Formal Modelling and Verification
Master CORO – M2 ERTS

Didier Lime

École Centrale de Nantes – LS2N

2018 – 2019

Plan I

Plan I

Concurrent Programming

Example

Let `x=0` be a variable shared between two tasks doing: `x=x+1` What is the value of `x`?

Concurrent Programming

Example

Let `x=0` be a variable shared between two tasks doing: `x=x+1` What is the value of `x`?

`x` is a critical resource: mutual exclusion is needed.

Concurrent Programming

Example

Let `x=0` be a variable shared between two tasks doing: `x=x+1` What is the value of `x`?

`x` is a critical resource: mutual exclusion is needed.

demo: concurrency.c

A few Problems

- Data consistency (concurrent access);

A few Problems

- Data consistency (concurrent access);
- Absence of deadlocks;

A few Problems

- Data consistency (concurrent access);
- Absence of deadlocks;
- Absence of starvation;

A few Problems

- Data consistency (concurrent access);
- Absence of deadlocks;
- Absence of starvation;
- ...

A few Problems

- Data consistency (concurrent access);
- Absence of deadlocks;
- Absence of starvation;
- ...

Two wide classes of properties:

A few Problems

- Data consistency (concurrent access);
- Absence of deadlocks;
- Absence of starvation;
- ...

Two wide classes of properties:

- Qualitative : safety and liveness;

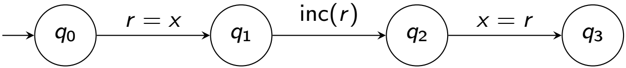
A few Problems

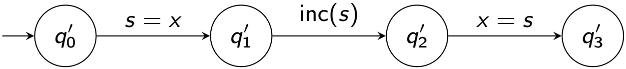
- Data consistency (concurrent access);
- Absence of deadlocks;
- Absence of starvation;
- ...

Two wide classes of properties:

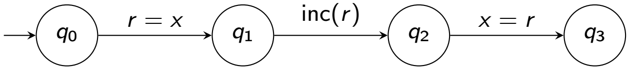
- Qualitative : safety and liveness;
- Quantitative: Deadlines, duration of execution, number of function calls...(Formalisms modelling time, probabilities, energy, ...)

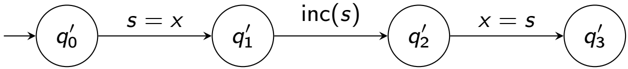
A First Model





A First Model

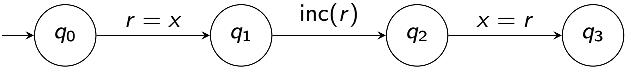


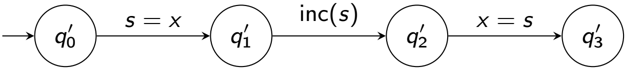


Informal property

We never have both $(q_1 \vee q_2)$ and $(q'_1 \vee q'_2)$ at the same time.

A First Model





Informal property

We never have both $(q_1 \vee q_2)$ and $(q'_1 \vee q'_2)$ at the same time.

Abstraction with respect to the property?

Peterson's Mutual Exclusion Algorithm [Ray84]

```
P :
    d ← false
loop
    <non critical section>
    d ← true
    turn ← 1
    attendre(¬d′∨ turn= 0)
    <critical section>
    d ← false
end_loop
```

```
P′ :
    d′ ← false
loop
    <non critical section>
    d′ ← true
    turn ← 0
    attendre(¬d∨ turn= 1)
    <critical section>
    d′ ← false
end_loop
```

Peterson's Mutual Exclusion Algorithm [Ray84]

```
P:
i0:  d ← false
loop
    <non critical section>
i1:  d ← true
i2:  turn ← 1
i3:  attendre(¬d'∨ turn= 0)
    <critical section>
i4:  d ← false
end_loop
```

```
P':
i'0:  d' ← false
loop
    <non critical section>
i'1:  d' ← true
i'2:  turn ← 0
i'3:  attendre(¬d∨ turn= 1)
    <critical section>
i'4:  d' ← false
end_loop
```

Peterson's Mutual Exclusion Algorithm [Ray84]

P:

i_0: $d \leftarrow$ `false`
`loop`
    `<non critical section>`
i_1: $d \leftarrow$ `true`
i_2: `turn` $\leftarrow$ 1
i_3: `attendre`($\neg d' \vee$ `turn`$= 0$)
    `<critical section>`
i_4: $d \leftarrow$ `false`
`end_loop`

P':

i'_0: $d' \leftarrow$ `false`
`loop`
    `<non critical section>`
i'_1: $d' \leftarrow$ `true`
i'_2: `turn` $\leftarrow$ 0
i'_3: `attendre`($\neg d \vee$ `turn`$= 1$)
    `<critical section>`
i'_4: $d' \leftarrow$ `false`
`end_loop`

How can we prove that this program works as intended?

Peterson's Mutual Exclusion Algorithm

- In the presentation, we have already abstracted some uninteresting parts;

Peterson's Mutual Exclusion Algorithm

- In the presentation, we have already abstracted some uninteresting parts;
- We have three shared variables

Peterson's Mutual Exclusion Algorithm

- In the presentation, we have already abstracted some uninteresting parts;
- We have three shared variables
  Why not only `turn`?

Peterson's Mutual Exclusion Algorithm

- In the presentation, we have already abstracted some uninteresting parts;
- We have three shared variables
  Why not only `turn`? Forced interleaving

Peterson's Mutual Exclusion Algorithm

- In the presentation, we have already abstracted some uninteresting parts;
- We have three shared variables
  Why not only `turn`? Forced interleaving
  Why not only `d` and `d'` ?

Peterson's Mutual Exclusion Algorithm

- In the presentation, we have already abstracted some uninteresting parts;
- We have three shared variables
  Why not only `turn`? Forced interleaving
  Why not only `d` and `d'` ? Deadlock

Peterson's Mutual Exclusion Algorithm

- In the presentation, we have already abstracted some uninteresting parts;
- We have three shared variables
  Why not only `turn`? Forced interleaving
  Why not only `d` and `d'` ? Deadlock
- The state of the system is $(d, d', \text{turn}, n, n') \in \mathbb{B} \times \mathbb{B} \times \{0, 1\} \times [0..4] \times [0..4]$, where n and n' represent the number of the next instruction to be executed for resp. P and P'

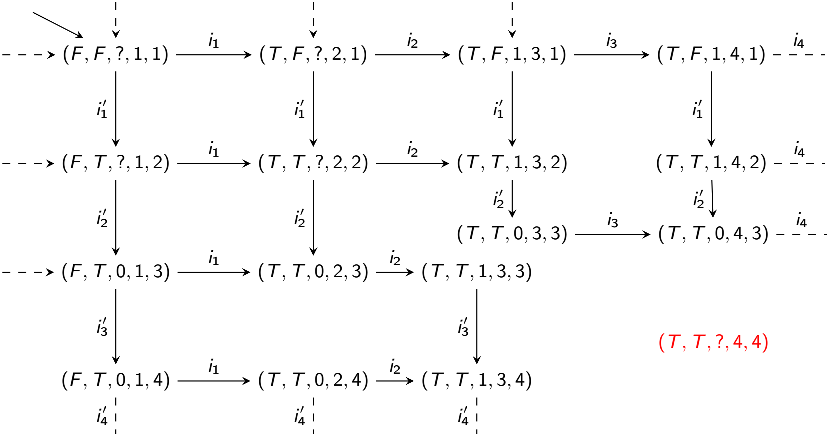
Peterson's Mutual Exclusion Algorithm

- In the presentation, we have already abstracted some uninteresting parts;
- We have three shared variables
  Why not only `turn`? Forced interleaving
  Why not only `d` and `d'` ? Deadlock
- The state of the system is $(d, d', \text{turn}, n, n') \in \mathbb{B} \times \mathbb{B} \times \{0, 1\} \times [0..4] \times [0..4]$, where n and n' represent the number of the next instruction to be executed for resp. P and P'
- The number of states is finite: we can explore exhaustively all possibilities.

Peterson's Mutual Exclusion Algorithm

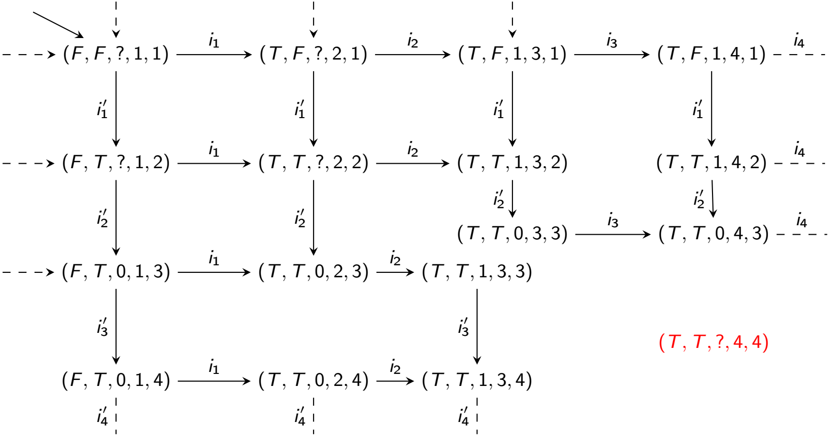
$$
\begin{tikzcd}
{} \arrow[dr] & {} \arrow[d, dashed] & {} \arrow[d, dashed] & {} \arrow[d, dashed] & & \\
{} \arrow[r, dashed] & (F, F, ?, 1, 1) \arrow[r, "i_1"] \arrow[d, "i'_1"'] & (T, F, ?, 2, 1) \arrow[r, "i_2"] \arrow[d, "i'_1"'] & (T, F, 1, 3, 1) \arrow[r, "i_3"] \arrow[d, "i'_1"'] & (T, F, 1, 4, 1) \arrow[r, dashed, "i_4"] \arrow[d, "i'_1"'] & {} \\
{} \arrow[r, dashed] & (F, T, ?, 1, 2) \arrow[r, "i_1"] \arrow[dd, "i'_2"'] & (T, T, ?, 2, 2) \arrow[r, "i_2"] \arrow[dd, "i'_2"'] & (T, T, 1, 3, 2) \arrow[d, "i'_2"'] & (T, T, 1, 4, 2) \arrow[r, dashed, "i_4"] \arrow[d, "i'_2"'] & {} \\
 & & & (T, T, 0, 3, 3) \arrow[r, "i_3"] & (T, T, 0, 4, 3) \arrow[r, dashed, "i_4"] & {} \\
{} \arrow[r, dashed] & (F, T, 0, 1, 3) \arrow[r, "i_1"] \arrow[dd, "i'_3"'] & (T, T, 0, 2, 3) \arrow[r, "i_2"] & (T, T, 1, 3, 3) \arrow[dd, "i'_3"'] & & \\
 & & & & \textcolor{red}{(T, T, ?, 4, 4)} & \\
 & (F, T, 0, 1, 4) \arrow[r, "i_1"] \arrow[d, dashed, "i'_4"'] & (T, T, 0, 2, 4) \arrow[r, "i_2"] \arrow[d, dashed, "i'_4"'] & (T, T, 1, 3, 4) \arrow[d, dashed, "i'_4"'] & & \\
 & {} & {} & {} & &
\end{tikzcd}
$$

Peterson's Mutual Exclusion Algorithm



Mutual Exclusion? Deadlocks? Access to the CS for P and P'?

Peterson's Mutual Exclusion Algorithm



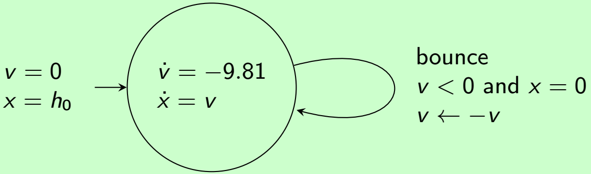
Mutual Exclusion? Deadlocks? Access to the CS for P and P'?
What about starvation?

Models for Complex Systems

- Many complex systems can be described by hybrid systems:
  - A continuous evolution governed by differential equations:
  - Discrete events, that change these these differential equations, or their initial conditions.

Example

Consider a ball dropped from some initial height h_0 and bouncing on the floor:



- The general setting is hard / impossible to analyze automatically
- We need to find an abstraction of the system relevant preserving the properties of interest.

Some Intuition About Decidability and Complexity

Some problems (e.g. termination of an arbitrary program) cannot be solved by an automatic procedure.

- Assume we have a function $H(f, x)$ that for any function f and any of its input x, returns whether the computation of $f(x)$ terminates;
- This might not be the case if f contains a `while (true)` loop for instance;
- Now consider the following function $H'(g)$, where g is a function with a function as input:

```
function H'(g):
    if H(g,g):
        while (true):  nop
    else:
        return true
```

Some Intuition About Decidability and Complexity

Some problems (e.g. termination of an arbitrary program) cannot be solved by an automatic procedure.

- Assume we have a function $H(f, x)$ that for any function f and any of its input x, returns whether the computation of $f(x)$ terminates;
- This might not be the case if f contains a `while (true)` loop for instance;
- Now consider the following function $H'(g)$, where g is a function with a function as input:

```
function H'(g):
    if H(g,g):
        while (true):  nop
    else:
        return true
```

- What happens for $H'(H')$?
  - if the computation terminates then $H(H', H')$ is true and the computation should loop forever (and hence not terminate);
  - if the computation does not terminate, then $H(H', H')$ is false and the computation should terminate (and return true).

Some Intuition About Decidability and Complexity

Some problems (e.g. termination of an arbitrary program) cannot be solved by an automatic procedure.

- Assume we have a function $H(f,x)$ that for any function f and any of its input x, returns whether the computation of $f(x)$ terminates;
- This might not be the case if f contains a `while (true)` loop for instance;
- Now consider the following function $H'(g)$, where g is a function with a function as input:

```
function H'(g):
    if H(g,g):
        while (true):  nop
    else:
        return true
```

- What happens for $H'(H')$?
  - if the computation terminates then $H(H',H')$ is true and the computation should loop forever (and hence not terminate);
  - if the computation does not terminate, then $H(H',H')$ is false and the computation should terminate (and return true).
- So function H cannot exist.

Some Intuition About Decidability and Complexity

- Problems with a yes/no answer are called decision problems;

Some Intuition About Decidability and Complexity

- Problems with a yes/no answer are called decision problems;
- A decision problem is decidable if there is an algorithm that always terminates and answer the good answer.

Some Intuition About Decidability and Complexity

- Problems with a yes/no answer are called decision problems;
- A decision problem is decidable if there is an algorithm that always terminates and answer the good answer.
- Even within decidable problems, we can make differences in:
  For a given model of computation, and considering a worst-case instance (or best-case, average-case, etc.)
  - the temporal complexity: number of instructions for the best algorithm to decide;
  - the spatial complexity: amount of memory for the best algorithm to decide.

Some Intuition About Decidability and Complexity

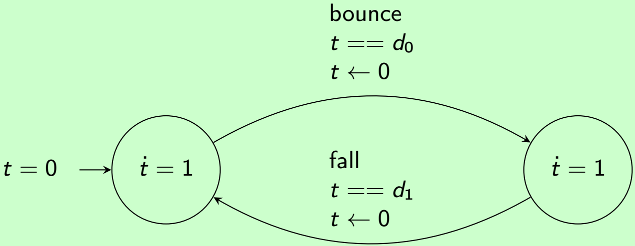
- Problems with a yes/no answer are called decision problems;
- A decision problem is decidable if there is an algorithm that always terminates and answer the good answer.
- Even within decidable problems, we can make differences in:
  For a given model of computation, and considering a worst-case instance (or best-case, average-case, etc.)
  - the temporal complexity: number of instructions for the best algorithm to decide;
  - the spatial complexity: amount of memory for the best algorithm to decide.
- For Turing machines and worst-case instances, we have, e.g.:
  - PTIME (aka P): CTL model-checking on finite automata (incl. reachability, liveness, etc.)
  - PSPACE: LTL model-checking on finite automata / TCTL model-checking on timed automata;
  - EXPTIME: Timed control for timed automata;
  - EXPSPACE: Timed language inclusion for strongly non-zeno timed automata;
  - Undecidable: Reachability in hybrid automata / language inclusion in timed automata.

Models for Complex Systems

- In the previous example, solve the equations to find the time to bouncing d_0, and the duration between bouncing and falling down again d_1:

Example

Consider a ball dropped from some initial height h_0, bouncing on the floor after d_0 seconds, and falling again d_1 seconds after bouncing:



- The time between the *bounce* events is preserved by this abstraction;
- It much easier to analyze automatically.

Models for Complex Systems

- Suppose now there is some dampening when bouncing: $v \leftarrow -v\epsilon$, with $0 < \epsilon < 1$;
- Now only an upper bound on the time between two *bounce* events is preserved in timed model;
- Still the number of bounces in the full model is infinite:

Exercise

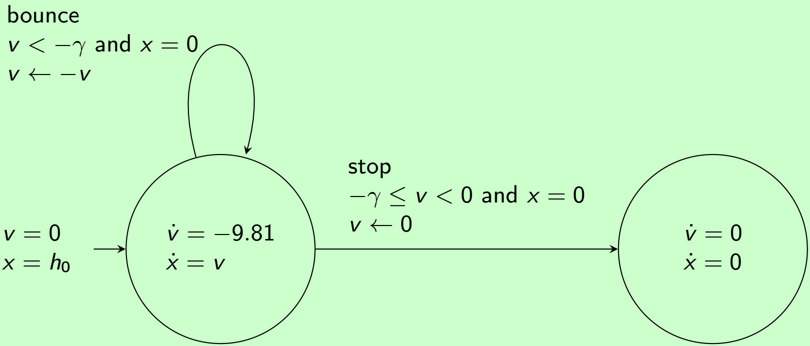
Modify the model so that bouncing eventually stops.

Models for Complex Systems

- Suppose now there is some dampening when bouncing: $v \leftarrow -v\epsilon$, with $0 < \epsilon < 1$;
- Now only an upper bound on the time between two *bounce* events is preserved in timed model;
- Still the number of bounces in the full model is infinite:

Exercise

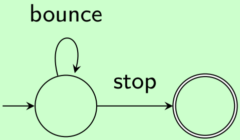
Modify the model so that bouncing eventually stops.



Models for Complex Systems

- Using no quantitative information, we can have a very coarse abstraction:

Example



- It is very easy to analyze and still preserves some properties, like:

  After bouncing a finite number of times, the ball eventually stops.

Plan I

Plan I

Definitions I

Definition (Labeled Transition Systems [BK08])

A Labeled Transition System (LTS) is a tuple $(S, S_0, \Sigma, \rightarrow)$, with:

- S is a set of elements called states ;
- S_0 is a non-empty subset of S containing the initial states ;
- Σ is a non-empty set of elements called actions;
- $\rightarrow \subset S \times \Sigma \times S$ is a relation called transition relation. We note $s \xrightarrow{a} s'$ when $(s, a, s') \in \rightarrow$.

Definition (Run)

A run of $\mathcal{S} = (S, S_0, \Sigma, \rightarrow)$ is possibly infinite sequence $s_0, a_0, s_1, a_1 \ldots, a_{n-1} s_n, \ldots$ such that $\forall i, s_i \xrightarrow{a_i} s_{i+1}$. We note $[\![\mathcal{S}]\!]$ the set of the runs of $\mathcal{S}$.

Definition (Reachable State)

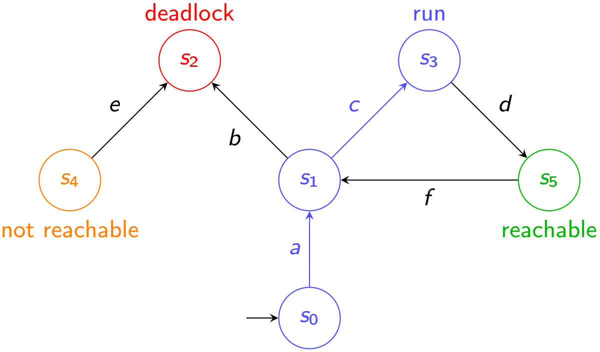
A state s' is reachable from a state s if there exists a run starting in s end ending in s'.

Definitions II

Definition (Deadlock State)

A state s is a deadlock if $\forall s' \in S, \forall a \in \Sigma, (s, a, s') \not\in \rightarrow$.

Example


deadlock
run
s_2
s_3
e
b
c
d
s_4
s_1
s_5
f
not reachable
reachable
a
s_0

Complete LTS

Definition

Complete LTS An LTS $(S, S_0, \Sigma, \rightarrow)$ is complete if from all states, every action is possible:

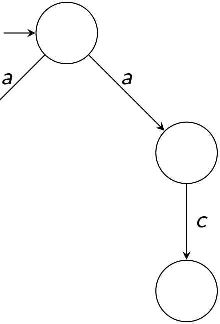
$$\forall s \in S, \forall a \in \sigma, \exists s' \in S \text{ s.t. } s \xrightarrow{a} s'$$

- A complete LTS means that all (sequential) behaviors are accounted for in the modeled system;
- It is particularly important for a specification;
- An LTS can always be made complete (preserving sequences of actions):
  - Add to S an *error* state s_{err};
  - For all $s \in S$ and $a \in \Sigma$ s.t. there is no s' s.t. $s \xrightarrow{a} s'$, add a transition (s, a, s_{err}) to $\rightarrow$.

Determinism

Definition (Deterministic LTS)

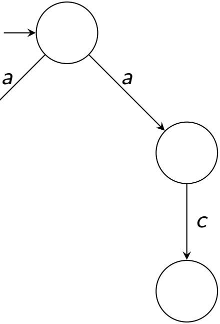
An LTS $\mathcal{S} = (S, S_0, \Sigma, \rightarrow)$ is deterministic if $|S_0| = 1$ and $s \xrightarrow{a} s' \land s \xrightarrow{a} s'' \Rightarrow s' = s''$.



Determinism

Definition (Deterministic LTS)

An LTS $\mathcal{S} = (S, S_0, \Sigma, \rightarrow)$ is deterministic if $|S_0| = 1$ and $s \xrightarrow{a} s' \wedge s \xrightarrow{a} s'' \Rightarrow s' = s''$.

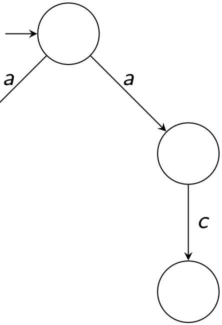


Usefulness ?

Determinism

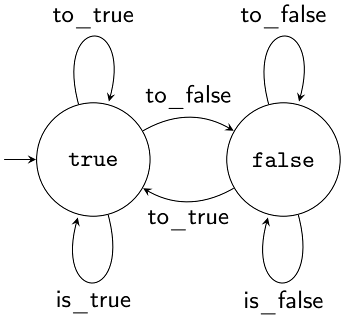
Definition (Deterministic LTS)

An LTS $\mathcal{S} = (S, S_0, \Sigma, \rightarrow)$ is deterministic if $|S_0| = 1$ and $s \xrightarrow{a} s' \wedge s \xrightarrow{a} s'' \Rightarrow s' = s''$.



Usefulness ? Abstraction.

LTS Example: Boolean Variable



Building Systems from Components

A complex system $\mathcal{S}$ is often described as the assembly of different components $(\mathcal{S}_i)_i$.

To analyze $\mathcal{S}$ we can:

- reason on the individual components $\mathcal{S}_i$ (for local properties);
- build $\mathcal{S}$ (possibly on-the-fly);
- compose local analyses (compositional approaches).

Synchronized Product of LTSs [BK08]

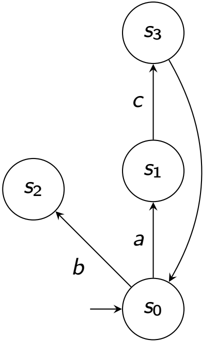
Consider n sets of actions Σ_i and n LTS $\mathcal{S}_i = (S^i, S^i_0, \Sigma_i, \rightarrow_i)_{n \in \mathbb{N}}$. We note $\Sigma_i^\bullet = \Sigma_i \cup \{\bullet\}$ with $\forall i, \bullet \notin \Sigma_i$.

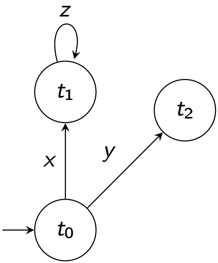
Definition (Synchronized Product)

The synchronised product $(\mathcal{S}_1, \ldots, \mathcal{S}_n)_f$ of the $\mathcal{S}_i$'s by the synchronization function $f : \Sigma_1^\bullet \times \cdots \times \Sigma_n^\bullet \rightarrow B$ is the LTS $(S, S_0, B, \rightarrow)$ such that:

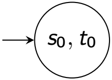
- $S = S^1 \times \ldots \times S^n$;
- $S_0 = S_0^1 \times \ldots \times S_0^n$;
- $\rightarrow \subset S \times B \times S$ is such that $(s_1, \ldots, s_n) \xrightarrow{b} (s'_1, \ldots, s'_n)$ iff $\forall i, \exists a_i \in \Sigma_i^\bullet$ s.t. $s_i \xrightarrow{a_i} s'_i$ and $f(a_1, \ldots, a_n) = b$.

Example

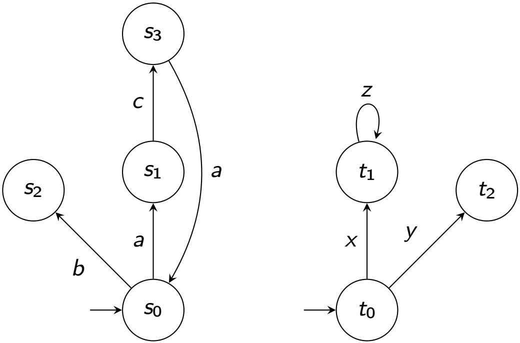




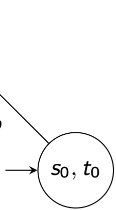
$$
\begin{aligned}
f(a, x) &= a \\
f(a, z) &= f(c, z) = a \\
f(b, \bullet) &= b \\
f(c, y) &= k
\end{aligned}
$$



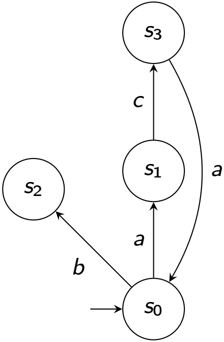
Example

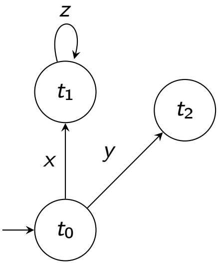


$$f(a, x) = a$$
$$f(a, z) = f(c, z) = a$$
$$f(b, \bullet) = b$$
$$f(c, y) = k$$

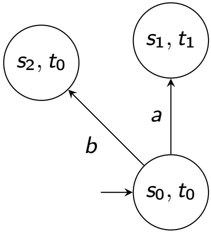


Example

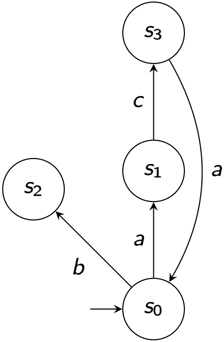


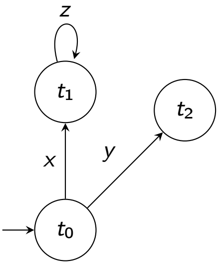


$$f(a, x) = a$$
$$f(a, z) = f(c, z) = a$$
$$f(b, \bullet) = b$$
$$f(c, y) = k$$

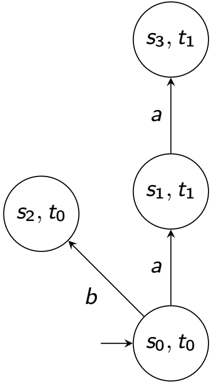


Example

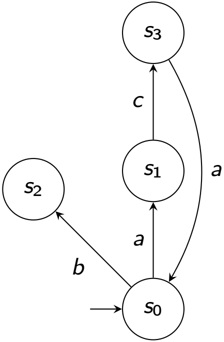


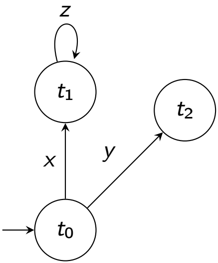


$$
\begin{aligned}
f(a,x) &= a\\
f(a,z) &= f(c,z) = a\\
f(b,\bullet) &= b\\
f(c,y) &= k
\end{aligned}
$$

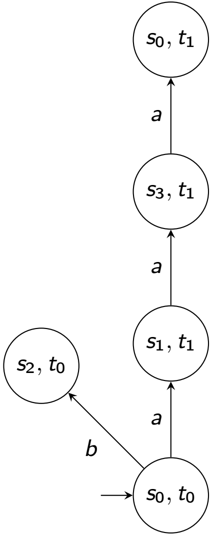


Example

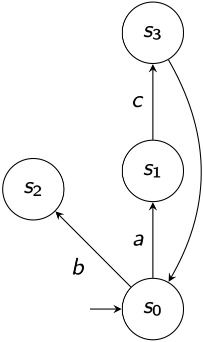


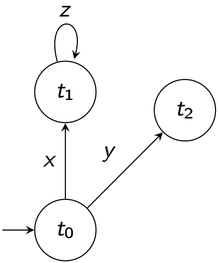


$$f(a, x) = a$$
$$f(a, z) = f(c, z) = a$$
$$f(b, \bullet) = b$$
$$f(c, y) = k$$

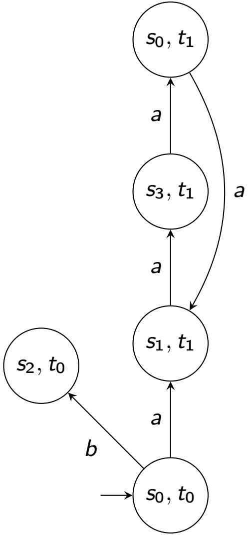


Example





$$f(a,x) = a$$
$$f(a,z) = f(c,z) = a$$
$$f(b,\bullet) = b$$
$$f(c,y) = k$$



Common Synchronization Schemes: No Synchronization

- We might have no synchronization at all between the components;
- This can be modeled by the asynchronous product;
- In f exactly one component is not $\bullet$; define all combinations of one action from some Σ_i with $\bullet$'s.

Common Synchronization Schemes: Complete Synchronization

- In some systems, components always progress all at the same time: e.g. hardware circuits;
- This can be modeled by the synchronous product;
- f never uses $\bullet$ and should be defined on all possible combinations of actions.

Common Synchronization Schemes: Rendez-vous

- When two processes need to wait for each other;

Common Synchronization Schemes: Rendez-vous

- When two processes need to wait for each other;
- We often denote rendez-vous actions with ? and !;

Common Synchronization Schemes: Rendez-vous

- When two processes need to wait for each other;
- We often denote rendez-vous actions with ? and !;
- For instance, sending a message $m!$ and recieving it $m?$ (both are blocking);

Common Synchronization Schemes: Rendez-vous

- When two processes need to wait for each other;
- We often denote rendez-vous actions with ? and !;
- For instance, sending a message $m!$ and recieving it $m?$ (both are blocking);
- The synchronization function is defined implicitly on such common labels by

$$f(a!, a?) = a$$

Common Synchronization Schemes: Rendez-vous

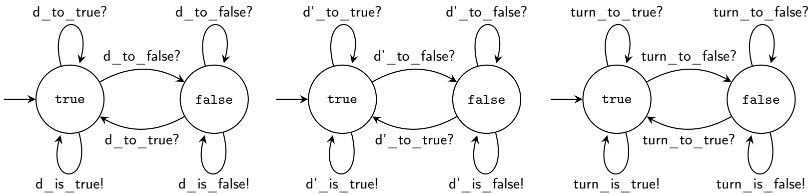
- When two processes need to wait for each other;
- We often denote rendez-vous actions with ? and !;
- For instance, sending a message $m!$ and recieving it $m?$ (both are blocking);
- The synchronization function is defined implicitly on such common labels by

$$f(a!, a?) = a$$

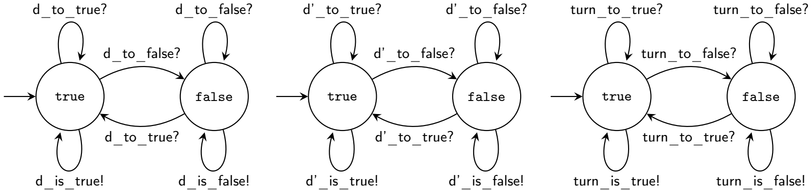
- In that context, actions without any synchronization mark are assumed independent:

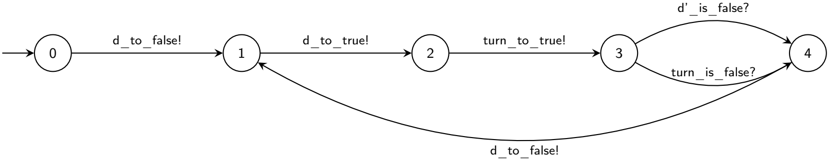
$$f(b, \bullet) = b \text{ or } f(\bullet, b) = b$$

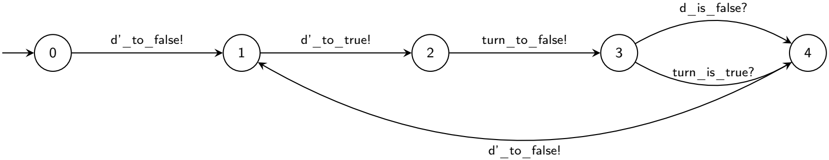
Example


d_to_true?
d_to_false?
d_to_false?
true
false
d_to_true?
d_is_true!
d_is_false!
d'_to_true?
d'_to_false?
d'_to_false?
true
false
d'_to_true?
d'_is_true!
d'_is_false!
turn_to_true?
turn_to_false?
turn_to_false?
true
false
turn_to_true?
turn_is_true!
turn_is_false!

Example


d_to_true?
d_to_false?
d_to_false?
true
false
d_to_true?
d_is_true!
d_is_false!
d'_to_true?
d'_to_false?
d'_to_false?
true
false
d'_to_true?
d'_is_true!
d'_is_false!
turn_to_true?
turn_to_false?
turn_to_false?
true
false
turn_to_true?
turn_is_true!
turn_is_false!


0
d_to_false!
1
d_to_true!
2
turn_to_true!
3
d'_is_false?
turn_is_false?
4
d_to_false!


0
d'_to_false!
1
d'_to_true!
2
turn_to_false!
3
d_is_false?
turn_is_true?
4
d'_to_false!

Common Synchronization Schemes: Broadcast

- In broadcasts, one process sends a message to several others;

Common Synchronization Schemes: Broadcast

- In broadcasts, one process sends a message to several others;
- Sending is not blocking, recieving is blocking.

Common Synchronization Schemes: Broadcast

- In broadcasts, one process sends a message to several others;
- Sending is not blocking, recieving is blocking.
- The notation with ! and ? (or !! and ??) is often used also;

Common Synchronization Schemes: Broadcast

- In broadcasts, one process sends a message to several others;
- Sending is not blocking, recieving is blocking.
- The notation with ! and ? (or !! and ??) is often used also;
- The non-blocking sending is often modeled by making sure that all processes can always recieve the message (possibly through a self-loop).

Exercise

Exercise

The philosophers' dinner. n philosopher are gathered around a big table with a spaghetti dish in the middle. There is one fork between each pair of adjacent philosophers. Each philospher is always either eating or thinking but never both at the same time. In order to eat, a philosopher needs the two forks, situated on its left and right.

1. Model a fork with a finite LTS;
2. Model a philosopher with a finite LTS (assuming that when it has taken a fork it will not release it before having eaten);
3. Build a substantial part of the product of two philosophers and two forks, including a deadlock.

Plan I

Verification

Objective

Verify (semi)automatically that the behavior of the system conforms to its specification.

- The specification can be described along two main formalisms:

Verification

Objective

Verify (semi)automatically that the behavior of the system conforms to its specification.

- The specification can be described along two main formalisms:
  - Algebraical specification: the specification describes what an ideal system should in the same formalism.

Verification

Objective

Verify (semi)automatically that the behavior of the system conforms to its specification.

- The specification can be described along two main formalisms:
  - Algebraical specification: the specification describes what an ideal system should in the same formalism.
  - Logical specification: the specification is formalized using a (temporal) logic.

Verification

Objective

Verify (semi)automatically that the behavior of the system conforms to its specification.

- The specification can be described along two main formalisms:
  - Algebraical specification: the specification describes what an ideal system should in the same formalism.
  - Logical specification: the specification is formalized using a (temporal) logic.
- The behaviors can be considered in two different ways:

Verification

Objective

Verify (semi)automatically that the behavior of the system conforms to its specification.

- The specification can be described along two main formalisms:
  - Algebraical specification: the specification describes what an ideal system should in the same formalism.
  - Logical specification: the specification is formalized using a (temporal) logic.
- The behaviors can be considered in two different ways:
  - A set of runs (traces);

Verification

Objective

Verify (semi)automatically that the behavior of the system conforms to its specification.

- The specification can be described along two main formalisms:
  - Algebraical specification: the specification describes what an ideal system should in the same formalism.
  - Logical specification: the specification is formalized using a (temporal) logic.
- The behaviors can be considered in two different ways:
  - A set of runs (traces);
  - A tree of executions, with branching points.

Plan I

Reachability and Safety

- The most basic property of model-checking is reachability:

Reachability and Safety

- The most basic property of model-checking is reachability:
  Let $(S, S_0, A, \rightarrow)$ be an LTS and $G \subseteq S$, can we reach a state in G?

Reachability and Safety

- The most basic property of model-checking is reachability:
  Let $(S, S_0, A, \to)$ be an LTS and $G \subseteq S$, can we reach a state in G?
- Its dual property is safety:

Reachability and Safety

- The most basic property of model-checking is reachability:
  Let $(S, S_0, A, \rightarrow)$ be an LTS and $G \subseteq S$, can we reach a state in G?
- Its dual property is safety:
  Let $(S, S_0, A, \rightarrow)$ be an LTS and $G \subseteq S$, can we stay in G forever?

Reachability and Safety

- The most basic property of model-checking is reachability:
  Let $(S, S_0, A, \rightarrow)$ *be an LTS and* $G \subseteq S$*, can we reach a state in* G*?*
- Its dual property is safety:
  Let $(S, S_0, A, \rightarrow)$ *be an LTS and* $G \subseteq S$*, can we stay in* G *forever?*
- The safety property can also be written as:

Reachability and Safety

- The most basic property of model-checking is reachability:
  Let $(S, S_0, A, \rightarrow)$ be an LTS and $G \subseteq S$, can we reach a state in G?
- Its dual property is safety:
  Let $(S, S_0, A, \rightarrow)$ be an LTS and $G \subseteq S$, can we stay in G forever?
- The safety property can also be written as:
  Let $(S, S_0, A, \rightarrow)$ be an LTS and $G \subseteq S$, are all states in $S \setminus G$ not reachable?

Erroneous Version of Peterson's Algorithm

P:

```
i0:  d ← false
loop
     <non critical section>
i1:  d ← true
i2:  turn ← 1
i3:  attendre(¬d' ∨ turn = 0)
     <critical section>
i4:  d ← false
end_loop
```

P':

```
i'0:  d' ← false
loop
      <non critical section>
i'1:  d' ← true
i'2:  turn ← 0
i'3:  attendre(¬d ∨ turn = 1)
      <critical section>
i'4:  d' ← false
end_loop
```

Erroneous Version of Peterson's Algorithm

P:
i_0: $d \leftarrow$ `false`
`loop`
    `<non critical section>`
i_1: $d \leftarrow$ `true`

i_3: `attendre(`$\neg d'$ `)`
    `<critical section>`
i_4: $d \leftarrow$ `false`
`end_loop`

P':
i'_0: $d' \leftarrow$ `false`
`loop`
    `<non critical section>`
i'_1: $d' \leftarrow$ `true`

i'_3: `attendre(`$\neg d$ `)`
    `<critical section>`
i'_4: $d' \leftarrow$ `false`
`end_loop`

We have removed `turn`.

Erroneous Version of Peterson's Algorithm

$$
\begin{tikzcd}
 & & {} \arrow[d, dashed] & & {} \arrow[d, dashed] & & & \\
{} \arrow[r, dashed] & {} \arrow[r, phantom] & (F,F,1,1) \arrow[rr, "i_1"] \arrow[d, "i_1'"'] & & (T,F,3,1) \arrow[rr, "i_3"] \arrow[d, "i_1'"'] & & (T,F,4,1) \arrow[r, dashed, dash, "i_4"] \arrow[d, "i_1'"'] & {} \\
{} \arrow[rr, dashed] & & (F,T,1,2) \arrow[rr, "i_1"] \arrow[d, "i_3'"'] & & \color{blue}{(T,T,3,3)} & & (T,T,4,3) \arrow[r, dashed, dash, "i_4"] & {} \\
 & & (F,T,1,4) \arrow[rr, "i_1"] \arrow[d, dashed, dash, "i_4'"] & & (T,T,3,4) \arrow[d, dashed, dash, "i_4'"] & & \color{red}{(T,T,4,4)} & \\
 & & {} & & {} & & &
\end{tikzcd}
$$

- **mutual exclusion**: is state $\color{red}{(T,T,4,4)}$ reachable?
- **non blocking**: is state $\color{blue}{(T,T,3,3)}$ reachable?

