



Software Verification

Verification of Real-time Systems

Carlo A. Furia

Program Verification: the very idea



P: a program

S: a specification

```
max (a, b: INTEGER): INTEGER
do
  if a > b then
    Result := a
  else
    Result := b
  end
end
```

```
require
  true
ensure
  Result >= a
  Result >= b
```

Does $P \models S$ hold?

The Program Verification problem:

- **Given:** a program P and a specification S
- **Determine:** if **every execution** of P , for every value of input parameters, **satisfies** S

Real-time Verification



P: a program

S: a specification

```
max (a, b: INTEGER): INTEGER
do
    if a > b then
        Result := a
    else
        Result := b
    end
end
```

```
ensure
    Result >= a
    Result >= b

ensure -- real-time
    "max terminates no sooner
    than 3 ms and no later than
    10 ms after invocation"
```

Does $P \models S$ hold?

The Real-time Verification problem:

- **Given:** program P (embedded in environment E) and real-time specification S
- **Determine:** if every execution of P (within E) satisfies S

Real-time Programs and Systems

Def. Real-time specification: specification that includes **exact timing** information.

Def. Real-time computation: computation whose specification is real-time. In other words: computation whose **correctness** depends not only on the value of the result but also on **when** the result is available.

- The **timing** of a piece of software is usually dependent on the **environment** where the computation takes place
- Hence, in real-time verification the **focus** shifts from programs to (software-intensive) **systems**
- The **purely computational** aspects can often be analyzed in isolation
- Real-time verification can then **focus on real-time** aspects of the **system**
 - e.g., synchronization, deadlines, delays, ...while abstracting away most of the rest

Decidability vs. Expressiveness Trade-Off

The Real-time Verification problem:

- **Given:** program P (embedded in environment E) and real-time specification S
- **Determine:** if **every execution** of P (within E) **satisfies** S

P : a system



$F(P)$: formal model of P

S : a real-time specification



$N(S)$: formal annotation for S

Does $F(P) \models N(S)$ hold?

- The **classes** of $F(P)$ and $N(S)$ should guarantee:
 - enough **expressiveness** to include a **quantitative** notion of **time**
 - **decidability** of the verification problem

Real-time Model-Checking



The Real-time Model Checking problem:

- **Given:** a **timed** automaton **A**
and a **metric** temporal-logic formula **F**
- **Determine:** if **every run** of **A** **satisfies F** or not
 - if **not**, also provide a **counterexample**:
a run of **A** where **F** does not hold

A: a timed automaton **A** $\stackrel{?}{\models}$ **F** **F:** a metric temporal-logic formula

- The **model-checking paradigm** is naturally **extended to real-time systems**
- Different **choices** are possible for the **family of automata** and of **formulae**
 - **Linear time** is the standard option for real-time (as opposed to branching time)
 - A different attribute of time that becomes **relevant in quantitative models** is **discrete vs. dense time**

Discrete vs. dense (continuous) time



Discrete time

- sequence of **isolated** "steps"
 - every instant has a unique **successor**
 - e.g.: the naturals $N = \{0, 1, 2, \dots\}$
-
- + **simple and intuitive**
 - + **verification usually decidable (and acceptably complex)**
 - + **robust and elegant theoretical framework**
 - **cannot model true asynchrony**
 - **unsuitable to model physical variables**

Dense (or continuous) time

- **arbitrarily small** distances
 - the successor of an instant is **not defined**
 - e.g.: the reals R
-
- + **can model true asynchrony**
 - + **accurate modeling of physical variables**
 - **tricky to understand**
 - **verification often undecidable (or highly complex)**
 - **lacks a unifying framework**



Discrete Real-time Model-Checking

Timed Automata and Metric Temporal Logic

Discrete Real-time Model-Checking

Discrete real-time model checking extends standard “untimed” model checking straightforwardly:

- **Discrete Timed Automata (TA)** extend the Finite-State Automata (FSA)
- **Metric Temporal Logic (MTL)** extends Linear Temporal Logic (LTL)

The Discrete Real-time Model Checking problem:

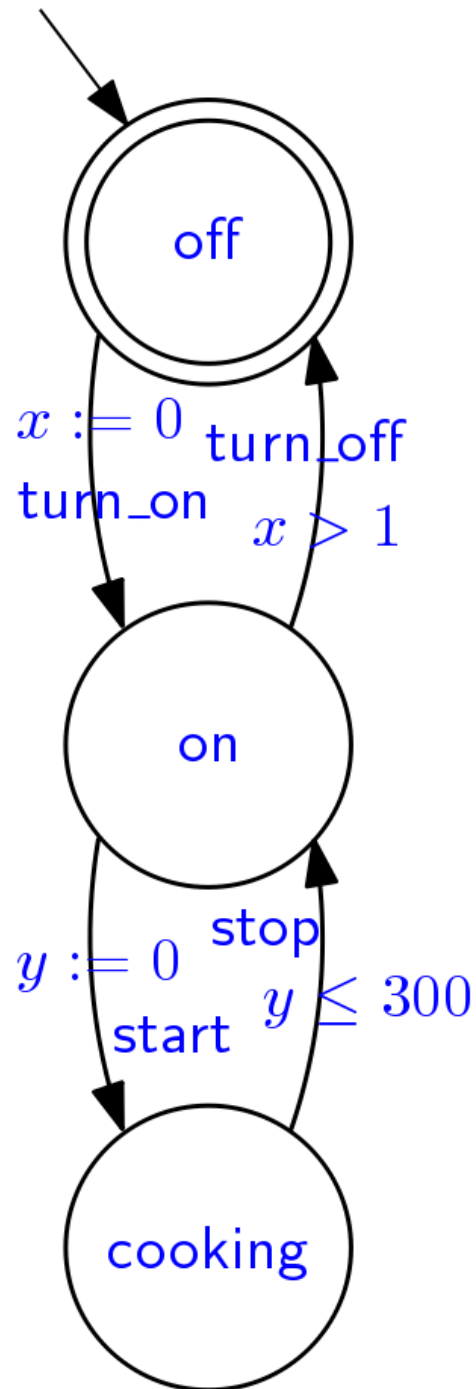
- **Given:** a **discrete TA** A and an **MTL** formula F
- **Determine:** if **every run** of A **satisfies** F or not
 - if **not**, also provide a **counterexample**: a run of A where F does not hold

A : a discrete TA

$A \models? F$

F : an MTL formula

Timed Automata: Syntax

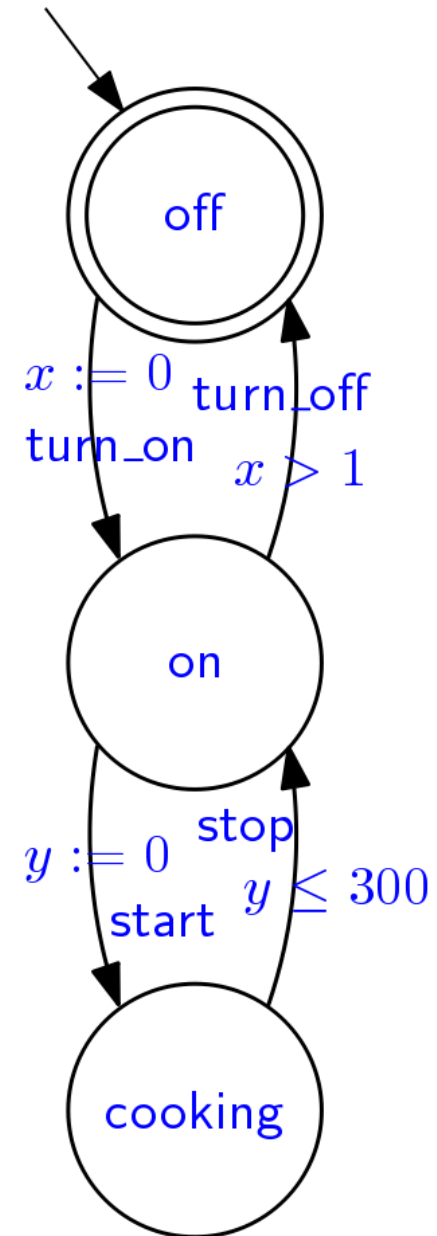


Timed Automata: Syntax

Def. Nondeterministic Timed Automaton (TA)

A tuple $[\Sigma, S, C, I, E, F]$:

- Σ : finite nonempty (input) **alphabet**
- S : finite nonempty set of **locations** (i.e., discrete states)
- C : finite set of **clocks**
- I, F : set of **initial/final** states
- E : finite set of **edges** $[s, \sigma, c, \rho, s']$
 - $s \in S$: **source** location
 - $s' \in S$: **target** location
 - $\sigma \in \Sigma$: **input** character (also "label")
 - c : **clock constraint** in the form:
 $c ::= x \approx k \mid \neg c \mid c1 \wedge c2$
 - $x, y \in C$ are clocks
 - $k \in \mathbb{N}$ is an integer constant
 - \approx is a comparison operator among $<, \leq, >, \geq, =$
 - $\rho \subseteq C$: set of clock that are **reset** (to 0)



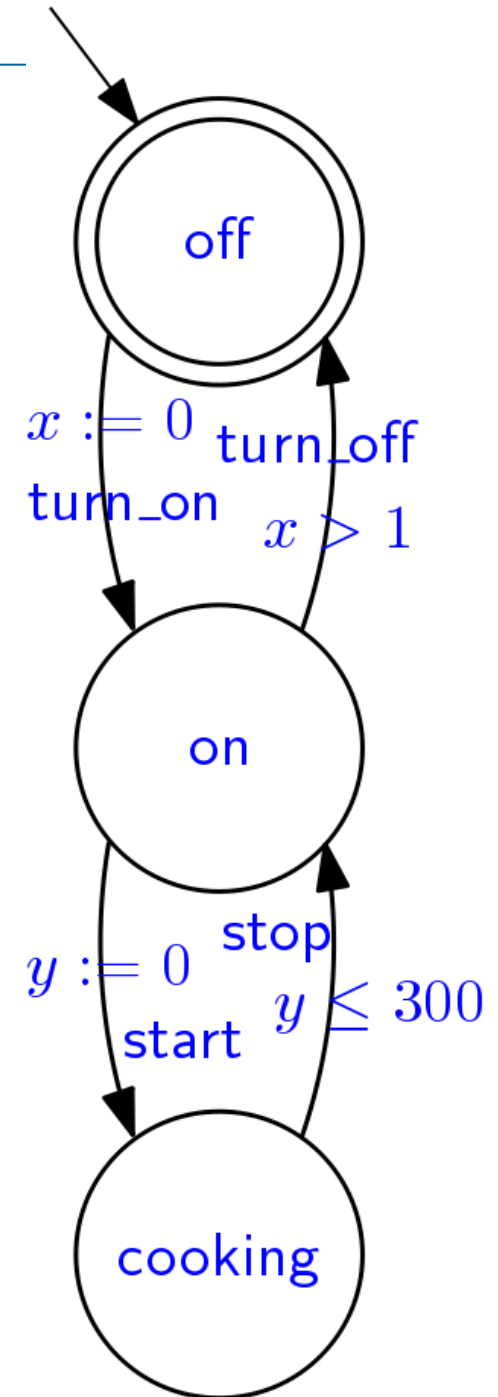
Timed Automata: Semantics

Accepting run:

