Timeline-based Planning: Theory and Practice

Andrea Orlandini
ISTC-CNR, Rome, Italy

Nicola Gigante and Angelo Montanari
University of Udine, Italy

PhD Course
University of Udine
December 2-6, 2019

Expressiveness and Complexity

Timeline-based planning: theoretical foundations

We have seen how timeline-based planning has been successfully employed in the last decades.

Despite this, timelines have not been thoroughly studied,
until recently, from a theoretical perspective:

- computational complexity of timeline-based planning problems;
- expressiveness of modeling languages;
- comparison with PDDL-like action-based languages.

Timeline-based modeling languages

The many different systems come with different concrete modeling languages:

- DDL.3 (APSI-TRF)
- NDDL (EUROPA,PLATINUm)
- AML (ASPEN)

```
SYNCHRONIZE RoverController.rover {

  VALUE TakeSample(?loc, ?file) {

    cd0 NavController.nav.At(?loc0);
    cd1 InstrPos.inst_position.Unstowed();
    cd2 InstrOp.inst_operation.Sampling(?loc2);

    DURING [0, +INF] [0, +INF] cd0;
    CONTAINS [0, +INF] [0, +INF] cd1;
    CONTAINS [0, +INF] [0, +INF] cd2;

    cd2 DURING [0, +INF] [0, +INF] cd1;

    ?loc0 = ?loc;
    ?loc2 = ?loc;
  }
}
```

The recent ANML language includes both action-based and timeline-based elements (Smith et al. 2008).

Timeline-based modeling languages (2)

The language spectrum is too broad to cover them all here.

Instead, we consider the formal language described by Andrea in previous lectures.

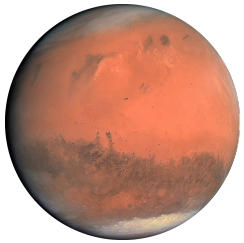
Cialdea Mayer et al. (2016)

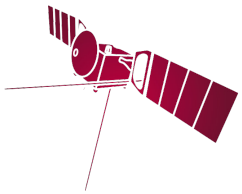
Marta Cialdea Mayer, Andrea Orlandini, and Alessandro Umbrico (2016). "Planning and Execution with Flexible Timelines: a Formal Account." In: *Acta Informatica* 53.6-8, pp. 649–680

Timelines

Running example

Mars orbiter



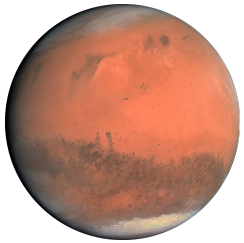


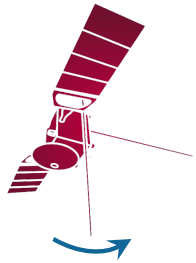
Toy example of a Mars orbiter doing scientific measurements:

1. Three “pointing modes”: Mars, Slewing, Earth
2. Four “activities”: Science, Communication, Maintenance, Idle
3. Temporal constraints:
   - Scientific measurements can be done only when pointing to Mars
   - Communication can happen:
     - only when pointing to Earth
     - only when a receiving ground station is visible
4. Goals:
   - Perform at least a given number of scientific measurements

Running example

Mars orbiter



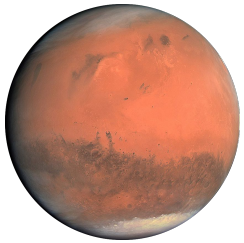


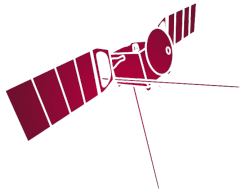
Toy example of a Mars orbiter doing scientific measurements:

1. Three “pointing modes”: Mars, Slewing, Earth
2. Four “activities”: Science, Communication, Maintenance, Idle
3. Temporal constraints:
   - Scientific measurements can be done only when pointing to Mars
   - Communication can happen:
     - only when pointing to Earth
     - only when a receiving ground station is visible
4. Goals:
   - Perform at least a given number of scientific measurements

Running example

Mars orbiter



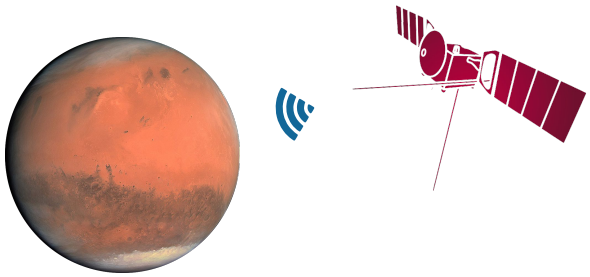


Toy example of a Mars orbiter doing scientific measurements:

1. Three “pointing modes”: Mars, Slewing, Earth
2. Four “activities”: Science, Communication, Maintenance, Idle
3. Temporal constraints:
   - Scientific measurements can be done only when pointing to Mars
   - Communication can happen:
     - only when pointing to Earth
     - only when a receiving ground station is visible
4. Goals:
   - Perform at least a given number of scientific measurements