A Reachability Algorithm

- For a finite LTS $(S, s_0, A, \rightarrow)$, reachability of $G \subseteq S$ can be checked by a graph walk:

```
W ← S0 ; P ← ∅; r ← false
while W ≠ ∅ and not r
    s ← next(W)
    if s ∈ G then
        r ← true
    else
        if s ∉ P then
            add s to P
            for all s′ ∈ S, a ∈ A s.t. s –a→ s′
                add s′ to W
            endfor
        endif
    endif
endwhile
```

Symbolic Algorithms for Reachability: Forward

- We compute the set of all reachable states;

Symbolic Algorithms for Reachability: Forward

- We compute the set of all reachable states;
- Then we check that it intersects G;

Symbolic Algorithms for Reachability: Forward

- We compute the set of all reachable states;
- Then we check that it intersects G;
- We define the successors of a state set X:

$$\mathrm{Succ}(X) = \{s' \in S | \exists s \in X, a \in A \text{ s.t. } s \xrightarrow{a} s'\}$$

Symbolic Algorithms for Reachability: Forward

- We compute the set of all reachable states;
- Then we check that it intersects G;
- We define the successors of a state set X:

$$\mathsf{Succ}(X) = \{s' \in S | \exists s \in X, a \in A \text{ s.t. } s \xrightarrow{a} s'\}$$

- We compute a least fix point: $(\mu X . S_0 \cup \mathsf{Succ}(X))$

$X \leftarrow \emptyset$
$Y \leftarrow S$
while $X \neq Y$
  $Y \leftarrow X$
  $X \leftarrow S_0 \cup \mathsf{Succ}(Y)$
endwhile

Symbolic Algorithms for Reachability: Forward

- We compute the set of all reachable states;
- Then we check that it intersects G;
- We define the successors of a state set X:

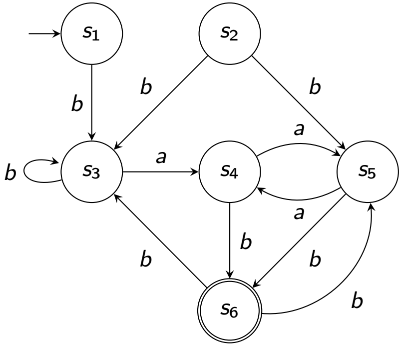
$$\mathsf{Succ}(X) = \{s' \in S | \exists s \in X, a \in A \text{ s.t. } s \xrightarrow{a} s'\}$$

- We compute a least fix point: $(\mu X. S_0 \cup \mathsf{Succ}(X))$

```
X ← ∅
Y ← S
while X ≠ Y
    Y ← X
    X ← S0 ∪ Succ(Y)
endwhile
```

Why does the algorithm terminate?

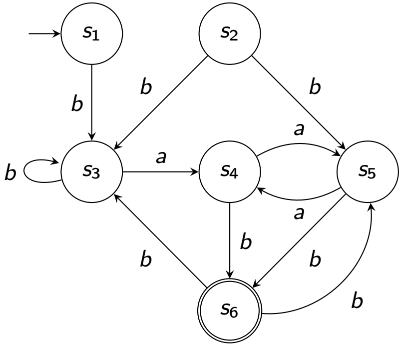
Forward Reachability: Example



We compute: $X_0 = \emptyset$ et $X_{n+1} = \{s_1\} \cup \mathrm{Succ}(X_n)$

- $X_0 = \emptyset$;

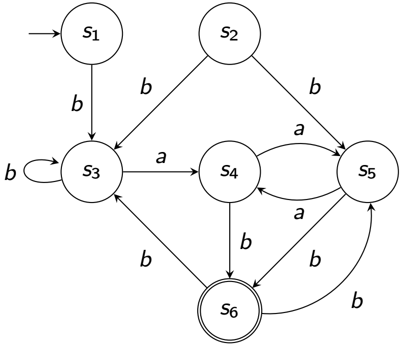
Forward Reachability: Example



We compute: $X_0 = \emptyset$ et $X_{n+1} = \{s_1\} \cup \mathrm{Succ}(X_n)$

- $X_0 = \emptyset$;
- $X_1 = \{s_1\}$;

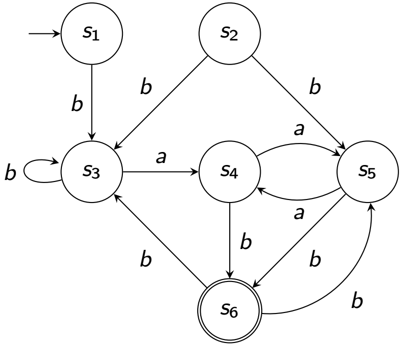
Forward Reachability: Example



We compute: $X_0 = \emptyset$ et $X_{n+1} = \{s_1\} \cup \mathrm{Succ}(X_n)$

- $X_0 = \emptyset$;
- $X_1 = \{s_1\}$;
- $X_2 = \{s_1, s_3\}$;

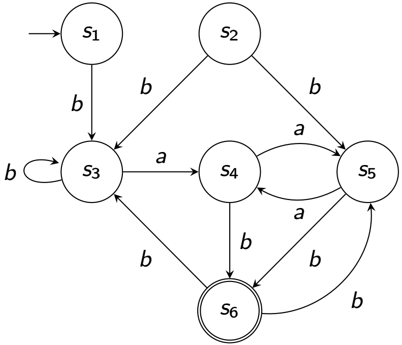
Forward Reachability: Example



We compute: $X_0 = \emptyset$ et $X_{n+1} = \{s_1\} \cup \text{Succ}(X_n)$

- $X_0 = \emptyset$;
- $X_1 = \{s_1\}$;
- $X_2 = \{s_1, s_3\}$;
- $X_3 = \{s_1, s_3, s_4\}$;

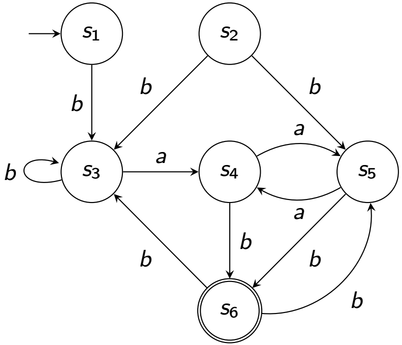
Forward Reachability: Example



We compute: $X_0 = \emptyset$ et $X_{n+1} = \{s_1\} \cup \mathrm{Succ}(X_n)$

- $X_0 = \emptyset$;
- $X_1 = \{s_1\}$;
- $X_2 = \{s_1, s_3\}$;
- $X_3 = \{s_1, s_3, s_4\}$;
- $X_4 = \{s_1, s_3, s_4, s_5, s_6\}$;

Forward Reachability: Example



We compute: $X_0 = \emptyset$ et $X_{n+1} = \{s_1\} \cup \mathrm{Succ}(X_n)$

- $X_0 = \emptyset$;
- $X_1 = \{s_1\}$;
- $X_2 = \{s_1, s_3\}$;
- $X_3 = \{s_1, s_3, s_4\}$;
- $X_4 = \{s_1, s_3, s_4, s_5, s_6\}$;
- $X_5 = X_4$.

Symbolic Algorithms for Reachability: Backward

- We compute the set of all co-reachable states for G
  i.e. from which G can be reached

Symbolic Algorithms for Reachability: Backward

- We compute the set of all co-reachable states for G
  i.e. from which G can be reached
- We then check whether it intersects S_0 or not;

Symbolic Algorithms for Reachability: Backward

- We compute the set of all co-reachable states for G
  i.e. from which G can be reached
- We then check whether it intersects S_0 or not;
- We define the predecessors of a state set X:

$$\text{Pred}(X) = \{s \in S | \exists s' \in X, a \in A \text{ s.t. } s \xrightarrow{a} s'\}$$

Symbolic Algorithms for Reachability: Backward

- We compute the set of all co-reachable states for G
  i.e. from which G can be reached
- We then check whether it intersects S_0 or not;
- We define the predecessors of a state set X:

$$\text{Pred}(X) = \{s \in S | \exists s' \in X, a \in A \text{ s.t. } s \xrightarrow{a} s'\}$$

- On fait alors un calcul de point fixe:

Symbolic Algorithms for Reachability: Backward

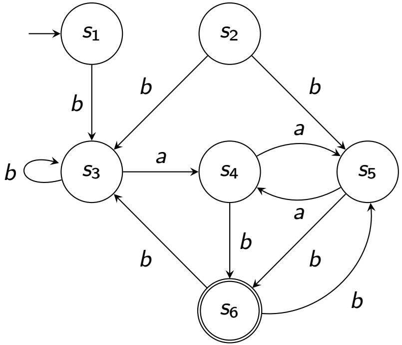
- We compute the set of all co-reachable states for G
  i.e. from which G can be reached
- We then check whether it intersects S_0 or not;
- We define the predecessors of a state set X:

$$\text{Pred}(X) = \{s \in S | \exists s' \in X, a \in A \text{ s.t. } s \xrightarrow{a} s'\}$$

- On fait alors un calcul de point fixe:
- We compute a smallest fix point: $\mu X.G \cup \text{Pred}(X)$

$X \leftarrow \emptyset$
$Y \leftarrow S$
while $X \neq Y$
    $Y \leftarrow X$
    $X \leftarrow G \cup \text{Pred}(Y)$
endwhile

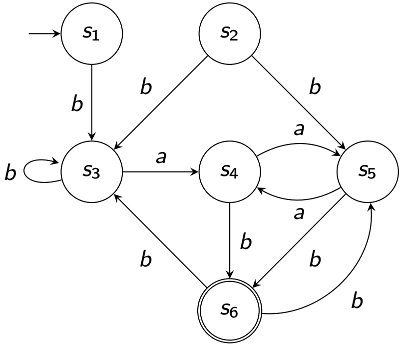
Backward Reachability: Example



We compute $X_0 = \emptyset$ and $X_{n+1} = \{s_6\} \cup \mathrm{Pred}(X_n)$

- $X_0 = \emptyset$;

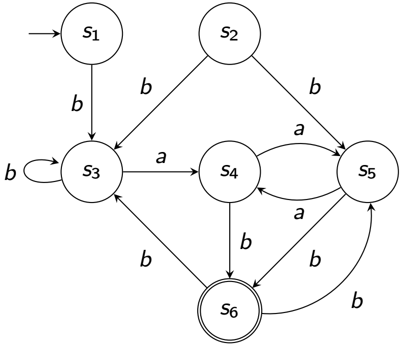
Backward Reachability: Example



We compute $X_0 = \emptyset$ and $X_{n+1} = \{s_6\} \cup \mathrm{Pred}(X_n)$

- $X_0 = \emptyset$;
- $X_1 = \{s_6\}$;

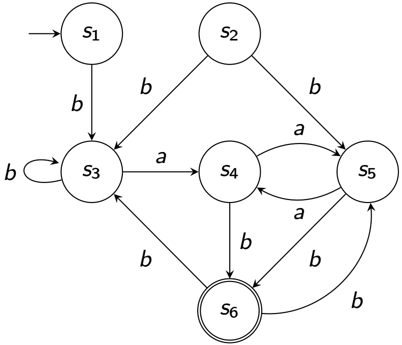
Backward Reachability: Example



We compute $X_0 = \emptyset$ and $X_{n+1} = \{s_6\} \cup \text{Pred}(X_n)$

- $X_0 = \emptyset$;
- $X_1 = \{s_6\}$;
- $X_2 = \{s_4, s_5, s_6\}$;

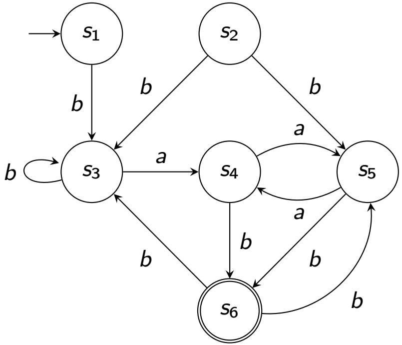
Backward Reachability: Example



We compute $X_0 = \emptyset$ and $X_{n+1} = \{s_6\} \cup \mathrm{Pred}(X_n)$

- $X_0 = \emptyset$;
- $X_1 = \{s_6\}$;
- $X_2 = \{s_4, s_5, s_6\}$;
- $X_3 = \{s_2, s_3, s_4, s_5, s_6\}$;

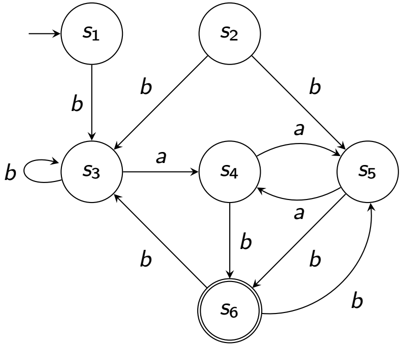
Backward Reachability: Example



We compute $X_0 = \emptyset$ and $X_{n+1} = \{s_6\} \cup \mathrm{Pred}(X_n)$

- $X_0 = \emptyset$;
- $X_1 = \{s_6\}$;
- $X_2 = \{s_4, s_5, s_6\}$;
- $X_3 = \{s_2, s_3, s_4, s_5, s_6\}$;
- $X_4 = \{s_1, s_2, s_3, s_4, s_5, s_6\}$;

Backward Reachability: Example



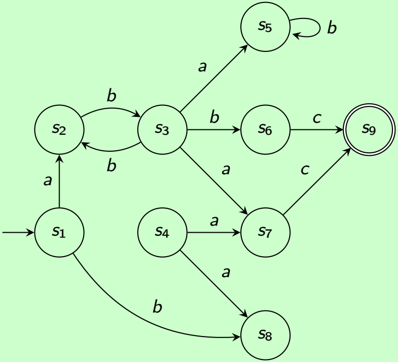
We compute $X_0 = \emptyset$ and $X_{n+1} = \{s_6\} \cup \mathrm{Pred}(X_n)$

- $X_0 = \emptyset$;
- $X_1 = \{s_6\}$;
- $X_2 = \{s_4, s_5, s_6\}$;
- $X_3 = \{s_2, s_3, s_4, s_5, s_6\}$;
- $X_4 = \{s_1, s_2, s_3, s_4, s_5, s_6\}$;
- $X_5 = X_4$.

Reachability: Exercises

Exercise

1. Compute the set of states that are reachable from s_1;
2. Compute the set of states that are co-reachable for s_9;
3. Compute the set of states that cannot reach s_9 (safety).



A Backward Symbolic Algorithm for Safety

- Let Y be the safe states for G and X the coreachable states for $\overline{G}$: we have $Y = \overline{X}$

A Backward Symbolic Algorithm for Safety

- Let Y be the safe states for G and X the coreachable states for $\overline{G}$: we have $Y = \overline{X}$
- The sequence $X_0 = \emptyset$ et $X_{n+1} = \overline{G} \cup \mathrm{Pred}(X_n)$ converges towards X;

A Backward Symbolic Algorithm for Safety

- Let Y be the safe states for G and X the coreachable states for $\overline{G}$: we have $Y = \overline{X}$
- The sequence $X_0 = \emptyset$ et $X_{n+1} = \overline{G} \cup \mathrm{Pred}(X_n)$ converges towards X;
- So $(Y_n)_n = (\overline{X_n})_n$ converges towards Y;

A Backward Symbolic Algorithm for Safety

- Let Y be the safe states for G and X the coreachable states for $\overline{G}$: we have $Y = \overline{X}$
- The sequence $X_0 = \emptyset$ et $X_{n+1} = \overline{G} \cup \text{Pred}(X_n)$ converges towards X;
- So $(Y_n)_n = (\overline{X_n})_n$ converges towards Y;
- $Y_0 = S$ and $Y_{n+1} = G \cap \overline{\text{Pred}(\overline{Y_n})}$;

A Backward Symbolic Algorithm for Safety

- Let Y be the safe states for G and X the coreachable states for $\overline{G}$: we have $Y = \overline{X}$
- The sequence $X_0 = \emptyset$ et $X_{n+1} = \overline{G} \cup \mathrm{Pred}(X_n)$ converges towards X;
- So $(Y_n)_n = (\overline{X_n})_n$ converges towards Y;
- $Y_0 = S$ and $Y_{n+1} = G \cap \overline{\mathrm{Pred}(\overline{Y_n})}$;
- $\mathrm{Pred}(\overline{Y_n}) = \{s | \exists s' \notin Y_n, s \rightarrow s'\}$;

A Backward Symbolic Algorithm for Safety

- Let Y be the safe states for G and X the coreachable states for $\overline{G}$: we have $Y = \overline{X}$
- The sequence $X_0 = \emptyset$ et $X_{n+1} = \overline{G} \cup \mathrm{Pred}(X_n)$ converges towards X;
- So $(Y_n)_n = (\overline{X_n})_n$ converges towards Y;
- $Y_0 = S$ and $Y_{n+1} = G \cap \overline{\mathrm{Pred}(\overline{Y_n})}$;
- $\mathrm{Pred}(\overline{Y_n}) = \{s | \exists s' \notin Y_n, s \rightarrow s'\}$;
- $\overline{\mathrm{Pred}(\overline{Y_n})} = \{s | \forall s' \notin Y_n, s \not\rightarrow s'\}$;

A Backward Symbolic Algorithm for Safety

- Let Y be the safe states for G and X the coreachable states for $\overline{G}$: we have $Y = \overline{X}$
- The sequence $X_0 = \emptyset$ et $X_{n+1} = \overline{G} \cup \text{Pred}(X_n)$ converges towards X;
- So $(Y_n)_n = (\overline{X_n})_n$ converges towards Y;
- $Y_0 = S$ and $Y_{n+1} = G \cap \overline{\text{Pred}(\overline{Y_n})}$;
- $\text{Pred}(\overline{Y_n}) = \{s | \exists s' \notin Y_n, s \to s'\}$;
- $\overline{\text{Pred}(\overline{Y_n})} = \{s | \forall s' \notin Y_n, s \not\to s'\}$;
- Let $\widetilde{\text{Pred}}(Z) = \{s | s \to s' \Rightarrow s' \in Z\}$, we have:

$$\begin{cases} Y_0 = S \\ Y_{n+1} = G \cap \widetilde{\text{Pred}}(Y_n) \end{cases}$$

A Backward Symbolic Algorithm for Safety

- Let Y be the safe states for G and X the coreachable states for $\overline{G}$: we have $Y = \overline{X}$
- The sequence $X_0 = \emptyset$ et $X_{n+1} = \overline{G} \cup \text{Pred}(X_n)$ converges towards X;
- So $(Y_n)_n = (\overline{X_n})_n$ converges towards Y;
- $Y_0 = S$ and $Y_{n+1} = G \cap \overline{\text{Pred}(\overline{Y_n})}$;
- $\text{Pred}(\overline{Y_n}) = \{s | \exists s' \notin Y_n, s \to s'\}$;
- $\overline{\text{Pred}(\overline{Y_n})} = \{s | \forall s' \notin Y_n, s \not\to s'\}$;
- Let $\widetilde{\text{Pred}}(Z) = \{s | s \to s' \Rightarrow s' \in Z\}$, we have:

$$\begin{cases} Y_0 = S \\ Y_{n+1} = G \cap \widetilde{\text{Pred}}(Y_n) \end{cases}$$

Exercise

Compute the set of states that are safe for $\overline{\{s_9\}}$ for the previous exemple.

Reachability: Exercise

Exercise

1. Prove that $s \in X_n$ iff there exists a run of length less than n that starts in s and ends in G;
2. Prove that the co-reachability algorithm is correct (gives states that are indeed coreachable) and complete (gives all of them).

Observers

- Reachability concerns the states of the system;

Observers

- Reachability concerns the states of the system;
- It can also be used for more complex properties (called regular properties) using observers;

Observers

- Reachability concerns the states of the system;
- It can also be used for more complex properties (called regular properties) using observers;
- An observer is:

Observers

- Reachability concerns the states of the system;
- It can also be used for more complex properties (called regular properties) using observers;
- An observer is:
  - A automaton that is synchronized with the model of the system;

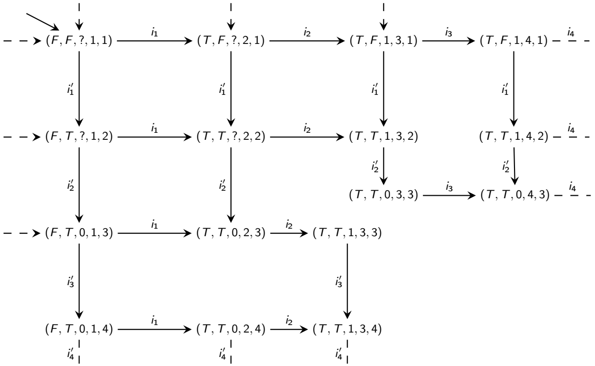
Observers

- Reachability concerns the states of the system;
- It can also be used for more complex properties (called regular properties) using observers;
- An observer is:
  - A automaton that is synchronized with the model of the system;
  - Non-intrusive: it does not modify the behavior of the system;

Observers

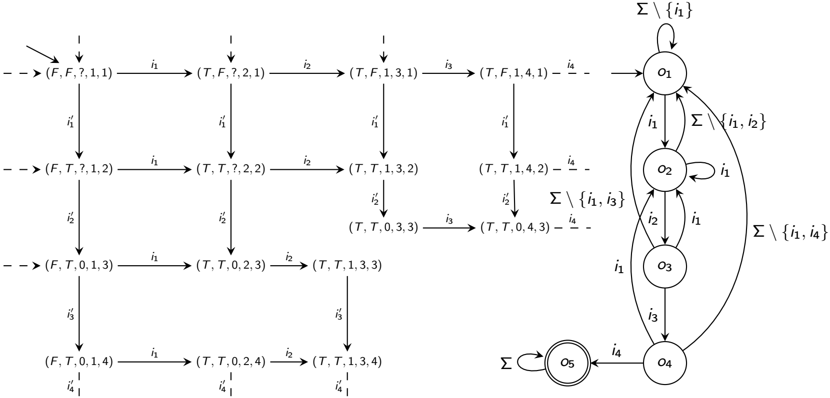
- Reachability concerns the states of the system;
- It can also be used for more complex properties (called regular properties) using observers;
- An observer is:
  - A automaton that is synchronized with the model of the system;
  - Non-intrusive: it does not modify the behavior of the system;
  - With one or more distinguished states in which the property can be decided.

Observers Example: Peterson's Algorithm



- **Access to the resource alone**: is the sequence i_1, i_2, i_3, i_4 feasible?

Observers Example: Peterson's Algorithm



- **Access to the resource alone**: is the sequence i_1, i_2, i_3, i_4 feasible?
  Is a state of the form $(*, *, *, *, *, o_5)$ reachable in the product?

Liveness

- Reachability is a very simple property;

Liveness

- Reachability is a very simple property;
- It cannot express the fact that something will always eventually happen;

Liveness

- Reachability is a very simple property;
- It cannot express the fact that something will always eventually happen;
- For that we use liveness:

Liveness

- Reachability is a very simple property;
- It cannot express the fact that something will always eventually happen;
- For that we use liveness:

Definition (Quasi-liveness)

Let $(S, S_0, \Sigma, \rightarrow)$ be an LTS and $s \in S$. $a \in \Sigma$ is quasi-live in s if there exists a state s' reachable from s, and from which a is possible ($\exists s''$ s.t. $s' \xrightarrow{a} s''$).

Liveness

- Reachability is a very simple property;
- It cannot express the fact that something will always eventually happen;
- For that we use liveness:

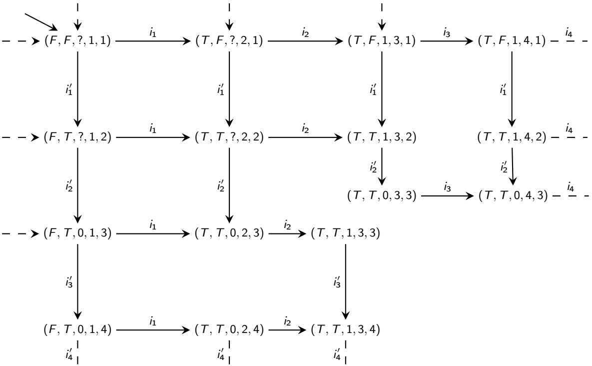
Definition (Quasi-liveness)

Let $(S, S_0, \Sigma, \to)$ be an LTS and $s \in S$. $a \in \Sigma$ is quasi-live in s if there exists a state s' reachable from s, and from which a is possible ($\exists s''$ s.t. $s' \xrightarrow{a} s''$).

Definition (Liveness)

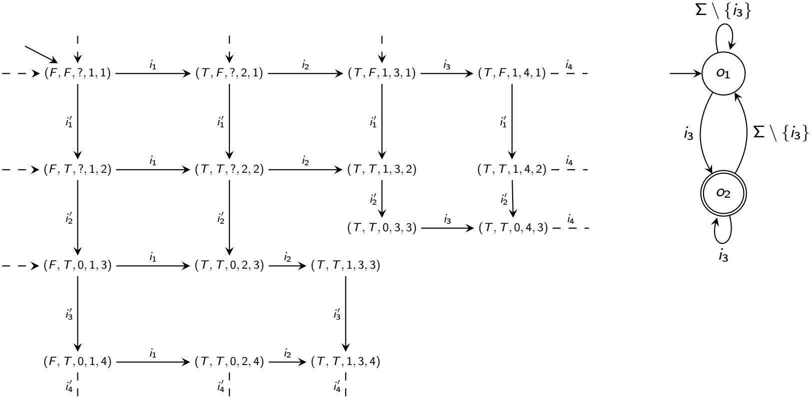
Let $(S, S_0, \Sigma, \to)$ be an LTS and $s \in S$. $a \in \Sigma$ is live from s if from all state s' reachable from s, a is quasi-live from s'.

Liveness Example: Peterson's Algorithm



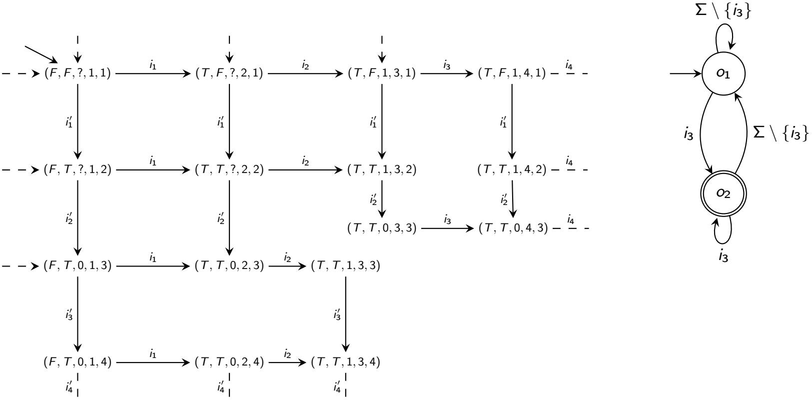
- **possible continuous access to the resource**: is i_3 live?

Liveness Example: Peterson's Algorithm



- **possible continuous access to the resource**: is i_3 live?
  Reachability of $(*, *, *, *, *, o_2)$ is not enough

Liveness Example: Peterson's Algorithm



- **possible continuous access to the resource**: is i_3 live?
  Reachability of $(*, *, *, *, *, o_2)$ is not enough
- We need (at least) the repeated reachability of $(*, *, *, *, *, o_2)$.

Repeated Reachability Algorithm

- Let $(S, S_0, \Sigma, \rightarrow)$ be an LTS and a set of states to repeat infinitely often $G \subseteq S$;
- We give a symbolic algorithm;
- We compute nested fix points: $\nu V.\mu X.\mathrm{Pred}(G \cap V) \cup \mathrm{Pred}(X)$

```
V ← S
W ← ∅
while  V ≠ W
      W ← V
      X ← ∅
      Y ← S
      while  X ≠ Y
            Y ← X
            X ← Pred(V ∩ G) ∪ Pred(Y)
      endwhile
      V ← X
endwhile
```

Repeated Reachability Algorithm

- Let $(S, S_0, \Sigma, \rightarrow)$ be an LTS and a set of states to repeat infinitely often $G \subseteq S$;
- We give a symbolic algorithm;
- We compute nested fix points: $\nu V.\mu X.\mathrm{Pred}(G \cap V) \cup \mathrm{Pred}(X)$

```
V ← S
W ← ∅
while V ≠ W
    W ← V
    V ← Co − reachable(Pred(V ∩ G))
endwhile
```

Repeated Reachability Algorithm

- Let $(S, S_0, \Sigma, \rightarrow)$ be an LTS and a set of states to repeat infinitely often $G \subseteq S$;
- We give a symbolic algorithm;
- We compute nested fix points: $\nu V.\mu X.\mathrm{Pred}(G \cap V) \cup \mathrm{Pred}(X)$

```
V ← S
W ← ∅
while V ≠ W
    W ← V
    V ← Co − reachable(Pred(V ∩ G))
endwhile
```

- After iteration $n \geq 1$, V contains the set of states that can reach G, in at least one step, and at least n times.

Repeated Reachability Algorithm

- States that can reach G:

$$\mathrm{Co}-\mathrm{reachable}(G)$$

Repeated Reachability Algorithm

- States that can reach G:

$$\mathrm{Co} - \mathrm{reachable}(G)$$

- States that can reach G in at least one step:

$$\mathrm{Co} - \mathrm{reachable}(\mathrm{Pred}(G))$$

Repeated Reachability Algorithm

- States that can reach G:

$$\mathrm{Co} - \mathrm{reachable}(G)$$

- States that can reach G in at least one step:

$$\mathrm{Co} - \mathrm{reachable}(\mathrm{Pred}(G))$$

- States in G that can reach G in at least one step:

$$G \cap \mathrm{Co} - \mathrm{reachable}(\mathrm{Pred}(G))$$

Repeated Reachability Algorithm

- States that can reach G:

$$\mathrm{Co} - \mathrm{reachable}(G)$$

- States that can reach G in at least one step:

$$\mathrm{Co} - \mathrm{reachable}(\mathrm{Pred}(G))$$

- States in G that can reach G in at least one step:

$$G \cap \mathrm{Co} - \mathrm{reachable}(\mathrm{Pred}(G))$$

- States that can reach a state in G that can reach a state in G in at least one step:

$$\mathrm{Co} - \mathrm{reachable}(G \cap \mathrm{Co} - \mathrm{reachable}(\mathrm{Pred}(G)))$$

Repeated Reachability Algorithm

- States that can reach G:

$$\mathrm{Co}-\mathrm{reachable}(G)$$

- States that can reach G in at least one step:

$$\mathrm{Co}-\mathrm{reachable}(\mathrm{Pred}(G))$$

- States in G that can reach G in at least one step:

$$G \cap \mathrm{Co}-\mathrm{reachable}(\mathrm{Pred}(G))$$

- States that can reach a state in G that can reach a state in G in at least one step:

$$\mathrm{Co}-\mathrm{reachable}(G \cap \mathrm{Co}-\mathrm{reachable}(\mathrm{Pred}(G)))$$

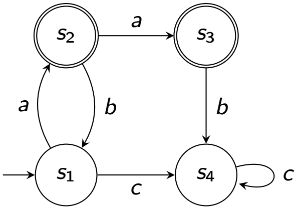
- States that can reach, in at least one step, a state in G that can reach in at least one step a state in G:

$$\mathrm{Co}-\mathrm{reachable}(\mathrm{Pred}(G \cap \mathrm{Co}-\mathrm{reachable}(\mathrm{Pred}(G))))$$

Repeated Reachability Algorithm

- States that can reach G:
$$\mathrm{Co}-\mathrm{reachable}(G)$$
- States that can reach G in at least one step:
$$\mathrm{Co}-\mathrm{reachable}(\mathrm{Pred}(G))$$
- States in G that can reach G in at least one step:
$$G \cap \mathrm{Co}-\mathrm{reachable}(\mathrm{Pred}(G))$$
- States that can reach a state in G that can reach a state in G in at least one step:
$$\mathrm{Co}-\mathrm{reachable}(G \cap \mathrm{Co}-\mathrm{reachable}(\mathrm{Pred}(G)))$$
- States that can reach, in at least one step, a state in G that can reach in at least one step a state in G:
$$\mathrm{Co}-\mathrm{reachable}(\mathrm{Pred}(G \cap \mathrm{Co}-\mathrm{reachable}(\mathrm{Pred}(G))))$$
- ...

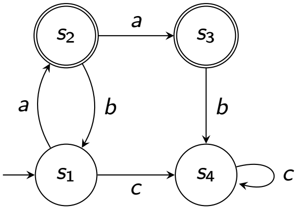
Repeated Reachability: Example



We compute $V_0 = S$ and
$V_{n+1} = \mathrm{Co} - \mathrm{reachable}(\mathrm{Pred}(\{s_2, s_3\} \cap V_n))$

- $V_0 = S$;

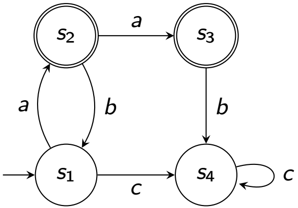
Repeated Reachability: Example



We compute $V_0 = S$ and
$V_{n+1} = \text{Co} - \text{reachable}(\text{Pred}(\{s_2, s_3\} \cap V_n))$

- $V_0 = S$;
- $V_1 = \text{Co} - \text{reachable}(\{s_1, s_2\})$;

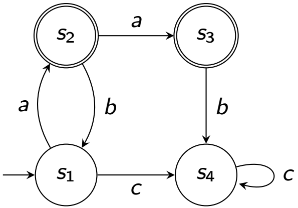
Repeated Reachability: Example



We compute $V_0 = S$ and
$V_{n+1} = \text{Co} - \text{reachable}(\text{Pred}(\{s_2, s_3\} \cap V_n))$

- $V_0 = S$;
- $V_1 = \text{Co} - \text{reachable}(\{s_1, s_2\})$;
- $V_1 = \{s_1, s_2\}$;

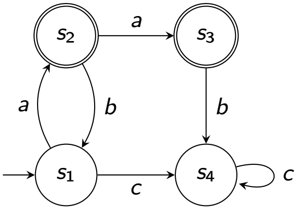
Repeated Reachability: Example



We compute $V_0 = S$ and
$V_{n+1} = \mathrm{Co}-\mathrm{reachable}(\mathrm{Pred}(\{s_2, s_3\} \cap V_n))$

- $V_0 = S$;
- $V_1 = \mathrm{Co}-\mathrm{reachable}(\{s_1, s_2\})$;
- $V_1 = \{s_1, s_2\}$;
- $V_2 = \mathrm{Co}-\mathrm{reachable}(\mathrm{Pred}(\{s_2\}))$;

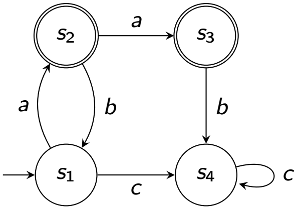
Repeated Reachability: Example



We compute $V_0 = S$ and
$V_{n+1} = \mathrm{Co} - \mathrm{reachable}(\mathrm{Pred}(\{s_2, s_3\} \cap V_n))$

- $V_0 = S$;
- $V_1 = \mathrm{Co} - \mathrm{reachable}(\{s_1, s_2\})$;
- $V_1 = \{s_1, s_2\}$;
- $V_2 = \mathrm{Co} - \mathrm{reachable}(\mathrm{Pred}(\{s_2\}))$;
- $V_2 = \mathrm{Co} - \mathrm{reachable}(\{s_1\})$;

Repeated Reachability: Example



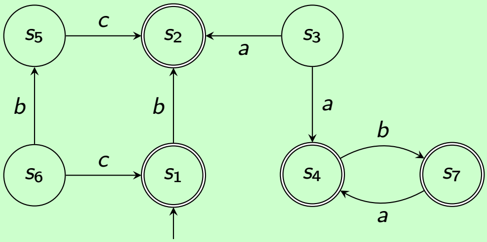
We compute $V_0 = S$ and
$V_{n+1} = \text{Co} - \text{reachable}(\text{Pred}(\{s_2, s_3\} \cap V_n))$

- $V_0 = S$;
- $V_1 = \text{Co} - \text{reachable}(\{s_1, s_2\})$;
- $V_1 = \{s_1, s_2\}$;
- $V_2 = \text{Co} - \text{reachable}(\text{Pred}(\{s_2\}))$;
- $V_2 = \text{Co} - \text{reachable}(\{s_1\})$;
- $V_2 = \{s_1, s_2\} = V_1$;

Repeated Reachability: Exercise

Exercise

Compute the set of states from which $\{s_1, s_2, s_4, s_7\}$ can be repeated infinitely often.



Towards Language Inclusion

- Even with repeated reachability of o_2 in the observer, we do not have the liveness of i_3;

Towards Language Inclusion

- Even with repeated reachability of o_2 in the observer, we do not have the liveness of i_3;
- We have:

Towards Language Inclusion

- Even with repeated reachability of o_2 in the observer, we do not have the liveness of i_3;
- We have:
  There exists an infinite run in the product that goes through o_2 infinitely often

Towards Language Inclusion

- Even with repeated reachability of o_2 in the observer, we do not have the liveness of i_3;
- We have:
  There exists an infinite run in the product that goes through o_2 infinitely often
- We want:

Towards Language Inclusion

- Even with repeated reachability of o_2 in the observer, we do not have the liveness of i_3;
- We have:
  There exists an infinite run in the product that goes through o_2 infinitely often
- We want:
  All infinite runs in the product go through o_2 infinitely often

Towards Language Inclusion

- Even with repeated reachability of o_2 in the observer, we do not have the liveness of i_3;
- We have:
  There exists an infinite run in the product that goes through o_2 infinitely often
- We want:
  All infinite runs in the product go through o_2 infinitely often
- This can also be written as:

Towards Language Inclusion

- Even with repeated reachability of o_2 in the observer, we do not have the liveness of i_3;
- We have:
  There exists an infinite run in the product that goes through o_2 infinitely often
- We want:
  All infinite runs in the product go through o_2 infinitely often
- This can also be written as:
  The set of infinite action sequences in the system is included in the set of infinite action sequences in the observer that go through o_2 infinitely often

Plan I

(Non-deterministic) Finite Automata (NFA)

- With observers, we are interested only in runs that lead to a distinguished state in which the property is decided;

(Non-deterministic) Finite Automata (NFA)

- With observers, we are interested only in runs that lead to a distinguished state in which the property is decided;
- This corresponds to the notion of Finite Automaton (NFA):

Definition (Finite Automaton)

A finite automaton is tuple $(S, S_0, A, \rightarrow, F)$ where:

- $(S, S_0, A, \rightarrow)$ is a finite LTS ;
- $F \subseteq S$ is a set of accepting states.

(Non-deterministic) Finite Automata (NFA)

- With observers, we are interested only in runs that lead to a distinguished state in which the property is decided;
- This corresponds to the notion of Finite Automaton (NFA):

Definition (Finite Automaton)

A finite automaton is tuple $(S, S_0, A, \rightarrow, F)$ where:

- $(S, S_0, A, \rightarrow)$ is a finite LTS ;
- $F \subseteq S$ is a set of accepting states.

- Sequences of actions can be characterized with the notion of language of a finite automaton.

Formal Languages

Letters and words:

- Consider a finite set Σ, called alphabet;
- Elements of Σ are called letters;
- Words are sequences of letters;
- We note Σ^* the set of all words on Σ;
- We note ϵ the empty word;
- We note uv the word obtained by the concatenation of words u and v;
- Concatenation is associative and ϵ is its neutral element;

Formal Languages

Letters and words:

- Consider a finite set Σ, called alphabet;
- Elements of Σ are called letters;
- Words are sequences of letters;
- We note Σ^* the set of all words on Σ;
- We note ϵ the empty word;
- We note uv the word obtained by the concatenation of words u and v;
- Concatenation is associative and ϵ is its neutral element;

Languages :

- A language on Σ is a subset of Σ^*;
- Let U and V be two languages on Σ, $UV = \{uv | u \in U, v \in V\}$;
- $U^* = \{u_0 \dots u_n | n \geq 0, \forall i, u_i \in U\}$.

Language Recognized by an NFA

Definition (Trace)

The trace of a run $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} s_n \cdots$ is the possibly infinite word $\text{trace}(\rho) = a_0 a_1 \ldots a_{n-1} \ldots$.

Language Recognized by an NFA

Definition (Trace)

The trace of a run $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} s_n \cdots$ is the possibly infinite word $\mathrm{trace}(\rho) = a_0 a_1 \ldots a_{n-1} \ldots$.

Definition (Word Recognized by an NFA)

A finite $w \in \Sigma^*$ is recognized by an NFA $(S, S_0, \Sigma, \rightarrow, F)$ if there exists a runs ρ of $\mathcal{S}$ starting from an initial state and ending in a state of F such that $\mathrm{trace}(\rho) = w$.

Language Recognized by an NFA

Definition (Trace)

The trace of a run $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} s_n \cdots$ is the possibly infinite word $\text{trace}(\rho) = a_0 a_1 \ldots a_{n-1} \ldots$.

Definition (Word Recognized by an NFA)

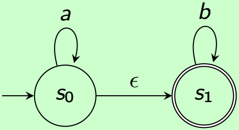
A finite $w \in \Sigma^*$ is recognized by an NFA $(S, S_0, \Sigma, \rightarrow, F)$ if there exists a runs ρ of $\mathcal{S}$ starting from an initial state and ending in a state of F such that $\text{trace}(\rho) = w$.

Definition (Language Recognized by an NFA)

The language $\mathcal{L}(\mathcal{S})$ recognized by an NFA $\mathcal{S}$ is the set of words it recognizes.

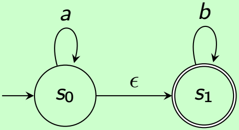
Finite Automata: Examples

Example



Finite Automata: Examples

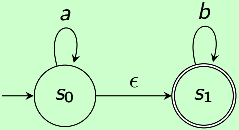
Example



$$\mathcal{L}(\mathcal{A}) = \{a^n b^m | m, n \in \mathbb{N}\}$$

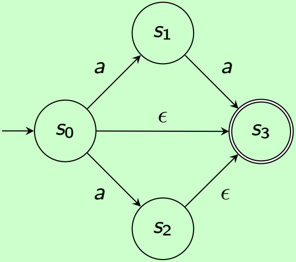
Finite Automata: Examples

Example



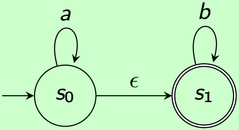
$$\mathcal{L}(\mathcal{A}) = \{a^n b^m | m, n \in \mathbb{N}\}$$

Example



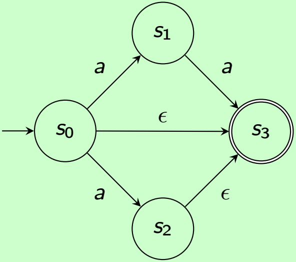
Finite Automata: Examples

Example



$$\mathcal{L}(\mathcal{A}) = \{a^n b^m | m, n \in \mathbb{N}\}$$

Example



$$\mathcal{L}(\mathcal{A}) = \{\epsilon, a, aa\}$$

Exercises

Exercise

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two finite automata. Show how to build a finite automaton the language of which is:

- The concatenation $\mathcal{L}(\mathcal{A}_1).\mathcal{L}(\mathcal{A}_2)$;
- The union $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$;
- The Kleene Star $\mathcal{L}(\mathcal{A}_1)^*$;
- The intersection $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

Exercise

Prove that the two following problems are equivalent:

- Given an LTS and s one of its states, is s reachable?
- Given a finite automaton, is its language empty?