$r =$ [off, (x=0, y=0)]
 [on, (x=0, y=3)]
 [cooking, (x=8, y=0)]
 [on, (x=81, y=73)]
 [off, (x=85, y=77)]

Over input **timed word**:

$w =$ [turn_on, 3]
 [start, 11]
 [stop, 84]
 [turn_off, 88]



Timed Automata: Semantics

Def. A **timed word** $w = w(1) w(2) \dots w(n) \in (\Sigma \times \mathbb{N})^*$ is a sequence of pairs $[\sigma(i), t(i)]$ such that:

- the sequence of timestamps $t(1), t(2), \dots, t(n)$ is **increasing**
- $[\sigma(i), t(i)]$ represents the i -th character $\sigma(i)$ read **at time $t(i)$**

Def. An **accepting run** of a TA $A = [\Sigma, S, C, I, E, F]$

over input timed word $w = [\sigma(1), t(1)] \dots [\sigma(n), t(n)] \in (\Sigma \times \mathbb{N})^*$ is a sequence $r = [s(0), v(0,1), \dots, v(0,|C|)] \dots [s(n), v(n,1), \dots, v(n,|C|)] \in (S \times \mathbb{N}^{|C|})^*$ of (extended) states such that:

- it **starts** from an initial and **ends** in an accepting state: $s(0) \in I, s(n) \in F$
- **initially** all clocks are reset to 0: $v(0,k) = 0$ for all $1 \leq k \leq |C|$
- for every **transition** ($0 \leq i < n$):
 $[s(i) v(i,1) \dots v(i,|C|)] \rightarrow [s(i+1) v(i+1,1) \dots v(i+1,|C|)]$
 some **edge** $[s(i), \sigma(i+1), c, \rho, s(i+1)]$ in E is followed:
 - the clock values $v(i,1) + (t(i+1) - t(i)) \dots v(i,|C|) + (t(i+1) - t(i))$ satisfy the constraint c
 - $v(i+1,k) =$ if k -th clock is in ρ then 0 else $v(i,k) + t(i+1) - t(i)$

Timed Automata: Semantics

Def. Any TA $A = [\Sigma, S, C, I, E, F]$ defines a set of input timed words $\langle A \rangle$:

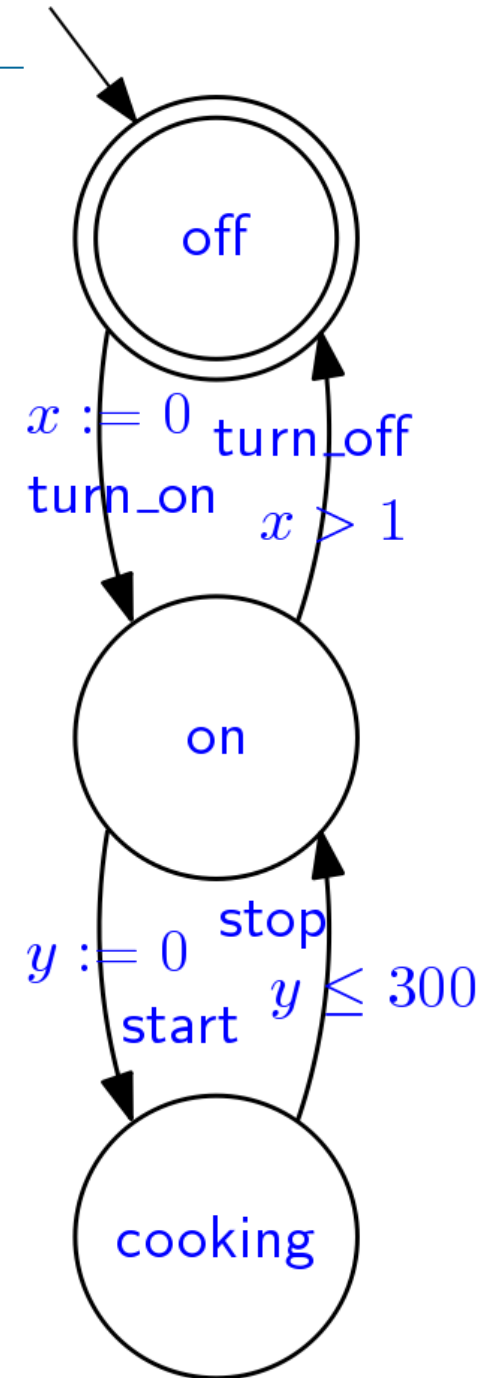
$\langle A \rangle \triangleq \{ w \in (\Sigma \times \mathbb{N})^* \mid \text{there is an accepting run of } A \text{ over } w \}$

$\langle A \rangle$ is called the language of A

With regular expressions and arithmetic:

$\langle A \rangle = ([\text{turn_on}, t_1] [\text{start}, t_2] [\text{stop}, t_3])^* [\text{turn_off}, t_4]^*$

with $t_3 - t_2 \leq 300$ and $t_4 - t_1 > 1$



Metric (Linear) Temporal Logic



$\langle \rangle [2,4)$ stop

"there is an occurrence of stop between 2 (included) and 4 (excluded) time units in the future"

- $[any, t \leq 1]^* [stop, 2] [stop, 3] [any, 4] [any, 7] \dots$
- $[any, t < 3]^* [stop, 3] [any, 4] [any, t > 4] \dots$

$[] (2,4]$ start

"start holds between 2 (excluded) and 4 (included) time units in the future"

- $[any, 0] [any, 1] [any, 2] [start, 3] [start, 4] [any, t > 4]^*$
- $[any, 0] [any, 1] [any, 2] [start, 3] [any, t > 4]^*$
- $[stop, 0] [stop, 1]$

Metric (Linear) Temporal Logic



\square (start $\Rightarrow \langle \rangle$ (3,10] stop)

“every occurrence of start is followed by an occurrence of stop between 3 (excluded) and 10 (included) time units in the future”

cook \cup (3,10] stop

“stop occurs between 3 (excluded) and 10 (included) time units in the future, and cook holds until then”

Metric (Linear) Temporal Logic: Syntax

Def. Propositional Metric Temporal Logic (MTL) formulae:

$$F ::= p \mid \neg F \mid F \wedge G \mid F U_{\langle a,b \rangle} G$$

with $p \in P$ any atomic proposition and $\langle a,b \rangle$ an interval of the time domain (w.l.o.g. with integer endpoints).

Temporal (modal) operators:

- next: $X F \triangleq \text{True } U[1,1] F$
- bounded until: $F U_{\langle a,b \rangle} G$
- bounded eventually: $\langle \rangle_{\langle a,b \rangle} F \triangleq \text{True } U_{\langle a,b \rangle} F$
- bounded always: $\square_{\langle a,b \rangle} F \triangleq \neg \langle \rangle_{\langle a,b \rangle} \neg F$
- intervals can be unbounded; e.g., $[3, \infty)$
- intervals with pseudo-arithmetic expressions; e.g.:
 - ≥ 3 for $[3, \infty)$
 - $= 1$ for $[1,1]$
 - $[0, \infty)$ is simply omitted

Metric Temporal Logic: Semantics

Def. A timed word $w = [\sigma(1), t(1)] [\sigma(2), t(2)] \dots [\sigma(n), t(n)] \in (P \times \mathbb{N})^*$ satisfies LTL formula F at position $1 \leq i \leq n$, denoted $w, i \models F$, when:

- $w, i \models p$ iff $p = \sigma(i)$
- $w, i \models \neg F$ iff $w, i \models F$ does **not** hold
- $w, i \models F \wedge G$ iff both $w, i \models F$ **and** $w, i \models G$ hold
- $w, i \models F \text{ U}_{\langle a, b \rangle} G$ iff for **some** $i \leq j \leq n$ such that $t(j) - t(i) \in \langle a, b \rangle$ it is: $w, j \models G$ and for **all** $i \leq k < j$ it is $w, k \models F$
 - i.e., F holds **until** G will hold **within** $\langle a, b \rangle$

For **derived operators**:

- $w, i \models \langle \rangle_{\langle a, b \rangle} F$ iff for **some** $i \leq j \leq n$ such that $t(j) - t(i) \in \langle a, b \rangle$ it is: $w, j \models F$
 - i.e., F holds **eventually within** $\langle a, b \rangle$
- $w, i \models []_{\langle a, b \rangle} F$ iff for **all** $i \leq j \leq n$ such that $t(j) - t(i) \in \langle a, b \rangle$ it is: $w, j \models F$
 - i.e., F holds **always within** $\langle a, b \rangle$

Metric Temporal Logic: Semantics



Def. Satisfaction:

$$w \models F \triangleq w, 1 \models F$$

i.e., timed word w satisfies formula F initially

Def. Any MTL formula F defines a set of timed words $\langle F \rangle$:

$$\langle F \rangle \triangleq \{ w \in (P \times \mathbb{N})^* \mid w \models F \}$$

$\langle F \rangle$ is called the language of F



Discrete Real-time Model-Checking

From Real-time to Untimed Model-Checking

Discrete-time Real-time Model Checking



An semantic view of the Real-time Model Checking problem:

Given: a timed automaton A and an MTL formula F

- if $\langle A \rangle \cap \langle \neg F \rangle$ is empty then every run of A satisfies F
- if $\langle A \rangle \cap \langle \neg F \rangle$ is not empty then some run of A does not satisfy F
 - any member of the nonempty intersection $\langle A \rangle \cap \langle \neg F \rangle$ is a counterexample

How to check $\langle A \rangle \cap \langle \neg F \rangle = \emptyset$ algorithmically (given A, F)?

