) need of being very careful when using more operations on clocks!
What if we allow more operations on clocks?

- that can be **transfer operations** (*i.e.* $x := y$), or **reinitialization operations** (*i.e.* $x := 4$), or ...

**[BDFP04]** Bouyer, Dufourd, Fleury, Petit. *Updatable Timed Automata* (*Theoretical Computer Science*).

<table>
<thead>
<tr>
<th></th>
<th>simple constraints</th>
<th>+ diagonal constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x := c, x := y$</td>
<td>decidable</td>
<td><strong>decidable</strong></td>
</tr>
<tr>
<td>$x := x + 1$</td>
<td><strong>decidable</strong></td>
<td><strong>undecidable</strong></td>
</tr>
<tr>
<td>$x := y + c$</td>
<td><strong>undecidable</strong></td>
<td></td>
</tr>
<tr>
<td>$x := x - 1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x &lt; c$</td>
<td><strong>decidable</strong></td>
<td><strong>decidable</strong></td>
</tr>
<tr>
<td>$x &gt; c$</td>
<td><strong>decidable</strong></td>
<td><strong>undecidable</strong></td>
</tr>
<tr>
<td>$x \sim y + c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y + c \leq x \leq y + d$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y + c \leq x \leq z + d$</td>
<td><strong>undecidable</strong></td>
<td></td>
</tr>
</tbody>
</table>

→ need of being very careful when using more operations on clocks!

A note on hybrid automata (see more on Friday)

- a discrete control (the mode of the system)
- continuous evolution of the variables within a mode

A note on hybrid automata (see more on Friday)

a discrete control (the mode of the system)
+ continuous evolution of the variables within a mode

The thermostat example

\[ \dot{T} = \begin{cases} 
  -0.5T & \text{for } T \geq 18 \\
  2.25 - 0.5T & \text{for } T \leq 22 
\end{cases} \]

Theorem [HKPV95]
The reachability problem is undecidable in hybrid automata, even for stopwatch automata.

(stopwatch automata: timed automata in which clocks can be stopped)

A relevant question: Is there something between timed automata and hybrid automata which is decidable?

\Rightarrow See Nicolas' afternoon lecture
Some extensions of timed automata

A note on hybrid automata (see more on Friday)

- A discrete control (the mode of the system)
- Continuous evolution of the variables within a mode

The thermostat example

\[ \dot{T} = \begin{cases} -0.5T & (T \geq 18) \\ 2.25 - 0.5T & (T \leq 22) \end{cases} \]

Theorem [HKPV95]

The reachability problem is undecidable in hybrid automata, even for stopwatch automata.

(stopwatch automata: timed automata in which clocks can be stopped)

A relevant question: Is there something between timed automata and hybrid automata which is decidable?

\[ \Rightarrow \text{See Nicolas' afternoon lecture} \]

A note on hybrid automata (see more on Friday)

- a discrete control (the mode of the system)
- continuous evolution of the variables within a mode

**The thermostat example**

- **Off**
  \[ \dot{T} = -0.5T \quad (T \geq 18) \]
- **On**
  \[ \dot{T} = 2.25 - 0.5T \quad (T \leq 22) \]

**Theorem** [HKPV95]

The reachability problem is **undecidable** in hybrid automata, even for stopwatch automata.

(stopwatch automata: timed automata in which clocks can be stopped)

---

A note on hybrid automata (see more on Friday)

a discrete control (the mode of the system)
+ continuous evolution of the variables within a mode

The thermostat example

Off
\[ \dot{T} = -0.5T \]
\((T \geq 18)\)

On
\[ \dot{T} = 2.25 - 0.5T \]
\((T \leq 22)\)

Theorem [HKPV95]
The reachability problem is undecidable in hybrid automata, even for stopwatch automata.

(stopwatch automata: timed automata in which clocks can be stopped)

A relevant question
Is there something between timed automata and hybrid automata which is decidable?
A note on hybrid automata (see more on Friday)

a discrete control (the mode of the system) 
+ continuous evolution of the variables within a mode

The thermostat example

Theorem [HKPV95]
The reachability problem is *undecidable* in hybrid automata, even for stopwatch automata.

(stopwatch automata: timed automata in which clocks can be stopped)

A relevant question
Is there something between timed automata and hybrid automata which is decidable?

⇝ See Nicolas’ afternoon lecture