Language Inclusion

- The language of an automaton describes its behavior;

Language Inclusion

- The language of an automaton describes its behavior;
- Consider automata $\mathcal{A}$ modeling the system and $\mathcal{S}$ modeling the specification:

Language Inclusion

- The language of an automaton describes its behavior;
- Consider automata $\mathcal{A}$ modeling the system and $\mathcal{S}$ modeling the specification:
  - All the behaviors of the system conform to the specification: $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S})$

Language Inclusion

- The language of an automaton describes its behavior;
- Consider automata $\mathcal{A}$ modeling the system and $\mathcal{S}$ modeling the specification:
  - All the behaviors of the system conform to the specification: $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S})$
  - All specified behaviors are present in the system: $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{A})$

Language Inclusion

- The language of an automaton describes its behavior;
- Consider automata $\mathcal{A}$ modeling the system and $\mathcal{S}$ modeling the specification:
  - All the behaviors of the system conform to the specification: $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S})$
  - All specified behaviors are present in the system: $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{A})$
- Language inclusion can be checked through intersection and complement:

$$\mathcal{L}(A) \subseteq \mathcal{L}(B) \Leftrightarrow \mathcal{L}(A) \cap \overline{\mathcal{L}(B)} = \emptyset$$

Language Inclusion

- The language of an automaton describes its behavior;
- Consider automata $\mathcal{A}$ modeling the system and $\mathcal{S}$ modeling the specification:
  - All the behaviors of the system conform to the specification: $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S})$
  - All specified behaviors are present in the system: $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{A})$
- Language inclusion can be checked through intersection and complement:

$$\mathcal{L}(A) \subseteq \mathcal{L}(B) \Leftrightarrow \mathcal{L}(A) \cap \overline{\mathcal{L}(B)} = \emptyset$$

- Equivalently on the automata:

$$\mathcal{L}(A) \subseteq \mathcal{L}(B) \Leftrightarrow \mathcal{L}(A \cap \overline{B}) = \emptyset$$

Language Inclusion

- The language of an automaton describes its behavior;
- Consider automata $\mathcal{A}$ modeling the system and $\mathcal{S}$ modeling the specification:
  - All the behaviors of the system conform to the specification: $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S})$
  - All specified behaviors are present in the system: $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{A})$
- Language inclusion can be checked through intersection and complement:

$$\mathcal{L}(A) \subseteq \mathcal{L}(B) \Leftrightarrow \mathcal{L}(A) \cap \overline{\mathcal{L}(B)} = \emptyset$$

- Equivalently on the automata:

$$\mathcal{L}(A) \subseteq \mathcal{L}(B) \Leftrightarrow \mathcal{L}(A \cap \overline{B}) = \emptyset$$

- Intersection: we can compute the product fully synchronized on common actions/labels;

Language Inclusion

- The language of an automaton describes its behavior;
- Consider automata $\mathcal{A}$ modeling the system and $\mathcal{S}$ modeling the specification:
  - All the behaviors of the system conform to the specification: $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S})$
  - All specified behaviors are present in the system: $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{A})$
- Language inclusion can be checked through intersection and complement:

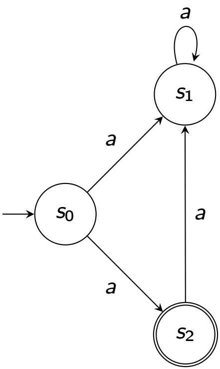
$$\mathcal{L}(A) \subseteq \mathcal{L}(B) \Leftrightarrow \mathcal{L}(A) \cap \overline{\mathcal{L}(B)} = \emptyset$$

- Equivalently on the automata:

$$\mathcal{L}(A) \subseteq \mathcal{L}(B) \Leftrightarrow \mathcal{L}(A \cap \overline{B}) = \emptyset$$

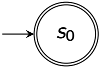
- Intersection: we can compute the product fully synchronized on common actions/labels;
- Complement: invert accepting states and non-accepting states but B should be complete et deterministic.

Complementation: Non-determinism



- Language:
$$\mathcal{L} = \{a\}$$
- Complement:
$$\overline{\mathcal{L}} = \{a^n | n \neq 1\}$$
- Inverting accepting and non-accepting states:
$$\mathcal{L}' = \{a^n | n \geq 0\}$$

Complementation: Incompleteness



- Language:
$$\mathcal{L} = \{\epsilon\}$$
- Complement ($\Sigma = \{a\}$):
$$\overline{\mathcal{L}} = \{a^n | n \geq 1\}$$
- Inverting accepting and non-accepting states:
$$\mathcal{L}' = \emptyset$$

Determinization of an NFA

Theorem

For every non-deterministic finite automaton $\mathcal{A}$, there exists a deterministic finite automaton $\Delta(\mathcal{A})$ with the same language as $\mathcal{A}$.

Determinization of an NFA

Theorem

For every non-deterministic finite automaton $\mathcal{A}$, there exists a deterministic *finite automaton $\Delta(\mathcal{A})$ with the same language as $\mathcal{A}$.*

Computing $\Delta(\mathcal{A})$ is called *determinization* of $\mathcal{A}$.

Determinization of an NFA

Let $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ be an NFA. We define $\Delta(\mathcal{A}) = (Q', \Sigma, \delta', q_0', F')$

- $Q' = 2^Q$;

Determinization of an NFA

Let $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ be an NFA. We define $\Delta(\mathcal{A}) = (Q', \Sigma, \delta', q_0', F')$

- $Q' = 2^Q$;
- $q_0' = Q_0$;

Determinization of an NFA

Let $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ be an NFA. We define $\Delta(\mathcal{A}) = (Q', \Sigma, \delta', q_0', F')$

- $Q' = 2^Q$;
- $q_0' = Q_0$;
- $F' = \{S \in 2^Q \,|F \cap S \neq \emptyset\}$;

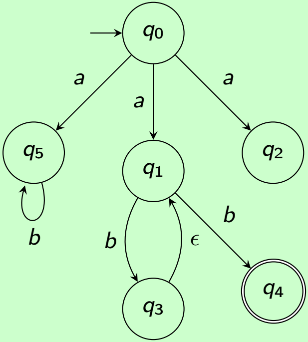
Determinization of an NFA

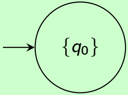
Let $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ be an NFA. We define $\Delta(\mathcal{A}) = (Q', \Sigma, \delta', q_0', F')$

- $Q' = 2^Q$;
- $q_0' = Q_0$;
- $F' = \{S \in 2^Q \,|F \cap S \neq \emptyset\}$;
- $\forall q' \in Q', \forall a \in \Sigma, \delta'(q') = \mathcal{F}_\epsilon(\bigcup_{q \in q'} \delta(q, a))$ where $\mathcal{F}_\epsilon$ is the fix point of the function: $S \mapsto S \cup \{q | \exists q', q \in \delta(q', \epsilon)\}$.

Determinization of an NFA: Example

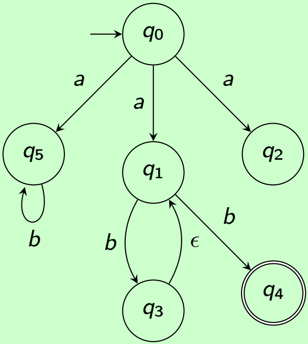
Example

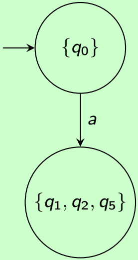




Determinization of an NFA: Example

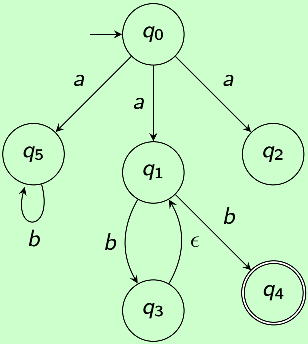
Example

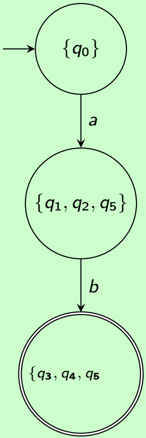




Determinization of an NFA: Example

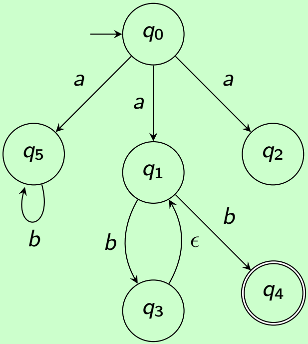
Example

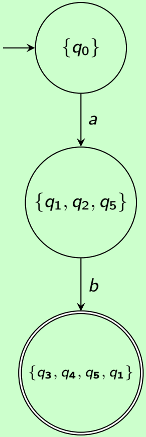




Determinization of an NFA: Example

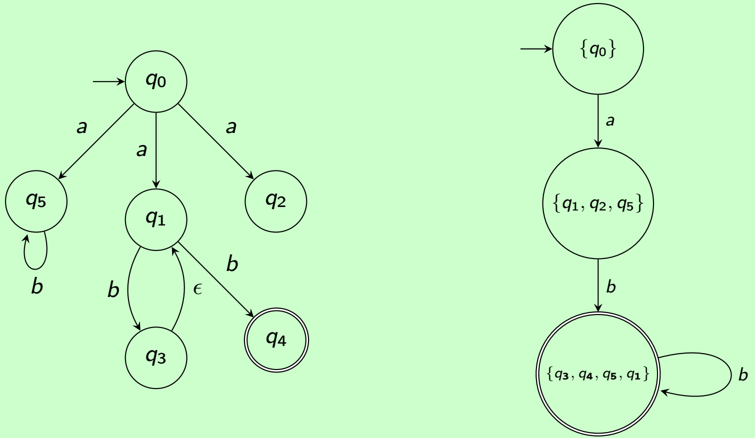
Example





Determinization of an NFA: Example

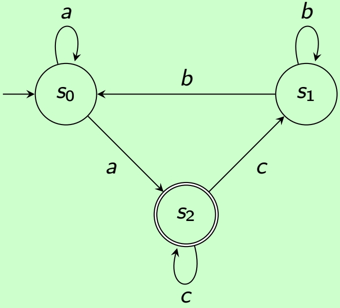
Example

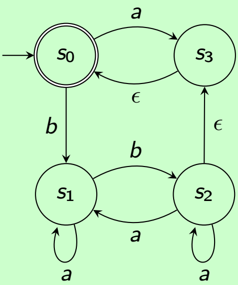


Determinization of an NFA: Exercise

Exercise

Determinize and complete the following NFAs:





Infinite Words and Büchi Automata

- For liveness properties, we need to reason on:
  - infinite runs;
  - and the repetition of certain states.

Infinite Words and Büchi Automata

- For liveness properties, we need to reason on:
  - infinite runs;
  - and the repetition of certain states.
- This corresponds to the notion of Büchi automata:

Definition (Büchi Automata)

A (non-deterministic) Büchi automaton (BA) is a tuple $(S, S_0, A, \rightarrow, R)$ where:

- $(S, S_0, A, \rightarrow)$ is a finite LTS ;
- $R \subseteq S$ is a set of repeated states.

Infinite Words and Büchi Automata

- For liveness properties, we need to reason on:
  - infinite runs;
  - and the repetition of certain states.
- This corresponds to the notion of Büchi automata:

Definition (Büchi Automata)

A (non-deterministic) Büchi automaton (BA) is a tuple $(S, S_0, A, \to, R)$ where:

- $(S, S_0, A, \to)$ is a finite LTS ;
- $R \subseteq S$ is a set of repeated states.

- The language of a Büchi automaton can be defined similarly as in the finite case.

Language recognized by a BA

Definition (Word recognized by a BA)

A infinite word $w \in \Sigma^\omega$ is recognized by a BA $(S, S_0, \Sigma, \rightarrow, R)$ if there exists an infinite run ρ of $\mathcal{S}$ starting from an initial state and ending in a state of R.

Language recognized by a BA

Definition (Word recognized by a BA)

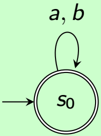
A infinite word $w \in \Sigma^\omega$ is recognized by a BA $(S, S_0, \Sigma, \rightarrow, R)$ if there exists an infinite run ρ of $\mathcal{S}$ starting from an initial state and ending in a state of R.

Definition (Language recognized by a BA)

The language $\mathcal{L}(\mathcal{S})$ recognized by a BA $\mathcal{S}$ is the set of its recognized words.

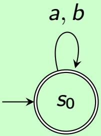
Examples

Example



Examples

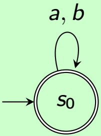
Example



$$\mathcal{L}(\mathcal{A}) = (a|b)^\omega$$

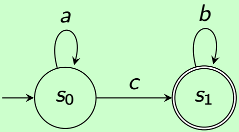
Examples

Example



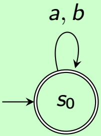
$$\mathcal{L}(\mathcal{A}) = (a|b)^\omega$$

Example



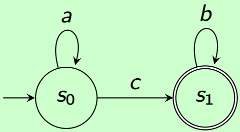
Examples

Example



$$\mathcal{L}(\mathcal{A}) = (a|b)^\omega$$

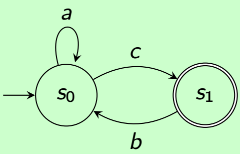
Example



$$\mathcal{L}(\mathcal{A}) = a^* c b^\omega$$

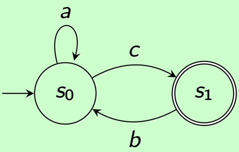
Examples

Example



Examples

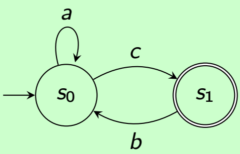
Example



$$\mathcal{L}(\mathcal{A}) = (a^* cb)^\omega$$

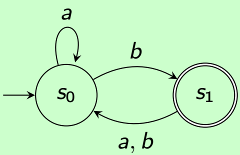
Examples

Example



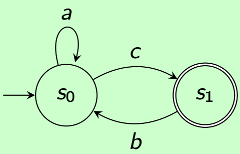
$$\mathcal{L}(\mathcal{A}) = (a^* cb)^\omega$$

Example



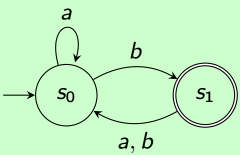
Examples

Example



$$\mathcal{L}(\mathcal{A}) = (a^* cb)^\omega$$

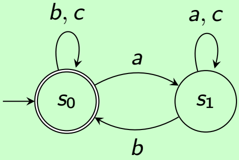
Example



$$\mathcal{L}(\mathcal{A}) = (a^* b)^\omega$$

Exercise

Exercise



1. Give an intuitive expression for the language of this BA;
2. Give a BA that recognizes the complement of that language.

Properties and Language Inclusion

- Büchi automata are closed for:

Properties and Language Inclusion

- Büchi automata are closed for:
  - union;

Properties and Language Inclusion

- Büchi automata are closed for:
  - union;
  - concatenation;

Properties and Language Inclusion

- Büchi automata are closed for:
  - union;
  - concatenation;
  - omega;

Properties and Language Inclusion

- Büchi automata are closed for:
  - union;
  - concatenation;
  - omega;
  - intersection;

Properties and Language Inclusion

- Büchi automata are closed for:
  - union;
  - concatenation;
  - omega;
  - intersection;
  - complement;

Properties and Language Inclusion

- Büchi automata are closed for:
  - union;
  - concatenation;
  - omega;
  - intersection;
  - complement;
- We can check language inclusion as before.

Properties and Language Inclusion

- Büchi automata are closed for:
  - union;
  - concatenation;
  - omega;
  - intersection;
  - complement;
- We can check language inclusion as before.
- But

Properties and Language Inclusion

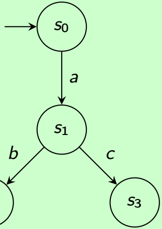
- Büchi automata are closed for:
  - union;
  - concatenation;
  - omega;
  - intersection;
  - complement;
- We can check language inclusion as before.
- But
  - BAs are not closed by determinization;

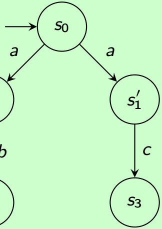
Properties and Language Inclusion

- Büchi automata are closed for:
  - union;
  - concatenation;
  - omega;
  - intersection;
  - complement;
- We can check language inclusion as before.
- But
  - BAs are not closed by determinization;
  - Building the complement is *hard* (but possible; we will skip it).

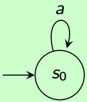
Limits of Trace Equivalence

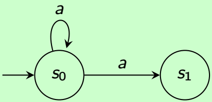
Example





Example





Simulation

Definition (Simulation)

Consider $\mathcal{A} = (S, S_0, A, \rightarrow)$. Let $\mathcal{R} \subseteq (S \times S)$ be a preorder (reflexive and transitive) relation. $\mathcal{R}$ is a simulation if: $\forall (s_1, s_2) \in \mathcal{R}, \forall s_2' \in S$ s.t. $s_2 \xrightarrow{a} s_2', \exists s_1' \in S$ s.t. $s_1 \xrightarrow{a} s_1'$ and $(s_1', s_2') \in \mathcal{R}$.

We often write $s\mathcal{R}s'$ for $(s, s') \in \mathcal{R}$.

Simulation

Definition (Simulation)

Consider $\mathcal{A} = (S, S_0, A, \rightarrow)$. Let $\mathcal{R} \subseteq (S \times S)$ be a preorder (reflexive and transitive) relation. $\mathcal{R}$ is a simulation if: $\forall (s_1, s_2) \in \mathcal{R}, \forall s_2' \in S$ s.t. $s_2 \xrightarrow{a} s_2', \exists s_1' \in S$ s.t. $s_1 \xrightarrow{a} s_1'$ and $(s_1', s_2') \in \mathcal{R}$.

We often write $s\mathcal{R}s'$ for $(s, s') \in \mathcal{R}$.
This can be straightforwardly extended ot the simulation of an LTS by another.

Theorem

$\mathcal{A}_1$ *simulate* $\mathcal{A}_2 \Rightarrow \mathcal{L}(\mathcal{A}_2) \subseteq \mathcal{L}(\mathcal{A}_1)$.

Simulation

Definition (Simulation)

Consider $\mathcal{A} = (S, S_0, A, \rightarrow)$. Let $\mathcal{R} \subseteq (S \times S)$ be a preorder (reflexive and transitive) relation. $\mathcal{R}$ is a simulation if: $\forall (s_1, s_2) \in \mathcal{R}, \forall s_2' \in S$ s.t. $s_2 \xrightarrow{a} s_2', \exists s_1' \in S$ s.t. $s_1 \xrightarrow{a} s_1'$ and $(s_1', s_2') \in \mathcal{R}$.

We often write $s\mathcal{R}s'$ for $(s, s') \in \mathcal{R}$.
This can be straightforwardly extended ot the simulation of an LTS by another.

Theorem

$\mathcal{A}_1$ *simulate* $\mathcal{A}_2 \Rightarrow \mathcal{L}(\mathcal{A}_2) \subseteq \mathcal{L}(\mathcal{A}_1)$.

what about the other direction?

Abstraction

- Some action might not be relevant for a given property: it makes sense to remove them.
- We can replace them with ϵ actions and want to compare the resulting abstracted LTSs:

Abstraction

- Some action might not be relevant for a given property: it makes sense to remove them.
- We can replace them with ϵ actions and want to compare the resulting abstracted LTSs:
- For instance, by considering that $\xrightarrow{a}\xrightarrow{\epsilon}\xrightarrow{b}\approx\xrightarrow{a}\xrightarrow{b}$;

Abstraction

- Some action might not be relevant for a given property: it makes sense to remove them.
- We can replace them with ϵ actions and want to compare the resulting abstracted LTSs:
- For instance, by considering that $\xrightarrow{a}\xrightarrow{\epsilon}\xrightarrow{b} \approx \xrightarrow{a}\xrightarrow{b}$;
- We define a simulation with respect to an abstraction criterion:

Definition (Simulation with respect to an abstraction criterion)

A preorder on $S \times S$, $\mathcal{R}$ is a simulation w.r.t. the abstraction criterion $\alpha \subseteq A^* \times A^*$ if: $\forall (s_1, s_2) \in \mathcal{R}, \forall s_2' \in S$ s.t. $s_2 \stackrel{u}{\Rightarrow} s_2', \exists v \in A^*$ s.t. $(u, v) \in \alpha, \exists s_1' \in S$ s.t. $s_1 \stackrel{v}{\Rightarrow} s_1'$ and $(s_1', s_2') \in \mathcal{R}$.

Abstraction

- Some action might not be relevant for a given property: it makes sense to remove them.
- We can replace them with ϵ actions and want to compare the resulting abstracted LTSs:
- For instance, by considering that $\xrightarrow{a}\xrightarrow{\epsilon}\xrightarrow{b}\approx\xrightarrow{a}\xrightarrow{b}$;
- We define a simulation with respect to an abstraction criterion:

Definition (Simulation with respect to an abstraction criterion)

A preorder on $S \times S$, $\mathcal{R}$ is a simulation w.r.t. the abstraction criterion $\alpha \subseteq A^* \times A^*$ if: $\forall (s_1, s_2) \in \mathcal{R}, \forall s_2' \in S$ s.t. $s_2 \overset{u}{\Rightarrow} s_2', \exists v \in A^*$ s.t. $(u, v) \in \alpha, \exists s_1' \in S$ s.t. $s_1 \overset{v}{\Rightarrow} s_1'$ and $(s_1', s_2') \in \mathcal{R}$.

- $\Rightarrow$ is the transitive and reflexive closure of $\rightarrow$;

Abstraction

- Some action might not be relevant for a given property: it makes sense to remove them.
- We can replace them with ϵ actions and want to compare the resulting abstracted LTSs:
- For instance, by considering that $\xrightarrow{a}\xrightarrow{\epsilon}\xrightarrow{b}\approx\xrightarrow{a}\xrightarrow{b}$;
- We define a simulation with respect to an abstraction criterion:

Definition (Simulation with respect to an abstraction criterion)

A preorder on $S \times S$, $\mathcal{R}$ is a simulation w.r.t. the abstraction criterion $\alpha \subseteq A^* \times A^*$ if: $\forall (s_1, s_2) \in \mathcal{R}, \forall s_2' \in S$ s.t. $s_2 \overset{u}{\Rightarrow} s_2', \exists v \in A^*$ s.t. $(u, v) \in \alpha, \exists s_1' \in S$ s.t. $s_1 \overset{v}{\Rightarrow} s_1'$ and $(s_1', s_2') \in \mathcal{R}$.

- $\Rightarrow$ is the transitive and reflexive closure of $\rightarrow$;
- Weak simulation : $\alpha = \{(a, \epsilon^* a \epsilon^*), a \in A\}$ (Milner's Observational Equivalence [Mil80]);

Abstraction

- Some action might not be relevant for a given property: it makes sense to remove them.
- We can replace them with ϵ actions and want to compare the resulting abstracted LTSs:
- For instance, by considering that $\xrightarrow{a}\xrightarrow{\epsilon}\xrightarrow{b}\approx\xrightarrow{a}\xrightarrow{b}$;
- We define a simulation with respect to an abstraction criterion:

Definition (Simulation with respect to an abstraction criterion)

A preorder on $S \times S$, $\mathcal{R}$ is a simulation w.r.t. the abstraction criterion $\alpha \subseteq A^* \times A^*$ if: $\forall (s_1, s_2) \in \mathcal{R}, \forall s_2' \in S$ s.t. $s_2 \overset{u}{\Rightarrow} s_2', \exists v \in A^*$ s.t. $(u, v) \in \alpha, \exists s_1' \in S$ s.t. $s_1 \overset{v}{\Rightarrow} s_1'$ and $(s_1', s_2') \in \mathcal{R}$.

- $\Rightarrow$ is the transitive and reflexive closure of $\rightarrow$;
- Weak simulation : $\alpha = \{(a, \epsilon^* a \epsilon^*), a \in A\}$ (Milner's Observational Equivalence [Mil80]);
- (Strong) Simulation : $\alpha = \{(a, a), a \in A\}$;

Bisimulation

- A system that is simulated by its specification conforms to it;

Bisimulation

- A system that is simulated by its specification conforms to it;
- A system that simulates its specification has at least all specified behaviors; indésirable ;

Bisimulation

- A system that is simulated by its specification conforms to it;
- A system that simulates its specification has at least all specified behaviors; indésirable ;
- We might want both:

Definition (Bisimulation)

A simulation $\mathcal{R} \subseteq S \times S$ of an LTS $\mathcal{S}$ is a bisimulation if $\mathcal{R}^{-1}$ is also a simulation of $\mathcal{S}$.
$s\mathcal{R}^{-1}s'$ iff $s'\mathcal{R}s$

Theorem

Computing the greatest bisimulation can be done in polynomial time for finite transition systems.

Bisimulation

- A system that is simulated by its specification conforms to it;
- A system that simulates its specification has at least all specified behaviors; indésirable ;
- We might want both:

Definition (Bisimulation)

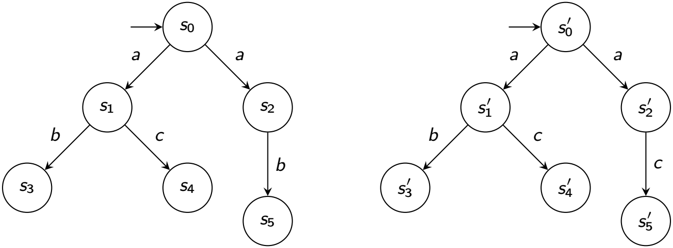
A simulation $\mathcal{R} \subseteq S \times S$ of an LTS $\mathcal{S}$ is a bisimulation if $\mathcal{R}^{-1}$ is also a simulation of $\mathcal{S}$.
$s\mathcal{R}^{-1}s'$ iff $s'\mathcal{R}s$

Theorem

Computing the greatest bisimulation can be done in polynomial time for finite transition systems.

- We can have $\mathcal{S}$ simulates $\mathcal{S}'$ by some relation $\mathcal{R}$, $\mathcal{S}'$ simulates $\mathcal{S}$ by some relation $\mathcal{R}'$ and $\mathcal{R}' \neq \mathcal{R}^{-1}$ (We call this a co-simulation).
  Example ?

Co-simulation



Co-simulation and language

- Bisimulation $\Rightarrow$ Co-simulation $\Rightarrow$ language equality

Co-simulation and language

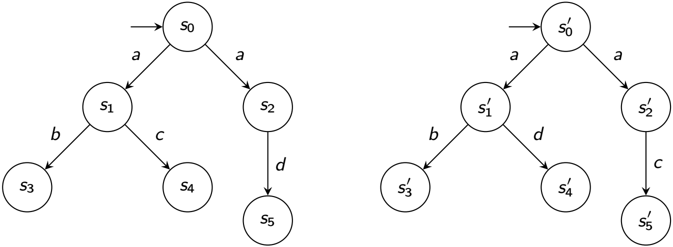
- Bisimulation $\Rightarrow$ Co-simulation $\Rightarrow$ language equality
- Co-simulation $\not\Rightarrow$ Bisimulation (cf. supra)

Co-simulation and language

- Bisimulation $\Rightarrow$ Co-simulation $\Rightarrow$ language equality
- Co-simulation $\not\Rightarrow$ Bisimulation (cf. supra)
- Language equality $\Rightarrow$ Co-simulation ?

Co-simulation and language

- Bisimulation $\Rightarrow$ Co-simulation $\Rightarrow$ language equality
- Co-simulation $\not\Leftarrow$ Bisimulation (cf. supra)
- Language equality $\Rightarrow$ Co-simulation ?
- No :



Computing the Greatest Bisimulation

- Bisimulation is an equivalence on arbitrarily long behaviors;

Computing the Greatest Bisimulation

- Bisimulation is an equivalence on arbitrarily long behaviors;
- We compute the greatest bisimulation $\approx$ step by step:

Computing the Greatest Bisimulation

- Bisimulation is an equivalence on arbitrarily long behaviors;
- We compute the greatest bisimulation $\approx$ step by step:
    1. for all states s_1 and s_2, we have $s_1 \approx_0 s_2$;

Computing the Greatest Bisimulation

- Bisimulation is an equivalence on arbitrarily long behaviors;
- We compute the greatest bisimulation $\approx$ step by step:
  1. for all states s_1 and s_2, we have $s_1 \approx_0 s_2$;
  2. $s_1 \approx_1 s_2$ iff $s_1 \xrightarrow{a} \Leftrightarrow s_2 \xrightarrow{a}$ (they can do exactly the same actions) and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_0 s_2'$;

Computing the Greatest Bisimulation

- Bisimulation is an equivalence on arbitrarily long behaviors;
- We compute the greatest bisimulation $\approx$ step by step:
  1. for all states s_1 and s_2, we have $s_1 \approx_0 s_2$;
  2. $s_1 \approx_1 s_2$ iff $s_1 \xrightarrow{a} \Leftrightarrow s_2 \xrightarrow{a}$ (they can do exactly the same actions) and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_0 s_2'$;
  3. $s_1 \approx_2 s_2$ iff $s_1 \approx_1 s_2$ and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_1 s_2'$;

Computing the Greatest Bisimulation

- Bisimulation is an equivalence on arbitrarily long behaviors;
- We compute the greatest bisimulation $\approx$ step by step:
  1. for all states s_1 and s_2, we have $s_1 \approx_0 s_2$;
  2. $s_1 \approx_1 s_2$ iff $s_1 \xrightarrow{a} \Leftrightarrow s_2 \xrightarrow{a}$ (they can do exactly the same actions) and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_0 s_2'$;
  3. $s_1 \approx_2 s_2$ iff $s_1 \approx_1 s_2$ and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_1 s_2'$;
  4. ...

Computing the Greatest Bisimulation

- Bisimulation is an equivalence on arbitrarily long behaviors;
- We compute the greatest bisimulation $\approx$ step by step:
  1. for all states s_1 and s_2, we have $s_1 \approx_0 s_2$;
  2. $s_1 \approx_1 s_2$ iff $s_1 \xrightarrow{a} \Leftrightarrow s_2 \xrightarrow{a}$ (they can do exactly the same actions) and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_0 s_2'$;
  3. $s_1 \approx_2 s_2$ iff $s_1 \approx_1 s_2$ and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_1 s_2'$;
  4. ...
  5. $s_1 \approx_n s_2$ iff $s_1 \approx_{n-1} s_2$ and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_{n-1} s_2'$;

Computing the Greatest Bisimulation

- Bisimulation is an equivalence on arbitrarily long behaviors;
- We compute the greatest bisimulation $\approx$ step by step:
  1. for all states s_1 and s_2, we have $s_1 \approx_0 s_2$;
  2. $s_1 \approx_1 s_2$ iff $s_1 \xrightarrow{a} \Leftrightarrow s_2 \xrightarrow{a}$ (they can do exactly the same actions) and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_0 s_2'$;
  3. $s_1 \approx_2 s_2$ iff $s_1 \approx_1 s_2$ and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_1 s_2'$;
  4. ...
  5. $s_1 \approx_n s_2$ iff $s_1 \approx_{n-1} s_2$ and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_{n-1} s_2'$;
  6. ...

Computing the Greatest Bisimulation

- Bisimulation is an equivalence on arbitrarily long behaviors;
- We compute the greatest bisimulation $\approx$ step by step:
  1. for all states s_1 and s_2, we have $s_1 \approx_0 s_2$;
  2. $s_1 \approx_1 s_2$ iff $s_1 \xrightarrow{a} \Leftrightarrow s_2 \xrightarrow{a}$ (they can do exactly the same actions) and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_0 s_2'$;
  3. $s_1 \approx_2 s_2$ iff $s_1 \approx_1 s_2$ and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_1 s_2'$;
  4. ...
  5. $s_1 \approx_n s_2$ iff $s_1 \approx_{n-1} s_2$ and if $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$ then $s_1' \approx_{n-1} s_2'$;
  6. ...
  7. Termination?

Computing the Greatest Bisimulation

- Let $F : S_1 \times S_2 \to S_1 \times S_2$ be defined by:

$$F(E) = \{(s_1, s_2) \in E | s_1 \xrightarrow{a} \Leftrightarrow s_2 \xrightarrow{a} \text{ et } s_1 \xrightarrow{a} s_1' \text{ et } s_2 \xrightarrow{a} s_2' \Rightarrow (s_1', s_2') \in E\}$$

Computing the Greatest Bisimulation

- Let $F : S_1 \times S_2 \rightarrow S_1 \times S_2$ be defined by:

$$F(E) = \{(s_1, s_2) \in E | s_1 \xrightarrow{a} \Leftrightarrow s_2 \xrightarrow{a} \text{ et } s_1 \xrightarrow{a} s_1' \text{ et } s_2 \xrightarrow{a} s_2' \Rightarrow (s_1', s_2') \in E\}$$

- We have $\approx_n = F^n(S_1 \times S_2)$ and this converges towards the greatest bisimulation $\approx$ between the two LTSs;

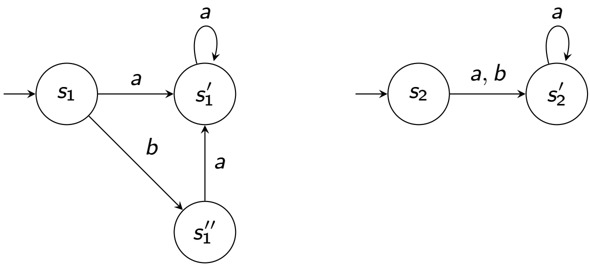
Computing the Greatest Bisimulation

- Let $F : S_1 \times S_2 \rightarrow S_1 \times S_2$ be defined by:

$$F(E) = \{(s_1, s_2) \in E | s_1 \xrightarrow{a} \Leftrightarrow s_2 \xrightarrow{a} \text{ et } s_1 \xrightarrow{a} s_1' \text{ et } s_2 \xrightarrow{a} s_2' \Rightarrow (s_1', s_2') \in E\}$$

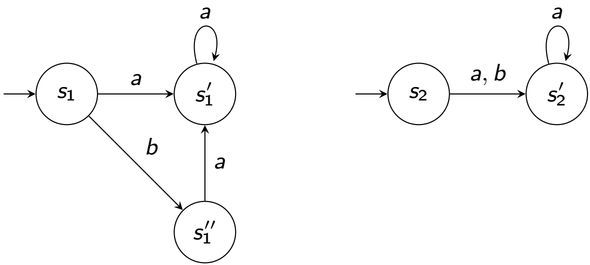
- We have $\approx_n = F^n(S_1 \times S_2)$ and this converges towards the greatest bisimulation $\approx$ between the two LTSs;
- They are bisimilar if $(s_1^0, s_2^0) \in \approx$.

Computing the Greatest Bisimulation: Example



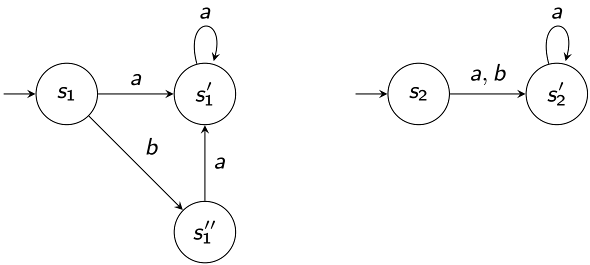
- $\approx_0 = S_1 \times S_2$

Computing the Greatest Bisimulation: Example



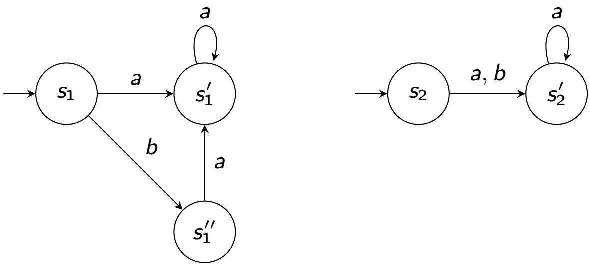
- $\approx_0 = S_1 \times S_2$
- $\approx_1 = F(\approx_0) = \{(s_1, s_2), (s_1', s_2'), (s_1'', s_2')\}$

Computing the Greatest Bisimulation: Example



- $\approx_0 = S_1 \times S_2$
- $\approx_1 = F(\approx_0) = \{(s_1, s_2), (s_1', s_2'), (s_1'', s_2')\}$
- $\approx_2 = F(\approx_1) = \approx_1$

Computing the Greatest Bisimulation: Example



- $\approx_0 = S_1 \times S_2$
- $\approx_1 = F(\approx_0) = \{(s_1, s_2), (s_1', s_2'), (s_1'', s_2')\}$
- $\approx_2 = F(\approx_1) = \approx_1$
- The two LTSs are bisimilar.

Quotient of an LTS by an equivalence relation

Definition (Quotient System)

Let $\mathcal{S} = (S, S_0, A, \rightarrow)$ be an LTS and $\approx$ be an equivalence relation on $S \times S$. The quotient of $\mathcal{S}$ by $\approx$ is the LTS $\mathcal{S}/\approx = (\hat{S}, \hat{S}_0, \hat{\rightarrow})$ defined by:

- $\hat{S} = S/\approx$ (equivalence classes of $\approx$) ;
- $\hat{S}_0 = \{\hat{s}, s \in S_0\}$;
- $\hat{s} \overset{a}{\hat{\rightarrow}} \hat{s}' \Leftrightarrow s \xrightarrow{a} s'$

($\hat{s}$ is the equivalence class of s.)

Quotient of an LTS by an equivalence relation

Definition (Quotient System)

Let $\mathcal{S} = (S, S_0, A, \rightarrow)$ be an LTS and $\approx$ be an equivalence relation on $S \times S$. The quotient of $\mathcal{S}$ by $\approx$ is the LTS $\mathcal{S}/\approx = (\hat{S}, \hat{S}_0, \hat{\rightarrow})$ defined by:

- $\hat{S} = S/\approx$ (equivalence classes of $\approx$) ;
- $\hat{S}_0 = \{\hat{s}, s \in S_0\}$;
- $\hat{s} \overset{a}{\hat{\rightarrow}} \hat{s}' \Leftrightarrow s \xrightarrow{a} s'$

($\hat{s}$ is the equivalence class of s.)

- By definition, $\mathcal{S} \approx \mathcal{S}/\approx$;
- The quotient system is an equivalent minimization (par $\approx$) of the original system;

Quotient of an LTS by an equivalence relation

Definition (Quotient System)

Let $\mathcal{S} = (S, S_0, A, \rightarrow)$ be an LTS and $\approx$ be an equivalence relation on $S \times S$. The quotient of $\mathcal{S}$ by $\approx$ is the LTS $\mathcal{S}/\approx = (\hat{S}, \hat{S}_0, \hat{\rightarrow})$ defined by:

- $\hat{S} = S/\approx$ (equivalence classes of $\approx$) ;
- $\hat{S}_0 = \{\hat{s}, s \in S_0\}$;
- $\hat{s} \overset{a}{\hat{\rightarrow}} \hat{s}' \Leftrightarrow s \xrightarrow{a} s'$

($\hat{s}$ is the equivalence class of s.)

- By definition, $\mathcal{S} \approx \mathcal{S}/\approx$;
- The quotient system is an equivalent minimization (par $\approx$) of the original system;
- Minimization by bisimulation, observation equivalence, etc.

Minimization by Bisimulation

- We can compute the greatest bisimulation between the states of one LTS using the previous algorithm;

Minimization by Bisimulation

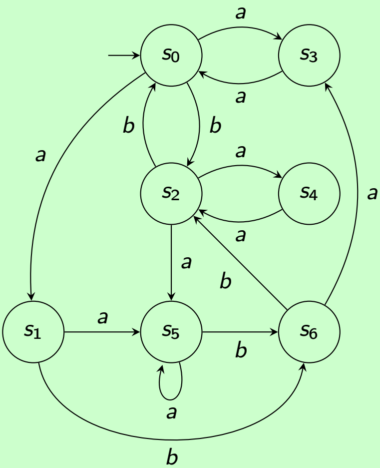
- We can compute the greatest bisimulation between the states of one LTS using the previous algorithm;
- Then we can merge bisimilar states as explained;

Minimization by Bisimulation

- We can compute the greatest bisimulation between the states of one LTS using the previous algorithm;
- Then we can merge bisimilar states as explained;
- This gives a minimized bisimilar LTS.