For a discrete time domain we can reduce real-time model checking to (untimed) model-checking:

- Transform timed automaton A into finite-state automaton A'

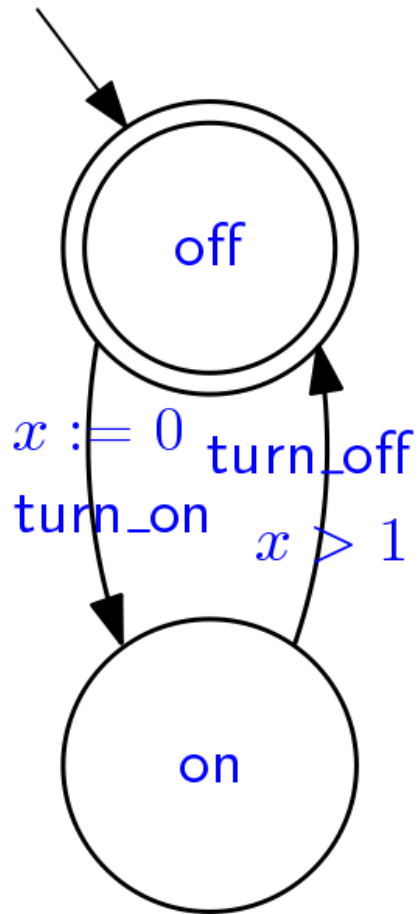
- Transform MTL formula F into LTL formula F'

$$\langle A \rangle \cap \langle \neg F \rangle = \emptyset \quad \text{iff} \quad \langle A' \rangle \cap \langle \neg F' \rangle = \emptyset$$

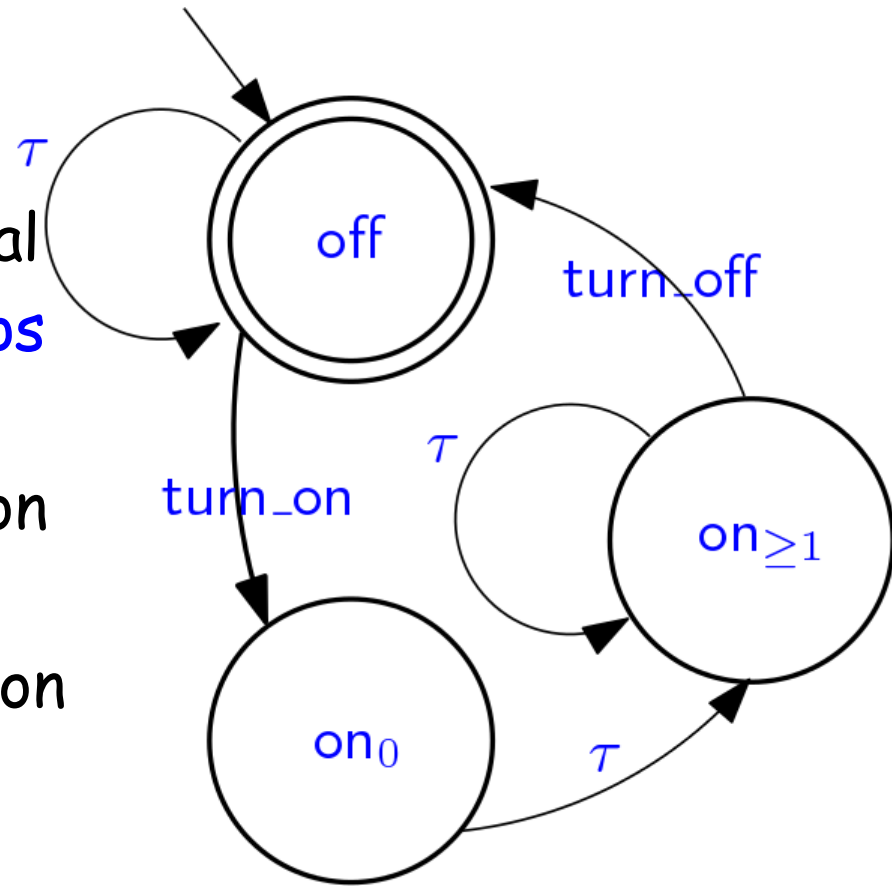
- Re-use standard model-checking algorithms

Reduce discrete-time TAs to FSAs

Use states of an FSA to "count" discrete time steps according to the semantics of the TA



- transitions with special events τ are time steps without events.
- on_0 represents location on with clock $x = 0$
- $on_{\geq 1}$ represents location on with clock $x \geq 1$



Reduce discrete-time MTL to LTL



Use next operator X to "count" discrete time steps according to the semantics of the MTL formula

- $\langle \rangle [1,3] p$ becomes $Xp \vee XXp \vee XXXp$
 - More compactly $X(p \vee X(p \vee Xp))$
- $[] \geq 5 p$ becomes $X^5 [](p \vee \tau)$
 - $X^5 p$ is a shorthand for $XXXXXp$
 - The disjunction is needed because we may have time increments without events
- The encoding for **bounded until** is a bit more complicated but not different in principle

Discrete-time Real-time MC: Complexity

There is an **exponential blow-up in complexity** when switching from (untimed) linear-time model checking to **discrete-time real-time model checking**:

- Discrete-time real-time **MTL** model checking:
EXSPACE-complete
 - in practice: **double-exponential time**
- LTL model checking: PSPACE-complete
 - in practice: singly-exponential time
- The blow up occurs only if the constants (in timed automata and MTL formulas) are **encoded succinctly in binary**
 - blow-up due to the “unrolling” of binary constants as FSA states or nested next operators



Dense Real-time Model-Checking

Timed Automata and Metric Temporal Logic

Dense Real-time Model-Checking



Dense real-time model checking considers the same model as discrete real-time model checking but with $\mathbb{R}_{\geq 0}$ as time domain:

- A **dense** Timed Automaton (TA) models the system
 - **Dense-time** Metric Temporal Logic (MTL) models the property
-
- The **syntax** of TA and MTL need not be changed for **dense time**
 - with the **possible exception** of allowing fractional time bounds
 - The **semantics** of TA and MTL is also unchanged except that:
 - $\mathbb{R}_{\geq 0}$ replaces \mathbb{N} as time domain
 - As we did with untimed model checking, we will use **finite-word** models for automata and logic.
 - **Unlike in untimed model checking, this choice** affects some results. (We will mention some details only later for simplicity.)

Dense Real-time Model-Checking

Dense real-time model checking extends standard “untimed” model checking:

- **Timed Automata (TA)** extend Finite-State Automata (FSA)
- **Metric Temporal Logic (MTL)** extends Linear Temporal Logic (LTL)

The Dense Real-time Model Checking problem:

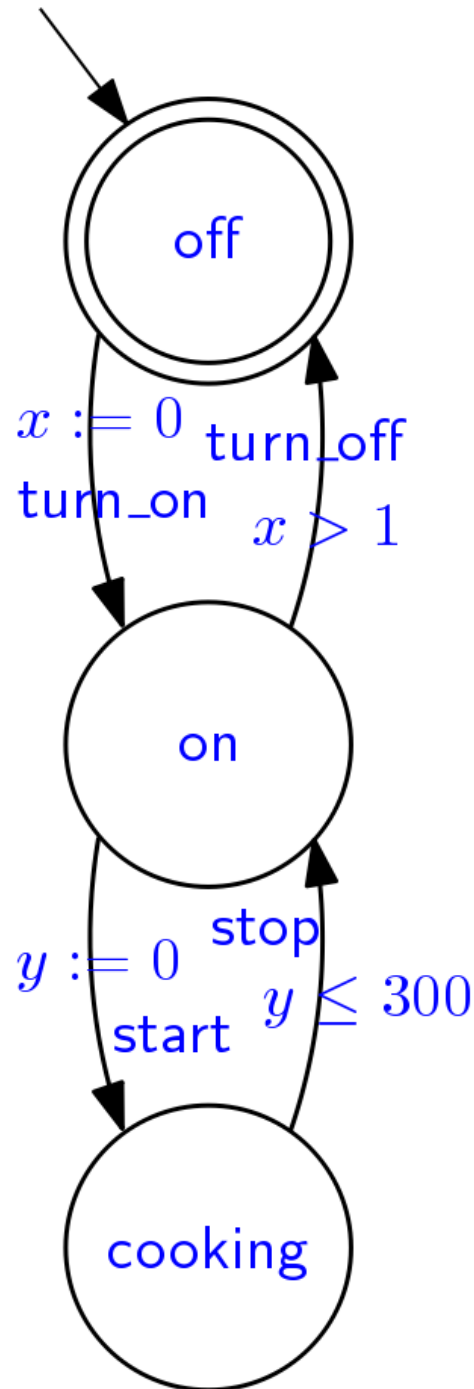
- **Given:** a dense TA A and an MTL formula F
- **Determine:** if every run of A satisfies F or not
 - if not, provide a counterexample: a run of A where F does not hold

A : a TA

$A \stackrel{?}{\models} F$

F : an MTL formula

Timed Automata: Syntax

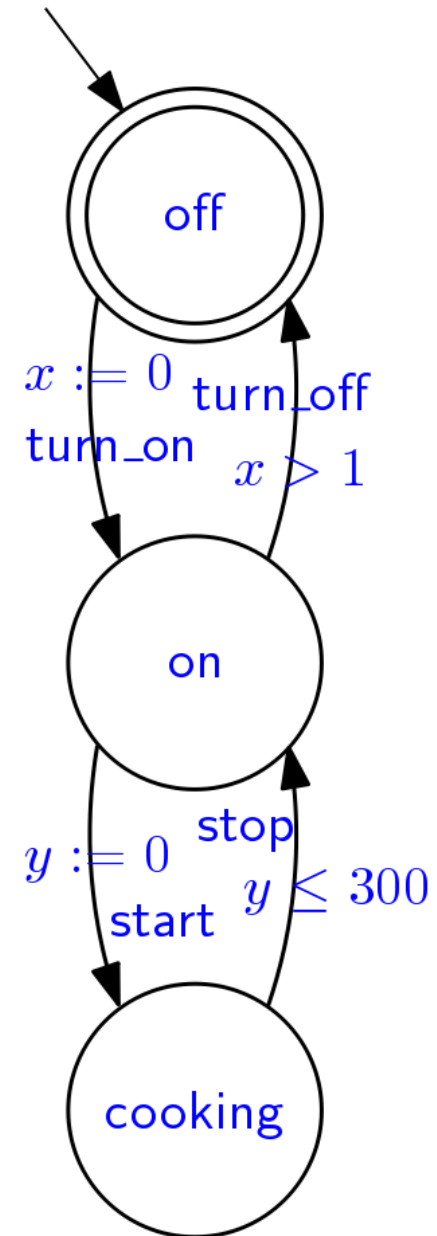


Timed Automata: Syntax

Def. Nondeterministic Timed Automaton (TA):

a tuple $[\Sigma, S, C, I, E, F]$:

- Σ : finite nonempty (input) **alphabet**
- S : finite nonempty set of **locations** (i.e., discrete states)
- C : finite set of **clocks**
- I, F : set of **initial/final** states
- E : finite set of **edges** $[s, \sigma, c, \rho, s']$
 - $s \in S$: **source** location
 - $s' \in S$: **target** location
 - $\sigma \in \Sigma$: **input** character (also "label")
 - c : **clock constraint** in the form:
 $c ::= x \approx k \mid \neg c \mid c1 \wedge c2$
 - $x, y \in C$ are clocks
 - $k \in \mathbb{N}$ is an integer constant
 - \approx is a comparison operator among $<, \leq, >, \geq, =$
 - $\rho \subseteq C$: set of clock that are **reset** (to 0)



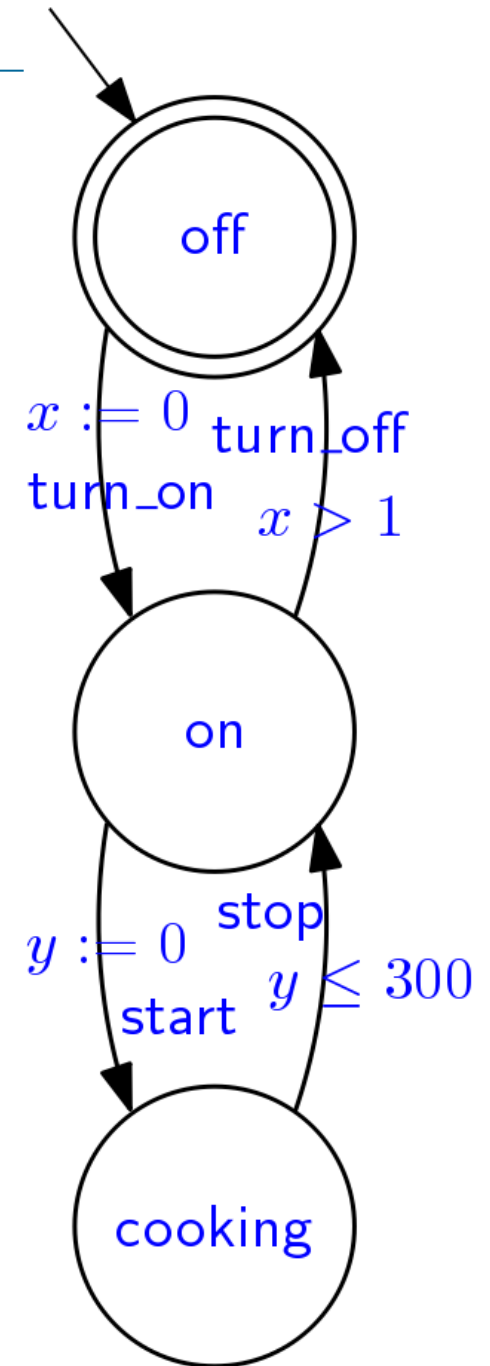
Timed Automata: Semantics

Accepting run:

$r =$ [off, (x=0, y=0)]
[on, (x=0, y=3.2)]
[cooking, (x=8.5, y=0)]
[on, (x=81.7, y=73.2)]
[off, (x=84.91, y=76.41)]

Over input **timed word**:

$w =$ [turn_on, 3.2]
[start, 11.7]
[stop, 84.9]
[turn_off, 88.11]



Timed Automata: Semantics

Def. A **timed word** $w = w(1) w(2) \dots w(n) \in (\Sigma \times \mathbb{R})^*$ is a sequence of pairs $[\sigma(i), t(i)]$ such that:

- the sequence of timestamps $t(1), t(2), \dots, t(n)$ is **increasing**
- $[\sigma(i), t(i)]$ represents the i -th character $\sigma(i)$ read **at time $t(i)$**

Def. An **accepting run** of a TA $A = [\Sigma, S, C, I, E, F]$ over input timed word $w = [\sigma(1), t(1)] \dots [\sigma(n), t(n)] \in (\Sigma \times \mathbb{R})^*$ is a sequence $r = [s(0), v(0,1), \dots, v(0,|C|)] \dots [s(n), v(n,1), \dots, v(n,|C|)] \in (S \times \mathbb{R}^{|C|})^*$ of (extended) states such that:

- it **starts** from an initial and **ends** in an accepting state: $s(0) \in I, s(n) \in F$
- **initially** all clocks are reset to 0: $v(0,k) = 0$ for all $1 \leq k \leq |C|$
- for every **transition** ($0 \leq i < n$):
 $[s(i) v(i,1) \dots v(i,|C|)] \rightarrow [s(i+1) v(i+1,1) \dots v(i+1,|C|)]$
some **edge** $[s(i), \sigma(i+1), c, \rho, s(i+1)]$ in E is followed:
 - the clock values $v(i,1) + (t(i+1) - t(i)) \dots v(i,|C|) + (t(i+1) - t(i))$ satisfy the constraint c
 - $v(i+1,k) =$ if k -th clock is in ρ then 0 else $v(i,k) + t(i+1) - t(i)$

Timed Automata: Semantics

Def. Any TA $A = [\Sigma, S, C, I, E, F]$ defines a set of input timed words $\langle A \rangle$:

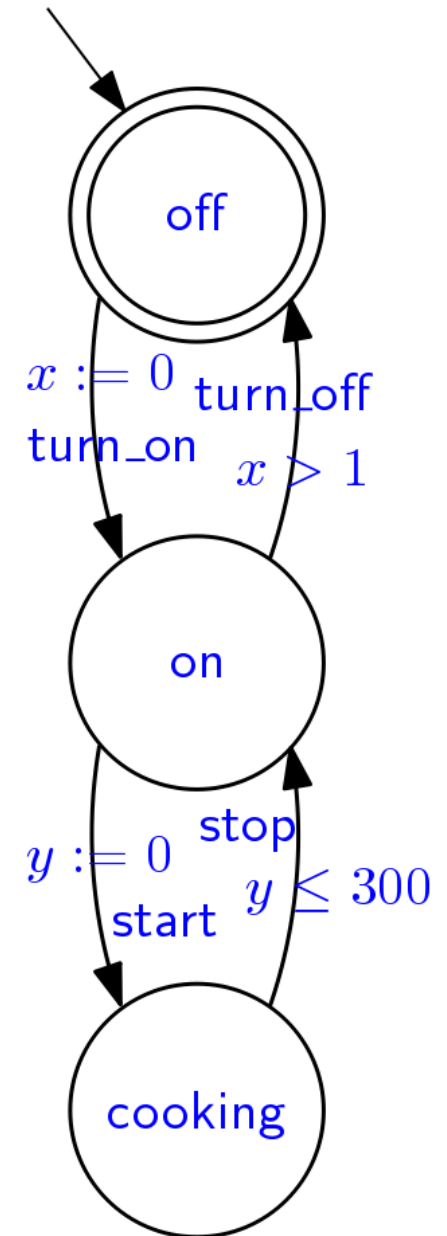
$$\langle A \rangle \triangleq \{ w \in (\Sigma \times \mathbb{R})^* \mid \text{there is an accepting run of } A \text{ over } w \}$$

$\langle A \rangle$ is called the language of A

With regular expressions and arithmetic:

$$\langle A \rangle = ([\text{turn_on}, t_1] [\text{start}, t_2] [\text{stop}, t_3])^* [\text{turn_off}, t_4]^*$$

$$\text{with } t_3 - t_2 \leq 300 \text{ and } t_4 - t_1 > 1$$



Metric (Linear) Temporal Logic



$\langle \rangle [2,4)$ stop

"there is an occurrence of stop between 2 (included) and 4 (excluded) time units in the future"

- $[any, t < 2]^* [stop, 2] [stop, 3] [any, 3.5] [any, 3.7] \dots$
- $[any, t < 3.99]^* [stop, 3.99] [any, 4] [any, t > 4] \dots$

$[] (2,4]$ start

"start holds between 2 (excluded) and 4 (included) time units in the future"

- $[any, t \leq 2] [start, 2.2] [start, 3] [start, 4] [any, t > 4] \dots$
- $[any, t \leq 2] [start, 4] [any, t > 4] \dots$
- $[stop, 0] [stop, 0.3] [stop, 0.7]$

Metric (Linear) Temporal Logic



\square (start $\Rightarrow \langle \rangle$ (3,10] stop)

“every occurrence of start is followed by an occurrence of stop between 3 (excluded) and 10 (included) time units in the future”

cook \cup (3,10] stop

“stop occurs between 3 (excluded) and 10 (included) time units in the future, and cook holds until then”

Metric (Linear) Temporal Logic: Syntax

Def. Propositional Metric Temporal Logic (MTL) formulae:

$$F ::= p \mid \neg F \mid F \wedge G \mid F U\langle a, b \rangle G$$

with $p \in P$ any atomic proposition and $\langle a, b \rangle$ an interval of the time domain (w.l.o.g. with integer endpoints).