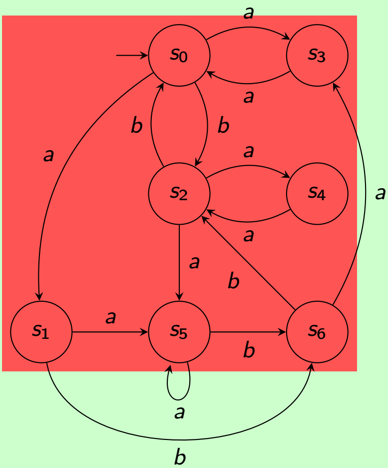
Minimization by Bisimulation

Example



Minimization by Bisimulation

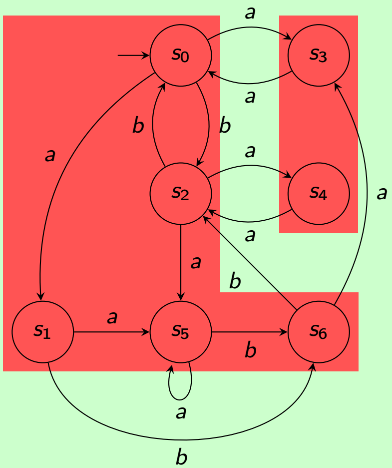
Example



Equivalence classes of $\approx_0$:
$A = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$

Minimization by Bisimulation

Example

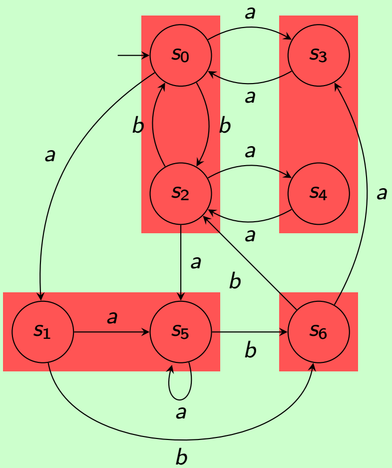


$\approx_1$	a	b
s_0	A	A
s_1	A	A
s_2	A	A
s_3	A	$\emptyset$
s_4	A	$\emptyset$
s_5	A	A
s_6	A	A

Equivalence classes of $\approx_1$:
$B = \{s_0, s_1, s_2, s_5, s_6\}$ and
$C = \{s_3, s_4\}$

Minimization by Bisimulation

Example

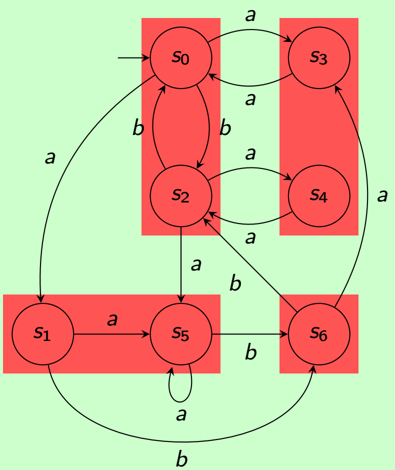


$\approx_2$	a	b
s_0	B, C	B
s_1	B	B
s_2	B, C	B
s_5	B	B
s_6	C	B
s_3	B	$\emptyset$
s_4	B	$\emptyset$

Equivalence classes of $\approx_2$:
$D = \{s_0, s_2\}$, $E = \{s_1, s_5\}$,
$F = \{s_6\}$, $G = \{s_3, s_4\}$

Minimization by Bisimulation

Example

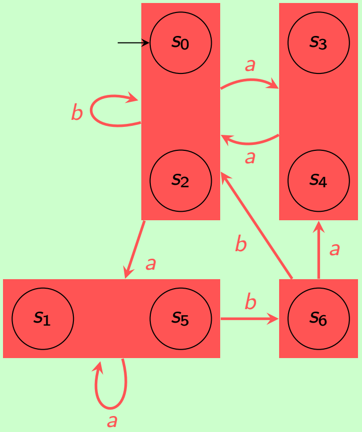


$\approx_3$	a	b
s_0	E, G	D
s_2	E, G	D
s_1	E	F
s_5	E	F
s_6	G	D
s_3	D	$\emptyset$
s_4	D	$\emptyset$

We have $\approx_3 = \approx_2$

Minimization by Bisimulation

Example



Application: Minimization of an NFA

- Determinization possibly gives very big automata;

Application: Minimization of an NFA

- Determinization possibly gives very big automata;
- We can minimize them using the previous technique;

Application: Minimization of an NFA

- Determinization possibly gives very big automata;
- We can minimize them using the previous technique;
- We need the bisimulation to be compatible with the acceptance condition:

$$(s, s') \in \mathcal{R} \Rightarrow (s \in F \Leftrightarrow s' \in F)$$

Application: Minimization of an NFA

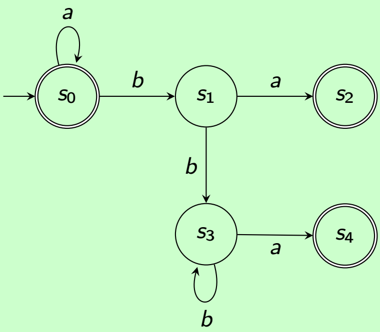
- Determinization possibly gives very big automata;
- We can minimize them using the previous technique;
- We need the bisimulation to be compatible with the acceptance condition:

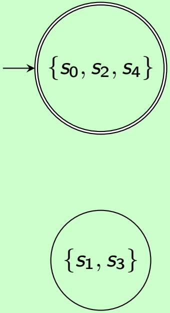
$$(s, s') \in \mathcal{R} \Rightarrow (s \in F \Leftrightarrow s' \in F)$$

- In the fix point, the initial equivalence classes of $\approx_0$ are F and $S \setminus F$.

Application: Minimization of an NFA

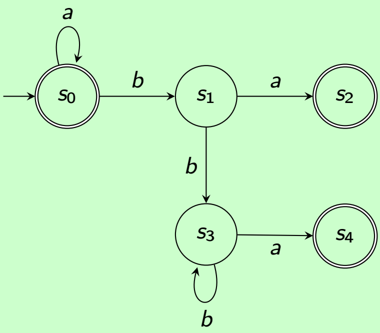
Example

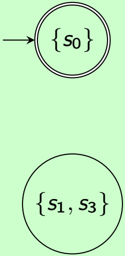




Application: Minimization of an NFA

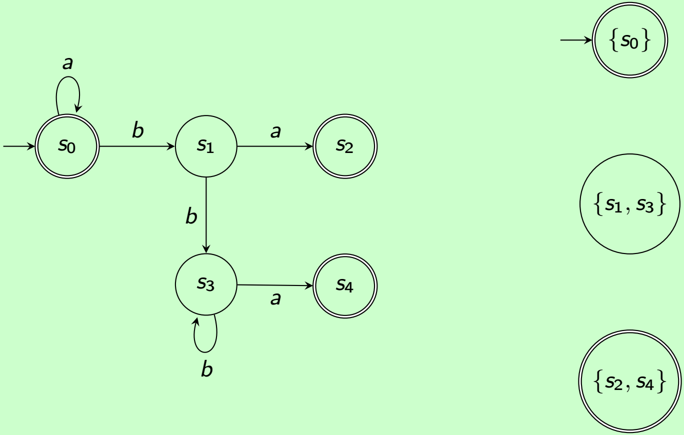
Example





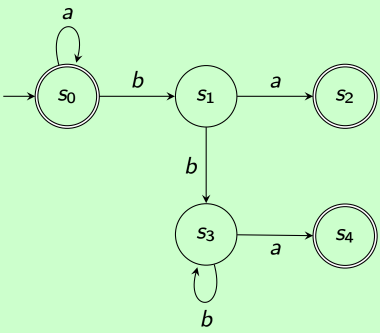
Application: Minimization of an NFA

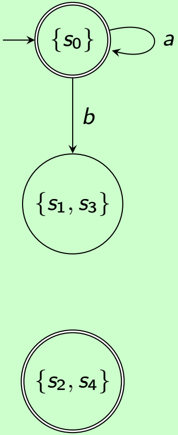
Example



Application: Minimization of an NFA

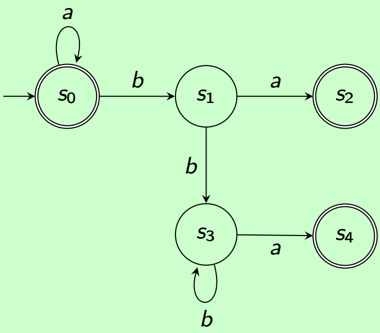
Example

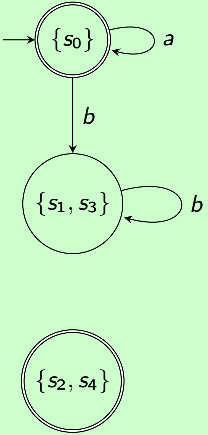




Application: Minimization of an NFA

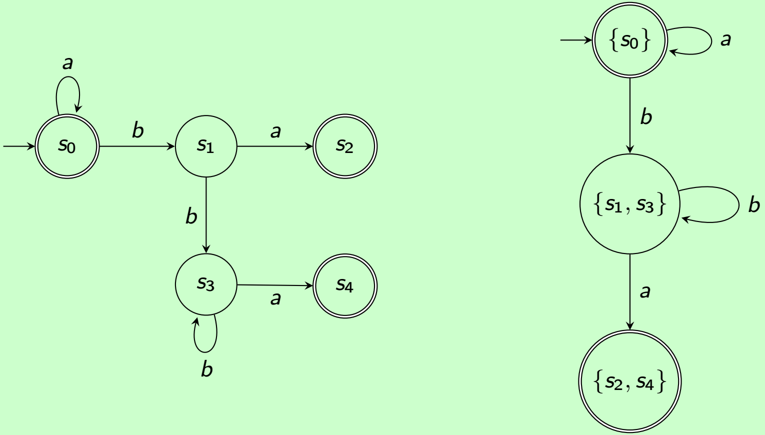
Example





Application: Minimization of an NFA

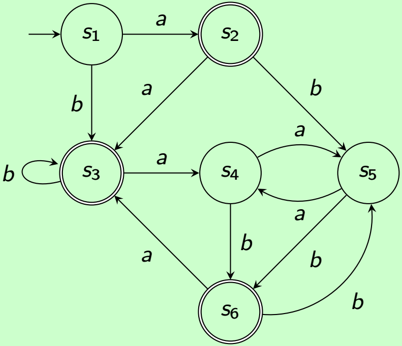
Example



Minimization of an NFA: Exercise

Exercise

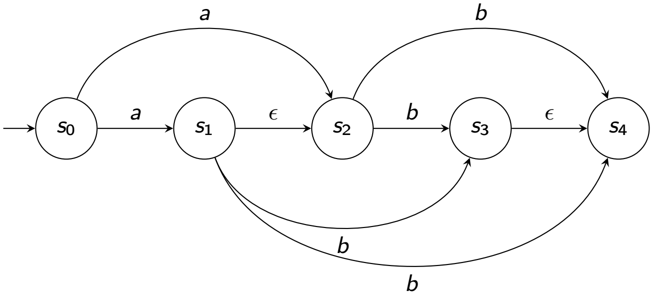
Minimize the following NFA, preserving bisimulation:



Minimizing using Observational Equivalence

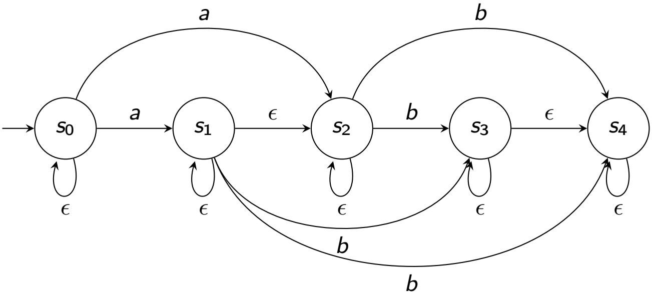
$$\rightarrow s_0 \xrightarrow{a} s_1 \xrightarrow{\epsilon} s_2 \xrightarrow{b} s_3 \xrightarrow{\epsilon} s_4$$

Minimizing using Observational Equivalence



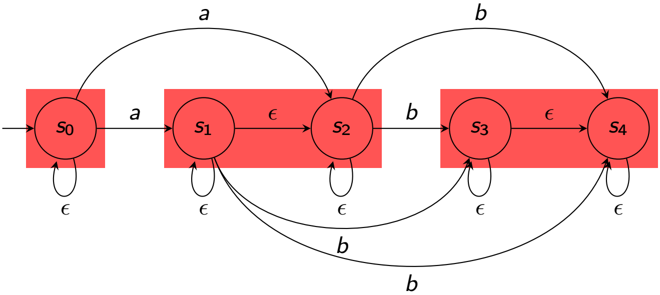
1. ϵ-saturation (transitive closure);

Minimizing using Observational Equivalence



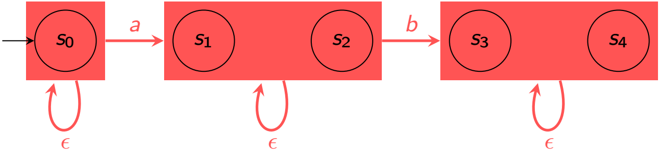
1. ϵ-saturation (transitive closure);
2. Add ϵ self-loops (reflexive closure);

Minimizing using Observational Equivalence



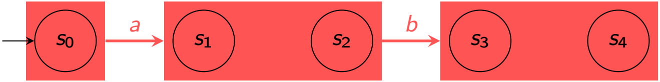
1. ϵ-saturation (transitive closure);
2. Add ϵ self-loops (reflexive closure);
3. Minimize using (strong) bisimulation;

Minimizing using Observational Equivalence



1. ϵ-saturation (transitive closure);
2. Add ϵ self-loops (reflexive closure);
3. Minimize using (strong) bisimulation;

Minimizing using Observational Equivalence

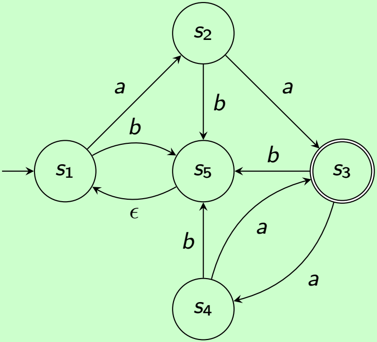


1. ϵ-saturation (transitive closure);
2. Add ϵ self-loops (reflexive closure);
3. Minimize using (strong) bisimulation;
4. Remove ϵ self-loops.

Minimizing using Observational Equivalence: Exercise

Exercise

Minimize the following NFA using observational equivalence (weak bisimulation):



Plan I

Introduction

- It is not always easy to model the specification as an automaton;

Introduction

- It is not always easy to model the specification as an automaton;
- It is usually more convenient to describe it as a set of requirements;

Introduction

- It is not always easy to model the specification as an automaton;
- It is usually more convenient to describe it as a set of requirements;
- Each requirement can then be check on the model of the system;

Introduction

- It is not always easy to model the specification as an automaton;
- It is usually more convenient to describe it as a set of requirements;
- Each requirement can then be check on the model of the system;
- For an automatic and sound procedure we need fomalized requirements:

Model-checking

Let $\mathcal{S}$ be a model of the system and let φ be a (temporal) logic formula

$$\mathcal{S} \models \varphi?$$

Classic Logics

- Propositionnal logic, under Backus-Naur Form (BNF)):

$$\varphi ::= \mathbf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

Classic Logics

- Propositionnal logic, under Backus-Naur Form (BNF)):

$$\varphi ::= \mathbf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

- First-order logic:

$$\varphi ::= \mathbf{p}(x) \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \mid \forall x.\varphi$$

Classic Logics

- Propositionnal logic, under Backus-Naur Form (BNF)):

$$\varphi ::= \mathbf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

- First-order logic:

$$\varphi ::= \mathbf{p}(x) \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \mid \forall x.\varphi$$

- Second-order logic, monadic second-order logic, etc.

Temporal Logic

- Temporal logics allow to refer to time in a qualitative manner;

Temporal Logic

- Temporal logics allow to refer to time in a qualitative manner;
- Until, since, always, never, eventually, etc.

Temporal Logic

- Temporal logics allow to refer to time in a qualitative manner;
- Until, since, always, never, eventually, etc.
- We study two such logics:

Temporal Logic

- Temporal logics allow to refer to time in a qualitative manner;
- Until, since, always, never, eventually, etc.
- We study two such logics:
  - LTL: properties on runs;

Temporal Logic

- Temporal logics allow to refer to time in a qualitative manner;
- Until, since, always, never, eventually, etc.
- We study two such logics:
  - LTL: properties on runs;
  - CTL: properties on execution trees.

Kripke Structures

Definition (Kripke Structure)

A Kripke Structure is a tuple $(AP, W, \rightarrow, \ell)$ where:

- AP is a set of atomic propositions;
- W is a non-empty set of states;
- $\rightarrow \subseteq W \times W$ is a (left-total) relation called transition relation;
- $\ell : W \rightarrow 2^{\mathrm{AP}}$ is a labeling (or interpretation) function.

Linear Temporal Logic

- The Propositional Linear Temporal Logic (LTL) [Pnu77] specifies properties of runs;

Linear Temporal Logic

- The Propositional Linear Temporal Logic (LTL) [Pnu77] specifies properties of runs;
- We have a set of AP of atomic propositions;

Syntax of LTL

$$\varphi ::= \mathbf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi$$

Linear Temporal Logic

- The Propositional Linear Temporal Logic (LTL) [Pnu77] specifies properties of runs;
- We have a set of AP of atomic propositions;

Syntax of LTL

$$\varphi ::= \mathbf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi$$

- true $= \mathbf{p} \vee \neg\mathbf{p}$, false $= \neg$true;

Linear Temporal Logic

- The Propositional Linear Temporal Logic (LTL) [Pnu77] specifies properties of runs;
- We have a set of AP of atomic propositions;

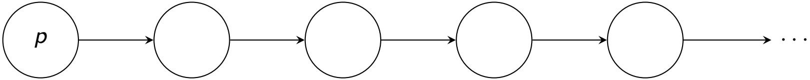
Syntax of LTL

$$\varphi ::= \mathbf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi$$

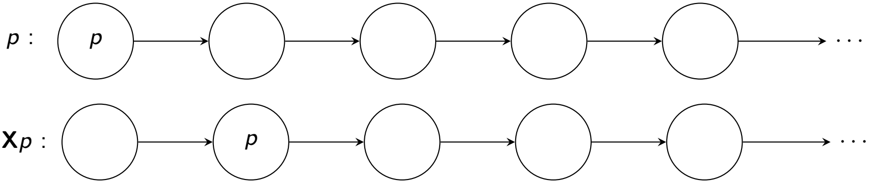
- true $= \mathbf{p} \vee \neg\mathbf{p}$, false $= \neg$true;
- Time modalities: Until **U** and neXt **X**.

Intuitive Semantics of LTL

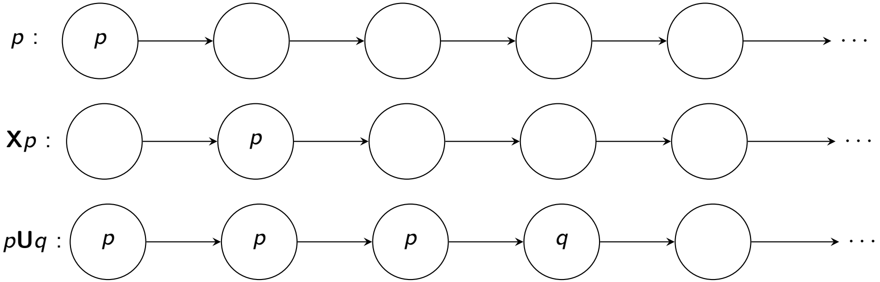
p :



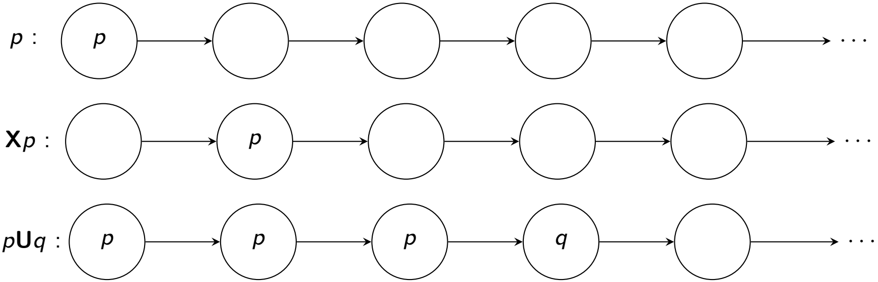
Intuitive Semantics of LTL



Intuitive Semantics of LTL



Intuitive Semantics of LTL



Exercise

Give an example run statisfying $p\mathbf{U}\neg(\mathtt{true}\mathbf{U}\neg q)$.

Formal Semantics of LTL

We consider the infinite traces (wrt. atomic propositions) of a Kripke structure:

$$a_0 a_1 a_2 \cdots \text{ with } \forall i, a_i \in 2^{\mathsf{AP}}$$

Formal Semantics of LTL

- $a_0 a_1 a_2 \cdots \models \mathbf{p}$ iff $\mathbf{p} \in a_0$;

Formal Semantics of LTL

We consider the infinite traces (wrt. atomic propositions) of a Kripke structure:

$$a_0 a_1 a_2 \cdots \text{ with } \forall i, a_i \in 2^{\mathsf{AP}}$$

Formal Semantics of LTL

- $a_0 a_1 a_2 \cdots \models \mathbf{p}$ iff $\mathbf{p} \in a_0$;
- $a_0 a_1 a_2 \cdots \models \neg\varphi$ iff $a_0 a_1 a_2 \cdots \not\models \varphi$;

Formal Semantics of LTL

We consider the infinite traces (wrt. atomic propositions) of a Kripke structure:

$$a_0 a_1 a_2 \cdots \text{ with } \forall i, a_i \in 2^{\mathsf{AP}}$$

Formal Semantics of LTL

- $a_0 a_1 a_2 \cdots \models \mathbf{p}$ iff $\mathbf{p} \in a_0$;
- $a_0 a_1 a_2 \cdots \models \neg\varphi$ iff $a_0 a_1 a_2 \cdots \not\models \varphi$;
- $a_0 a_1 a_2 \cdots \models \varphi \vee \psi$ iff $a_0 a_1 a_2 \cdots \models \varphi$ or $a_0 a_1 a_2 \cdots \models \psi$;

Formal Semantics of LTL

We consider the infinite traces (wrt. atomic propositions) of a Kripke structure:

$$a_0 a_1 a_2 \cdots \text{ with } \forall i, a_i \in 2^{\mathrm{AP}}$$

Formal Semantics of LTL

- $a_0 a_1 a_2 \cdots \models \mathbf{p}$ iff $\mathbf{p} \in a_0$;
- $a_0 a_1 a_2 \cdots \models \neg\varphi$ iff $a_0 a_1 a_2 \cdots \not\models \varphi$;
- $a_0 a_1 a_2 \cdots \models \varphi \vee \psi$ iff $a_0 a_1 a_2 \cdots \models \varphi$ or $a_0 a_1 a_2 \cdots \models \psi$;
- $a_0 a_1 a_2 \cdots \models \mathbf{X}\varphi$ iff $a_1 a_2 \cdots \models \varphi$;

Formal Semantics of LTL

We consider the infinite traces (wrt. atomic propositions) of a Kripke structure:

$$a_0 a_1 a_2 \cdots \text{ with } \forall i, a_i \in 2^{\mathrm{AP}}$$

Formal Semantics of LTL

- $a_0 a_1 a_2 \cdots \models \mathbf{p}$ iff $\mathbf{p} \in a_0$;
- $a_0 a_1 a_2 \cdots \models \neg\varphi$ iff $a_0 a_1 a_2 \cdots \not\models \varphi$;
- $a_0 a_1 a_2 \cdots \models \varphi \vee \psi$ iff $a_0 a_1 a_2 \cdots \models \varphi$ or $a_0 a_1 a_2 \cdots \models \psi$;
- $a_0 a_1 a_2 \cdots \models \mathbf{X}\varphi$ iff $a_1 a_2 \cdots \models \varphi$;
- $a_0 a_1 a_2 \cdots \models \varphi \mathbf{U} \psi$ iff $\exists u \geq 0$ s.t. $a_u a_{u+1} \cdots \models \psi$ and $\forall v$ s.t. $0 \leq v < u, a_v a_{v+1} \cdots \models \varphi$.

Other Classic Modalities

- We can define **F** (finally/future) and **G** (globally):

Other Classic Modalities

- We can define **F** (finally/future) and **G** (globally):
  - $\mathbf{F}\varphi = \mathtt{true}\mathbf{U}\varphi$;

Other Classic Modalities

- We can define **F** (finally/future) and **G** (globally):
  - $\mathbf{F}\varphi = \mathtt{true}\mathbf{U}\varphi$;
  - $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$;

Other Classic Modalities

- We can define **F** (finally/future) and **G** (globally):
  - $\mathbf{F}\varphi = \mathtt{true}\mathbf{U}\varphi$;
  - $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$;
- Combinations:

Other Classic Modalities

- We can define **F** (finally/future) and **G** (globally):
  - $\mathbf{F}\varphi = \mathtt{true}\mathbf{U}\varphi$;
  - $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$;
- Combinations:
  - **GF**: ;

Other Classic Modalities

- We can define **F** (finally/future) and **G** (globally):
  - $\mathbf{F}\varphi = \mathtt{true}\mathbf{U}\varphi$;
  - $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$;
- Combinations:
  - **GF**:infinitely often ;

Other Classic Modalities

- We can define **F** (finally/future) and **G** (globally):
  - $\mathbf{F}\varphi = \mathtt{true}\mathbf{U}\varphi$;
  - $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$;
- Combinations:
  - **GF**:infinitely often ;
  - **FG**: ;

Other Classic Modalities

- We can define **F** (finally/future) and **G** (globally):
  - $\mathbf{F}\varphi = \mathtt{true}\mathbf{U}\varphi$;
  - $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$;
- Combinations:
  - **GF**:infinitely often ;
  - **FG**:almost always ;

Other Classic Modalities

- We can define **F** (finally/future) and **G** (globally):
  - $\mathbf{F}\varphi = \mathtt{true}\mathbf{U}\varphi$;
  - $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$;
- Combinations:
  - **GF**:infinitely often ;
  - **FG**:almost always ;
- Weak until: $\varphi\mathbf{W}\psi = \mathbf{G}\varphi \vee \varphi\mathbf{U}\psi$.

Other Classic Modalities

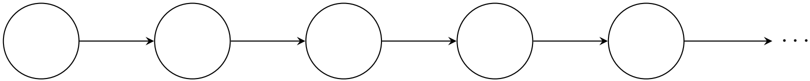
- We can define $\mathbf{F}$ (finally/future) and $\mathbf{G}$ (globally):
  - $\mathbf{F}\varphi = \mathtt{true}\mathbf{U}\varphi$;
  - $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$;
- Combinations:
  - **GF**:infinitely often ;
  - **FG**:almost always ;
- Weak until: $\varphi\mathbf{W}\psi = \mathbf{G}\varphi \vee \varphi\mathbf{U}\psi$.

Exercise

Release is the dual of Until: $\varphi\mathbf{R}\psi = \neg(\neg\varphi\mathbf{U}\neg\psi)$. Express $\mathbf{R}$ in function of $\mathbf{U}$ and $\mathbf{G}$ without any negation.

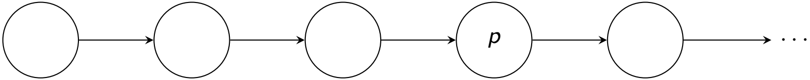
Examples

$\mathbf{F}p$:

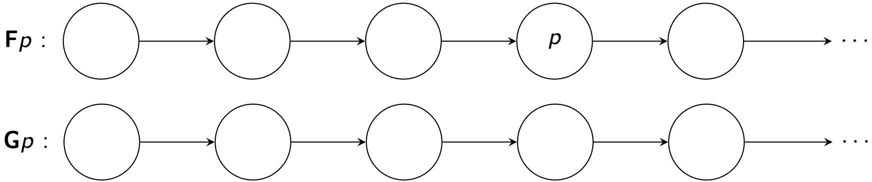


Examples

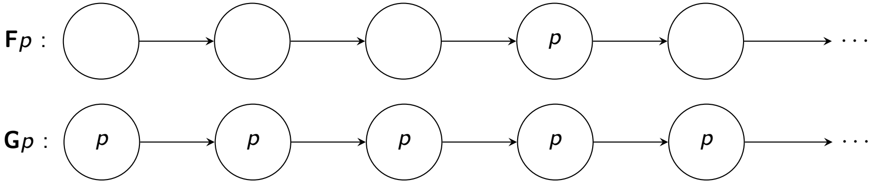
$\mathbf{F}p$:



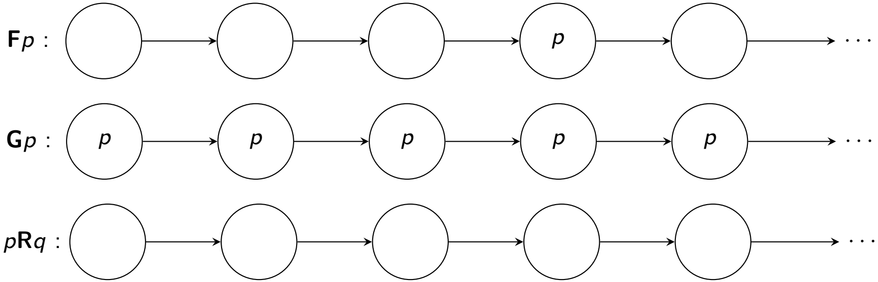
Examples



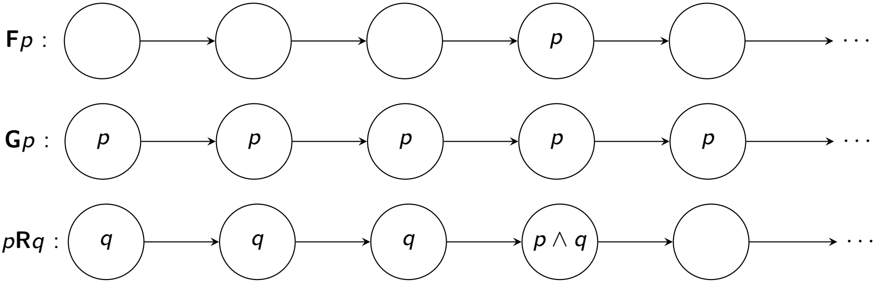
Examples



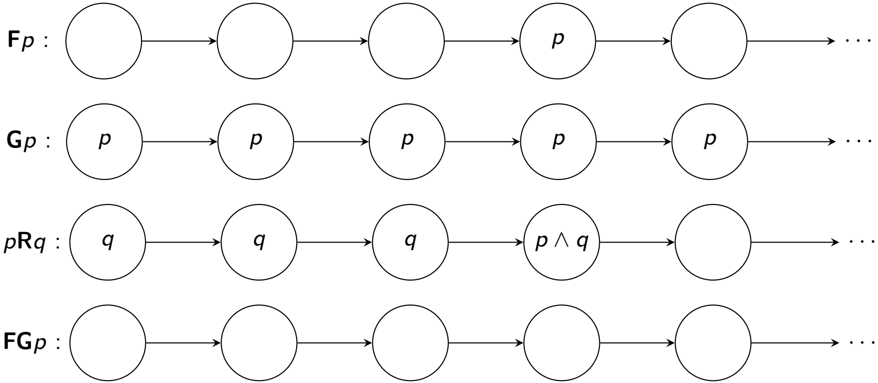
Examples



Examples

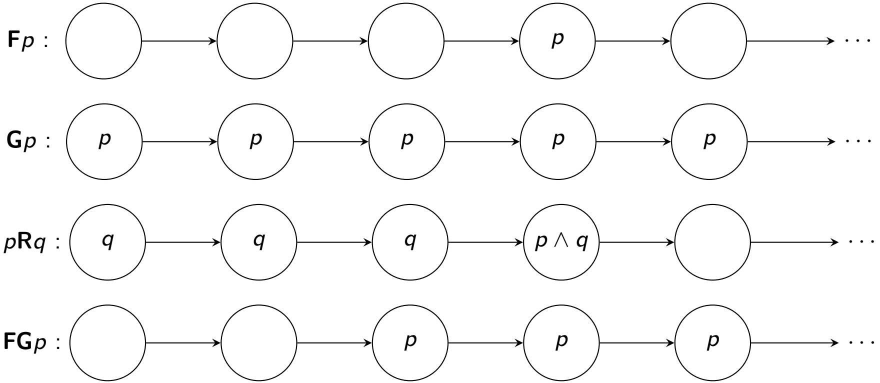


Examples

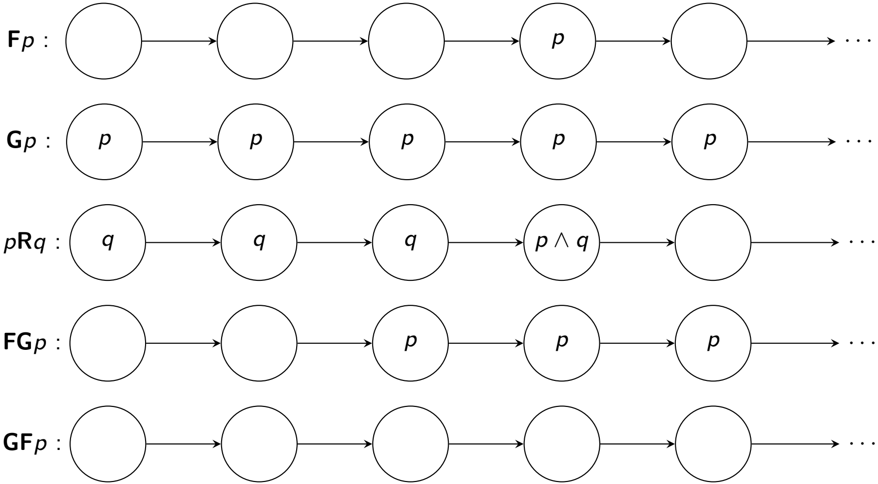


$\mathbf{F}p$:
p
· · ·
$\mathbf{G}p$:
p
p
p
p
p
· · ·
$p\mathbf{R}q$:
q
q
q
$p \wedge q$
· · ·
$\mathbf{FG}p$:
· · ·

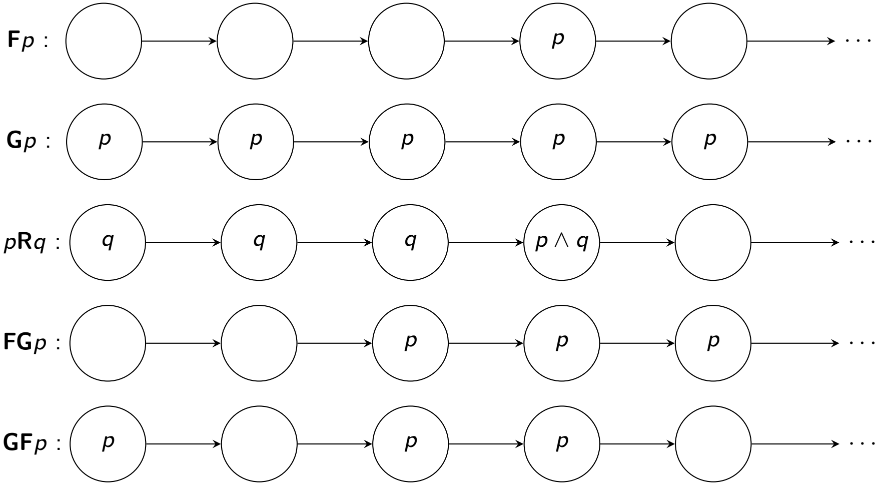
Examples



Examples



Examples



Some Classic Properties

- Reachability: $\mathbf{F}p$;

Some Classic Properties

- Reachability: $\mathbf{F}p$;
- Safety: $\mathbf{G}(p \Rightarrow \neg q)$;

Some Classic Properties

- Reachability: $\mathbf{F}p$;
- Safety: $\mathbf{G}(p \Rightarrow \neg q)$;
- Liveness: $\mathbf{GF}p$;

Some Classic Properties

- Reachability: $\mathbf{F}p$;
- Safety: $\mathbf{G}(p \Rightarrow \neg q)$;
- Liveness: $\mathbf{GF}p$;
- Response: $\mathbf{G}(p \Rightarrow \mathbf{F}q)$;

LTL Properties: Exercise

Exercise

Consider a plane transporting passengers between Nantes and Amsterdam. It continuously executes the following cycle: the plane is empty in Nantes; wait 1h for next departure (might be skipped or repeated); passengers embark; the plane flies to Amsterdam; passengers disembark; wait 1h for next departure (might be skipped or repeated); new passengers embark; the plane flies back to Nantes; passenger disembark.

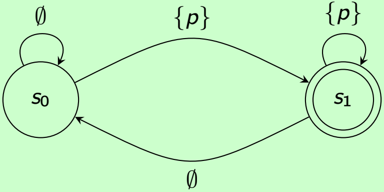
Model this problem as a Kripke Structure and write LTL formulas for the following properties and assess their truth on the model:

1. The plane will eventually be empty in Amsterdam;
2. The plane is always infinitely often in Nantes;
3. If it never waits indefinitely, the plane is always infinitely often in Nantes;
4. Whenever the plane is in Amsterdam, it is next full in Nantes;
5. Whenever the plane is full, it will be empty and then full again (assuming fairness);
6. Whenever the plane is full in Nantes and ready to fly out, it cannot be empty in Nantes before being first empty in Amsterdam.

LTL and Büchi Automata

Example

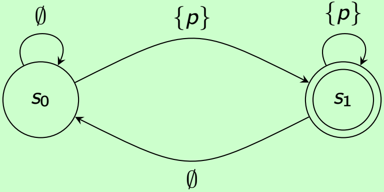
$\mathbf{GF}p$



LTL and Büchi Automata

Example

GFp



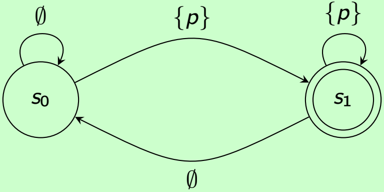
Example

G$(p \Rightarrow$ **F**$q)$

LTL and Büchi Automata

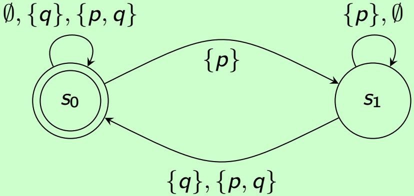
Example

GFp



Example

G$(p \Rightarrow$ **F**$q)$



LTL and Büchi Automata

Theorem

For every formula φ of LTL, there exists a Büchi automaton $\mathcal{A}_\varphi$ such that the language of $\mathcal{A}_\varphi$ is exactly the set of sequences of sets of atomic propositions verifying φ.

LTL and Büchi Automata

Theorem

For every formula φ of LTL, there exists a Büchi automaton $\mathcal{A}_\varphi$ such that the language of $\mathcal{A}_\varphi$ is exactly the set of sequences of sets of atomic propositions verifying φ.

- Let $\mathcal{S}$, be a BA, we have:

$$\mathcal{S} \models \varphi \Leftrightarrow \mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(A_\varphi)$$

LTL and Büchi Automata

Theorem

For every formula φ of LTL, there exists a Büchi automaton $\mathcal{A}_\varphi$ such that the language of $\mathcal{A}_\varphi$ is exactly the set of sequences of sets of atomic propositions verifying φ.

- Let $\mathcal{S}$, be a BA, we have:

$$\mathcal{S} \models \varphi \Leftrightarrow \mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(A_\varphi)$$

- That is:

$$\mathcal{S} \models \varphi \Leftrightarrow \mathcal{L}(\mathcal{S}) \cap \overline{\mathcal{L}(A_\varphi)} = \emptyset$$

LTL and Büchi Automata

Theorem

For every formula φ of LTL, there exists a Büchi automaton $\mathcal{A}_\varphi$ such that the language of $\mathcal{A}_\varphi$ is exactly the set of sequences of sets of atomic propositions verifying φ.

- Let $\mathcal{S}$, be a BA, we have:

$$\mathcal{S} \models \varphi \Leftrightarrow \mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(A_\varphi)$$

- That is:

$$\mathcal{S} \models \varphi \Leftrightarrow \mathcal{L}(\mathcal{S}) \cap \overline{\mathcal{L}(A_\varphi)} = \emptyset$$

- And finally:

$$\mathcal{S} \models \varphi \Leftrightarrow \mathcal{L}(\mathcal{S} \times A_{\neg\varphi}) = \emptyset$$

Büchi Automaton of an LTL formula

Definition (Fermeture)

The closure $Cl(\varphi)$ of a formula φ is the smallest set of LTL formulas that is closed under the following rules:

- $\varphi \in Cl(\varphi)$;
- if $\psi \in Cl(\varphi)$ then $\neg\psi \in Cl(\varphi)$ (recall that $\neg\neg\psi = \psi$);
- if $\psi_1 \wedge \psi_2 \in Cl(\varphi)$ then $\psi_1 \in Cl(\varphi)$ and $\psi_2 \in Cl(\varphi)$;
- if $\psi_1 \vee \psi_2 \in Cl(\varphi)$ then $\psi_1 \in Cl(\varphi)$ and $\psi_2 \in Cl(\varphi)$;
- if $\mathbf{X}\psi \in Cl(\varphi)$ then $\psi \in Cl(\varphi)$;
- if $\psi_1 \mathbf{U} \psi_2 \in Cl(\varphi)$ then $\psi_1 \in Cl(\varphi)$, $\psi_2 \in Cl(\varphi)$ and $\mathbf{X}(\psi_1 \mathbf{U} \psi_2) \in Cl(\varphi)$.

Example

The closure of $\varphi = (p \vee q)\mathbf{U}(\mathbf{X}\mathit{true})$ is:

$$\{\varphi, \neg\varphi, (p \vee q), \neg(p \vee q), \mathbf{X}\mathtt{true}, \neg(\mathbf{X}\mathtt{true}), p, \neg p, q, \neg q, \mathtt{true}, \mathtt{false}\}$$

Büchi Automaton of an LTL formula

Definition (Maximally Consistent Subset)

A subset C of $Cl(\varphi)$ is maximally consistent if:

1. C is consistent:
   - $\texttt{true} \in Cl(\varphi) \Rightarrow \texttt{true} \in C$;
   - $\forall \psi \in Cl(\varphi), \psi \in C$ iff $\neg\psi \notin C$;
   - $\forall \psi = \psi_1 \wedge \psi_2 \in Cl(\varphi), \psi \in C$ iff $\psi_1 \in C$ $\psi_2 \in C$;
   - $\forall \psi = \psi_1 \vee \psi_2 \in Cl(\varphi), \psi \in C$ iff $\psi_1 \in C$ or $\psi_2 \in C$;
   - $\forall \psi = \psi_1 \mathbf{U} \psi_2 \in Cl(\varphi)$:
     - if $\psi_2 \in C$ then $\psi \in C$;
     - if $\psi \in C$ and $\psi_2 \notin C$ then $\psi_1 \in C$.
2. C is maximal: $\forall \psi \in Cl(\varphi)$, either ψ or $\neg\psi$ belongs to C.

Example

The following sets are maximally consistent for $\varphi = (p \vee q)\mathbf{U}(\mathbf{X} true)$:

1. $\{\varphi, p \vee q, \neg(\mathbf{X}\texttt{true}), \neg p, q, \texttt{true}\}$,
2. $\{\varphi, \neg(p \vee q), \mathbf{X}\texttt{true}, \neg p, \neg q, \texttt{true}\}$,
3. $\{\neg\varphi, p \vee q, \neg(\mathbf{X}\texttt{true}), p, q, \texttt{true}\}$,
4. ...

Büchi Automaton of an LTL formula

We build a generalized Büchi automaton $\mathcal{A}_\varphi = (Q, \Sigma, \delta, Q_0, F)$ where:

- Q is the set of maximally consistent subsets of $Cl(\varphi)$;

Büchi Automaton of an LTL formula

We build a generalized Büchi automaton $\mathcal{A}_\varphi = (Q, \Sigma, \delta, Q_0, F)$ where:

- Q is the set of maximally consistent subsets of $Cl(\varphi)$;
- $\Sigma = 2^{\mathsf{AP}}$;

Büchi Automaton of an LTL formula

We build a generalized Büchi automaton $\mathcal{A}_\varphi = (Q, \Sigma, \delta, Q_0, F)$ where:

- Q is the set of maximally consistent subsets of $Cl(\varphi)$;
- $\Sigma = 2^{\mathsf{AP}}$;
- $Q_0 = \{s \in Q | \varphi \in s\}$;

Büchi Automaton of an LTL formula

We build a generalized Büchi automaton $\mathcal{A}_\varphi = (Q, \Sigma, \delta, Q_0, F)$ where:

- Q is the set of maximally consistent subsets of $Cl(\varphi)$;
- $\Sigma = 2^{\mathrm{AP}}$;
- $Q_0 = \{s \in Q | \varphi \in s\}$;
- $\forall s, t \in Q, \forall a \in \Sigma, t \in \delta(s, a)$ iff:
  - $\forall p \in \mathrm{AP}, p \in s$ iff $p \in a$;
  - $\forall \mathbf{X}\psi \in Cl(\varphi), \mathbf{X}\psi \in s$ iff $\psi \in t$.
  - $\forall \psi_1 \mathbf{U} \psi_2 \in Cl(\varphi), \psi_1 \mathbf{U} \psi_2 \in s$ iff $\psi_2 \in s$ or $(\psi_1 \in s$ and $\psi_1 \mathbf{U} \psi_2 \in t)$

Büchi Automaton of an LTL formula

We build a generalized Büchi automaton $\mathcal{A}_\varphi = (Q, \Sigma, \delta, Q_0, F)$ where:

- Q is the set of maximally consistent subsets of $Cl(\varphi)$;
- $\Sigma = 2^{\mathrm{AP}}$;
- $Q_0 = \{s \in Q | \varphi \in s\}$;
- $\forall s, t \in Q, \forall a \in \Sigma, t \in \delta(s, a)$ iff:
  - $\forall p \in \mathrm{AP}, p \in s$ iff $p \in a$;
  - $\forall \mathbf{X}\psi \in Cl(\varphi), \mathbf{X}\psi \in s$ iff $\psi \in t$.
  - $\forall \psi_1 \mathbf{U} \psi_2 \in Cl(\varphi), \psi_1 \mathbf{U} \psi_2 \in s$ iff $\psi_2 \in s$ or $(\psi_1 \in s$ and $\psi_1 \mathbf{U} \psi_2 \in t)$
- $F = \{F_{\psi_1}, \dots, F_{\psi_n}\}$ where $\forall \psi = \varphi_1 \mathbf{U} \varphi_2 \in Cl(\varphi), F_\psi = \{s \in Q | \neg\psi \in s$ or $\varphi_2 \in s\}$.
  An accepted path should go infinitely often through at least one state of each F_{ψ_i}

Büchi Automaton of an LTL formula

We build a generalized Büchi automaton $\mathcal{A}_\varphi = (Q, \Sigma, \delta, Q_0, F)$ where:

- Q is the set of maximally consistent subsets of $Cl(\varphi)$;
- $\Sigma = 2^{\mathrm{AP}}$;
- $Q_0 = \{s \in Q | \varphi \in s\}$;
- $\forall s, t \in Q, \forall a \in \Sigma, t \in \delta(s, a)$ iff:
  - $\forall p \in \mathrm{AP}, p \in s$ iff $p \in a$;
  - $\forall \mathbf{X}\psi \in Cl(\varphi), \mathbf{X}\psi \in s$ iff $\psi \in t$.
  - $\forall \psi_1 \mathbf{U} \psi_2 \in Cl(\varphi), \psi_1 \mathbf{U} \psi_2 \in s$ iff $\psi_2 \in s$ or $(\psi_1 \in s$ and $\psi_1 \mathbf{U} \psi_2 \in t)$
- $F = \{F_{\psi_1}, \ldots, F_{\psi_n}\}$ where $\forall \psi = \varphi_1 \mathbf{U} \varphi_2 \in Cl(\varphi), F_\psi = \{s \in Q | \neg\psi \in s$ or $\varphi_2 \in s\}$.
  An accepted path should go infinitely often through at least one state of each F_{ψ_i}

Theorem

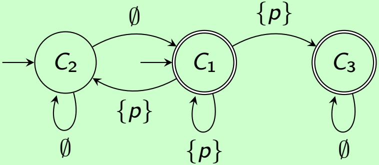
- *$\mathcal{A}_\varphi$ recognizes exactly the sequences satisfying φ ;*
- *$\mathcal{A}_\varphi$ has at most $2^{4|\varphi|}$ states.*

Exercises

Example

We build the automaton for $\varphi = \mathbf{Fp} = \texttt{true}\mathbf{U}p$, with $\mathsf{AP} = \{p\}$.

- $Cl(\varphi) = \{\varphi, \neg\varphi, p, \neg p, \texttt{true}, \texttt{false}\}$;
- The maximally consistent subsets are:
  - $C_1 = \{\varphi, p, \texttt{true}\}$;
  - $C_2 = \{\varphi, \neg p, \texttt{true}\}$;
  - $C_3 = \{\neg\varphi, \neg p, \texttt{true}\}$;
- The automaton:



Exercises

Exercise

Build a Büchi automaton for each of the following formula:

1. $\mathbf{Gp}$;
2. $\mathbf{pUXq}$;
3. $\mathbf{G}(\mathbf{p} \Rightarrow \mathbf{Xq})$.

Computation Tree Logic

- CTL (*Computation Tree Logic) [CES86] expresses properties on execution trees;*

Computation Tree Logic

- CTL (*Computation Tree Logic) [CES86] expresses properties on execution trees;*
- We still have a set AP of atomic propositions;

Computation Tree Logic

- CTL (*Computation Tree Logic) [CES86] expresses properties on execution trees;*
- We still have a set AP of atomic propositions;
- The syntax of CTL is given by:

Syntax of CTL

$$\varphi ::= \mathbf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{A}\psi \mid \mathbf{E}\psi$$
$$\psi ::= \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi$$

Computation Tree Logic

- CTL (*Computation Tree Logic) [CES86] expresses properties on execution trees;*
- We still have a set AP of atomic propositions;
- The syntax of CTL is given by:

Syntax of CTL

$$\varphi ::= \mathbf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{A}\psi \mid \mathbf{E}\psi$$
$$\psi ::= \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi$$

- Modalities Until **U** and neXt **X**.

Computation Tree Logic

- CTL (*Computation Tree Logic) [CES86] expresses properties on execution trees;*
- We still have a set AP of atomic propositions;
- The syntax of CTL is given by:

Syntax of CTL

$$\varphi ::= \mathbf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{A}\psi \mid \mathbf{E}\psi$$
$$\psi ::= \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi$$

- Modalities Until **U** and neXt **X**.
- Path quantification For *All* **A** and *Exists* **E**.

Semantics of CTL

Sémantics of CTL

Let $\mathcal{S} = (W, \rightarrow, \ell)$ be a Kripke structure, $t \in W$ and $\mathcal{S}$ (so $c_0 = t$):

- $(\mathcal{S}, t) \models \mathbf{p}$ iff $\mathbf{p} \in \ell(t)$;

Semantics of CTL

Sémantics of CTL

Let $\mathcal{S} = (W, \rightarrow, \ell)$ be a Kripke structure, $t \in W$ and $\mathcal{S}$ (so $c_0 = t$):

- $(\mathcal{S}, t) \models \mathbf{p}$ iff $\mathbf{p} \in \ell(t)$;
- $(\mathcal{S}, t) \models \neg\varphi$ iff $(\mathcal{S}, t) \not\models \varphi$;

Semantics of CTL

Sémantics of CTL

Let $\mathcal{S} = (W, \rightarrow, \ell)$ be a Kripke structure, $t \in W$ and $\mathcal{S}$ (so $c_0 = t$):

- $(\mathcal{S}, t) \models \mathbf{p}$ iff $\mathbf{p} \in \ell(t)$;
- $(\mathcal{S}, t) \models \neg\varphi$ iff $(\mathcal{S}, t) \not\models \varphi$;
- $(\mathcal{S}, t) \models \varphi_1 \vee \varphi_2$ iff $(\mathcal{S}, t) \models \varphi_1$ or $(\mathcal{S}, t) \models \varphi_2$;

Semantics of CTL

Sémantics of CTL

Let $\mathcal{S} = (W, \rightarrow, \ell)$ be a Kripke structure, $t \in W$ and $\mathcal{S}$ (so $c_0 = t$):

- $(\mathcal{S}, t) \models \mathbf{p}$ iff $\mathbf{p} \in \ell(t)$;
- $(\mathcal{S}, t) \models \neg\varphi$ iff $(\mathcal{S}, t) \not\models \varphi$;
- $(\mathcal{S}, t) \models \varphi_1 \vee \varphi_2$ iff $(\mathcal{S}, t) \models \varphi_1$ or $(\mathcal{S}, t) \models \varphi_2$;
- $(\mathcal{S}, t) \models \mathbf{A}\psi$ iff $\forall b \in Path(t), (\mathcal{S}, b, t) \models \psi$;

Semantics of CTL

Sémantics of CTL

Let $\mathcal{S} = (W, \rightarrow, \ell)$ be a Kripke structure, $t \in W$ and $\mathcal{S}$ (so $c_0 = t$):

- $(\mathcal{S}, t) \models \mathbf{p}$ iff $\mathbf{p} \in \ell(t)$;
- $(\mathcal{S}, t) \models \neg\varphi$ iff $(\mathcal{S}, t) \not\models \varphi$;
- $(\mathcal{S}, t) \models \varphi_1 \vee \varphi_2$ iff $(\mathcal{S}, t) \models \varphi_1$ or $(\mathcal{S}, t) \models \varphi_2$;
- $(\mathcal{S}, t) \models \mathbf{A}\psi$ iff $\forall b \in \mathit{Path}(t), (\mathcal{S}, b, t) \models \psi$;
- $(\mathcal{S}, t) \models \mathbf{E}\psi$ iff $\exists b \in \mathit{Path}(t), (\mathcal{S}, b, t) \models \psi$;

Semantics of CTL

Sémantics of CTL

Let $\mathcal{S} = (W, \rightarrow, \ell)$ be a Kripke structure, $t \in W$ and $\mathcal{S}$ (so $c_0 = t$):

- $(\mathcal{S}, t) \models \mathbf{p}$ iff $\mathbf{p} \in \ell(t)$;
- $(\mathcal{S}, t) \models \neg\varphi$ iff $(\mathcal{S}, t) \not\models \varphi$;
- $(\mathcal{S}, t) \models \varphi_1 \vee \varphi_2$ iff $(\mathcal{S}, t) \models \varphi_1$ or $(\mathcal{S}, t) \models \varphi_2$;
- $(\mathcal{S}, t) \models \mathbf{A}\psi$ iff $\forall b \in \mathit{Path}(t), (\mathcal{S}, b, t) \models \psi$;
- $(\mathcal{S}, t) \models \mathbf{E}\psi$ iff $\exists b \in \mathit{Path}(t), (\mathcal{S}, b, t) \models \psi$;
- $(\mathcal{S}, b, t) \models \mathbf{X}\varphi$ iff $(\mathcal{S}, c_1) \models \varphi$;

Semantics of CTL

Sémantics of CTL

Let $\mathcal{S} = (W, \rightarrow, \ell)$ be a Kripke structure, $t \in W$ and $\mathcal{S}$ (so $c_0 = t$):

- $(\mathcal{S}, t) \models \mathbf{p}$ iff $\mathbf{p} \in \ell(t)$;
- $(\mathcal{S}, t) \models \neg\varphi$ iff $(\mathcal{S}, t) \not\models \varphi$;
- $(\mathcal{S}, t) \models \varphi_1 \vee \varphi_2$ iff $(\mathcal{S}, t) \models \varphi_1$ or $(\mathcal{S}, t) \models \varphi_2$;
- $(\mathcal{S}, t) \models \mathbf{A}\psi$ iff $\forall b \in \mathit{Path}(t), (\mathcal{S}, b, t) \models \psi$;
- $(\mathcal{S}, t) \models \mathbf{E}\psi$ iff $\exists b \in \mathit{Path}(t), (\mathcal{S}, b, t) \models \psi$;
- $(\mathcal{S}, b, t) \models \mathbf{X}\varphi$ iff $(\mathcal{S}, c_1) \models \varphi$;
- $(\mathcal{S}, b, t) \models \varphi_1 \mathbf{U} \varphi_2$ iff $\exists j$ s.t. $(\mathcal{S}, c_j) \models \varphi_2$ and $\forall i < j, (\mathcal{S}, c_i) \models \varphi_1$.

Classic Abbreviations

- $\mathbf{AF}\varphi = \mathbf{A}\mathtt{true}\mathbf{U}\varphi$:

Classic Abbreviations

- $\mathbf{AF}\varphi = \mathbf{A}\mathtt{true}\mathbf{U}\varphi$: φ is inevitable;

Classic Abbreviations

- $\mathbf{AF}\varphi = \mathbf{A}\mathtt{true}\mathbf{U}\varphi$: φ is inevitable;
- $\mathbf{EF}\varphi = \mathbf{E}\mathtt{true}\mathbf{U}\varphi$:

Classic Abbreviations

- $\mathbf{AF}\varphi = \mathbf{A}\mathtt{true}\mathbf{U}\varphi$: φ is inevitable;
- $\mathbf{EF}\varphi = \mathbf{E}\mathtt{true}\mathbf{U}\varphi$: φ is possible;

Classic Abbreviations

- $\mathbf{AF}\varphi = \mathbf{A}\mathtt{true}\mathbf{U}\varphi$: φ is inevitable;
- $\mathbf{EF}\varphi = \mathbf{E}\mathtt{true}\mathbf{U}\varphi$: φ is possible;
- $\mathbf{EG}\varphi = \neg\mathbf{AF}\neg\varphi$:

Classic Abbreviations

- $\mathbf{AF}\varphi = \mathbf{A}\mathtt{true}\mathbf{U}\varphi$: φ is inevitable;
- $\mathbf{EF}\varphi = \mathbf{E}\mathtt{true}\mathbf{U}\varphi$: φ is possible;
- $\mathbf{EG}\varphi = \neg\mathbf{AF}\neg\varphi$: φ can be preserved;

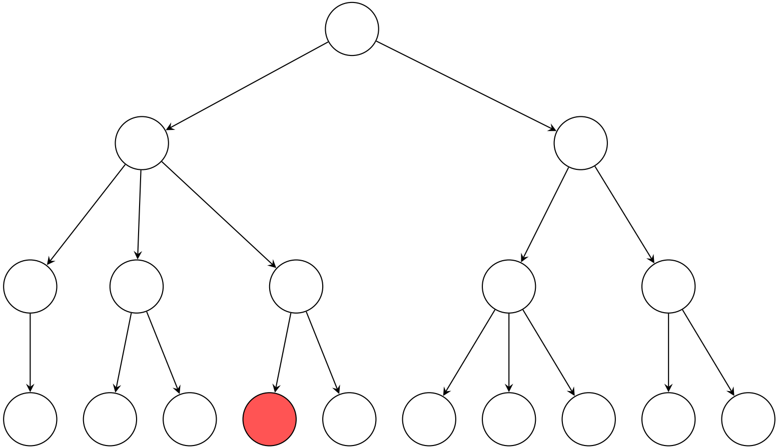
Classic Abbreviations

- $\mathbf{AF}\varphi = \mathbf{A}\mathtt{true}\mathbf{U}\varphi$: φ is inevitable;
- $\mathbf{EF}\varphi = \mathbf{E}\mathtt{true}\mathbf{U}\varphi$: φ is possible;
- $\mathbf{EG}\varphi = \neg\mathbf{AF}\neg\varphi$: φ can be preserved;
- $\mathbf{AG}\varphi = \neg\mathbf{EF}\neg\varphi$:

Classic Abbreviations

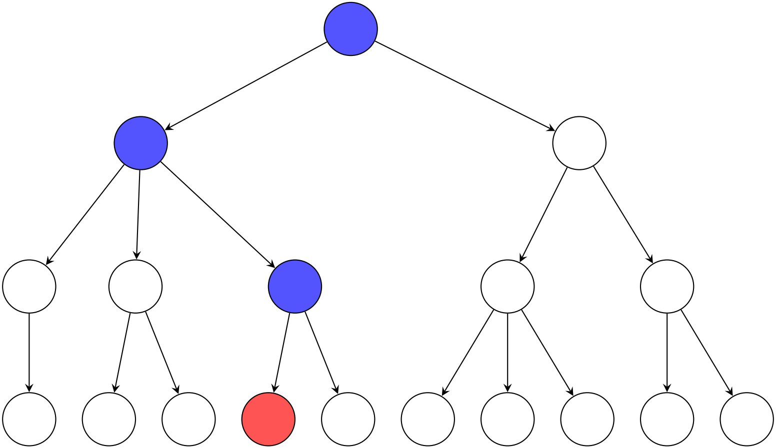
- $\mathbf{AF}\varphi = \mathbf{A}\mathtt{true}\mathbf{U}\varphi$: φ is inevitable;
- $\mathbf{EF}\varphi = \mathbf{E}\mathtt{true}\mathbf{U}\varphi$: φ is possible;
- $\mathbf{EG}\varphi = \neg\mathbf{AF}\neg\varphi$: φ can be preserved;
- $\mathbf{AG}\varphi = \neg\mathbf{EF}\neg\varphi$: φ always holds.

Classic Abbreviations



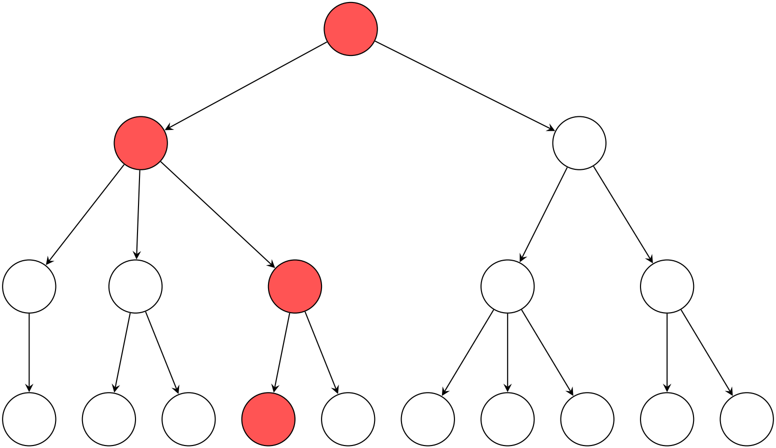
EFφ

Classic Abbreviations



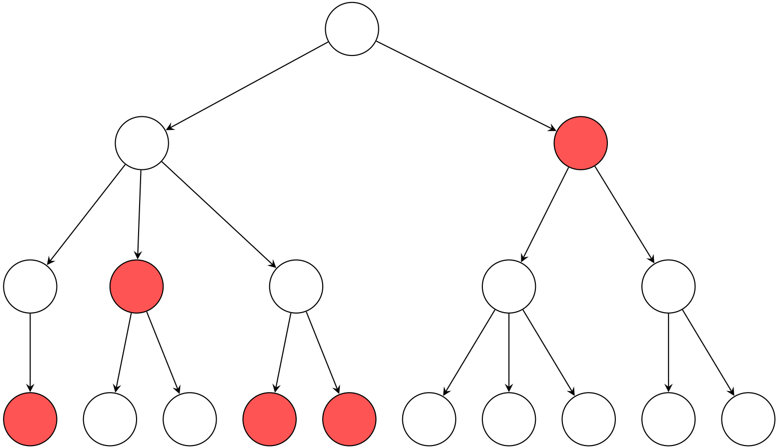
$\mathbf{E}\varphi\mathbf{U}\psi$

Classic Abbreviations



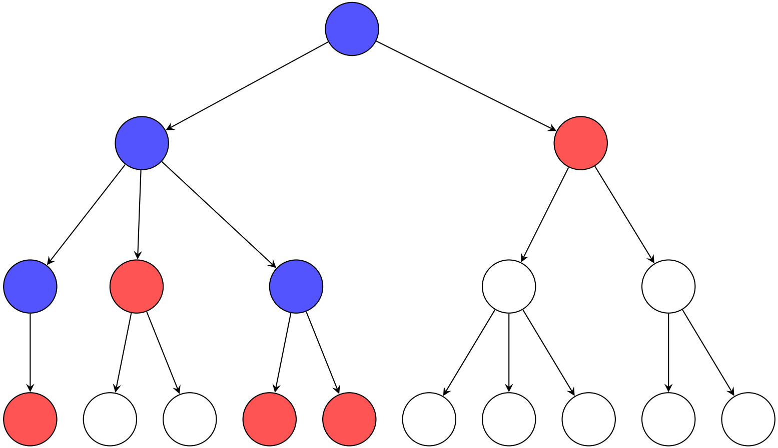
EGφ

Classic Abbreviations



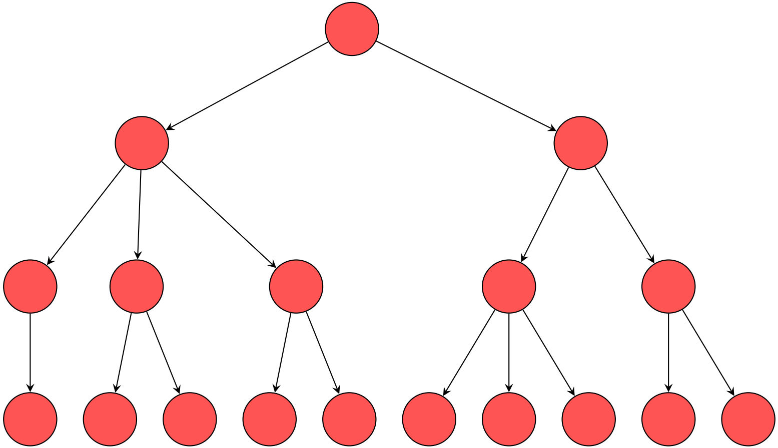
AFφ

Classic Abbreviations



$\mathbf{A}\varphi\mathbf{U}\psi$

Classic Abbreviations

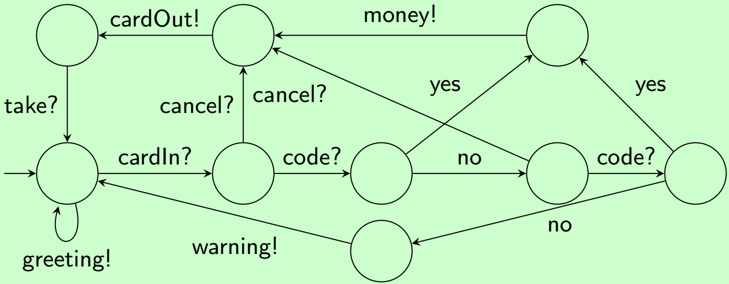


AGφ

CTL: Exercise

Exercise

Consider the following (labeled) Kripke Structure representing a simplified ATM:

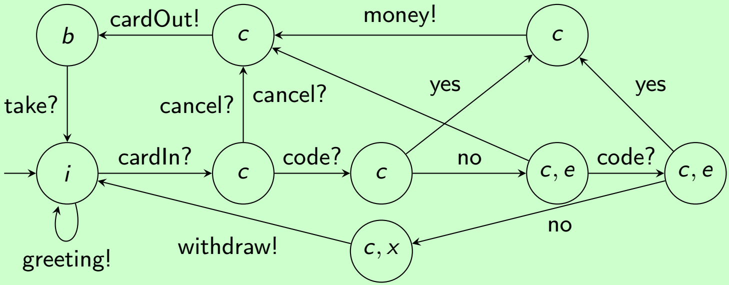


1. Label the states with the following properties:
   - *i* the ATM is available for a new transaction;
   - *c* the card has been inserted in the ATM (and not withdrawn);
   - *b* the card has been ejected and is ready to take;
   - *e* exactly one error with the code has occurred;
   - *x* two errors with the code have occurred.

CTL: Exercise

Exercise

Consider the following (labeled) Kripke Structure representing a simplified ATM:



2. Write formulas for the following properties and assess their truth value:
   - There is no deadlock;
   - A new session is always eventually started;
   - While less than two errors have been, the card can still be gotten back;
   - Whenever a session is started, if less than two errors are made, we can always take the card back;
   - When two errors have been made, no card can be taken back until a new one is inserted.

Evaluation of CTL Formulas

- There is an equivalence between Alternating Tree Automata and CTL [BVW94] ;

Evaluation of CTL Formulas

- There is an equivalence between Alternating Tree Automata and CTL [BVW94] ;
- We focus here on a direct evaluation of the truth values;

Evaluation of CTL Formulas

- There is an equivalence between Alternating Tree Automata and CTL [BVW94] ;
- We focus here on a direct evaluation of the truth values;
- It is based on symbolic set operations, using fix points.

Symbolic Set Evaluation: Propositional Logic

- States satisfying a given atomic proposition p: $[\![p]\!] = \{s | p \in \ell(s)\}$;
- States satisfying $\varphi_1 \vee \varphi_2$:
- States satisfying $\varphi_1 \wedge \varphi_2$:
- States satisfying $\neg\varphi$:

Symbolic Set Evaluation: Propositional Logic

- States satisfying a given atomic proposition p: $[\![p]\!] = \{s | p \in \ell(s)\}$;
- States satisfying $\varphi_1 \vee \varphi_2$: $[\![\varphi_1 \vee \varphi_2]\!] = [\![\varphi_1]\!] \cup [\![\varphi_2]\!]$;
- States satisfying $\varphi_1 \wedge \varphi_2$:
- States satisfying $\neg\varphi$:

Symbolic Set Evaluation: Propositional Logic

- States satisfying a given atomic proposition p: $[\![p]\!] = \{s | p \in \ell(s)\}$;
- States satisfying $\varphi_1 \vee \varphi_2$: $[\![\varphi_1 \vee \varphi_2]\!] = [\![\varphi_1]\!] \cup [\![\varphi_2]\!]$;
- States satisfying $\varphi_1 \wedge \varphi_2$: $[\![\varphi_1 \wedge \varphi_2]\!] = [\![\varphi_1]\!] \cap [\![\varphi_2]\!]$;
- States satisfying $\neg\varphi$:

Symbolic Set Evaluation: Propositional Logic

- States satisfying a given atomic proposition p: $[\![p]\!] = \{s | p \in \ell(s)\}$;
- States satisfying $\varphi_1 \vee \varphi_2$: $[\![\varphi_1 \vee \varphi_2]\!] = [\![\varphi_1]\!] \cup [\![\varphi_2]\!]$;
- States satisfying $\varphi_1 \wedge \varphi_2$: $[\![\varphi_1 \wedge \varphi_2]\!] = [\![\varphi_1]\!] \cap [\![\varphi_2]\!]$;
- States satisfying $\neg\varphi$: $[\![\neg\varphi]\!] = W \setminus [\![\varphi]\!]$.

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\text{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\text{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;
  - $\widetilde{\text{Pred}}(X) = \{s \in W | \forall s' \in W, s \longrightarrow s' \Rightarrow s' \in X'\}$;

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\mathrm{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;
  - $\widetilde{\mathrm{Pred}}(X) = \{s \in W | \forall s' \in W, s \longrightarrow s' \Rightarrow s' \in X'\}$;
  - Duality: $\widetilde{\mathrm{Pred}}(X) =$;

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\text{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;
  - $\widetilde{\text{Pred}}(X) = \{s \in W | \forall s' \in W, s \longrightarrow s' \Rightarrow s' \in X'\}$;
  - Duality: $\widetilde{\text{Pred}}(X) = W \setminus \text{Pred}(W \setminus X)$;

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\text{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;
  - $\widetilde{\text{Pred}}(X) = \{s \in W | \forall s' \in W, s \longrightarrow s' \Rightarrow s' \in X'\}$;
  - Duality: $\widetilde{\text{Pred}}(X) = W \setminus \text{Pred}(W \setminus X)$;
- Next:

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\text{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;
  - $\widetilde{\text{Pred}}(X) = \{s \in W | \forall s' \in W, s \longrightarrow s' \Rightarrow s' \in X'\}$;
  - Duality: $\widetilde{\text{Pred}}(X) = W \setminus \text{Pred}(W \setminus X)$;
- Next:
  - $[\![\mathbf{EX}\varphi]\!] =$;

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\mathrm{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;
  - $\widetilde{\mathrm{Pred}}(X) = \{s \in W | \forall s' \in W, s \longrightarrow s' \Rightarrow s' \in X'\}$;
  - Duality: $\widetilde{\mathrm{Pred}}(X) = W \setminus \mathrm{Pred}(W \setminus X)$;
- Next:
  - $[\![\mathbf{EX}\varphi]\!] = \mathrm{Pred}([\![\varphi]\!])$;

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\text{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;
  - $\widetilde{\text{Pred}}(X) = \{s \in W | \forall s' \in W, s \longrightarrow s' \Rightarrow s' \in X'\}$;
  - Duality: $\widetilde{\text{Pred}}(X) = W \setminus \text{Pred}(W \setminus X)$;
- Next:
  - $[\![\mathbf{EX}\varphi]\!] = \text{Pred}([\![\varphi]\!])$;
  - $[\![\mathbf{AX}\varphi]\!] =$;

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\mathrm{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;
  - $\widetilde{\mathrm{Pred}}(X) = \{s \in W | \forall s' \in W, s \longrightarrow s' \Rightarrow s' \in X'\}$;
  - Duality: $\widetilde{\mathrm{Pred}}(X) = W \setminus \mathrm{Pred}(W \setminus X)$;
- Next:
  - $[\![\mathbf{EX}\varphi]\!] = \mathrm{Pred}([\![\varphi]\!])$;
  - $[\![\mathbf{AX}\varphi]\!] = \widetilde{\mathrm{Pred}}([\![\varphi]\!])$;

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\mathrm{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;
  - $\widetilde{\mathrm{Pred}}(X) = \{s \in W | \forall s' \in W, s \longrightarrow s' \Rightarrow s' \in X'\}$;
  - Duality: $\widetilde{\mathrm{Pred}}(X) = W \setminus \mathrm{Pred}(W \setminus X)$;
- Next:
  - $[\![\mathbf{EX}\varphi]\!] = \mathrm{Pred}([\![\varphi]\!])$;
  - $[\![\mathbf{AX}\varphi]\!] = \widetilde{\mathrm{Pred}}([\![\varphi]\!])$;
- Until:

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\mathrm{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;
  - $\widetilde{\mathrm{Pred}}(X) = \{s \in W | \forall s' \in W, s \longrightarrow s' \Rightarrow s' \in X'\}$;
  - Duality: $\widetilde{\mathrm{Pred}}(X) = W \setminus \mathrm{Pred}(W \setminus X)$;
- Next:
  - $[\![\mathbf{EX}\varphi]\!] = \mathrm{Pred}([\![\varphi]\!])$;
  - $[\![\mathbf{AX}\varphi]\!] = \widetilde{\mathrm{Pred}}([\![\varphi]\!])$;
- Until:
  - $[\![\mathbf{E}\varphi_1\mathbf{U}\varphi_2]\!] =$.

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\mathrm{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;
  - $\widetilde{\mathrm{Pred}}(X) = \{s \in W | \forall s' \in W, s \longrightarrow s' \Rightarrow s' \in X'\}$;
  - Duality: $\widetilde{\mathrm{Pred}}(X) = W \setminus \mathrm{Pred}(W \setminus X)$;
- Next:
  - $[\![\mathbf{EX}\varphi]\!] = \mathrm{Pred}([\![\varphi]\!])$;
  - $[\![\mathbf{AX}\varphi]\!] = \widetilde{\mathrm{Pred}}([\![\varphi]\!])$;
- Until:
  - $[\![\mathbf{E}\varphi_1\mathbf{U}\varphi_2]\!] = \mu Z.([\![\varphi_2]\!] \cup ([\![\varphi_1]\!] \cap \mathrm{Pred}(Z)))$.

Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\mathrm{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;
  - $\widetilde{\mathrm{Pred}}(X) = \{s \in W | \forall s' \in W, s \longrightarrow s' \Rightarrow s' \in X'\}$;
  - Duality: $\widetilde{\mathrm{Pred}}(X) = W \setminus \mathrm{Pred}(W \setminus X)$;
- Next:
  - $[\![\mathbf{EX}\varphi]\!] = \mathrm{Pred}([\![\varphi]\!])$;
  - $[\![\mathbf{AX}\varphi]\!] = \widetilde{\mathrm{Pred}}([\![\varphi]\!])$;
- Until:
  - $[\![\mathbf{E}\varphi_1\mathbf{U}\varphi_2]\!] = \mu Z.([\![\varphi_2]\!] \cup ([\![\varphi_1]\!] \cap \mathrm{Pred}(Z)))$.
  - $[\![\mathbf{A}\varphi_1\mathbf{U}\varphi_2]\!] =$;

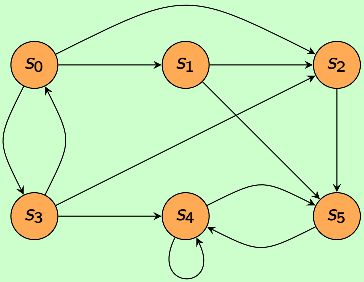
Symbolic Set Evaluation: Temporal Operators

- Useful set operations:
  - Let $X \subseteq W$
  - $\mathrm{Pred}(X) = \{s \in W | \exists s' \in X, s \longrightarrow s'\}$;
  - $\widetilde{\mathrm{Pred}}(X) = \{s \in W | \forall s' \in W, s \longrightarrow s' \Rightarrow s' \in X'\}$;
  - Duality: $\widetilde{\mathrm{Pred}}(X) = W \setminus \mathrm{Pred}(W \setminus X)$;
- Next:
  - $[\![\mathbf{EX}\varphi]\!] = \mathrm{Pred}([\![\varphi]\!])$;
  - $[\![\mathbf{AX}\varphi]\!] = \widetilde{\mathrm{Pred}}([\![\varphi]\!])$;
- Until:
  - $[\![\mathbf{E}\varphi_1\mathbf{U}\varphi_2]\!] = \mu Z.([\![\varphi_2]\!] \cup ([\![\varphi_1]\!] \cap \mathrm{Pred}(Z)))$.
  - $[\![\mathbf{A}\varphi_1\mathbf{U}\varphi_2]\!] = \mu Z.([\![\varphi_2]\!] \cup ([\![\varphi_1]\!] \cap \mathrm{Pred}(Z) \cap \widetilde{\mathrm{Pred}}(Z)))$;

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

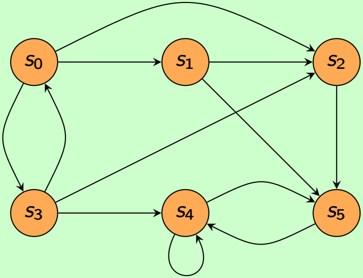


$Z_0 = \emptyset$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

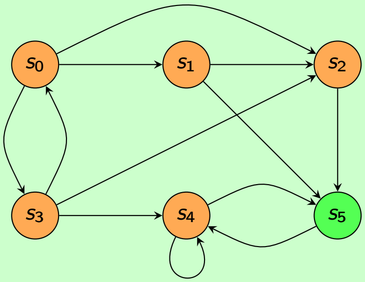


$Z_0 = \emptyset$
$Z_1 = [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0))$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

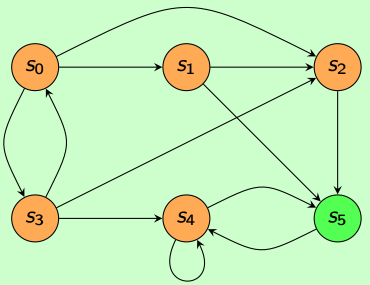


$$Z_0 = \emptyset$$
$$Z_1 = [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0))$$
$$Z_1 = \{s_5\}$$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

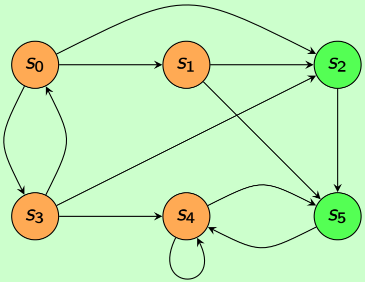


$Z_0 = \emptyset$
$Z_1 = [\![s_5]\!] \cup ([\![\texttt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0))$
$Z_1 = \{s_5\}$
$Z_2 = [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1)$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

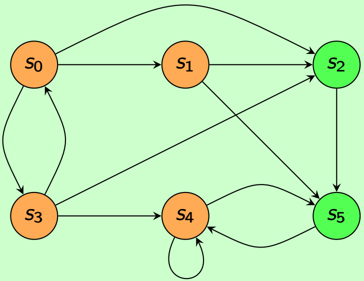


$$
\begin{aligned}
Z_0 &= \emptyset \\
Z_1 &= [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0)) \\
Z_1 &= \{s_5\} \\
Z_2 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1) \\
Z_2 &= \{s_5\} \cup \{s_2\} = \{s_2, s_5\}
\end{aligned}
$$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

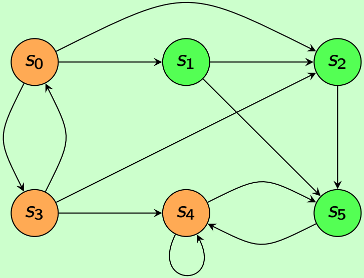


$Z_0 = \emptyset$
$Z_1 = [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0))$
$Z_1 = \{s_5\}$
$Z_2 = [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1)$
$Z_2 = \{s_5\} \cup \underbrace{\{s_2\}} = \{s_2, s_5\}$
$Z_3 = [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_2)$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

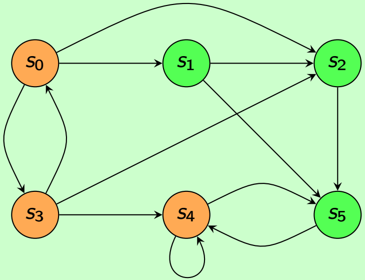


$$
\begin{aligned}
Z_0 &= \emptyset \\
Z_1 &= [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0)) \\
Z_1 &= \{s_5\} \\
Z_2 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1) \\
Z_2 &= \{s_5\} \cup \{s_2\} = \{s_2, s_5\} \\
Z_3 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_2) \\
Z_3 &= \{s_5\} \cup \{s_1, s_2\} = \{s_1, s_2, s_5\}
\end{aligned}
$$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF} s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

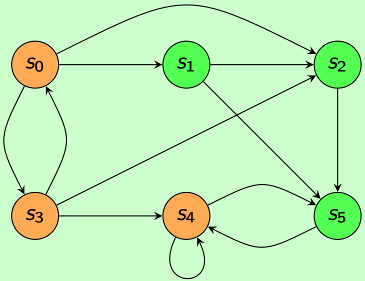


$$
\begin{aligned}
Z_0 &= \emptyset \\
Z_1 &= [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0)) \\
Z_1 &= \{s_5\} \\
Z_2 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1) \\
Z_2 &= \{s_5\} \cup \{s_2\} = \{s_2, s_5\} \\
Z_3 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_2) \\
Z_3 &= \{s_5\} \cup \{s_1, s_2\} = \{s_1, s_2, s_5\} \\
Z_4 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_3)
\end{aligned}
$$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

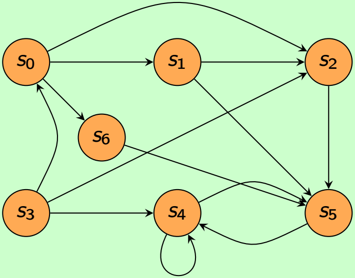


$$
\begin{aligned}
Z_0 &= \emptyset \\
Z_1 &= [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0)) \\
Z_1 &= \{s_5\} \\
Z_2 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1) \\
Z_2 &= \{s_5\} \cup \{s_2\} = \{s_2, s_5\} \\
Z_3 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_2) \\
Z_3 &= \{s_5\} \cup \{s_1, s_2\} = \{s_1, s_2, s_5\} \\
Z_4 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_3) \\
Z_4 &= \{s_5\} \cup \{s_1, s_2\} = Z_3
\end{aligned}
$$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