Temporal (modal) operators:

- next: $X F \triangleq \text{True } U[1,1] F$
- bounded until: $F U\langle a, b \rangle G$
- bounded eventually: $\langle \rangle \langle a, b \rangle F \triangleq \text{True } U\langle a, b \rangle F$
- bounded always: $[]\langle a, b \rangle F \triangleq \neg \langle \rangle \langle a, b \rangle \neg F$
- intervals can be unbounded; e.g., $[3, \infty)$
- intervals with pseudo-arithmetic expressions; e.g.:
 - ≥ 3 for $[3, \infty)$
 - $= 1$ for $[1,1]$
 - $[0, \infty)$ is simply omitted

Metric Temporal Logic: Semantics



Def. A timed word $w = [\sigma(1), t(1)] [\sigma(2), t(2)] \dots [\sigma(n), t(n)] \in (P \times \mathbb{R})^*$ satisfies LTL formula F at position $1 \leq i \leq n$, denoted $w, i \models F$, when:

- $w, i \models p$ iff $p = \sigma(i)$
- $w, i \models \neg F$ iff $w, i \models F$ does **not** hold
- $w, i \models F \wedge G$ iff both $w, i \models F$ **and** $w, i \models G$ hold
- $w, i \models F \text{ U}_{\langle a, b \rangle} G$ iff for **some** $i \leq j \leq n$ such that $t(j) - t(i) \in \langle a, b \rangle$ it is: $w, j \models G$ and for **all** $i \leq k < j$ it is $w, k \models F$
 - i.e., F holds **until** G will hold **within** $\langle a, b \rangle$

For **derived operators**:

- $w, i \models \langle \rangle_{\langle a, b \rangle} F$ iff for **some** $i \leq j \leq n$ such that $t(j) - t(i) \in \langle a, b \rangle$ it is: $w, j \models F$
 - i.e., F holds **eventually within** $\langle a, b \rangle$
- $w, i \models []_{\langle a, b \rangle} F$ iff for **all** $i \leq j \leq n$ such that $t(j) - t(i) \in \langle a, b \rangle$ it is: $w, j \models F$
 - i.e., F holds **always within** $\langle a, b \rangle$

Metric Temporal Logic: Semantics



Def. Satisfaction:

$$w \models F \triangleq w, 1 \models F$$

i.e., timed word w satisfies formula F initially

Def. Any MTL formula F defines a set of timed words $\langle F \rangle$:

$$\langle F \rangle \triangleq \{ w \in (P \times R)^* \mid w \models F \}$$

$\langle F \rangle$ is called the language of F



Dense Real-time Model-Checking

What's Decidable?

Automata-theoretic real-time model-checking?

Let's try to extend the automata-theoretic model checking paradigm to real-time.

Its three algorithmic components:

- **MTL2TA**: given MTL formula F build TA $a(F)$ such that $\langle F \rangle = \langle a(F) \rangle$
- **TA-Intersection**: given TAs A, B build TA C such that $\langle A \rangle \cap \langle B \rangle = \langle C \rangle$
- **TA-Emptiness**: given TA A check whether $\langle A \rangle = \emptyset$ is the case

Which of these algorithms are feasible over real-time?

TAs not Closed under Complement

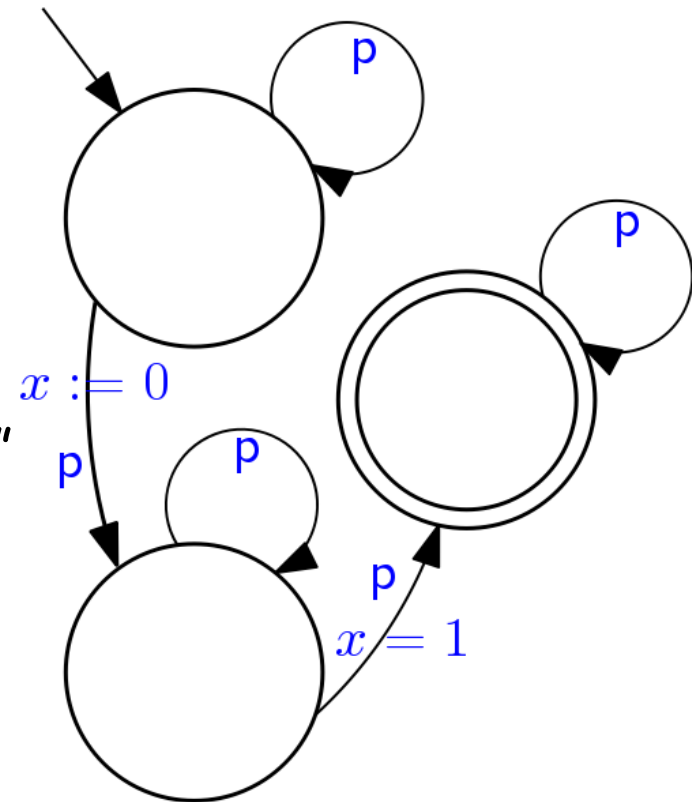
A : a dense TA $A \stackrel{?}{\models} F$ F : a dense-time MTL formula

Fundamental problem:

Dense timed automata are **not closed under complement**

The **complement** of the language of this TA **isn't accepted by any TA**:

- **language** of this TA:
"there exist two p 's separated by one t.u."
- **complement** language:
"no two p 's are separated by one t.u."
- **intuition**: need a clock for each p within the past time unit, but there can be an **unbounded** number of such p 's because time is dense

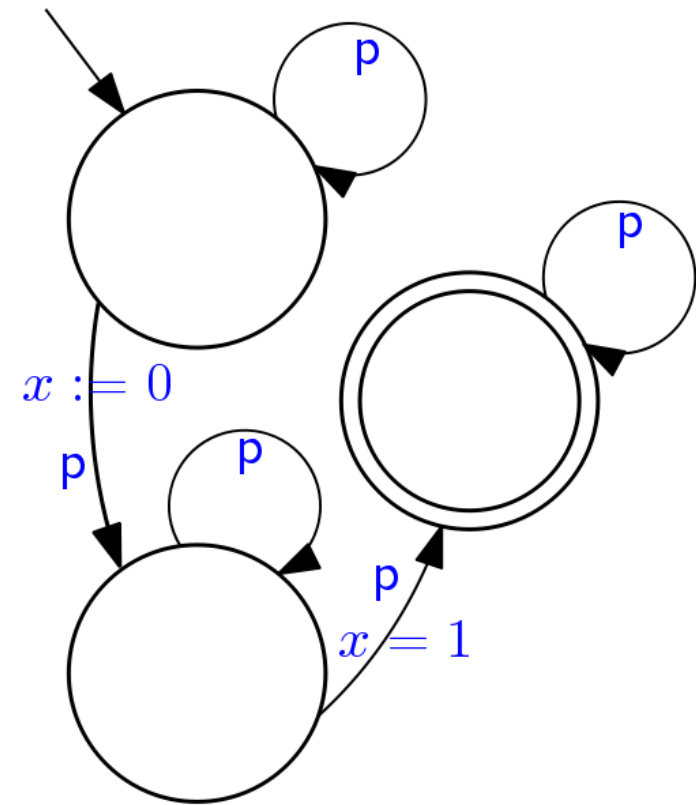


TAs not Closed under Complement



Fundamental problem:

- Dense TAs are **not closed under complement**
- **MTL** is clearly **closed under complement**
 - Language of the TA: $\langle \rangle (p \wedge \langle \rangle = 1 p)$
 - **Complement** language of the TA:
 $\neg \langle \rangle (p \wedge \langle \rangle = 1 p) = \langle \rangle (p \Rightarrow \neg \langle \rangle = 1 p)$
- Hence, automata-theoretic dense real-time model-checking is **unfeasible** (in general)



Dense MTL Model Checking is Undecidable



An even more fundamental problem:

The **dense-time model-checking problem** for MTL and TAs is **undecidable** (for **infinite** words)

- no approach is going to work, not just the automata-theoretic one

MTL and TAs are “**too expressive**” over dense time

What's Decidable about Timed Automata

Let's revisit the three algorithmic components of automata-theoretic model checking:

- **MTL2TA**: given MTL formula F build TA $a(F)$ such that $\langle F \rangle = \langle a(F) \rangle$
 - **undecidable** problem*
- **TA-Intersection**: given TAs A, B build TA C such that $\langle A \rangle \cap \langle B \rangle = \langle C \rangle$
 - **decidable**
- **TA-Emptiness**: given TA A check whether $\langle A \rangle = \emptyset$ is the case
 - **decidable!**

* (for infinite words: see clarification later)



Dense Real-time Model-Checking

Intersection of Timed Automata

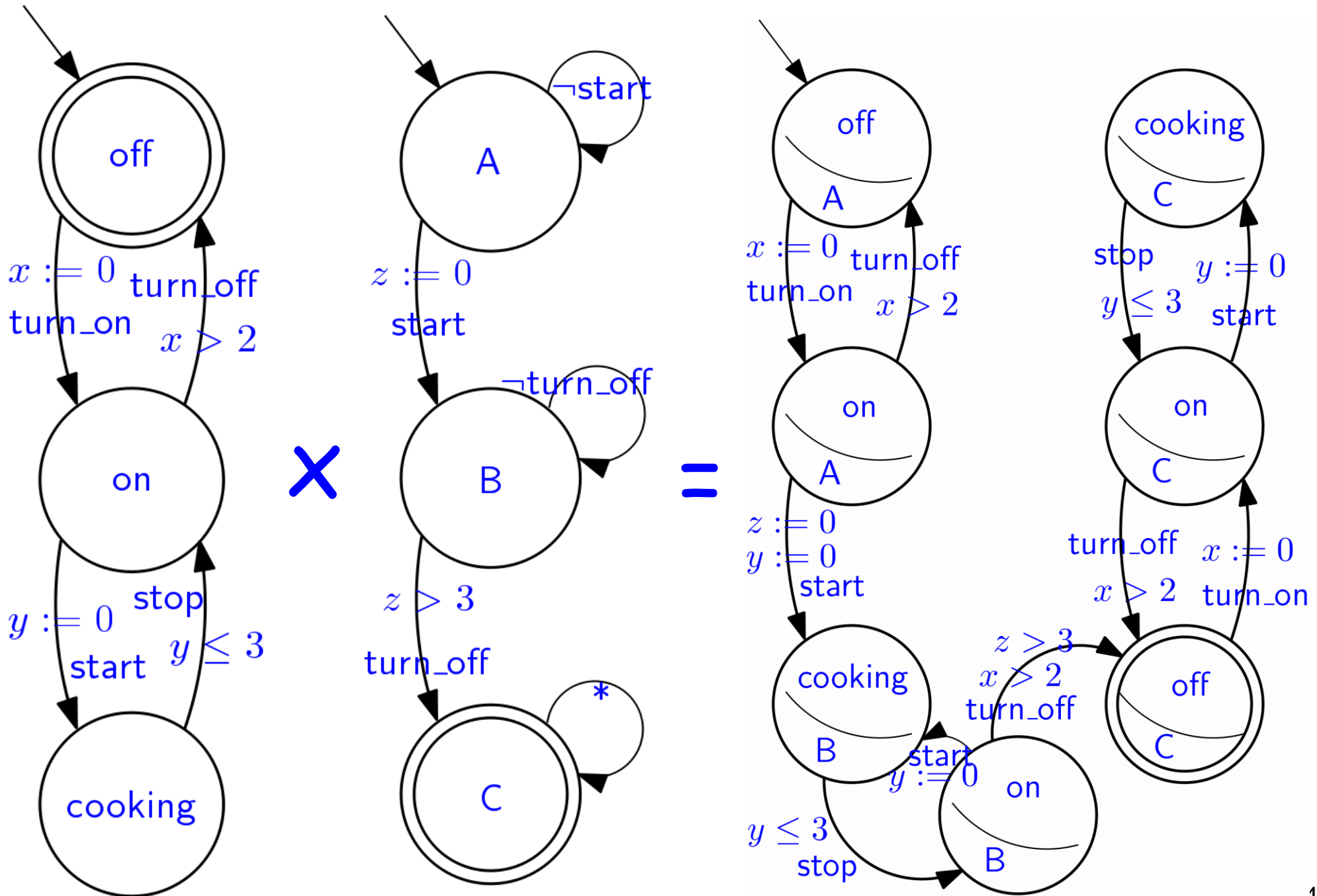
TA-Intersection: running TAs in parallel



Given TAs A , B it is always possible to **build** automatically a TA C that **accepts** precisely the **words** that **both** A and B accept.

TA C represents all possible **parallel runs** of A and B where a timed word is accepted if and only if both A and B would accept it. The construction is called "**product automaton**".

TA-Intersection: Example



TA-Intersection: running TAs in parallel

Def. Given TAs $A = [\Sigma, S^A, C^A, I^A, E^A, F^A]$ and $B = [\Sigma, S^B, C^B, I^B, E^B, F^B]$
let $C \triangleq A \times B \triangleq [\Sigma, S^C, C^C, I^C, E^C, F^C]$ be defined as:

- $S^C \triangleq S^A \times S^B$
- $C^C \triangleq C^A \cup C^B$ (assuming w.l.o.g. that they are disjoint sets)
- $I^C \triangleq \{ (s, t) \mid s \in I^A \text{ and } t \in I^B \}$
- $[(s, t), \sigma, c^A \wedge c^B, \rho^A \cup \rho^B, (s', t')] \in E^C$ iff
 $[s, \sigma, c^A, \rho^A, s'] \in E^A$ and $[t, \sigma, c^B, \rho^B, t'] \in E^B$
- $F^C \triangleq \{ (s, t) \mid s \in F^A \text{ and } t \in F^B \}$

Theorem.

$\langle A \times B \rangle$

=

$\langle A \rangle \cap \langle B \rangle$



Dense Real-time Model-Checking

Checking the Emptiness of Timed Automata

TA-Emptiness

Given a TA A it is always possible to **check** automatically if it **accepts some timed word**.

Outline of the **algorithm**:

- Assume that clock constraints involve **integer constants** only
- Define an **equivalence relation** over **extended states** (location + clocks)
- All extended states in the same equivalence class are **equivalent w.r.t.** satisfaction of **clock constraints**
 - The equivalence relation is such that there is a **finite number of equivalence classes** for any given TA
- Given a TA A , build an FSA $\text{reg}(A)$ - the "**region automaton**":
 - the **states** of $\text{reg}(A)$ represent the **equivalence classes** of the extended states of any run of A
 - the **edges** of $\text{reg}(A)$ represent **evolution of any extended state** within the equivalence class over any run of A
- Checking the **emptiness of $\text{reg}(A)$** is **equivalent** to checking A 's **emptiness**

Integer vs. Rational vs. Irrational

The domain for time is $\mathbb{R}_{\geq 0}$

What about the domain for time constraints?

– constants in clock constraints of TAs (e.g.: $x < k$)

1. Same as the domain for time: $\mathbb{R}_{\geq 0}$

- $x < \pi$
- emptiness becomes undecidable!

2. Discrete time domain: integers \mathbb{Z}

- e.g.: $x < 5$
- emptiness fully decidable (see algorithm next)

3. Dense but not continuous: rationals $\mathbb{Q}_{\geq 0}$

- $x < 1/3$
- emptiness is reducible to the discrete case

Integer vs. Rational

Dense but not continuous: rationals $\mathbb{Q}_{\geq 0}$

- Let A be a TA with rational constants
 - let m be the least common multiple of denominators of all constants appearing in the clock constraints of A
 - let A^*m be the TA obtained from A by multiplying every constants in the clock constraints by m
 - A^*m has only integers constants in its clock constraints
- A accepts any timed word
 $[\sigma(1), t(1)] [\sigma(2), t(2)] \dots [\sigma(n), t(n)]$
iff A^*m accepts the "scaled" timed word
 $[\sigma(1), m^*t(1)] [\sigma(2), m^*t(2)] \dots [\sigma(n), m^*t(n)]$
- Hence checking the emptiness of A^*m is equivalent to checking the emptiness of A

Equivalence Relation over Extended States



Let us fix a TA $A = [\Sigma, S, C, I, E, F]$ with $C = [x(1), \dots, x(n)]$

- For any clock $x(i)$ in C let $M(i)$ be the largest constant involving clock $x(i)$ in any clock constraint in E
- Let $[v(1), \dots, v(n)] \in \mathbb{R}_{\geq 0}^n$ denote a "clock evaluation" representing any assignment of values to clocks
- **Equivalence** of two clock evaluations:
 $[v(1), \dots, v(n)] \sim [v'(1), \dots, v'(n)]$ iff all of the following hold:
 1. For all $1 \leq i \leq n$: $\text{int}(v(i)) = \text{int}(v'(i))$ or $v(i), v'(i) > M(i)$
 2. For all $1 \leq i, j \leq n$ such that $v(i) \leq M(i)$ and $v(j) \leq M(j)$:
 $\text{frac}(v(i)) \leq \text{frac}(v(j))$ iff $\text{frac}(v'(i)) \leq \text{frac}(v'(j))$
 3. For all $1 \leq i \leq n$ such that $v(i) \leq M(i)$:
 $\text{frac}(v(i)) = 0$ iff $\text{frac}(v'(i)) = 0$

Note: $\text{int}(x)$ is the integer part of x ;
 $\text{frac}(x)$ is the fractional part of x

For example: $\text{int}(3.12) = 3$ $\text{frac}(3.12) = 0.12$

Clock Regions



Def. A clock region is an equivalence class of clock evaluations induced by the equivalence relation \sim

- For a clock evaluation $v = [v(1), \dots, v(n)] \in \mathbb{R}_{\geq 0}^n$, $[[v]]$ denotes the clock region v belongs to
- As a consequence of the definition of \sim , any clock region can be uniquely characterized by a finite set of constraints on clocks
 - $v = [0.4; 0.9; 0.7; 0]$ for 4 clocks w, x, y, z
 - $[[v]]$ is $z = 0 < w < y < x < 1$
- Fact: clock regions are always in finite number

Clock Regions (cont'd)



More *systematically*:

- given a set of clocks $C = [x(1), \dots, x(n)]$
- with $M(i)$ the largest constant appearing in constraints on clock $x(i)$

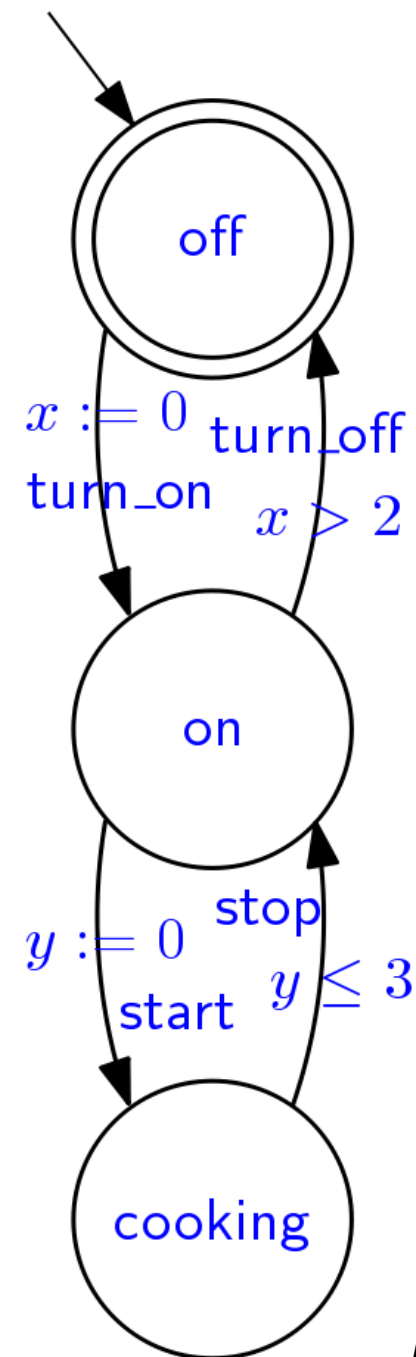
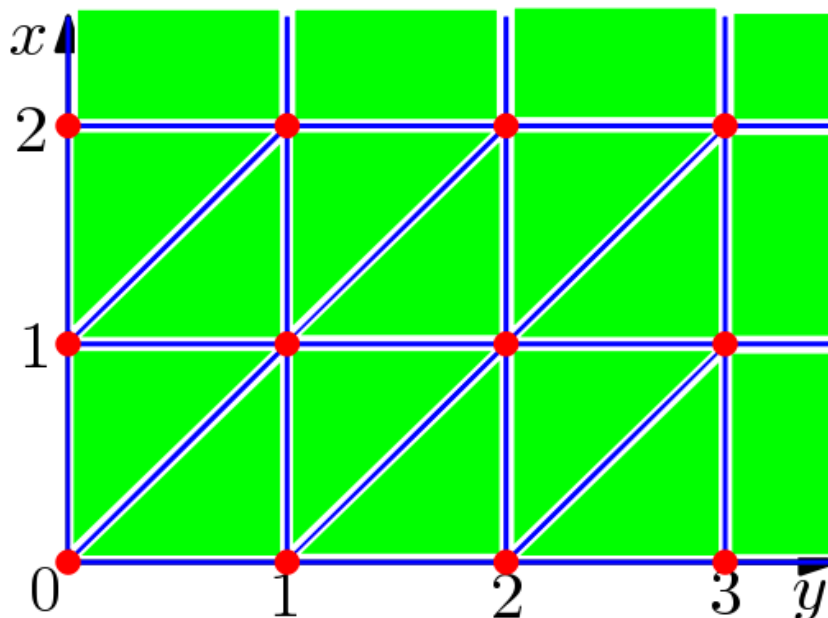
a **clock region** is *uniquely characterized* by