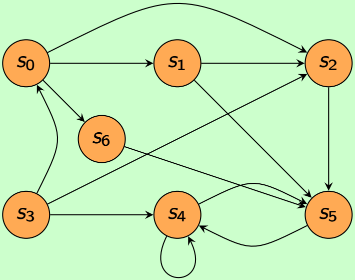
Example



Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

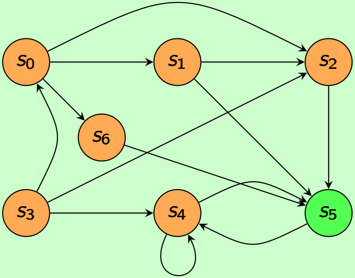


$Z_0 = \emptyset$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

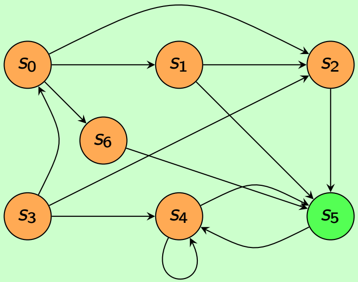


$$Z_0 = \emptyset$$
$$Z_1 = [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0)) = \{s_5\}$$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF} s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

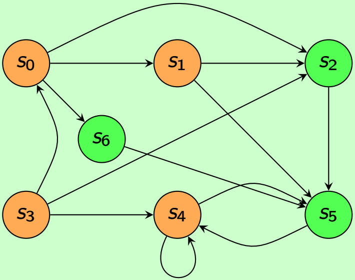


$$Z_0 = \emptyset$$
$$Z_1 = [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0)) = \{s_5\}$$
$$Z_2 = [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1)$$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

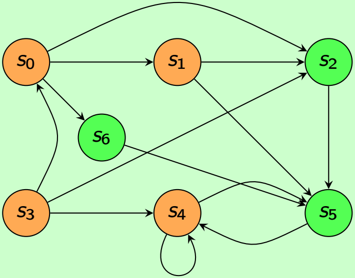


$$Z_0 = \emptyset$$
$$Z_1 = [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0)) = \{s_5\}$$
$$Z_2 = [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1)$$
$$Z_2 = \{s_5\} \cup \{s_2, s_6\} = \{s_2, s_5, s_6\}$$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

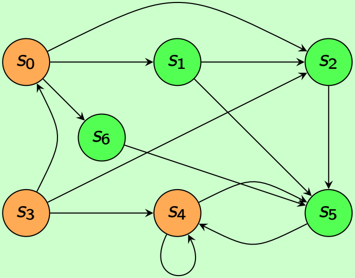


$$
\begin{aligned}
Z_0 &= \emptyset \\
Z_1 &= [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0)) = \{s_5\} \\
Z_2 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1) \\
Z_2 &= \{s_5\} \cup \{s_2, s_6\} = \{s_2, s_5, s_6\} \\
Z_3 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_2)
\end{aligned}
$$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

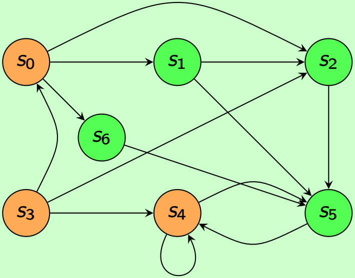


$$
\begin{aligned}
Z_0 &= \emptyset \\
Z_1 &= [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0)) = \{s_5\} \\
Z_2 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1) \\
Z_2 &= \{s_5\} \cup \{s_2, s_6\} = \{s_2, s_5, s_6\} \\
Z_3 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_2) \\
Z_3 &= \{s_5\} \cup \{s_1, s_2, s_6\} = \{s_1, s_2, s_5, s_6\}
\end{aligned}
$$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

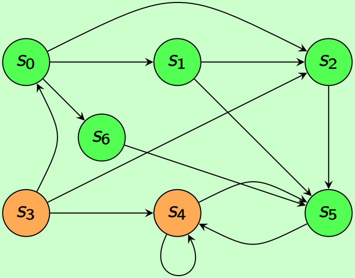


$$
\begin{aligned}
Z_0 &= \emptyset \\
Z_1 &= [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0)) = \{s_5\} \\
Z_2 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1) \\
Z_2 &= \{s_5\} \cup \{s_2, s_6\} = \{s_2, s_5, s_6\} \\
Z_3 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_2) \\
Z_3 &= \{s_5\} \cup \{s_1, s_2, s_6\} = \{s_1, s_2, s_5, s_6\} \\
Z_4 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_3)
\end{aligned}
$$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

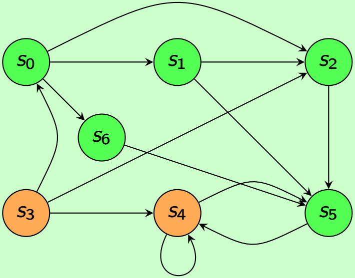


$$
\begin{aligned}
Z_0 &= \emptyset \\
Z_1 &= [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0)) = \{s_5\} \\
Z_2 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1) \\
Z_2 &= \{s_5\} \cup \{s_2, s_6\} = \{s_2, s_5, s_6\} \\
Z_3 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_2) \\
Z_3 &= \{s_5\} \cup \{s_1, s_2, s_6\} = \{s_1, s_2, s_5, s_6\} \\
Z_4 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_3) \\
Z_4 &= \{s_5\} \cup \{s_0, s_1, s_2, s_6\} = \{s_0, s_1, s_2, s_5, s_6\}
\end{aligned}
$$

Symbolic Set Evaluation: Example

Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example

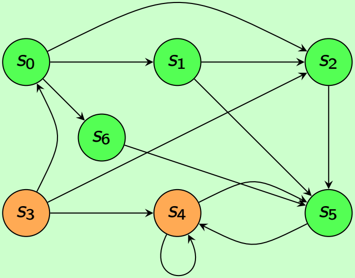


$$
\begin{aligned}
Z_0 &= \emptyset \\
Z_1 &= [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0)) = \{s_5\} \\
Z_2 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1) \\
Z_2 &= \{s_5\} \cup \{s_2, s_6\} = \{s_2, s_5, s_6\} \\
Z_3 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_2) \\
Z_3 &= \{s_5\} \cup \{s_1, s_2, s_6\} = \{s_1, s_2, s_5, s_6\} \\
Z_4 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_3) \\
Z_4 &= \{s_5\} \cup \{s_0, s_1, s_2, s_6\} = \{s_0, s_1, s_2, s_5, s_6\} \\
Z_5 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_4)
\end{aligned}
$$

Symbolic Set Evaluation: Example

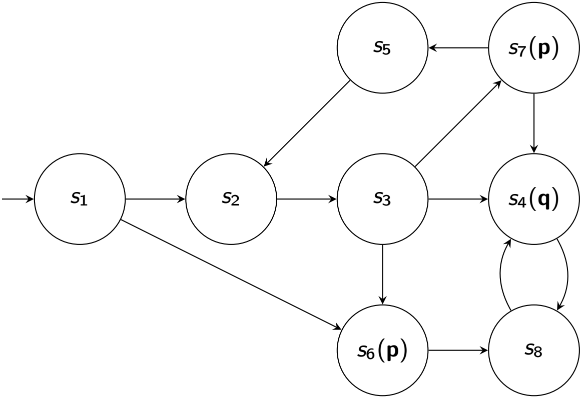
Consider $\varphi = \mathbf{AF}s_5$. Does $(\mathcal{S}, s_0) \models \varphi$?

Example



$$
\begin{aligned}
Z_0 &= \emptyset \\
Z_1 &= [\![s_5]\!] \cup ([\![\mathtt{true}]\!] \cap \mathrm{Pred}(Z_0) \cap \widetilde{\mathrm{Pred}}(Z_0)) = \{s_5\} \\
Z_2 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_1) \\
Z_2 &= \{s_5\} \cup \{s_2, s_6\} = \{s_2, s_5, s_6\} \\
Z_3 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_2) \\
Z_3 &= \{s_5\} \cup \{s_1, s_2, s_6\} = \{s_1, s_2, s_5, s_6\} \\
Z_4 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_3) \\
Z_4 &= \{s_5\} \cup \{s_0, s_1, s_2, s_6\} = \{s_0, s_1, s_2, s_5, s_6\} \\
Z_5 &= [\![s_5]\!] \cup \widetilde{\mathrm{Pred}}(Z_4) \\
Z_5 &= \{s_5\} \cup \{s_0, s_1, s_2, s_6\} = Z_4
\end{aligned}
$$

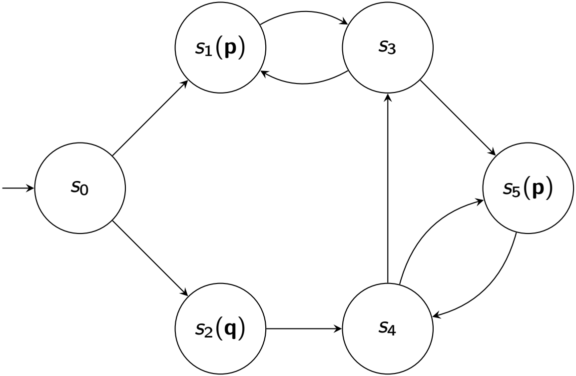
Symbolic Set Evaluation: Exercise



1. Does the Kripke Structure satisfy the CTL property **AF q**? Give details of the computation.
2. Does the Kripke Structure satisfy the CTL property **EF (p** $\wedge$ **AF q)**? Give details of the computation.

Symbolic Set Evaluation: Exercise

Check the CTL property $\phi = \mathbf{AF\ AG}(\mathbf{p} \vee \neg\mathbf{q})$ on the following Kripke structure:



LTL and CTL

Exercise

1. Exhibit two Kripke structures (KS) such that:
   - there exists a CTL formula that distinguishes them (true for one KS and false for the other one);
   - there is no LTL formula that distinguishes them.
2. Exhibit a CTL formula with no equivalent in LTL;
   Two formulas are equivalent if they are true exactly for the same KS
3. Exhibit an LTL formula with no equivalent in CTL.

Plan I

Control

- Does the system conform to the specification? $\mathcal{S} \models \varphi$?

Control

- Does the system conform to the specification? $\mathcal{S} \models \varphi$?
- A more general problem:

Control Problem

Given a model for the environment and the possible actions of the controller $\mathcal{S}$ and a specification φ,

$$\exists C, (C || \mathcal{S}) \models \varphi?$$

Control

- Does the system conform to the specification? $\mathcal{S} \models \varphi$?
- A more general problem:

Control Problem

Given a model for the environment and the possible actions of the controller $\mathcal{S}$ and a specification φ,

$$\exists C, (C||\mathcal{S}) \models \varphi?$$

- In the model-checking problem we assume the model of the system to be closed;

Control

- Does the system conform to the specification? $\mathcal{S} \models \varphi$?
- A more general problem:

Control Problem

Given a model for the environment and the possible actions of the controller $\mathcal{S}$ and a specification φ,

$$\exists C, (C || \mathcal{S}) \models \varphi?$$

- In the model-checking problem we assume the model of the system to be closed;
- In the control problem we assume an open system that we want to close by synthesizing a controller;

Control

- Does the system conform to the specification? $\mathcal{S} \models \varphi$?
- A more general problem:

Control Problem

Given a model for the environment and the possible actions of the controller $\mathcal{S}$ and a specification φ,

$$\exists C, (C || \mathcal{S}) \models \varphi?$$

- In the model-checking problem we assume the model of the system to be closed;
- In the control problem we assume an open system that we want to close by synthesizing a controller;
- We model the control problem in the framework of the theory of games on graphs.

Control Game

- We define a two-player game;

Control Game

- We define a two-player game;
- The plant is given by a game automaton:

Definition (Game Automaton)

A game automaton is a finite automaton (Q, A, δ, Q_0) such that $A = A_u \cup A_c$ with $A_u \cap A_c = \emptyset$.

Control Game

- We define a two-player game;
- The plant is given by a game automaton:

Definition (Game Automaton)

A game automaton is a finite automaton (Q, A, δ, Q_0) such that $A = A_u \cup A_c$ with $A_u \cap A_c = \emptyset$.

- The two players are:
  - the environment playing actions in A_u;
  - the controller playing actions in A_c;

Control Game

- We define a two-player game;
- The plant is given by a game automaton:

Definition (Game Automaton)

A game automaton is a finite automaton (Q, A, δ, Q_0) such that $A = A_u \cup A_c$ with $A_u \cap A_c = \emptyset$.

- The two players are:
  - the environment playing actions in A_u;
  - the controller playing actions in A_c;
- We define a control objective for the controller:
  The objective for the environment will be to prevent the controller to win

Control Game

- We define a two-player game;
- The plant is given by a game automaton:

Definition (Game Automaton)

A game automaton is a finite automaton (Q, A, δ, Q_0) such that $A = A_u \cup A_c$ with $A_u \cap A_c = \emptyset$.

- The two players are:
  - the environment playing actions in A_u;
  - the controller playing actions in A_c;
- We define a control objective for the controller:
  The objective for the environment will be to prevent the controller to win
  - reachability: enforce the reachability of some given states;

Control Game

- We define a two-player game;
- The plant is given by a game automaton:

Definition (Game Automaton)

A game automaton is a finite automaton (Q, A, δ, Q_0) such that $A = A_u \cup A_c$ with $A_u \cap A_c = \emptyset$.

- The two players are:
  - the environment playing actions in A_u;
  - the controller playing actions in A_c;
- We define a control objective for the controller:
  The objective for the environment will be to prevent the controller to win
  - reachability: enforce the reachability of some given states;
  - safety: enforce staying in some given states. donnés ;

Control Game

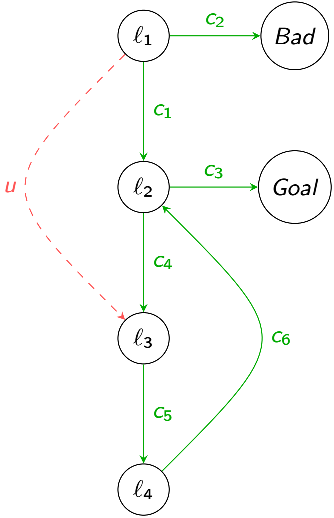
- We define a two-player game;
- The plant is given by a game automaton:

Definition (Game Automaton)

A game automaton is a finite automaton (Q, A, δ, Q_0) such that $A = A_u \cup A_c$ with $A_u \cap A_c = \emptyset$.

- The two players are:
  - the environment playing actions in A_u;
  - the controller playing actions in A_c;
- We define a control objective for the controller:
  The objective for the environment will be to prevent the controller to win
  - reachability: enforce the reachability of some given states;
  - safety: enforce staying in some given states. donnés ;
  - others, e.g. ω-regular objectives on Büchi game automata, etc.

Example



Strategies

Definition (Strategy)

Let $\mathcal{A} = (Q, A, \delta, Q_0)$ be a game automaton. A strategy f on $\mathcal{A}$ is a partial function of $[\![\mathcal{A}]\!]$ in A_c. If $f : Q \to A_c$ then f is said to be memoryless/positional.

Strategies

Definition (Strategy)

Let $\mathcal{A} = (Q, A, \delta, Q_0)$ be a game automaton. A strategy f on $\mathcal{A}$ is a partial function of $[\![\mathcal{A}]\!]$ in A_c. If $f : Q \to A_c$ then f is said to be memoryless/positional.

Definition (Outcome of a strategy)

Let $f : \mathcal{A} \to A_c$ be a strategy on $\mathcal{A}$. The outcome $Outcome(\rho, f)$ of applying f from state q is the subset of runs of $\mathcal{A}$ inductively defined by;

Strategies

Definition (Strategy)

Let $\mathcal{A} = (Q, A, \delta, Q_0)$ be a game automaton. A strategy f on $\mathcal{A}$ is a partial function of $[\![\mathcal{A}]\!]$ in A_c. If $f : Q \to A_c$ then f is said to be memoryless/positional.

Definition (Outcome of a strategy)

Let $f : \mathcal{A} \to A_c$ be a strategy on $\mathcal{A}$. The outcome $Outcome(\rho, f)$ of applying f from state q is the subset of runs of $\mathcal{A}$ inductively defined by;

- $q \in Outcome(q, f)$;

Strategies

Definition (Strategy)

Let $\mathcal{A} = (Q, A, \delta, Q_0)$ be a game automaton. A strategy f on $\mathcal{A}$ is a partial function of $[\![\mathcal{A}]\!]$ in A_c. If $f : Q \to A_c$ then f is said to be memoryless/positional.

Definition (Outcome of a strategy)

Let $f : \mathcal{A} \to A_c$ be a strategy on $\mathcal{A}$. The outcome $Outcome(\rho, f)$ of applying f from state q is the subset of runs of $\mathcal{A}$ inductively defined by;

- $q \in Outcome(q, f)$;
- if $\rho \in Outcome(q, f)$, then $\rho' = \rho \xrightarrow{a} q' \in Outcome(q, f)$ if $\rho' \in [\![\mathcal{A}]\!]$ and either $a \in A_u$, or $a \in A_c$ and $a = f(\rho)$.

Strategies

Definition (Strategy)

Let $\mathcal{A} = (Q, A, \delta, Q_0)$ be a game automaton. A strategy f on $\mathcal{A}$ is a partial function of $[\![\mathcal{A}]\!]$ in A_c. If $f : Q \to A_c$ then f is said to be memoryless/positional.

Definition (Outcome of a strategy)

Let $f : \mathcal{A} \to A_c$ be a strategy on $\mathcal{A}$. The outcome $Outcome(\rho, f)$ of applying f from state q is the subset of runs of $\mathcal{A}$ inductively defined by;

- $q \in Outcome(q, f)$;
- if $\rho \in Outcome(q, f)$, then $\rho' = \rho \xrightarrow{a} q' \in Outcome(q, f)$ if $\rho' \in [\![\mathcal{A}]\!]$ and either $a \in A_u$, or $a \in A_c$ and $a = f(\rho)$.
- An infinite run belongs to $Outcome(q, f)$ if all its finite prefixes do.

Objectives, Winning

Definition

A run is maximal within some set of runs R if it is infinite or cannot be extended within R by a controllable action. We simply say that it is maximal if $R = [\![\mathcal{A}]\!]$.

Definition

An objective for game $\mathcal{A}$ is a set of maximal runs of $\mathcal{A}$.

- A strategy f is Winning for objective Obj from state q, if for all runs ρ that is maximal in $Outcome(q, f)$, we have $\rho \in Obj$;

Objectives, Winning

Definition

A run is maximal within some set of runs R if it is infinite or cannot be extended within R by a controllable action. We simply say that it is maximal if $R = [\![\mathcal{A}]\!]$.

Definition

An objective for game $\mathcal{A}$ is a set of maximal runs of $\mathcal{A}$.

- A strategy f is Winning for objective Obj from state q, if for all runs ρ that is maximal in $Outcome(q, f)$, we have $\rho \in Obj$;
- A state s is winning if there exists a winning strategy from s;

Objectives, Winning

Definition

A run is maximal within some set of runs R if it is infinite or cannot be extended within R by a controllable action. We simply say that it is maximal if $R = [\![\mathcal{A}]\!]$.

Definition

An objective for game $\mathcal{A}$ is a set of maximal runs of $\mathcal{A}$.

- A strategy f is Winning for objective Obj from state q, if for all runs ρ that is maximal in $Outcome(q, f)$, we have $\rho \in Obj$;
- A state s is winning if there exists a winning strategy from s;
- The game is winning if all its initial states are winning.

Reachability Games

- Let *Goal* be a set of target states;

Reachability Games

- Let *Goal* be a set of target states;
- For a run $\rho = s_0 \xrightarrow{a_0} \cdots \xrightarrow{a_{n-1}} s_n$, we define $States(\rho) = \{s_0, s_1, \ldots, s_n\}$;

Reachability Games

- Let *Goal* be a set of target states;
- For a run $\rho = s_0 \xrightarrow{a_0} \cdots \xrightarrow{a_{n-1}} s_n$, we define $States(\rho) = \{s_0, s_1, \ldots, s_n\}$;
- The reachability objective for *Goal* contains exactly the maximal runs ρ s.t.: $States(\rho) \cap Goal \neq \emptyset$ and there is always a possible controllable transition in all states before *Goal*.

Winning States and Controllable Predecessors

- In order to compute the winning states, we adapt the co-reachable states computation;

Winning States and Controllable Predecessors

- In order to compute the winning states, we adapt the co-reachable states computation;
- In order to replace Pred, what are the states from which we are sure that we have a strategy to go to some set of states X?

Winning States and Controllable Predecessors

- In order to compute the winning states, we adapt the co-reachable states computation;
- In order to replace Pred, what are the states from which we are sure that we have a strategy to go to some set of states X?
- We define the controllable predecessors operator $\pi(X)$:

$$\forall a \in A, \mathrm{Pred}_a(X) = \{q \in Q | q \xrightarrow{a} q', q' \in X\}$$

$$\pi(X) =$$

Winning States and Controllable Predecessors

- In order to compute the winning states, we adapt the co-reachable states computation;
- In order to replace Pred, what are the states from which we are sure that we have a strategy to go to some set of states X?
- We define the controllable predecessors operator $\pi(X)$:

$$\forall a \in A, \mathrm{Pred}_a(X) = \{q \in Q | q \xrightarrow{a} q', q' \in X\}$$

$$\pi(X) = \mathrm{Pred}_c(X) \cap \widetilde{\mathrm{Pred}}_u(X)$$

Winning States and Controllable Predecessors

- In order to compute the winning states, we adapt the co-reachable states computation;
- In order to replace Pred, what are the states from which we are sure that we have a strategy to go to some set of states X?
- We define the controllable predecessors operator $\pi(X)$:

$$\forall a \in A, \mathrm{Pred}_a(X) = \{q \in Q | q \xrightarrow{a} q', q' \in X\}$$

$$\pi(X) = \mathrm{Pred}_c(X) \cap \widetilde{\mathrm{Pred}}_u(X)$$

Exercise

Give an alternative expression of $\pi(X)$ that does not use $\widetilde{\mathrm{Pred}}$.

Winning States and Controllable Predecessors

- In order to compute the winning states, we adapt the co-reachable states computation;
- In order to replace Pred, what are the states from which we are sure that we have a strategy to go to some set of states X?
- We define the controllable predecessors operator $\pi(X)$:

$$\forall a \in A, \mathrm{Pred}_a(X) = \{q \in Q | q \xrightarrow{a} q', q' \in X\}$$

$$\pi(X) = \mathrm{Pred}_c(X) \cap \widetilde{\mathrm{Pred}}_u(X)$$

Exercise

Give an alternative expression of $\pi(X)$ that does not use $\widetilde{\mathrm{Pred}}$.

- The set of winning states is given by:

$$Win =$$

Winning States and Controllable Predecessors

- In order to compute the winning states, we adapt the co-reachable states computation;
- In order to replace Pred, what are the states from which we are sure that we have a strategy to go to some set of states X?
- We define the controllable predecessors operator $\pi(X)$:

$$\forall a \in A, \mathrm{Pred}_a(X) = \{q \in Q | q \xrightarrow{a} q', q' \in X\}$$

$$\pi(X) = \mathrm{Pred}_c(X) \cap \widetilde{\mathrm{Pred}}_u(X)$$

Exercise

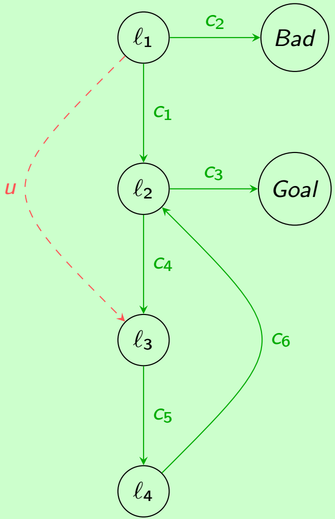
Give an alternative expression of $\pi(X)$ that does not use $\widetilde{\mathrm{Pred}}$.

- The set of winning states is given by:

$$Win = \mu X.(Goal \cup \pi(X))$$

Example

Example



Controllable Predecessors:

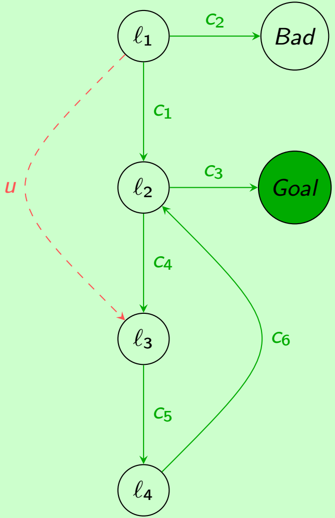
$$\pi(X) = \big(\mathrm{Pred}_c(X) \setminus \mathrm{Pred}_u(\overline{X})\big)$$

Itérer π:

1. $Win_0 = \emptyset$

Example

Example



Controllable Predecessors:

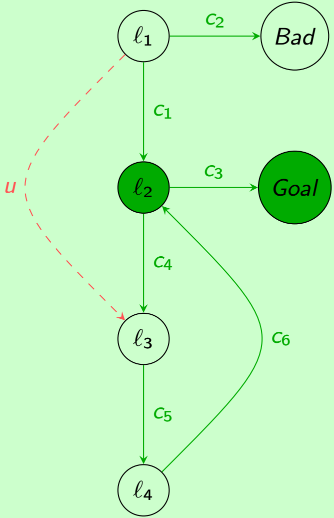
$$\pi(X) = \big(\mathrm{Pred}_c(X) \setminus \mathrm{Pred}_u(\overline{X})\big)$$

Itérer π:

1. $Win_0 = \emptyset$
2. $Win_1 = \{Goal\}$

Example

Example



Controllable Predecessors:

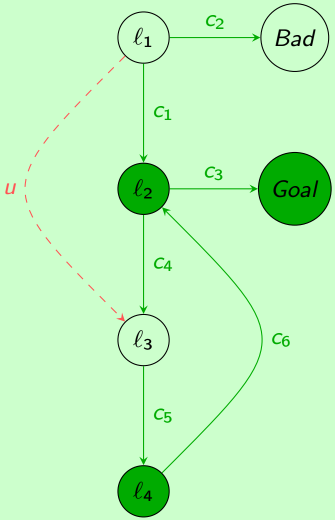
$$\pi(X) = \big(\mathrm{Pred}_c(X) \setminus \mathrm{Pred}_u(\overline{X})\big)$$

Itérer π:

1. $Win_0 = \emptyset$
2. $Win_1 = \{Goal\}$
3. $Win_2 = \{Goal, \ell_2\}$

Example

Example



Controllable Predecessors:

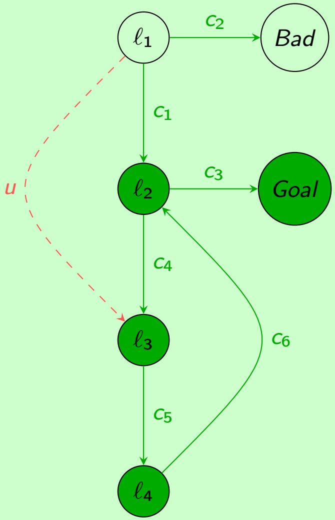
$$\pi(X) = \left(\mathrm{Pred}_c(X) \setminus \mathrm{Pred}_u(\overline{X})\right)$$

Itérer π:

1. $Win_0 = \emptyset$
2. $Win_1 = \{Goal\}$
3. $Win_2 = \{Goal, \ell_2\}$
4. $Win_3 = \{Goal, \ell_2, \ell_4\}$

Example

Example



Controllable Predecessors:

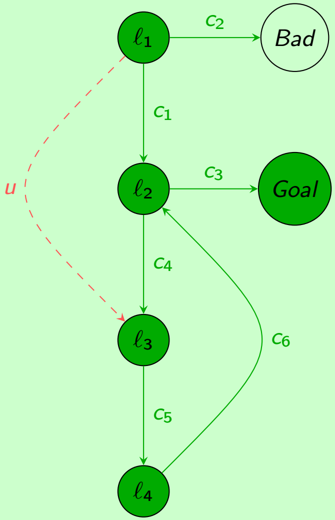
$$\pi(X) = \left(\mathrm{Pred}_c(X) \setminus \mathrm{Pred}_u(\overline{X})\right)$$

Itérer π:

1. $Win_0 = \emptyset$
2. $Win_1 = \{Goal\}$
3. $Win_2 = \{Goal, \ell_2\}$
4. $Win_3 = \{Goal, \ell_2, \ell_4\}$
5. $Win_4 = \{Goal, \ell_2, \ell_4, \ell_3\}$

Example

Example



Controllable Predecessors:

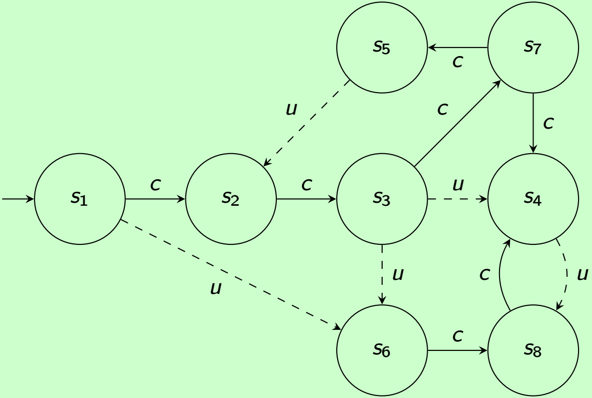
$$\pi(X) = \left(\mathrm{Pred}_c(X) \setminus \mathrm{Pred}_u(\overline{X})\right)$$

Itérer π:

1. $Win_0 = \emptyset$
2. $Win_1 = \{Goal\}$
3. $Win_2 = \{Goal, \ell_2\}$
4. $Win_3 = \{Goal, \ell_2, \ell_4\}$
5. $Win_4 = \{Goal, \ell_2, \ell_4, \ell_3\}$
6. $Win_5 = \{Goal, \ell_2, \ell_4, \ell_3, \ell_1\}$

Exercise

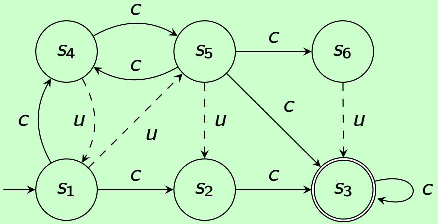
Exercise



Assuming the transitions labeled by c are controllable and those labeled by u are uncontrollable, can the controller enforce the reachability of s_4? Give the details of the computation.

Exercise

Exercise



Assuming the transitions labeled by c are controllable and those labeled by u are uncontrollable, can the controller enforce the reachability of s_3? Give the details of the computation.

Exercise

Exercise

1. Prove that checking the CTL property EF can be done by solving a reachability game;
2. Prove the same for AF.

Exercise

Given a set of safe states G, the safety game consists in forcing the system to stay in G.

1. define the corresponding objective in terms of set of runs;
2. show that the winning states are not the complement of those for which the environment has a strategy to enforce the reachability of $\overline{G}$;
3. propose an algorithm to compute the winning states (and a winning strategy) in a safety game;
4. how can we define a notion of "most permissive strategy"?

Plan I

Timed Models

- To make models more realistic, we can account for execution times of actions, delays between them, etc.

Timed Models

- To make models more realistic, we can account for execution times of actions, delays between them, etc.
- We might also want to enforce timing constraints in the specification;

Timed Models

- To make models more realistic, we can account for execution times of actions, delays between them, etc.
- We might also want to enforce timing constraints in the specification;
- We then need to add time to discrete models:

Timed Models

- To make models more realistic, we can account for execution times of actions, delays between them, etc.
- We might also want to enforce timing constraints in the specification;
- We then need to add time to discrete models:
  - a discrete time (previous case);

Timed Models

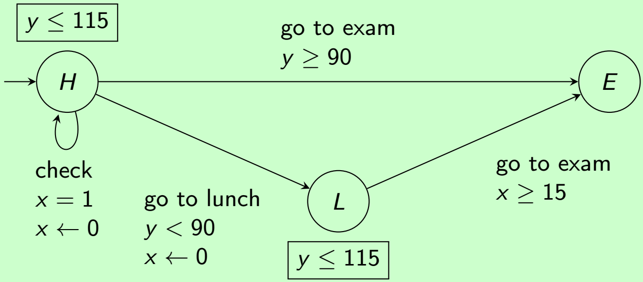
- To make models more realistic, we can account for execution times of actions, delays between them, etc.
- We might also want to enforce timing constraints in the specification;
- We then need to add time to discrete models:
  - a discrete time (previous case);
  - dense time.

Plan I

Timed Automata: Example

Example

It is noon, you are at home (H), and you want to go out for a lunch (L) with a friend before the FMOV exam (E) at 2pm. You have to wait for your friend to be ready and however, so you check every minute.



Do all paths lead to E in less than 120 minutes?

Timed Automata

Definition (Timed Automata [AD94, HNSY94])

A timed automaton is a tuple (L, l_0, X, A, E, Inv) where:

- L is a finite set of locations;
- l_0 is the initial location;
- X is a finite set of clocks with non-negative real values;
- A is a finite set of actions;
- $E \subset L \times \mathcal{C}(X) \times A \times 2^X \times L$ is a finite set of edges. Let $e = (l, \delta, \alpha, R, l') \in E$. Edge e links location l to location l', with guard δ, action α, set of clocks to reset to zero R.
- $Inv \in \mathcal{C}(X)^L$ assigns an *invariant* to each location.

$\mathcal{C}(X)$ is the set of conjunctions of simple constraints on X: $x \sim c$ with $x \in X, c \in \mathbb{Q}$ and $\{\sim \in <, \leq, \geq, >\}$

Semantics of Timed Automata

- The possible actions in a TA are defined by the current location and the value of all clocks;
- We call this a concrete state of a TA;
- The behavior of a TA is defined as a timed transition system, called its behavioral semantics.

Timed Transition Systems

Definition (Timed Transition System)

A Timed Transition System (TTS) is a tuple $(S, A, s_0, \rightarrow)$ where:

- S is a set of states;
- A is a set of actions;
- $s_0 \in S$ is the initial state;
- $\rightarrow$ is the transition relation, decomposed into:
  - continuous/time transitions: $\xrightarrow{d \in \mathbb{R}^+}$,
  - discrete transitions: $\xrightarrow{a \in A}$.

For $d \in \mathbb{R}^+$, $\xrightarrow{d}$ is the action consisting in letting d time units elapse.

Semantics of Timed Automata

Definition (Semantics of a Timed Automaton)

The (concrete/behavioral) semantics of a timed automaton $\mathcal{A}$ is the TTS $\mathcal{S}_{\mathcal{A}} = (Q, A \cup \{d\}_{d \in \mathbb{R}^+}, Q_0, \rightarrow)$ where:

- $Q = \{(\ell, \nu) \in L \times (\mathbb{R}^+)^X | \nu \models Inv(\ell)\}$,
- $Q_0 = (\ell_0, \mathbf{0})$,
- $\rightarrow \in Q \times (A \cup \{d\}_{d \in \mathbb{R}^+}) \times Q$ is defined for $a \in A$ and $d \in \mathbb{R}^+$ by:
  - $(l, \nu) \xrightarrow{a} (\ell', \nu')$ iff $\exists(\ell, \delta, a, R, \ell') \in E$ s.t.:

$$\begin{cases} \delta(\nu) = \mathtt{true}, \\ \nu' = \nu[R \leftarrow 0], \\ Inv(\ell')(\nu') = \mathtt{true} \end{cases}$$

Semantics of Timed Automata

Definition (Semantics of a Timed Automaton)

The (concrete/behavioral) semantics of a timed automaton $\mathcal{A}$ is the TTS $\mathcal{S}_\mathcal{A} = (Q, A \cup \{d\}_{d \in \mathbb{R}^+}, Q_0, \rightarrow)$ where:

- $Q = \{(\ell, \nu) \in L \times (\mathbb{R}^+)^X | \nu \models Inv(\ell)\}$,
- $Q_0 = (\ell_0, \mathbf{0})$,
- $\rightarrow \in Q \times (A \cup \{d\}_{d \in \mathbb{R}^+}) \times Q$ is defined for $a \in A$ and $d \in \mathbb{R}^+$ by:
  - $(l, \nu) \xrightarrow{a} (\ell', \nu')$ iff $\exists(\ell, \delta, a, R, \ell') \in E$ s.t.:

$$\begin{cases} \delta(\nu) = \mathtt{true}, \\ \nu' = \nu[R \leftarrow 0], \\ Inv(\ell')(\nu') = \mathtt{true} \end{cases}$$

  - $(l, \nu) \xrightarrow{d} (l, \nu')$ iff:

$$\begin{cases} \nu' = \nu + d, \\ \forall d' \in [0, d], Inv(l)(\nu + d') = \mathtt{true} \end{cases}$$

Runs in Timed Transition Systems

Definition (Run)

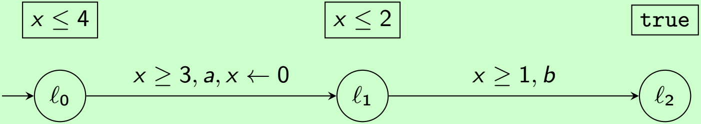
A run σ from s in a TTS $(S, A, s_0, \rightarrow)$ is a sequence $(s_i)_{i \geq 1}$ s.t.:
$s_1 \xrightarrow{d_1} s_2 \xrightarrow{a_1} s_3 \xrightarrow{d_2} s_4 \xrightarrow{a_2} \cdots$ with $\forall i, a_i \in A, d_i \in \mathbb{R}^+$ and $s_1 = s$. We note
$\sigma = s_1 \xrightarrow{(d_1, a_1)} s_3 \xrightarrow{(d_2, a_2)} \cdots$.

Definition (Duration, divergence)

Let $\sigma = s_1 \xrightarrow{(d_1, a_1)} s_3 \xrightarrow{(d_2, a_2)} \cdots$ be a run in a TTS. La duration of σ is $d(\sigma) = \sum_i d_i$. If $d(\sigma) = \infty$, we say that σ is divergent.

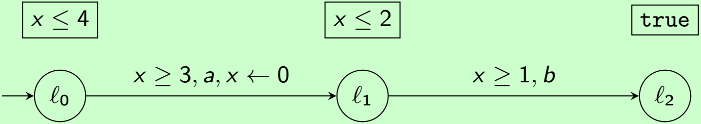
Timed Automata: Another Example

Example



Timed Automata: Another Example

Example



$$\rho = (\ell_0, 0) \xrightarrow{(3.2, a)} (\ell_1, 0) \xrightarrow{(1.6, b)} (\ell_2, 1.6)$$

Timed Automata: Exercise

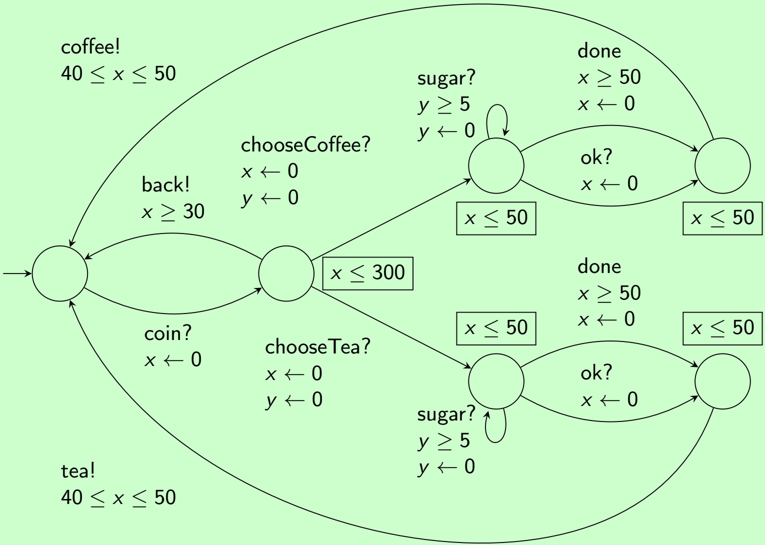
Exercise

Consider a beverage vending machine that serves coffee and tea. First you insert one coin, then you get to choose either coffee or tea. If no choice is made within 30 seconds, the session is aborted and the coin is given back. When a beverage has been selected the number of sugar doses must be chosen by repeatedly pressing a button. To avoid unwanted presses of the button, consecutive presses are only taken into account if they are separated by at least 0.5s. After 5s, or whenever the OK button is pressed, the machine delivers the beverage, which takes between 4s and 5s, and gets back to the initial state.

Model this system using a timed automaton.

Timed Automata: Exercise

Exercise



Plan I

Languages of Timed Automata

- Given a set of accepting (or repeated) locations F, we can define the language of a timed automaton as before:

Languages of Timed Automata

- Given a set of accepting (or repeated) locations F, we can define the language of a timed automaton as before:

Definition (Trace of Timed Run)

The trace of a run $\rho = (\ell_0, \nu_0) \xrightarrow{(a_0, d_0)} \ell_1, \nu_1) \xrightarrow{(a_1, d_1)} \cdots$ is the (timed) word $(a_0, d_0)(a_1, d_0 + d_1)(a_2, d_0 + d_1 + d_2) \cdots$

Languages of Timed Automata

- Given a set of accepting (or repeated) locations F, we can define the language of a timed automaton as before:

Definition (Trace of Timed Run)

The trace of a run $\rho = (\ell_0, \nu_0) \xrightarrow{(a_0, d_0)} (\ell_1, \nu_1) \xrightarrow{(a_1, d_1)} \cdots$ is the (timed) word $(a_0, d_0)(a_1, d_0 + d_1)(a_2, d_0 + d_1 + d_2) \cdots$

- It is now a timed language where each letter/action has an (absolute) date.