- For each clock $x(i)$ a constraint in the form:
 - $x(i) = c$ with $c = 0, 1, \dots, M(i)$; or
 - $c - 1 < x(i) < c$ with $c = 1, \dots, M(i)$; or
 - $x(i) > M(i)$
- For each pair of clocks $x(i), x(j)$ a constraint in the form
 - $\text{frac}(x(i)) < \text{frac}(x(j))$
 - $\text{frac}(x(i)) = \text{frac}(x(j))$
 - $\text{frac}(x(i)) > \text{frac}(x(j))$

(These are unnecessary if $x(i) = c, x(j) = c, x(i) > M(i),$ or $x(j) > M(j)$)

Clock Regions: Example

- Clocks $C = [x, y]$
- $M(x) = 2; M(y) = 3$
- All 60 possible clock regions:
 - 12 corner points
 - 30 open line segments
 - 18 open regions



Time-successors of Regions

Fact: a clock evaluation v satisfies a clock constraint c iff every other clock evaluation in $[[v]]$ satisfies c

Hence, we can say that a "clock region satisfies a clock constraint"

Def. The **time successor** $\text{time-succ}(R)$ of a clock region R is the set of all **clock regions** (including R itself) that **can be reached from R** by **letting time pass** (i.e., without resetting any clock).

Given a clock region R it is always possible to compute all other clock regions that **can be reached from R** by **letting time pass** (i.e., without resetting any clock)

Graphically:

the time-successors of a region R are the regions that can be reached by moving along a **line parallel to the diagonal** in the **upward direction**, starting from any point in R

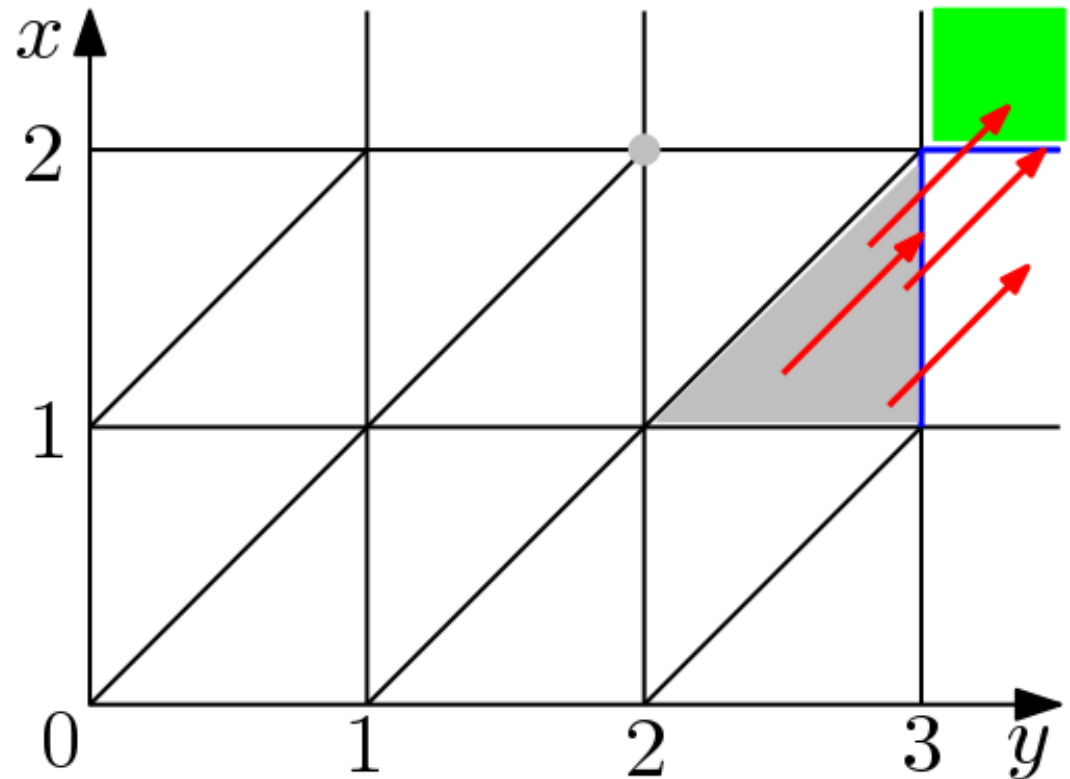
(For a **formal definition** see e.g.: Alur & Dill, 1994)

Time-successors of Regions: Example

Graphically: the time-successors of a region R are the regions that can be reached by moving along a line parallel to the diagonal in the upward direction, starting from any point in R

Example:

- successors of region $2 < y < 3; 1 < x < y-1$ (other than the region itself):
 - $y > 3; 1 < x < 2$
 - $y > 3; x = 2$
 - $y = 3; 1 < x < 2$
 - $y > 3; x > 2$
- successors of region $y = 2; x = 2$ (other than the region itself):
 - $2 < y < 3; x > 2$
 - ...



Region Automaton Construction

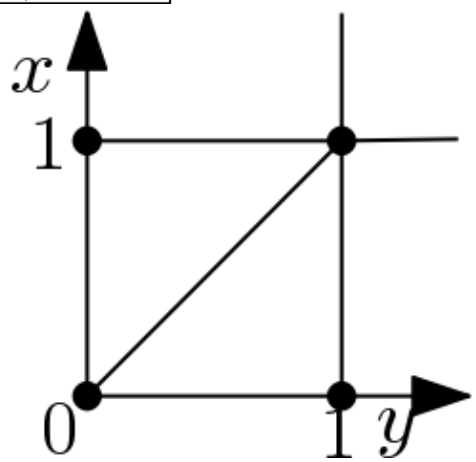
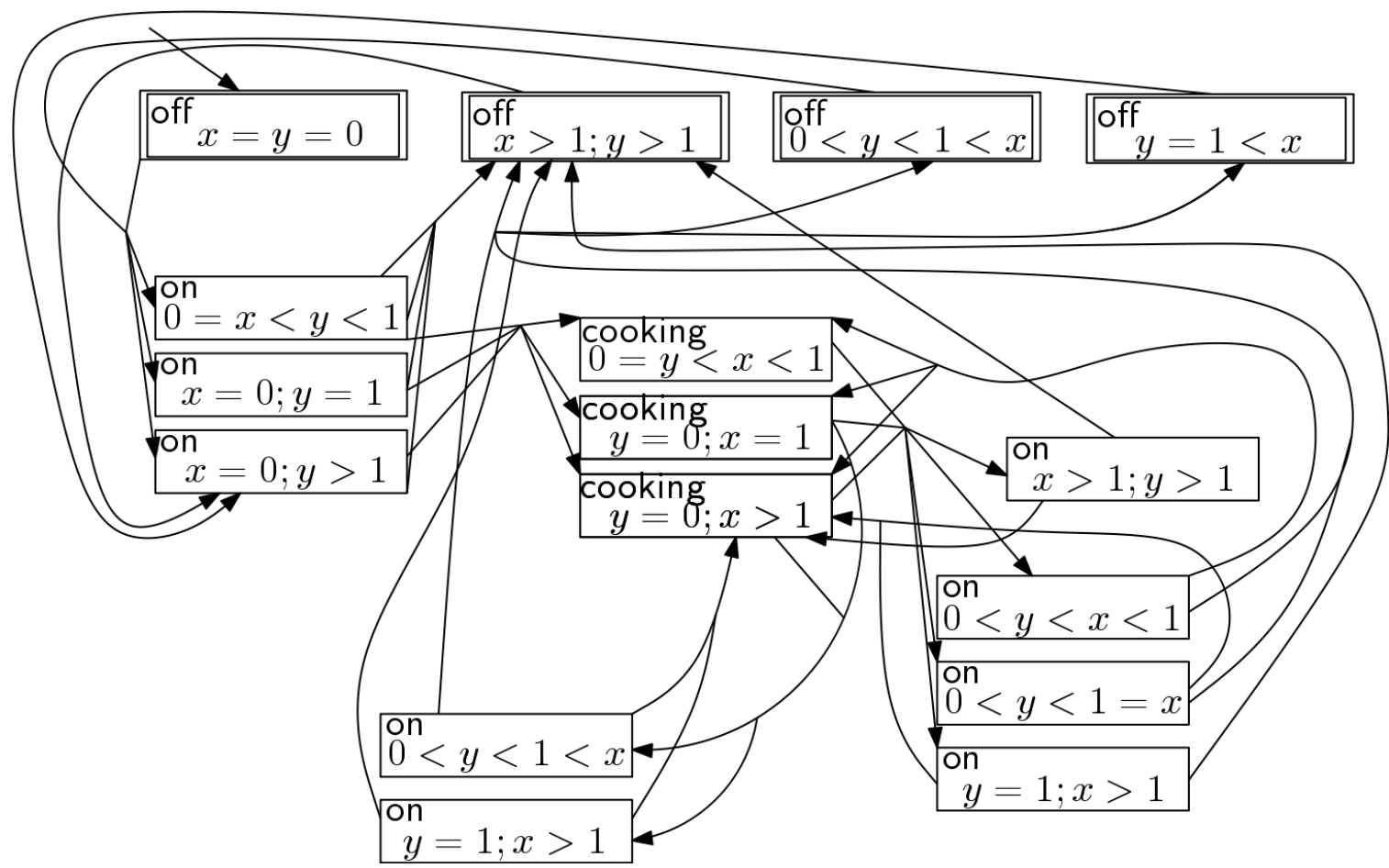
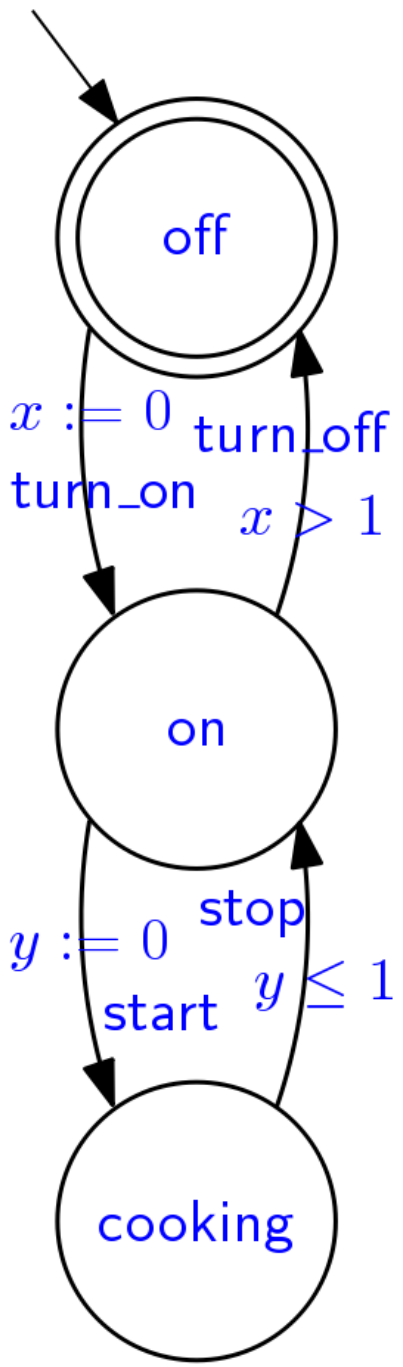
For a timed automaton A it is always possible to build an FSA $\text{reg}(A)$ (the "region automaton" of A) such that:

$$\langle A \rangle = \emptyset \quad \text{iff} \quad \langle \text{reg}(A) \rangle = \emptyset$$

Def. Given a TA $A = [\Sigma, S, C, I, E, F]$ its region automaton $\text{reg}(A) \triangleq [\Sigma, rS, rI, rE, rF]$ is defined as:

- $rS \triangleq \{ (s, r) \mid s \in S \text{ and } r \text{ is a clock region} \}$
- $rI \triangleq \{ (s, [[0, 0, \dots, 0]]) \mid s \in I \}$
 - the clock region where all clocks are reset to 0
- $rE(\sigma, [s, r]) \triangleq \{ (s', r') \mid [s, \sigma, c, \rho, s'] \in E \}$
and there exists a region $r'' \in \text{time-succ}(r)$
such that r'' satisfies c , and r' is obtained
from r'' by resetting all clocks in ρ to 0 }
- $rF \triangleq \{ (s, r) \mid s \in F \}$

Region Automaton: Example





Dense Real-time Model-Checking

Complexity, Variants, and Tools



Complexity of Emptiness Checking for TAs

- Building the region automaton and **checking** its **emptiness** takes **time exponential** in the size of the clock constraints
- **Checking emptiness** of a **TA** is a **PSPACE-complete** problem
 - Hence the **region-automaton** algorithm is **worst-case optimal**
- However, **variants** of the emptiness checking algorithm can **achieve better performances** in practice
 - mostly by using **ad hoc data structures** and **symbolic representations of regions** that can be manipulated efficiently



Variants of TA Emptiness Checking

Variants of the **Emptiness Checking Algorithm** are typically based on more efficient (on average) **representations of regions**

- **Representatives**
 - a clock region is represented by a **concrete extended state** that belongs to it
 - the concrete state is a **"representative"** of the region
 - if it is suitably chosen, manipulating it is **equivalent** to manipulating the whole region
- **Clock constraints** (a.k.a. **zones**)
 - a region is represented symbolically as a **Boolean combination of clock constraints**
 - **successors** are computed symbolically **directly** on the **Boolean expression**
- Other **equivalence relations** (e.g., **bisimulation**)
 - they can produce **fewer equivalence classes**



Tools for the Analysis of TAs

- Uppaal (Larsen, Petterson, Yi et al., ~1995)
- Kronos (Tripakis, Yovine et al., ~1995)
- HyTech (Henzinger et al., ~1994)
- PHAVer (Frehse, ~2005)

Remark: emptiness checking is also called
"reachability analysis"

the language of a TA A is empty iff the accepting states of A cannot be reached in any computation



Dense Real-time Model-Checking

Getting Decidability Back

Decidable Dense Real-time MC

Model checking is **undecidable** over dense-time infinite words
for **TAs** and **MTL** formulas

As usual, we can **trade-off** some **expressiveness** in exchange for decidability.

In particular, not mutually exclusively:

- **Syntactic restrictions:** use a real-time temporal logic with **less expressiveness**
- **Semantic restrictions:** **restrict** (the **density** of) the time domain in some way
 - discretization
 - finite words
 - bounded variability
 - bounded time

Reducing the Expressiveness of MTL



There exist different real-time temporal logics for which dense-time model checking is decidable.

Some examples:

- Strict subsets of MTL:
 - MITL: MTL without punctual (i.e., singleton) intervals (Alur, Henzinger; Hirshfeld, Rabinovich et al.)
 - BMTL, SMTL, ... (Ouaknine, Worrell et al.)
- Branching-time real-time logics:
 - TCTL (Henzinger, Nicollin, Sifakis, Yovine, et al.)

Discretization of Dense Real-time M-C

Build **approximations** of TAs and MTL dense-time semantics over **discrete time**, such that **some** results of the discrete-time analysis **apply to the dense-time semantics as well**.

In general these approaches are **incomplete**, that is they **can't be applied** to certain classes of formulas or they ignore certain classes of dense timed word.

- **Digitization** (Henzinger, Manna, Pnueli, 1992)
- **Sampling** (F., 2006)

Restrict the Semantics to Finite Words

Real-time model-checking of TAs and MTL is undecidable for infinite timed words

- infinite sequences of timestamped input symbols

It is **decidable for finite words**

(which we used in formally defining the semantics)

This result came somewhat **unexpectedly** in ~2005 (**Ouaknine & Worrell**) and it **contradicted** the “folk belief” that the undecidability for infinite words carries over to finite words

Restrict the Semantics to Finite Words

There are various reasons, however, that **lessen the practical (and didactical) relevance** of this decidability result. Mainly:

- While decidable, the problem has **non-primitive recursive complexity**
 - as complex as a computable function can be!
- The **(current) algorithm** for decidability is nontrivial and **difficult to present concisely**
 - it uses techniques different than the region automaton construction for TAs
 - no efficient symbolic techniques have been developed yet

Bounded Variability and Time

Other semantic restrictions to dense time that makes that model-checking problem decidable (over infinite time as well)

- **Bounded variability:**
"at most k events can occur within a time unit"
 - Wilke, 1994; F., 2008
- **Bounded time:**
"time only goes up to B "
 - Ouaknine, Rabinovich, Worrell, 2009



Dense Real-time Model-Checking

Other Models for Real-time

Other Models for Real-time

Research and practice in real-time systems has a **wide spectrum and heterogeneous concerns**

There exist **many different models** that go **beyond the model-checking paradigm**

Let us briefly consider two of them:

- **Timed Petri nets**: another concurrency model
- **TRIO** (and others): very expressive real-time temporal logics

Further reading:

F. et al. "Modeling time in computing", Springer 2012

Timed Petri Nets (in a slide)

Petri Nets (PN) are a popular model for concurrency.

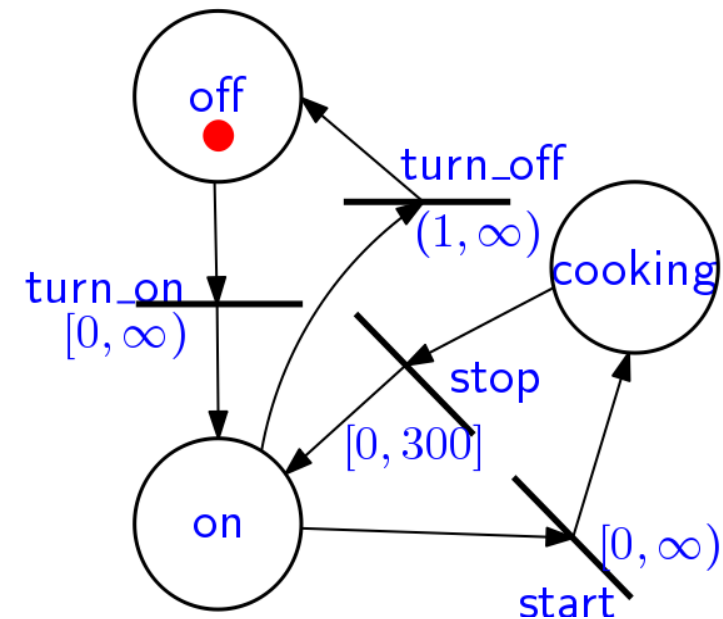
Many variants are available, including (real-)timed ones.

- PNs and timed PNs pre-date TAs but are less common in automated verification
- More suitable for “natural” modeling of asynchrony
- Places store tokens
- Transitions fire, moving tokens around
- Time bounds on the firing time of transitions

Model of the microwave oven
(not equivalent to the TA models we've seen)

Bounded Petri nets: bound on the maximum number of tokens that can be in any place in any run

- Essentially equivalent to TAs in expressiveness (with some semantic subtleties)



Full-fledged Real-time Temporal Logics



Another, quite **different, approach** to real-time modeling and analysis uses **very expressive first- (or even higher-) order temporal logic** to formalize any aspect of the system under analysis.

Example: the **TRIO** temporal logic, which includes:

- a core real-time temporal logic with real-time temporal operators
- first-order quantification and arithmetic
- object-oriented constructs
- higher-order extensions

Usage:

- **partial requirements** (and formal documentation)
- **semi-automated analysis**
- development by **refinement**
- ...