Languages of Timed Automata

- Given a set of accepting (or repeated) locations F, we can define the language of a timed automaton as before:

Definition (Trace of Timed Run)

The trace of a run $\rho = (\ell_0, \nu_0) \xrightarrow{(a_0, d_0)} \ell_1, \nu_1) \xrightarrow{(a_1, d_1)} \cdots$ is the (timed) word $(a_0, d_0)(a_1, d_0 + d_1)(a_2, d_0 + d_1 + d_2) \cdots$

- It is now a timed language where each letter/action has an (absolute) date.
- For finite automata, most interesting properties are decidable;

Languages of Timed Automata

- Given a set of accepting (or repeated) locations F, we can define the language of a timed automaton as before:

Definition (Trace of Timed Run)

The trace of a run $\rho = (\ell_0, \nu_0) \xrightarrow{(a_0,d_0)} \ell_1, \nu_1) \xrightarrow{(a_1,d_1)} \cdots$ is the (timed) word $(a_0, d_0)(a_1, d_0 + d_1)(a_2, d_0 + d_1 + d_2)\cdots$

- It is now a timed language where each letter/action has an (absolute) date.
- For finite automata, most interesting properties are decidable;
- Not for timed automata:

Languages of Timed Automata

- Given a set of accepting (or repeated) locations F, we can define the language of a timed automaton as before:

Definition (Trace of Timed Run)

The trace of a run $\rho = (\ell_0, \nu_0) \xrightarrow{(a_0, d_0)} \ell_1, \nu_1) \xrightarrow{(a_1, d_1)} \cdots$ is the (timed) word $(a_0, d_0)(a_1, d_0 + d_1)(a_2, d_0 + d_1 + d_2) \cdots$

- It is now a timed language where each letter/action has an (absolute) date.
- For finite automata, most interesting properties are decidable;
- Not for timed automata:

Theorem (Universality)

The universality problem (does the automata accept the language of all timed words) is undecidable for TA.

Languages of Timed Automata

- Given a set of accepting (or repeated) locations F, we can define the language of a timed automaton as before:

Definition (Trace of Timed Run)

The trace of a run $\rho = (\ell_0, \nu_0) \xrightarrow{(a_0, d_0)} (\ell_1, \nu_1) \xrightarrow{(a_1, d_1)} \cdots$ is the (timed) word $(a_0, d_0)(a_1, d_0 + d_1)(a_2, d_0 + d_1 + d_2) \cdots$

- It is now a timed language where each letter/action has an (absolute) date.
- For finite automata, most interesting properties are decidable;
- Not for timed automata:

Theorem (Universality)

The universality problem (does the automata accept the language of all timed words) is undecidable for TA.

- Neither is inclusion of languages: how to prove this?

Languages of Timed Automata

- Given a set of accepting (or repeated) locations F, we can define the language of a timed automaton as before:

Definition (Trace of Timed Run)

The trace of a run $\rho = (\ell_0, \nu_0) \xrightarrow{(a_0,d_0)} \ell_1, \nu_1) \xrightarrow{(a_1,d_1)} \cdots$ is the (timed) word $(a_0, d_0)(a_1, d_0 + d_1)(a_2, d_0 + d_1 + d_2) \cdots$

- It is now a timed language where each letter/action has an (absolute) date.
- For finite automata, most interesting properties are decidable;
- Not for timed automata:

Theorem (Universality)

The universality problem (does the automata accept the language of all timed words) is undecidable for TA.

- Neither is inclusion of languages: how to prove this?
  Just check:

$$\mathcal{A}_{\text{universel}} \subseteq \mathcal{L}$$

Languages of Timed Automata

- The emptiness problem is decidable however:

Languages of Timed Automata

- The emptiness problem is decidable however:

Theorem (Emptiness)

The emptiness problem for TA (is the timed language of a TA empty?) is decidable and PSPACE-complete.

Languages of Timed Automata

- The emptiness problem is decidable however:

Theorem (Emptiness)

The emptiness problem for TA (is the timed language of a TA empty?) is decidable and PSPACE-complete.

- And so is reachability:

Languages of Timed Automata

- The emptiness problem is decidable however:

Theorem (Emptiness)

The emptiness problem for TA (is the timed language of a TA empty?) is decidable and PSPACE-complete.

- And so is reachability:

Corollary (Reachability)

Reachability of a concrete state is PSPACE-complete in TA.

Languages of Timed Automata

- The emptiness problem is decidable however:

Theorem (Emptiness)

The emptiness problem for TA (is the timed language of a TA empty?) is decidable and PSPACE-complete.

- And so is reachability:

Corollary (Reachability)

Reachability of a concrete state is PSPACE-complete in TA.

- This is proved (and done in practice) by building finite abstractions of the state-space: region and zone graphs.

Other Properties

Theorem (Union and intersection)

The intersection of two TA, and their union, is a TA and can be computed.

Other Properties

Theorem (Union and intersection)

The intersection of two TA, and their union, is a TA and can be computed.

Theorem (Complement)

- *There might be no TA that accepts the complement of the timed language of another TA.*
- *Knowing whether there exists a TA that accepts the complement of the timed language of a TA is undecidable.*

Other Properties

Theorem (Union and intersection)

The intersection of two TA, and their union, is a TA and can be computed.

Theorem (Complement)

- *There might be no TA that accepts the complement of the timed language of another TA.*
- *Knowing whether there exists a TA that accepts the complement of the timed language of a TA is undecidable.*

Theorem (Determinisation)

- *There might be no deterministic TA that accepts the timed language of a given TA.*
- *Knowing whether there exists a deterministic TA that accepts the same timed language as a TA is undecidable.*

temporisés.

Abstractions

- A (concrete) state of a TA is a pair (l, ν);

Abstractions

- A (concrete) state of a TA is a pair (l, ν);
- There is in general an infinite number of states;

Abstractions

- A (concrete) state of a TA is a pair (l, ν);
- There is in general an infinite number of states;
- They cannot be enumerated exhaustively;

Abstractions

- A (concrete) state of a TA is a pair (l, ν);
- There is in general an infinite number of states;
- They cannot be enumerated exhaustively;
- We can group states together wrt. some good equivalence relation;

Abstractions

- A (concrete) state of a TA is a pair (l, ν);
- There is in general an infinite number of states;
- They cannot be enumerated exhaustively;
- We can group states together wrt. some good equivalence relation;
- We study here:

Abstractions

- A (concrete) state of a TA is a pair (l, ν);
- There is in general an infinite number of states;
- They cannot be enumerated exhaustively;
- We can group states together wrt. some good equivalence relation;
- We study here:
  - regions;

Abstractions

- A (concrete) state of a TA is a pair (l, ν);
- There is in general an infinite number of states;
- They cannot be enumerated exhaustively;
- We can group states together wrt. some good equivalence relation;
- We study here:
  - regions;
  - zones.

Regions

- We need an equivalence relation $\equiv$ that respects the timing constraints:

$$\nu \equiv \nu' \Rightarrow \begin{cases} \nu \models g \Leftrightarrow \nu' \models g \\ \forall t \in \mathbb{R}^+, \exists t' \in \mathbb{R}^+ \text{ s.t. } \nu + t \equiv \nu' + t' \end{cases}$$

Regions

- We need an equivalence relation $\equiv$ that respects the timing constraints:

$$\nu \equiv \nu' \Rightarrow \begin{cases} \nu \models g \Leftrightarrow \nu' \models g \\ \forall t \in \mathbb{R}^+, \exists t' \in \mathbb{R}^+ \text{ s.t. } \nu + t \equiv \nu' + t' \end{cases}$$

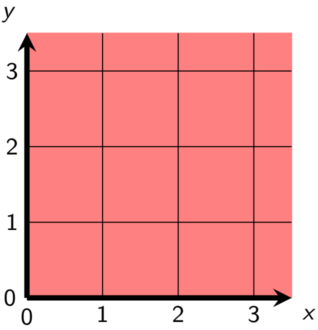
- Assume coefficient in guards and invariants are integers:

Regions

- We need an equivalence relation $\equiv$ that respects the timing constraints:

$$\nu \equiv \nu' \Rightarrow \begin{cases} \nu \models g \Leftrightarrow \nu' \models g \\ \forall t \in \mathbb{R}^+, \exists t' \in \mathbb{R}^+ \text{ s.t. } \nu + t \equiv \nu' + t' \end{cases}$$

- Assume coefficient in guards and invariants are integers:

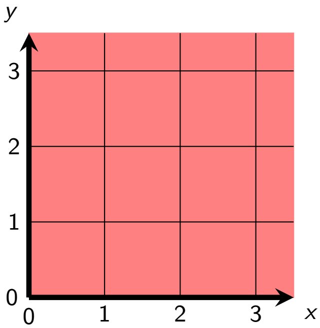


Regions

- We need an equivalence relation $\equiv$ that respects the timing constraints:

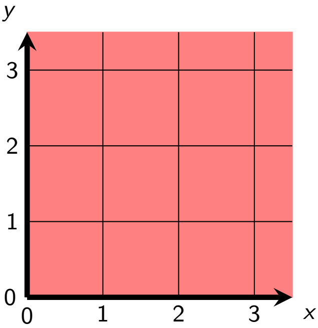
$$\nu \equiv \nu' \Rightarrow \begin{cases} \nu \models g \Leftrightarrow \nu' \models g \\ \forall t \in \mathbb{R}^+, \exists t' \in \mathbb{R}^+ \text{ s.t. } \nu + t \equiv \nu' + t' \end{cases}$$

- Assume coefficient in guards and invariants are integers:

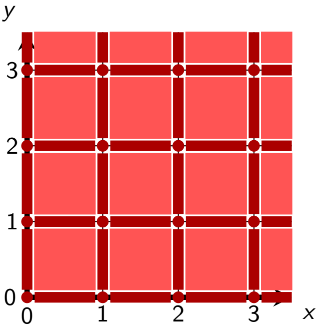


What about strict constraints?

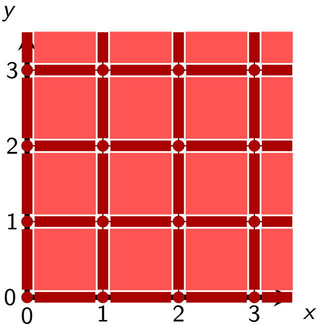
Regions



Regions

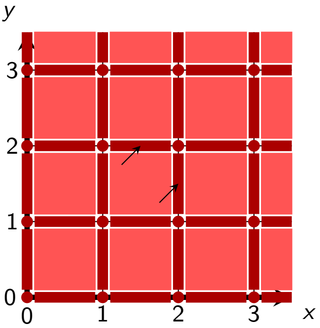


Regions



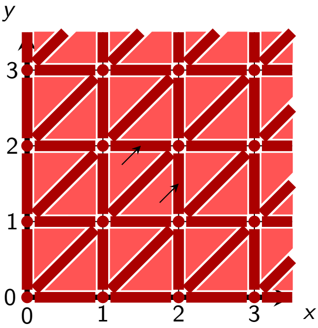
With this definition, the successor by time elapsing of a “region” is not unique.

Regions



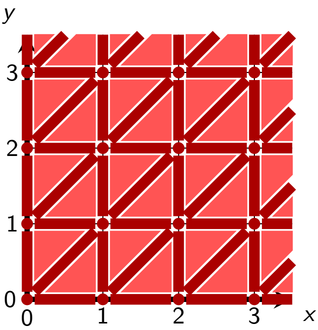
With this definition, the successor by time elapsing of a “region” is not unique.

Regions



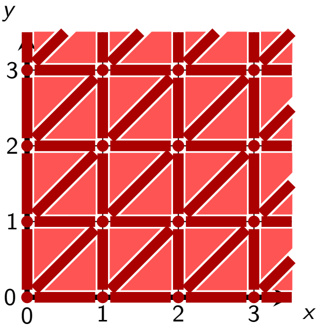
With this definition, the successor by time elapsing of a "region" is not unique.

Regions



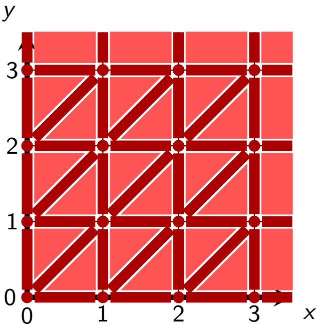
- We still do not have a finite partitioning;

Regions



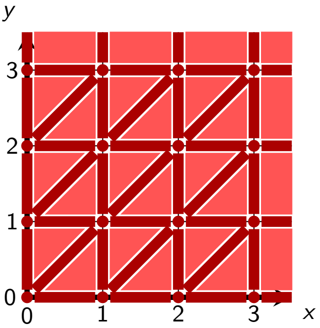
- We still do not have a finite partitioning;
- Let *max* be the maximal value of the constants in guards and invariants;

Regions



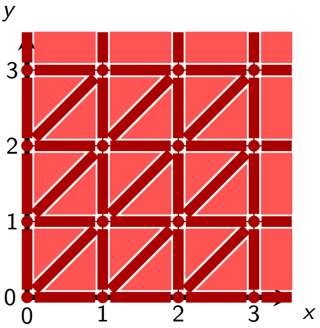
- We still do not have a finite partitioning;
- Let *max* be the maximal value of the constants in guards and invariants;
- The value of clocks beyond *max* is irrelevant (extrapolation or *k*-approximation).

Regions



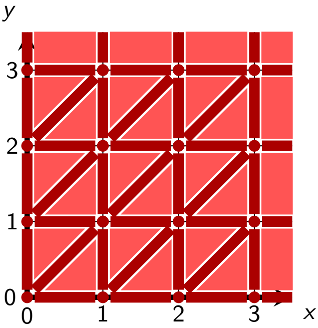
- We still do not have a finite partitioning;
- Let *max* be the maximal value of the constants in guards and invariants;
- The value of clocks beyond *max* is irrelevant (extrapolation or *k*-approximation).
- It can be further improved:

Regions



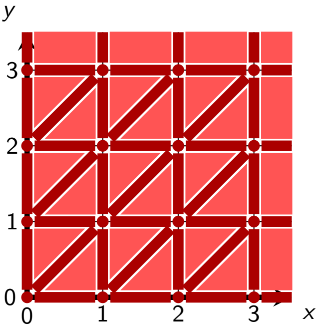
- We still do not have a finite partitioning;
- Let *max* be the maximal value of the constants in guards and invariants;
- The value of clocks beyond *max* is irrelevant (extrapolation or *k*-approximation).
- It can be further improved:
  - One constant per clock;

Regions



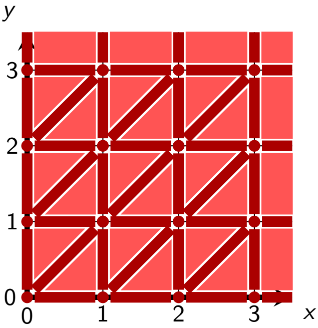
- We still do not have a finite partitioning;
- Let *max* be the maximal value of the constants in guards and invariants;
- The value of clocks beyond *max* is irrelevant (extrapolation or *k*-approximation).
- It can be further improved:
  - One constant per clock;
  - One constant per clock and per location.

Regions



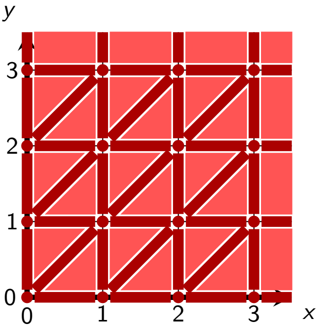
$$\nu \equiv \nu' \Leftrightarrow \begin{cases} E(\nu(x)) = E(\nu'(x)) \\ \quad \text{or } (\nu(x) > max \text{ and } \nu'(x) > max) \\ \text{if } (\nu(x) \leq max \text{ and } \nu'(y) \leq max) \text{ then} \\ \quad \text{frac}(\nu(x)) < \text{frac}(\nu(y)) \Leftrightarrow \\ \quad \text{frac}(\nu'(x)) < \text{frac}(\nu'(y)) \end{cases}$$

Region Automaton



- Let $Succ(R)$ be the set of successor regions of R by time elapsing:
  $Succ(R) = \{R' | \exists \nu \in R, \exists t \in \mathbb{R}^+, \nu + t \in R'\}$

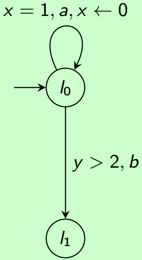
Region Automaton



- Let $Succ(R)$ be the set of successor regions of R by time elapsing: $Succ(R) = \{R' | \exists \nu \in R, \exists t \in \mathbb{R}^+, \nu + t \in R'\}$
- The region automaton $\mathcal{A} = (Q, A, q_0, \rightarrow)$ is defined by:
  - $Q = L \times \mathbb{R}^+ / \equiv$;
  - $q_0 = (l_0, \mathbf{0})$;
  - $\rightarrow = \{(q, R) \xrightarrow{a} (q', R') | \exists q \xrightarrow{\delta,\alpha,r} q' \in E$ and $\exists R'' \in Succ(R)$ s.t. $R'' \subseteq \delta$ and $R' = R''[r \leftarrow 0]\}$.

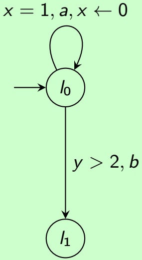
Region Automaton

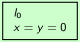
Example



Region Automaton

Example



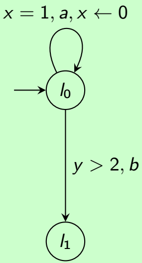


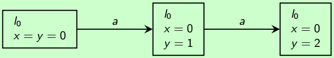
Region Automaton

Example

Region Automaton

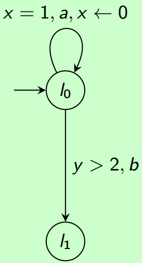
Example

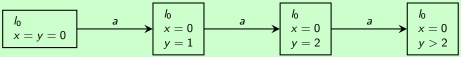




Region Automaton

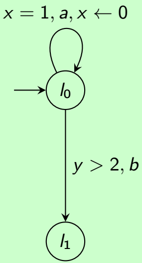
Example

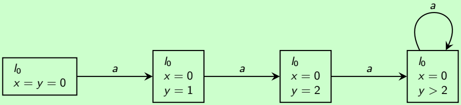




Region Automaton

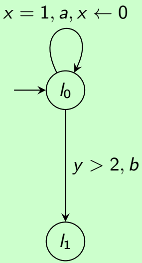
Example

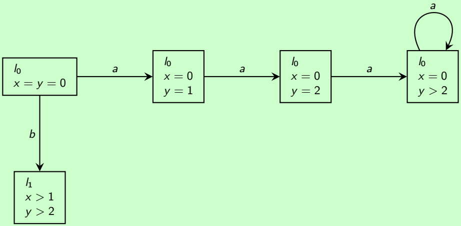




Region Automaton

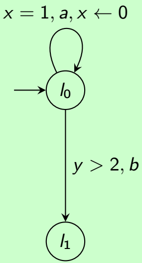
Example

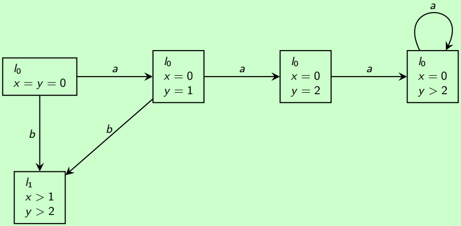




Region Automaton

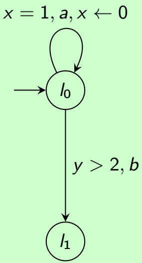
Example

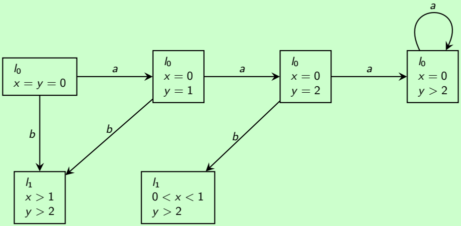




Region Automaton

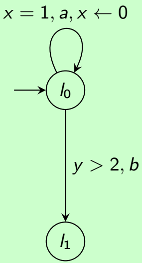
Example

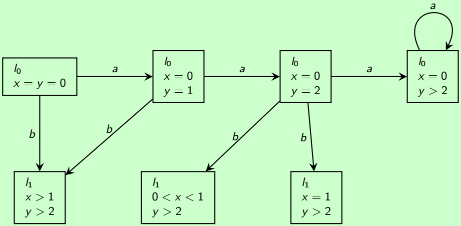




Region Automaton

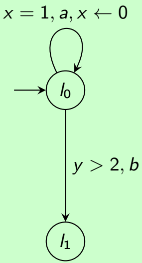
Example

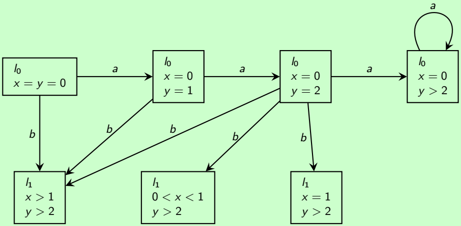




Region Automaton

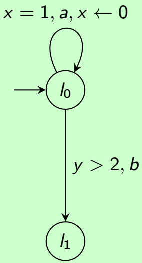
Example

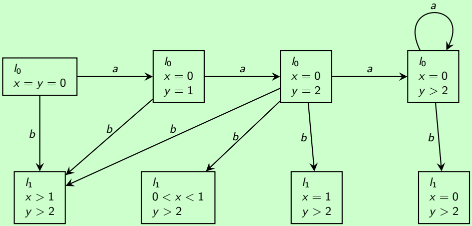




Region Automaton

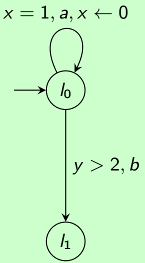
Example

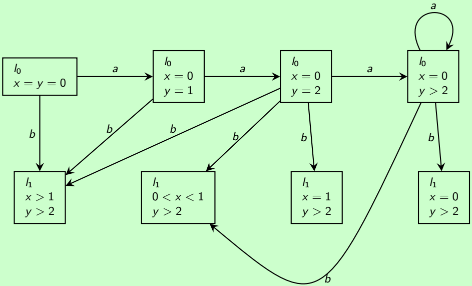




Region Automaton

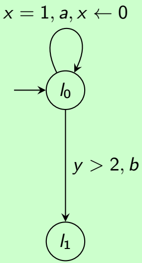
Example

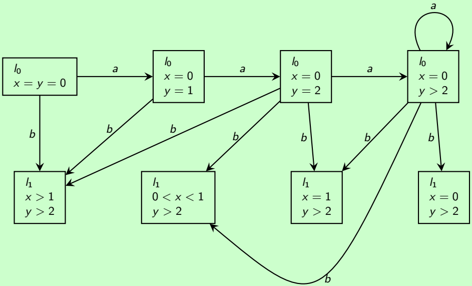




Region Automaton

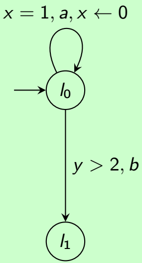
Example

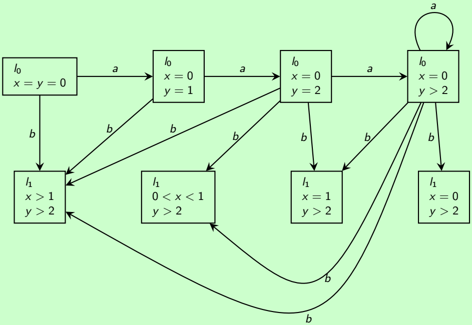




Region Automaton

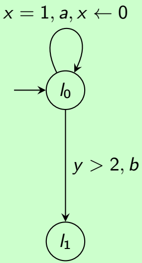
Example

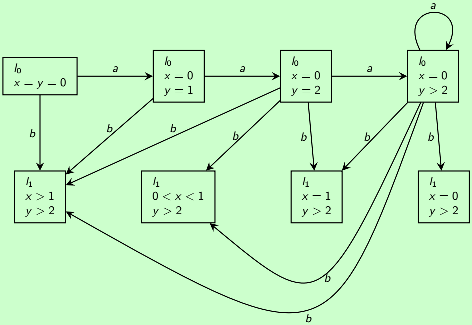




Region Automaton

Example





Exercise

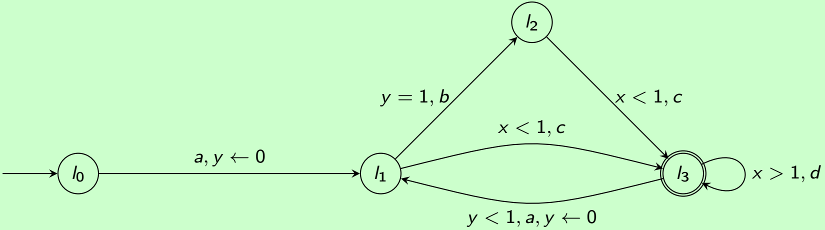
Draw the transitions in the clock space.

Region Automaton

Exercise

We assume **only for this exercise** that we can take a transition only after having waited for a positive duration (> 0).

Compute the region graph of the following TA from [AD94]:

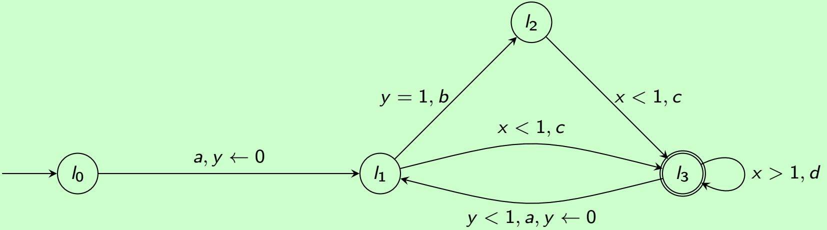


Region Automaton

Exercise

We assume **only for this exercise** that we can take a transition only after having waited for a positive duration (> 0).

Compute the region graph of the following TA from [AD94]:



Remark: This automaton can do cycle without letting time elapse.

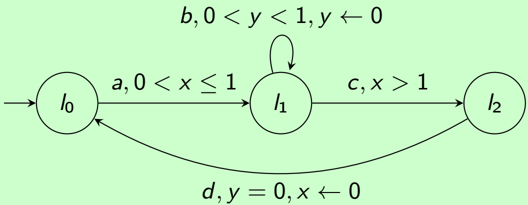
Definition (Zeno Run)

An infinite run is Zeno if its duration is finite.

Region Automaton

Exercise

Compute the region automaton of the following TA:

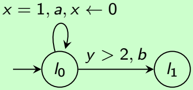


Model-checking

- The region automaton preserves time-abstract bisimulation
  bisimulation were dad' is equivalent to a whenever d and d' are delays
- We can then use it to verify CTL properties on TA:
  Ignoring sequences of delays without any action

Exercise

1. Does the following TA (from the example for the region automaton) verify $\mathbf{AF} l_1$?



2. What about $\mathbf{EF}(l_0 \text{ and } x = 1)$?

Plan I

Timed Computation Tree Logic

- We may want to refer to time explicitly in formulas;

Timed Computation Tree Logic

- We may want to refer to time explicitly in formulas;
- We thus add time to temporal logics LTL and CTL;

Timed Computation Tree Logic

- We may want to refer to time explicitly in formulas;
- We thus add time to temporal logics LTL and CTL;
- We study here the timed version of CTL: Timed CTL (ou TCTL) [AD94]:

Timed Computation Tree Logic

- We may want to refer to time explicitly in formulas;
- We thus add time to temporal logics LTL and CTL;
- We study here the timed version of CTL: Timed CTL (ou TCTL) [AD94]:
- Let X be a set of clocks and AP a set of atomic propositions, I an interval of $\mathbb{R}_{\geq 0}$:

Syntax of TCTL

$$\varphi ::= \mathbf{p} \,|\, x \sim k \,|\, \neg\varphi \,|\, \varphi \vee \varphi \,|\, z \,\mathbf{in}\, \varphi \,|\, \mathbf{A}\varphi\mathbf{U}_I\varphi \,|\, \mathbf{E}\varphi\mathbf{U}\varphi$$

Timed Computation Tree Logic

- We may want to refer to time explicitly in formulas;
- We thus add time to temporal logics LTL and CTL;
- We study here the timed version of CTL: Timed CTL (ou TCTL) [AD94]:
- Let X be a set of clocks and AP a set of atomic propositions, I an interval of $\mathbb{R}_{\geq 0}$:

Syntax of TCTL

$$\varphi ::= \mathbf{p} \,|\, x \sim k \,|\, \neg\varphi \,|\, \varphi \vee \varphi \,|\, z \,\mathbf{in}\, \varphi \,|\, \mathbf{A}\varphi\mathbf{U}_I\varphi \,|\, \mathbf{E}\varphi\mathbf{U}\varphi$$

- neXt has been removed (Why ?) ;

Timed Computation Tree Logic

- We may want to refer to time explicitly in formulas;
- We thus add time to temporal logics LTL and CTL;
- We study here the timed version of CTL: Timed CTL (ou TCTL) [AD94]:
- Let X be a set of clocks and AP a set of atomic propositions, I an interval of $\mathbb{R}_{\geq 0}$:

Syntax of TCTL

$$\varphi ::= \mathbf{p} \,|\, x \sim k \,|\, \neg\varphi \,|\, \varphi \vee \varphi \,|\, z\ \mathbf{in}\ \varphi \,|\, \mathbf{A}\varphi\mathbf{U}_I\varphi \,|\, \mathbf{E}\varphi\mathbf{U}\varphi$$

- neXt has been removed (Why ?) ;
- Until now has a timed interval attached: $\varphi_1\mathbf{U}_I\varphi_2$ holds for a run ρ if there exists a prefix ρ' of ρ that satisfies $\varphi_1\mathbf{U}\varphi_2$ and with a duration in I;

Timed Computation Tree Logic

- We may want to refer to time explicitly in formulas;
- We thus add time to temporal logics LTL and CTL;
- We study here the timed version of CTL: Timed CTL (ou TCTL) [AD94]:
- Let X be a set of clocks and AP a set of atomic propositions, I an interval of $\mathbb{R}_{\geq 0}$:

Syntax of TCTL

$$\varphi ::= \mathbf{p} \,|\, x \sim k \,|\, \neg\varphi \,|\, \varphi \vee \varphi \,|\, z \textbf{ in } \varphi \,|\, \mathbf{A}\varphi\mathbf{U}_I\varphi \,|\, \mathbf{E}\varphi\mathbf{U}\varphi$$

- neXt has been removed (Why ?) ;
- Until now has a timed interval attached: $\varphi_1\mathbf{U}_I\varphi_2$ holds for a run ρ if there exists a prefix ρ' of ρ that satisfies $\varphi_1\mathbf{U}\varphi_2$ and with a duration in I;
- $x \in X$, $k \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$ is a constraint on the clocks of the system.

Semantics of TCTL

Sémantics of TCTL

Let $\mathcal{S} = (W, \rightarrow, \ell)$ be a timed Kripke structure, $s = (l, \nu) \in W$ and $c = (c_i)_{i \in \mathbb{N}} \in Path(t)A$, where $Path(t)$ is the set of paths starting at s in $\mathcal{S}$ (thus $c_0 = t$). Let $\Delta(c, i)$ be the duration of $c_0 \longrightarrow \cdots \longrightarrow c_i$

- $(\mathcal{S}, s) \models \mathbf{p}$ iff $\mathbf{p} \in \ell(t)$;

Semantics of TCTL

Sémantics of TCTL

Let $\mathcal{S} = (W, \rightarrow, \ell)$ be a timed Kripke structure, $s = (l, \nu) \in W$ and $c = (c_i)_{i \in \mathbb{N}} \in Path(t)A$, where $Path(t)$ is the set of paths starting at s in $\mathcal{S}$ (thus $c_0 = t$). Let $\Delta(c, i)$ be the duration of $c_0 \longrightarrow \cdots \longrightarrow c_i$

- $(\mathcal{S}, s) \models \mathbf{p}$ iff $\mathbf{p} \in \ell(t)$;
- $(\mathcal{S}, s) \models \neg\varphi$ iff $(\mathcal{S}, t) \not\models \varphi$;

Semantics of TCTL

Sémantics of TCTL

Let $\mathcal{S} = (W, \rightarrow, \ell)$ be a timed Kripke structure, $s = (l, \nu) \in W$ and $c = (c_i)_{i\in\mathbb{N}} \in Path(t)A$, where $Path(t)$ is the set of paths starting at s in $\mathcal{S}$ (thus $c_0 = t$). Let $\Delta(c, i)$ be the duration of $c_0 \longrightarrow \cdots \longrightarrow c_i$

- $(\mathcal{S}, s) \models \mathbf{p}$ iff $\mathbf{p} \in \ell(t)$;
- $(\mathcal{S}, s) \models \neg\varphi$ iff $(\mathcal{S}, t) \not\models \varphi$;
- $(\mathcal{S}, s) \models \varphi_1 \vee \varphi_2$ iff $(\mathcal{S}, s, w) \models \varphi_1$ or $(\mathcal{S}, s, w) \models \varphi_2$;

Semantics of TCTL

Sémantics of TCTL

Let $\mathcal{S} = (W, \rightarrow, \ell)$ be a timed Kripke structure, $s = (l, \nu) \in W$ and $c = (c_i)_{i \in \mathbb{N}} \in Path(t)A$, where $Path(t)$ is the set of paths starting at s in $\mathcal{S}$ (thus $c_0 = t$). Let $\Delta(c, i)$ be the duration of $c_0 \longrightarrow \cdots \longrightarrow c_i$

- $(\mathcal{S}, s) \models \mathbf{p}$ iff $\mathbf{p} \in \ell(t)$;
- $(\mathcal{S}, s) \models \neg\varphi$ iff $(\mathcal{S}, t) \not\models \varphi$;
- $(\mathcal{S}, s) \models \varphi_1 \vee \varphi_2$ iff $(\mathcal{S}, s, w) \models \varphi_1$ or $(\mathcal{S}, s, w) \models \varphi_2$;
- $(\mathcal{S}, s) \models x \sim k$ iff $\nu(x) \sim k$;

Semantics of TCTL

Sémantics of TCTL

Let $\mathcal{S} = (W, \rightarrow, \ell)$ be a timed Kripke structure, $s = (l, \nu) \in W$ and $c = (c_i)_{i\in\mathbb{N}} \in Path(t)A$, where $Path(t)$ is the set of paths starting at s in $\mathcal{S}$ (thus $c_0 = t$). Let $\Delta(c, i)$ be the duration of $c_0 \longrightarrow \cdots \longrightarrow c_i$

- $(\mathcal{S}, s) \models \mathbf{p}$ iff $\mathbf{p} \in \ell(t)$;
- $(\mathcal{S}, s) \models \neg\varphi$ iff $(\mathcal{S}, t) \not\models \varphi$;
- $(\mathcal{S}, s) \models \varphi_1 \vee \varphi_2$ iff $(\mathcal{S}, s, w) \models \varphi_1$ or $(\mathcal{S}, s, w) \models \varphi_2$;
- $(\mathcal{S}, s) \models x \sim k$ iff $\nu(x) \sim k$;
- $(\mathcal{S}, s) \models \mathbf{A}\varphi_1\mathbf{U}\varphi_2$ iff $\forall c \in Path(s), \exists i, (\mathcal{S}, c_i) \models \varphi_2$, $\Delta(c, i) \in I$, and $\forall j < i, (\mathcal{S}, c_j) \models \varphi_1$;

Semantics of TCTL

Sémantics of TCTL

Let $\mathcal{S} = (W, \rightarrow, \ell)$ be a timed Kripke structure, $s = (l, \nu) \in W$ and $c = (c_i)_{i \in \mathbb{N}} \in Path(t)A$, where $Path(t)$ is the set of paths starting at s in $\mathcal{S}$ (thus $c_0 = t$). Let $\Delta(c, i)$ be the duration of $c_0 \longrightarrow \cdots \longrightarrow c_i$

- $(\mathcal{S}, s) \models \mathbf{p}$ iff $\mathbf{p} \in \ell(t)$;
- $(\mathcal{S}, s) \models \neg\varphi$ iff $(\mathcal{S}, t) \not\models \varphi$;
- $(\mathcal{S}, s) \models \varphi_1 \vee \varphi_2$ iff $(\mathcal{S}, s, w) \models \varphi_1$ or $(\mathcal{S}, s, w) \models \varphi_2$;
- $(\mathcal{S}, s) \models x \sim k$ iff $\nu(x) \sim k$;
- $(\mathcal{S}, s) \models \mathbf{A}\varphi_1\mathbf{U}\varphi_2$ iff $\forall c \in Path(s), \exists i, (\mathcal{S}, c_i) \models \varphi_2$, $\Delta(c, i) \in I$, and $\forall j < i, (\mathcal{S}, c_j) \models \varphi_1$;
- $(\mathcal{S}, s) \models \mathbf{E}\varphi_1\mathbf{U}\varphi_2$ iff $\exists c \in Path(s), \exists i, (\mathcal{S}, c_i) \models \varphi_2$, $\Delta(c, i) \in I$, and $\forall j < i, (\mathcal{S}, c_j) \models \varphi_1$.

Timed Until

We often use shorthands for intervals in the Until operator:

- $\mathbf{U}_{<2}$ is $\mathbf{U}_{[0,2]}$;
- $\mathbf{U}_{\geq 2}$ is $\mathbf{U}_{[2,+\infty)}$;
- $\mathbf{U}_{=2}$ is $\mathbf{U}_{[2,2]}$;
- etc.

Timed Properties: Examples

- Bounded response:

Timed Properties: Examples

- Bounded response:

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{AF}_{\leq 3}\mathbf{q})$$

Timed Properties: Examples

- Bounded response:

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{AF}_{\leq 3}\mathbf{q})$$

- Periodicity:

Timed Properties: Examples

- Bounded response:

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{AF}_{\leq 3}\mathbf{q})$$

- Periodicity:

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{A}\neg\mathbf{p}\mathbf{U}_{=3}\mathbf{p})$$

Timed Properties: Examples

- Bounded response:

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{AF}_{\leq 3}\mathbf{q})$$

- Periodicity:

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{A}\neg\mathbf{p}\mathbf{U}_{=3}\mathbf{p})$$

- Minimum delay (e.g. sporadic tasks):

Timed Properties: Examples

- Bounded response:

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{AF}_{\leq 3}\mathbf{q})$$

- Periodicity:

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{A}\neg\mathbf{p}\mathbf{U}_{=3}\mathbf{p})$$

- Minimum delay (e.g. sporadic tasks):

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{A}\neg\mathbf{p}\mathbf{U}_{\geq 3}\mathbf{p})$$

Timed Properties: Examples

- Bounded response:

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{AF}_{\leq 3}\mathbf{q})$$

- Periodicity:

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{A}\neg\mathbf{p}\mathbf{U}_{=3}\mathbf{p})$$

- Minimum delay (e.g. sporadic tasks):

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{A}\neg\mathbf{p}\mathbf{U}_{\geq 3}\mathbf{p})$$

- Minimum interval (e.g. between 3 and 10):

Timed Properties: Examples

- Bounded response:

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{AF}_{\leq 3}\mathbf{q})$$

- Periodicity:

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{A}\neg\mathbf{p}\mathbf{U}_{=3}\mathbf{p})$$

- Minimum delay (e.g. sporadic tasks):

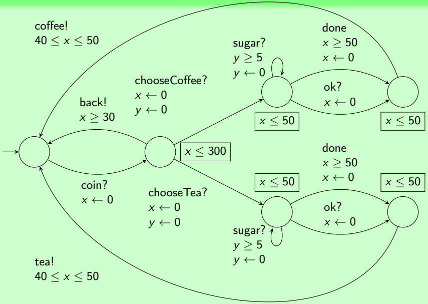
$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{A}\neg\mathbf{p}\mathbf{U}_{\geq 3}\mathbf{p})$$

- Minimum interval (e.g. between 3 and 10):

$$\mathbf{AG}(\mathbf{p} \Rightarrow \mathbf{A}\neg\mathbf{p}\mathbf{U}_{=3}\mathbf{AF}_{\leq 7}\mathbf{p})$$

Timed Properties: Exercise

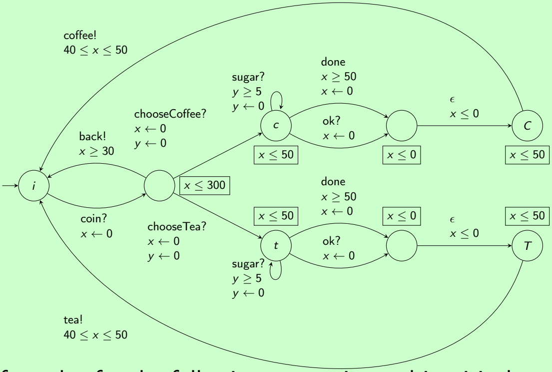
Exercise



1. Add atomic propositions: i no session started, c coffee choosed, t tea choosed, C coffee served, T tea served (add some locations for the last two);

Timed Properties: Exercise

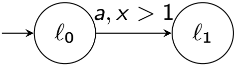
Exercise

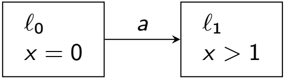


2. Write TCTL formulas for the following properties and intuitively assess their truth:
   - When a vending session has started, a beverage is always obtained in less than 40s;
   - When coffee has been chosen, tea cannot be obtained before the session is ended;
   - When coffee has been chosen, tea cannot be obtained before at least 5s;
   - It is possible to never have any coffee or tea;
   - It is possible to never have any coffe or tea but still start sessions infinitely often.

TCTL Model-checking with Untimed Until

- If all until operators have interval $[0, inf)$ we can reduce the verification to CTL;
- We cannot use the previous region construction though, because it does not distinguish the values of clocks when time elapses:





Is $\mathbf{EF}(x = 1)$ satisfied?

- We build a variant: a region automaton with delays.

Region Automaton With Delays

- We make explicit the passing from one region to another by delaying [ACD93]:

Region Automaton With Delays

- We make explicit the passing from one region to another by delaying [ACD93]:
  - actions: $(l, r) \xrightarrow{a \in \Sigma} (l', r')$ iff $\exists (l, \gamma, a, R, l')$ s.t.:

$$\left\{ \begin{array}{l} r' = r[R \leftarrow 0] \\ r \models \gamma \\ r' \models Inv(l') \end{array} \right.$$

Region Automaton With Delays

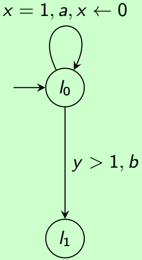
- We make explicit the passing from one region to another by delaying [ACD93]:
  - actions: $(l, r) \xrightarrow{a \in \Sigma} (l', r')$ iff $\exists (l, \gamma, a, R, l')$ s.t.:

$$\begin{cases} r' = r[R \leftarrow 0] \\ r \models \gamma \\ r' \models Inv(l') \end{cases}$$

  - time: $(l, r) \xrightarrow{\delta} (l, Succ_1(r))$ with $Succ_1(r)$ the first region that is reachable from r by delaying or r if none exist (beyond the maximal constant).

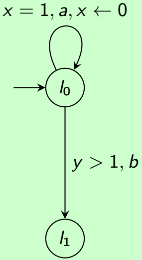
Region Automaton With Delays

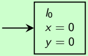
Example



Region Automaton With Delays

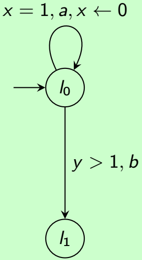
Example

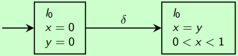




Region Automaton With Delays

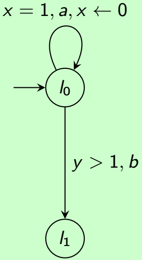
Example

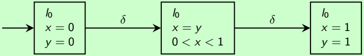




Region Automaton With Delays

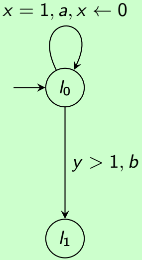
Example

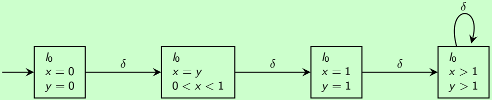




Region Automaton With Delays

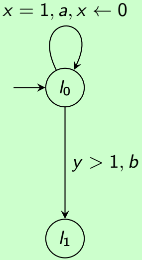
Example

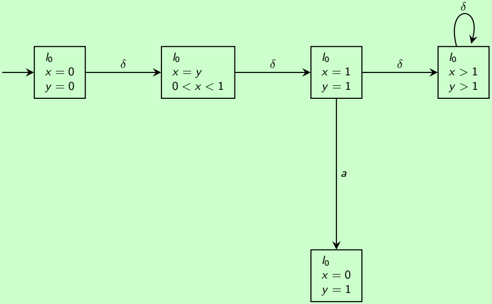




Region Automaton With Delays

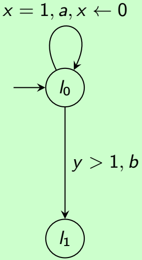
Example

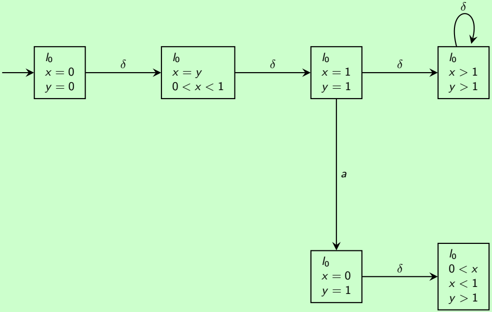




Region Automaton With Delays

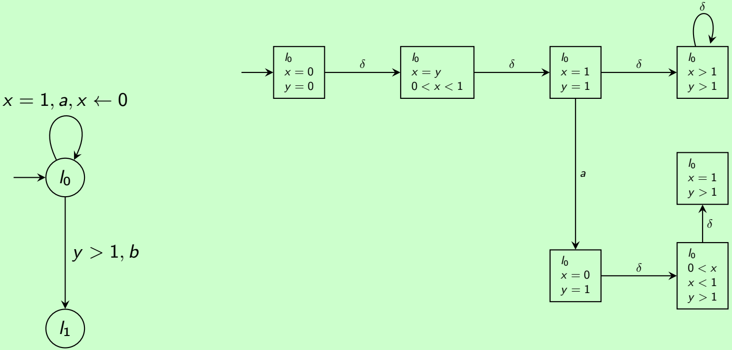
Example





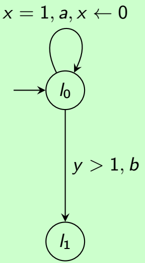
Region Automaton With Delays

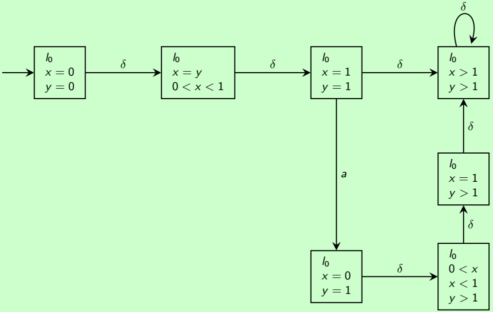
Example



Region Automaton With Delays

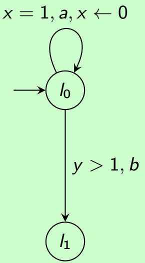
Example

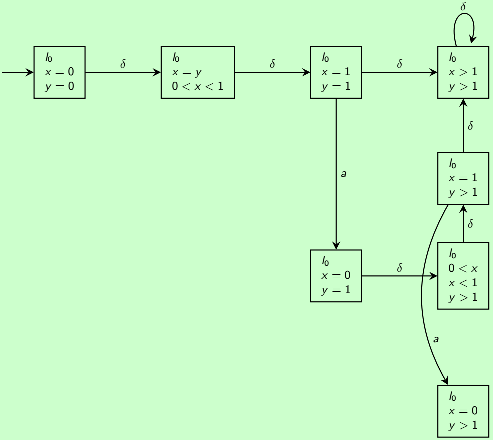




Region Automaton With Delays

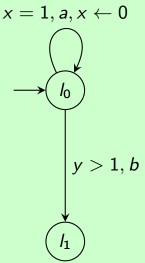
Example

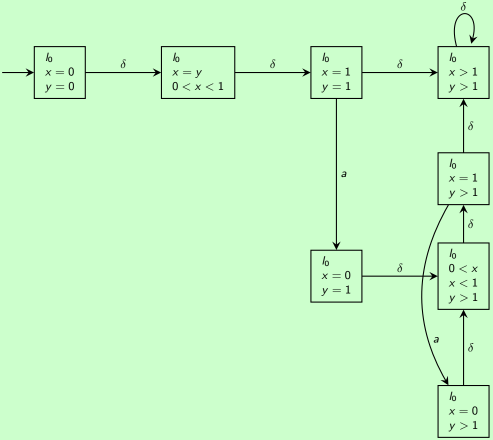




Region Automaton With Delays

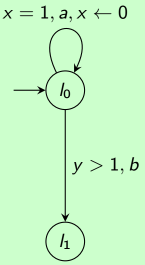
Example

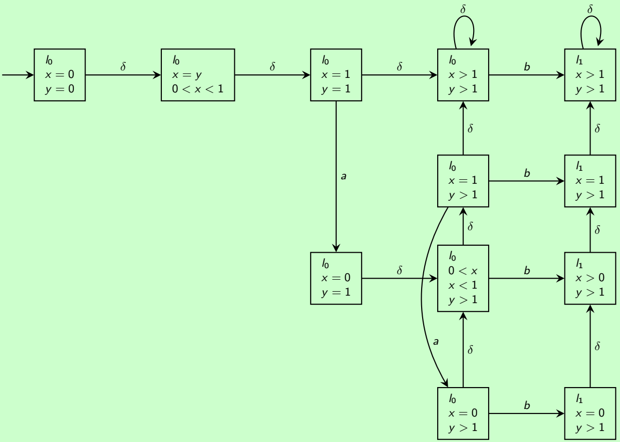




Region Automaton With Delays

Example





TCTL Model-checking with Timed Untils

- We want to check $\varphi_1 U_I \varphi_2$ in state $s = (\ell, \nu)$;
- This is equivalent to $\varphi_1 U(z \in I \wedge \varphi_2)$ in s with an additional fresh clock z such that $\nu(z) = 0$;
- In case of a non-nested formula, just add z and proceed as before;
- In case of a nested formula, we need to test the nested timed until subformulas for each state of the region automaton;

Example

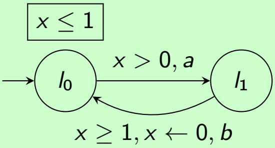
Consider formula $\varphi = \mathbf{EF}\psi$ with $\psi = \varphi_1 U_{[1,2]} \varphi_2$

1. build the region automaton (with delays);
2. for each of its states (ℓ, r), check $\psi' = \varphi_2 U(z \in [1,2] \wedge \varphi_2$ in the region automaton starting from $(\ell, r[z \leftarrow 0])$, where $r[z \leftarrow 0]$ is r plus an additional clock z with constraint $z = 0$;
3. label each of the (ℓ, r) that satisfy ψ by a new atomic proposition p_ψ;
4. check $\mathbf{EF} p_\psi$.

TCTL Model-checking

Exercise

Is property $\varphi = \mathbf{AG}(l_1 \Rightarrow \mathbf{AF}_{\leq 1} l_0)$ satisfied by the following TA:



Plan I

Zones

- Enumerating regions is usually very inefficient in practice (though optimal in theory) ;

Zones

- Enumerating regions is usually very inefficient in practice (though optimal in theory) ;
- We group regions into zones (convex unions of regions) s.t.:

$$(l,\nu) \equiv (l',\nu') \Leftrightarrow \exists \rho,\rho' \in [\![\mathcal{S}]\!], \left\{ \begin{array}{l} \mathit{last}(\rho) = (l,\nu), \mathit{last}(\rho') = (l',\nu') \\ \text{and } \mathit{Untimed}(\rho) = \mathit{Untimed}(\rho') \end{array} \right.$$

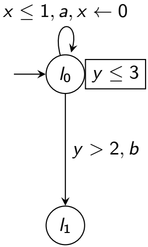
Zones

- Enumerating regions is usually very inefficient in practice (though optimal in theory) ;
- We group regions into zones (convex unions of regions) s.t.:

$$(l, \nu) \equiv (l', \nu') \Leftrightarrow \exists \rho, \rho' \in [\![\mathcal{S}]\!], \left\{ \begin{array}{l} \mathit{last}(\rho) = (l, \nu), \mathit{last}(\rho') = (l', \nu') \\ \text{and } \mathit{Untimed}(\rho) = \mathit{Untimed}(\rho') \end{array} \right.$$

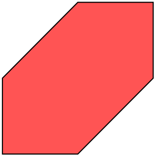
- Zones (like regions) are particular convex polyhedra.

Zones: Example



Zones

- Zones can be encoded by Difference Bound Matrix (DBM):
  $x \leq 1 \Leftrightarrow x - x_0 \leq 1, x_0 = 0$

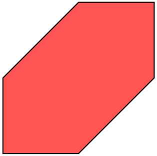


$$\begin{cases} 0 \leq x \leq 2 \\ 0 \leq y \leq 2 \\ -1 \leq x - y \leq 1 \end{cases}$$

$$\begin{bmatrix} (0, \leq) & (0, \leq) & (0, \leq) \\ (2, \leq) & (0, \leq) & (1, \leq) \\ (2, \leq) & (1, \leq) & (0, \leq) \end{bmatrix}$$

Zones

- Zones can be encoded by Difference Bound Matrix (DBM):
  $x \leq 1 \Leftrightarrow x - x_0 \leq 1, x_0 = 0$



$$\begin{cases} 0 \leq x \leq 2 \\ 0 \leq y \leq 2 \\ -1 \leq x - y \leq 1 \end{cases}$$

$$\begin{bmatrix} (0, \leq) & (0, \leq) & (0, \leq) \\ (2, \leq) & (0, \leq) & (1, \leq) \\ (2, \leq) & (1, \leq) & (0, \leq) \end{bmatrix}$$

Exercise

Write the system of inequalities encoded by the following DBM and draw the corresponding polyhedron:

$$\begin{bmatrix} (0, \leq) & (-1, <) & (-2, \leq) \\ (2, \leq) & (0, \leq) & (0, \leq) \\ (4, \leq) & (2, \leq) & (0, \leq) \end{bmatrix}$$

Canonical Form of DBMs

Exercise

Write the systems of inequalities encoded by the following DBMs and draw the corresponding polyhedra:

$$\begin{bmatrix} (0,\leq) & (0,\leq) & (-1,\leq) \\ (2,\leq) & (0,\leq) & (0,\leq) \\ (4,\leq) & (4,\leq) & (0,\leq) \end{bmatrix} \text{ and } \begin{bmatrix} (0,\leq) & (0,\leq) & (-1,\leq) \\ (2,\leq) & (0,\leq) & (0,\leq) \\ (4,\leq) & (5,\leq) & (0,\leq) \end{bmatrix}$$

Canonical Form of DBMs

Exercise

Write the systems of inequalities encoded by the following DBMs and draw the corresponding polyhedra:

$$\begin{bmatrix} (0,\leq) & (0,\leq) & (-1,\leq) \\ (2,\leq) & (0,\leq) & (0,\leq) \\ (4,\leq) & (4,\leq) & (0,\leq) \end{bmatrix} \text{ and } \begin{bmatrix} (0,\leq) & (0,\leq) & (-1,\leq) \\ (2,\leq) & (0,\leq) & (0,\leq) \\ (4,\leq) & (5,\leq) & (0,\leq) \end{bmatrix}$$

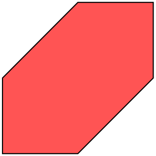
- A DBM is in canonical form if all its coefficients are minimal;
- Allows for efficient equality and inclusion tests;
- Any DBM can be put into canonical form using the Floyd-Warshall algorithm:

Floyd-Warshall Algorithm

for k **from** `1` **to** `N`:
  for i **from** `1` **to** `N`:
    for j **from** `1` **to** `N`:
      $D(i,j) \leftarrow min(D(i,j), D(i,k) + D(k,j))$

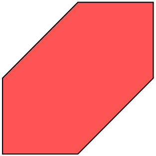
Forward Reachability Using Zones

- We define two specific operations on zones:



Reset of y (to zero):

$$Z[y \leftarrow 0]$$

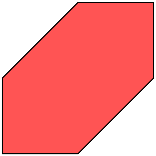


Future of Z (time elapsing):

$$Z^{\nearrow}$$

Forward Reachability Using Zones

- We define two specific operations on zones:



Reset of y (to zero):

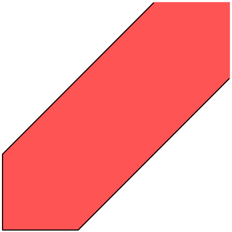
$$Z[y \leftarrow 0]$$

Future of Z (time elapsing):

$$Z^{\nearrow}$$

Forward Reachability Using Zones

- We define two specific operations on zones:



Reset of y (to zero):

$$Z[y \leftarrow 0]$$

Future of Z (time elapsing):

$$Z^{\nearrow}$$

Reset, future, intersection using DBMs

Exercise

1. if D is a canonical form DBM representing zone Z, compute canonical form DBM $D^{\nearrow}$ representing $Z^{\nearrow}$;
2. if D_1 and D_2 are two DBMs, compute the DBM D' representing the intersection of the two corresponding zones; if D_1 and D_2 are in canonical form, is your D' always in canonical form?
3. if D is the canonical form DBM representing zone Z, compute the canonical form DBM $D[x \leftarrow 0]$ representing $Z[x \leftarrow 0]$.

Forward Reachability Using Zones

- For a TA $\mathcal{A} = (L, l_0, X, A, E, Inv)$, we have:

Forward Reachability Using Zones

- For a TA $\mathcal{A} = (L, l_0, X, A, E, Inv)$, we have:
- Initially:

$$Z_0 = \mathbf{0}^{\nearrow} \wedge Inv(l_0)$$

Forward Reachability Using Zones

- For a TA $\mathcal{A} = (L, l_0, X, A, E, Inv)$, we have:
- Initially:

$$Z_0 = \mathbf{0}^{\nearrow} \wedge Inv(l_0)$$

- For an edge $e = (l, \delta, \alpha, R, l')$,

$$Z' = ((Z \wedge \delta)[R \leftarrow 0])^{\nearrow} \wedge Inv(l')$$

Forward Reachability Using Zones

- For a TA $\mathcal{A} = (L, l_0, X, A, E, Inv)$, we have:
- Initially:

$$Z_0 = \mathbf{0}^{\nearrow} \wedge Inv(l_0)$$

- For an edge $e = (l, \delta, \alpha, R, l')$,

$$Z' = ((Z \wedge \delta)[R \leftarrow 0])^{\nearrow} \wedge Inv(l')$$

- We can thus compute a "zone automaton" also called simulation graph;

Forward Reachability Using Zones

- For a TA $\mathcal{A} = (L, l_0, X, A, E, Inv)$, we have:
- Initially:

$$Z_0 = \mathbf{0}^{\nearrow} \wedge Inv(l_0)$$

- For an edge $e = (l, \delta, \alpha, R, l')$,

$$Z' = ((Z \wedge \delta)[R \leftarrow 0])^{\nearrow} \wedge Inv(l')$$

- We can thus compute a "zone automaton" also called simulation graph;
- It is made finite by converging either by equality or inclusion (of zones, for a given location):

Forward Reachability Using Zones

- For a TA $\mathcal{A} = (L, l_0, X, A, E, Inv)$, we have:
- Initially:
$$Z_0 = \mathbf{0}^{\nearrow} \wedge Inv(l_0)$$
- For an edge $e = (l, \delta, \alpha, R, l')$,
$$Z' = ((Z \wedge \delta)[R \leftarrow 0])^{\nearrow} \wedge Inv(l')$$
- We can thus compute a "zone automaton" also called simulation graph;
- It is made finite by converging either by equality or inclusion (of zones, for a given location):
- When computing (l', Z'), we loop on a previously computed (l, Z) when:

Forward Reachability Using Zones

- For a TA $\mathcal{A} = (L, l_0, X, A, E, Inv)$, we have:
- Initially:
$$Z_0 = \mathbf{0}^{\nearrow} \wedge Inv(l_0)$$
- For an edge $e = (l, \delta, \alpha, R, l')$,
$$Z' = ((Z \wedge \delta)[R \leftarrow 0])^{\nearrow} \wedge Inv(l')$$
- We can thus compute a "zone automaton" also called simulation graph;
- It is made finite by converging either by equality or inclusion (of zones, for a given location):
- When computing (l', Z'), we loop on a previously computed (l, Z) when:
  - equality: $l' = l$ and $Z' = Z$ (Preserves);

Forward Reachability Using Zones

- For a TA $\mathcal{A} = (L, l_0, X, A, E, Inv)$, we have:
- Initially:

$$Z_0 = \mathbf{0}^{\nearrow} \wedge Inv(l_0)$$

- For an edge $e = (l, \delta, \alpha, R, l')$,

$$Z' = ((Z \wedge \delta)[R \leftarrow 0])^{\nearrow} \wedge Inv(l')$$

- We can thus compute a "zone automaton" also called simulation graph;
- It is made finite by converging either by equality or inclusion (of zones, for a given location):
- When computing (l', Z'), we loop on a previously computed (l, Z) when:
  - equality: $l' = l$ and $Z' = Z$ (Preserves the untimed language);

Forward Reachability Using Zones

- For a TA $\mathcal{A} = (L, l_0, X, A, E, Inv)$, we have:
- Initially:

$$Z_0 = \mathbf{0}^{\nearrow} \wedge Inv(l_0)$$

- For an edge $e = (l, \delta, \alpha, R, l')$,

$$Z' = ((Z \wedge \delta)[R \leftarrow 0])^{\nearrow} \wedge Inv(l')$$

- We can thus compute a "zone automaton" also called simulation graph;
- It is made finite by converging either by equality or inclusion (of zones, for a given location):
- When computing (l', Z'), we loop on a previously computed (l, Z) when:
  - equality: $l' = l$ and $Z' = Z$ (Preserves the untimed language);
  - inclusion: $l' = l$ and $Z' \subseteq Z$ (Preserves).

Forward Reachability Using Zones

- For a TA $\mathcal{A} = (L, l_0, X, A, E, Inv)$, we have:
- Initially:

$$Z_0 = \mathbf{0}^{\nearrow} \wedge Inv(l_0)$$

- For an edge $e = (l, \delta, \alpha, R, l')$,

$$Z' = ((Z \wedge \delta)[R \leftarrow 0])^{\nearrow} \wedge Inv(l')$$

- We can thus compute a "zone automaton" also called simulation graph;
- It is made finite by converging either by equality or inclusion (of zones, for a given location):
- When computing (l', Z'), we loop on a previously computed (l, Z) when:
  - equality: $l' = l$ and $Z' = Z$ (Preserves the untimed language);
  - inclusion: $l' = l$ and $Z' \subseteq Z$ (Preserves reachability).

Forward Reachability Using Zones

- For a TA $\mathcal{A} = (L, l_0, X, A, E, Inv)$, we have:
- Initially:

$$Z_0 = \mathbf{0}^{\nearrow} \wedge Inv(l_0)$$

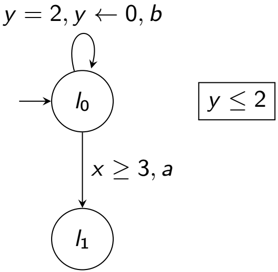
- For an edge $e = (l, \delta, \alpha, R, l')$,

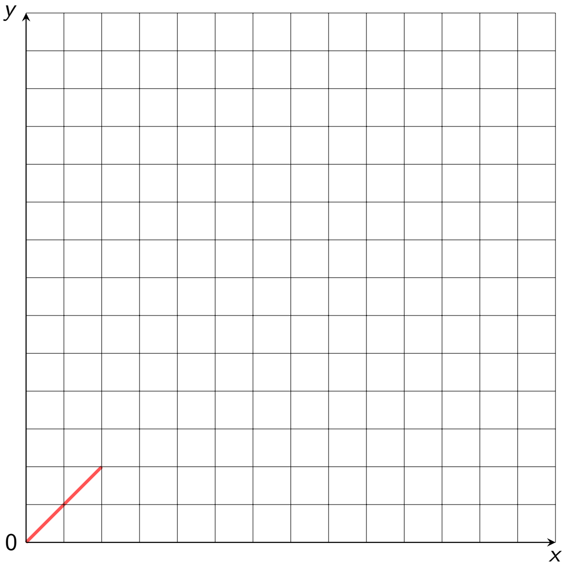
$$Z' = ((Z \wedge \delta)[R \leftarrow 0])^{\nearrow} \wedge Inv(l')$$

- We can thus compute a "zone automaton" also called simulation graph;
- It is made finite by converging either by equality or inclusion (of zones, for a given location):
- When computing (l', Z'), we loop on a previously computed (l, Z) when:
  - equality: $l' = l$ and $Z' = Z$ (Preserves the untimed language);
  - inclusion: $l' = l$ and $Z' \subseteq Z$ (Preserves reachability).

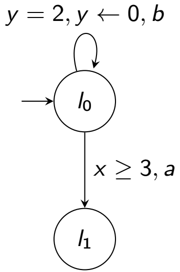
  These tests benefit from the canonical form of DBMs

Extrapolation

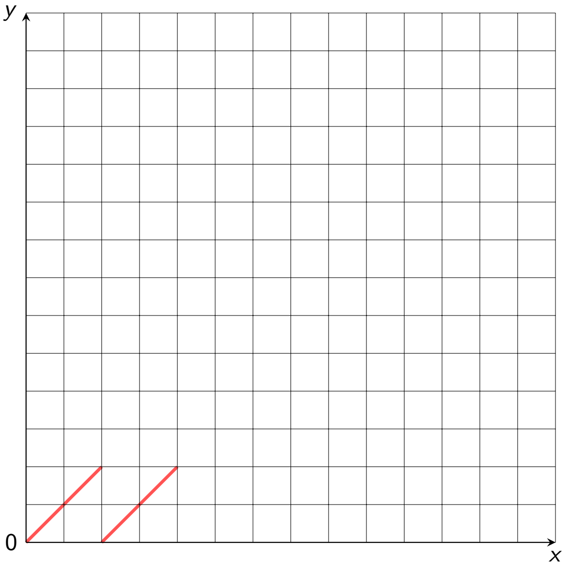




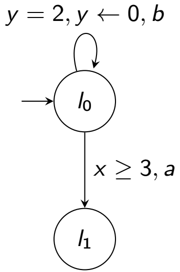
Extrapolation



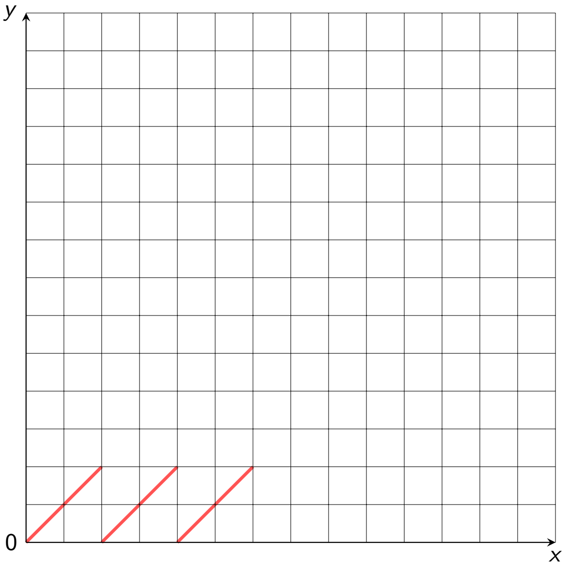
$\boxed{y \leq 2}$



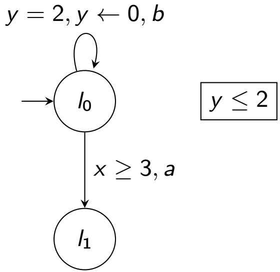
Extrapolation

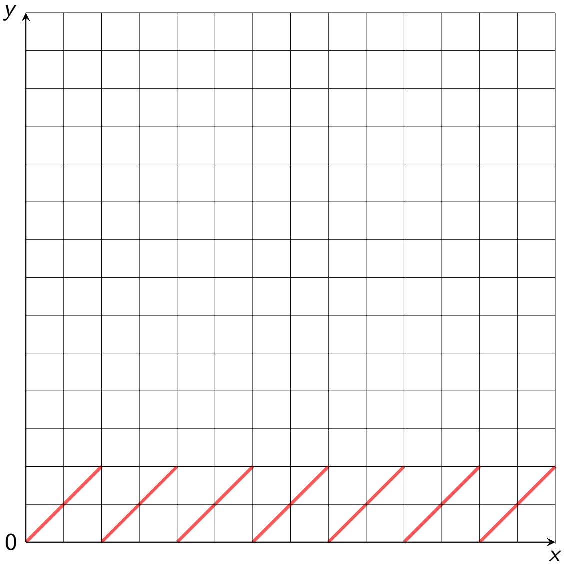


$\boxed{y \leq 2}$

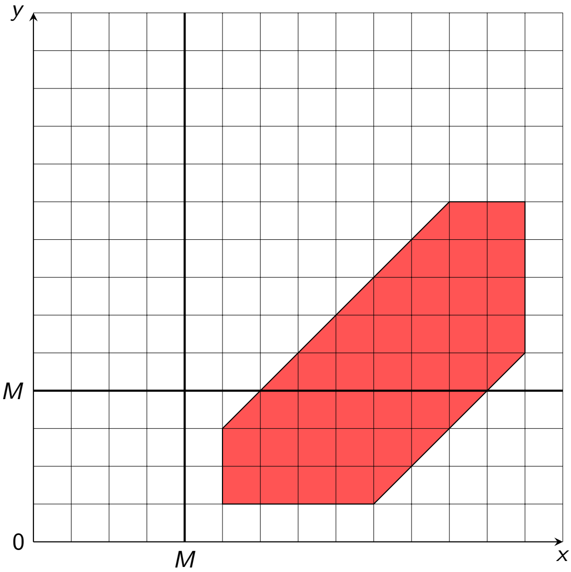


Extrapolation

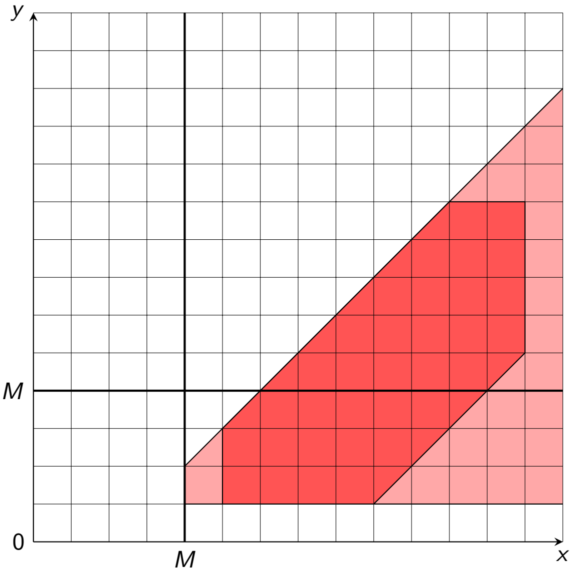




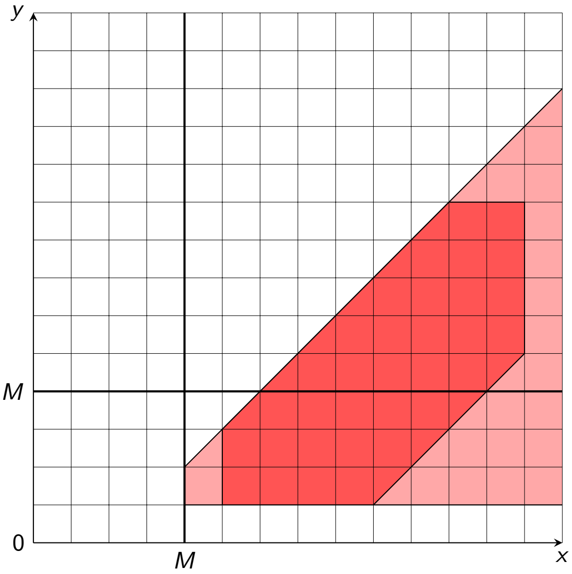
Extrapolation



Extrapolation

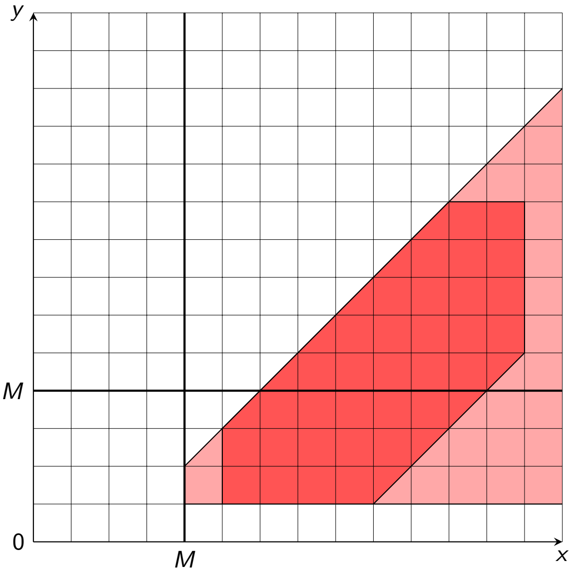


Extrapolation



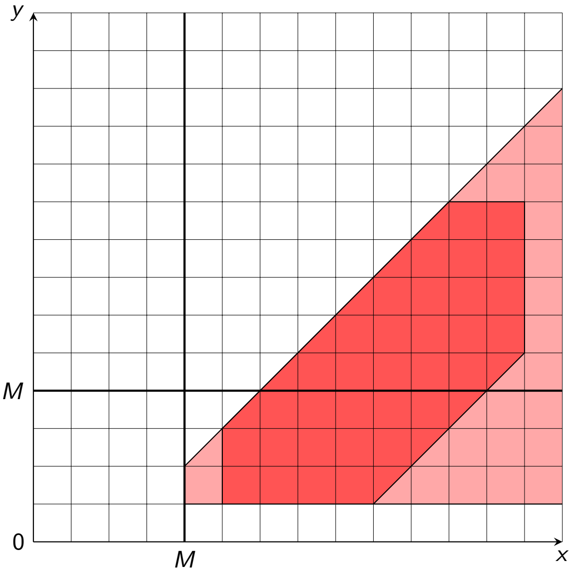
If $D(i,j) > M$ then
$D(i,j) \leftarrow$

Extrapolation



If $D(i,j) > M$ then $D(i,j) \leftarrow +\infty$

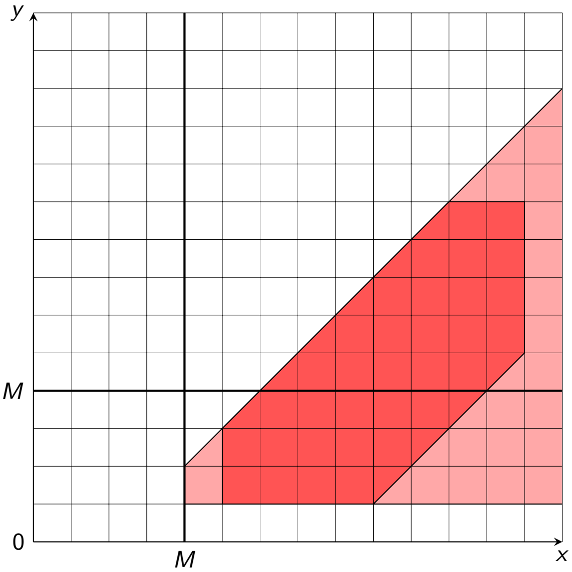
Extrapolation



If $D(i,j) > M$ then
$D(i,j) \leftarrow +\infty$

If $D(i,j) < -M$ then
$D(i,j) \leftarrow$

Extrapolation

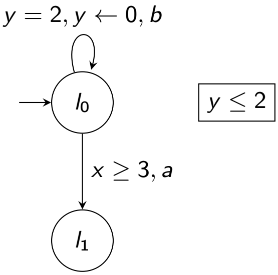


If $D(i,j) > M$ then $D(i,j) \leftarrow +\infty$

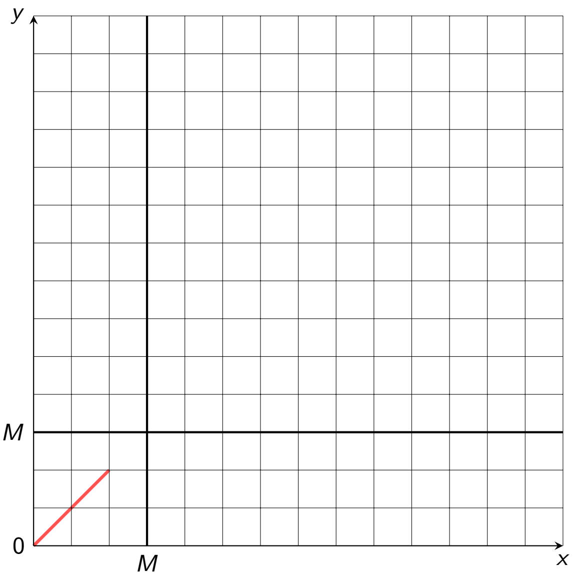
If $D(i,j) < -M$ then $D(i,j) \leftarrow -M$

Not necessarily "optimal": here both constraints could be removed.

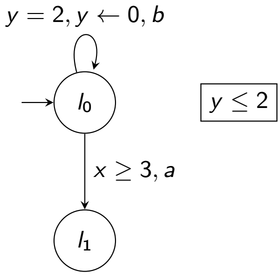
Extrapolation



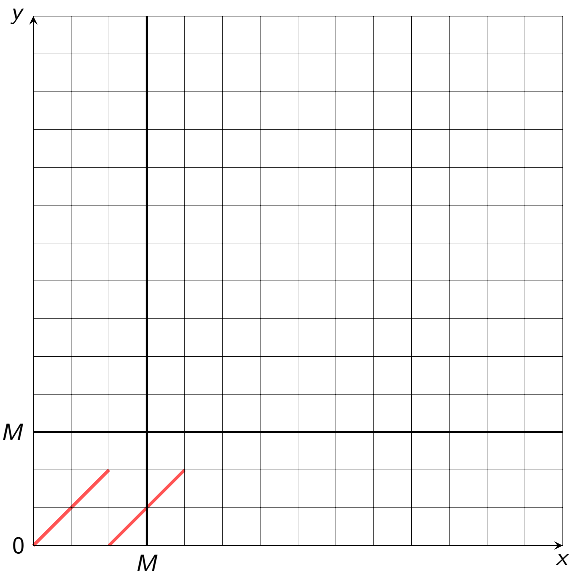
We have M = 3



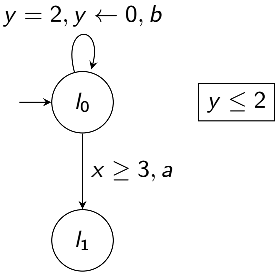
Extrapolation



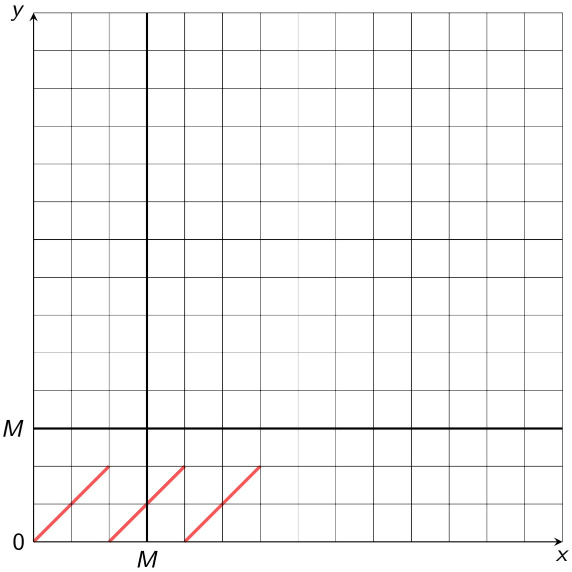
We have M = 3



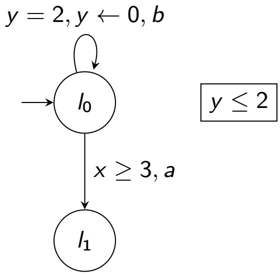
Extrapolation



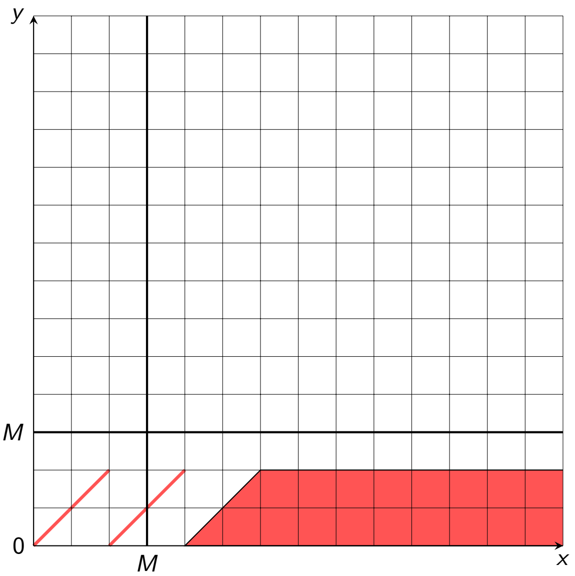
We have M = 3



Extrapolation



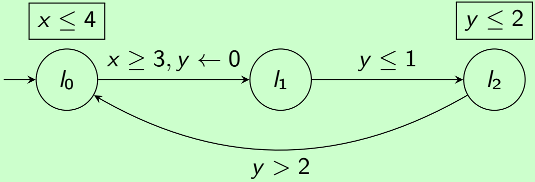
We have M = 3



Forward Reachability Using Zones

Exercise

Compute the simulation graph of the following TA:



On-the-fly TCTL Model-checking

- For a subset of TCTL we can write more efficient algorithms;

On-the-fly TCTL Model-checking

- For a subset of TCTL we can write more efficient algorithms;
- E.g, for EU and AU, we can devise simple on-the-fly algorithms:

On-the-fly TCTL Model-checking

- For a subset of TCTL we can write more efficient algorithms;
- E.g, for EU and AU, we can devise simple on-the-fly algorithms:
- We need only compute the simulation graph with an additional clock z for the timed Until:

Algorithm for $\mathbf{Ep U_{[a,b]} q}$

bool checkEU(l,Z):
  $passed \leftarrow passed \cup \{(l, Z)\}$
  if $(\min(Z_{|z}) > b)$
    return false
  else
    return $(q \in \ell(l)$ and $\max(Z_{|z}) \geq a)$
      or $(p \in \ell(l)$ and $\bigvee_{e=(l,\alpha,\delta,R,l')\in E}((l', next(Z, e)) \notin passed$
        ?checkEU$(l', next(Z, e))$
        :false$))$

Rajeev Alur, Costas Courcoubetis, and David Dill.
Model-checking in dense real-time.
Information and Computation, 104(1):2–34, 1993.

R. Alur and D. Dill.
A Theory of Timed Automata.
Theoretical Computer Science, 126(2):183–235, 1994.

Christel Baier and Joost-Pieter Katoen.
Principles of Model Checking.
MIT Press, 2008.

Orna Bernholtz, Moshe Y. Vardi, and Pierre Wolper.
An automata-theoretic approach to branching-time model checking (extended abstract).
In David L. Dill, editor, *6th Internation Conference on Computer Aided Verification (CAV '94)*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155, Stanford, California, USA, June 1994. Springer.

E. M. Clarke, E. A. Emerson, and A. P. Sistla.
Automatic verification of finite-state concurrent systems using temporal logic specifications.
ACM Transactions on Programming Languages and Systems, 8:244–263, 1986.

Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine.

Symbolic model checking for real-time systems.
Information and Computation, 111(2):193–244, 1994.

Robin Milner.
A Calculus of Communicating Systems, volume 92 of *Lecture Notes in Computer Science*.
Springer, 1980.

Amir Pnueli.
The temporal logic of programs.
18th Annual IEEE Symposium on Foundations of Computer Science, pages 46–57, 1977.

Michel Raynal.
Algorithmique du parallélisme : le problème de l'exclusion mutuelle.
Dunod, 1984.