Running example

Mars orbiter

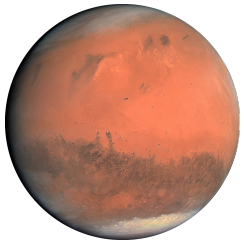


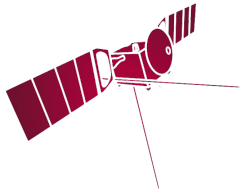
Toy example of a Mars orbiter doing scientific measurements:

1. Three “pointing modes”: Mars, Slewing, Earth
2. Four “activities”: Science, Communication, Maintenance, Idle
3. Temporal constraints:
   - Scientific measurements can be done only when pointing to Mars
   - Communication can happen:
     - only when pointing to Earth
     - only when a receiving ground station is visible
4. Goals:
   - Perform at least a given number of scientific measurements

Running example

Mars orbiter



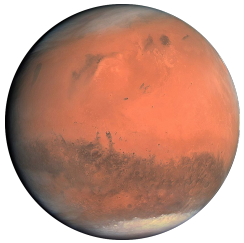


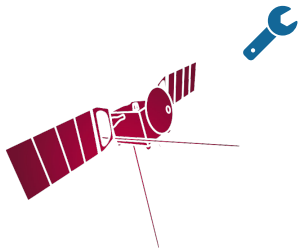
Toy example of a Mars orbiter doing scientific measurements:

1. Three “pointing modes”: Mars, Slewing, Earth
2. Four “activities”: Science, Communication, Maintenance, Idle
3. Temporal constraints:
   - Scientific measurements can be done only when pointing to Mars
   - Communication can happen:
     - only when pointing to Earth
     - only when a receiving ground station is visible
4. Goals:
   - Perform at least a given number of scientific measurements

Running example

Mars orbiter



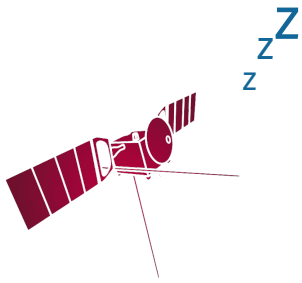


Toy example of a Mars orbiter doing scientific measurements:

1. Three “pointing modes”: Mars, Slewing, Earth
2. Four “activities”: Science, Communication, Maintenance, Idle
3. Temporal constraints:
   - Scientific measurements can be done only when pointing to Mars
   - Communication can happen:
     - only when pointing to Earth
     - only when a receiving ground station is visible
4. Goals:
   - Perform at least a given number of scientific measurements

Running example

Mars orbiter



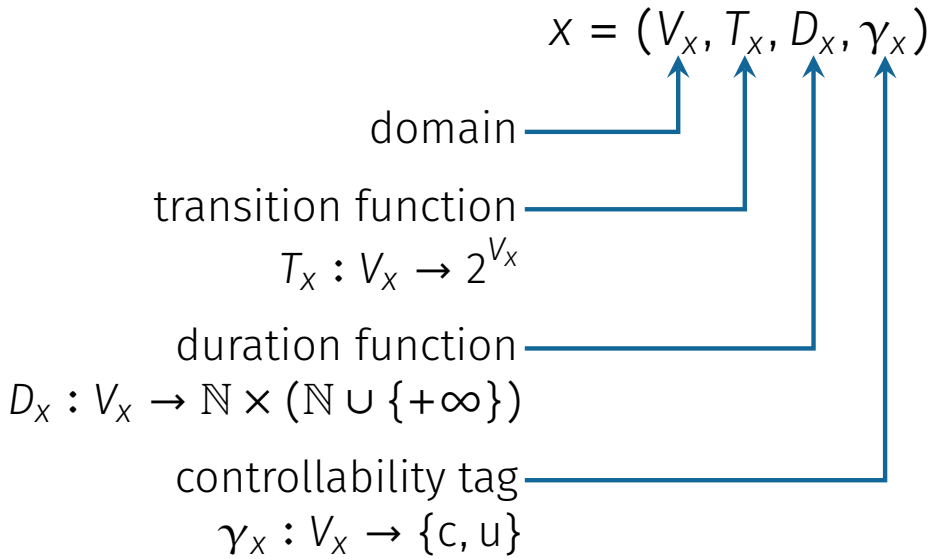
Toy example of a Mars orbiter doing scientific measurements:

1. Three “pointing modes”: Mars, Slewing, Earth
2. Four “activities”: Science, Communication, Maintenance, Idle
3. Temporal constraints:
   - Scientific measurements can be done only when pointing to Mars
   - Communication can happen:
     - only when pointing to Earth
     - only when a receiving ground station is visible
4. Goals:
   - Perform at least a given number of scientific measurements

Timeline-based planning problems

State variables

State variables represent the components of the system:

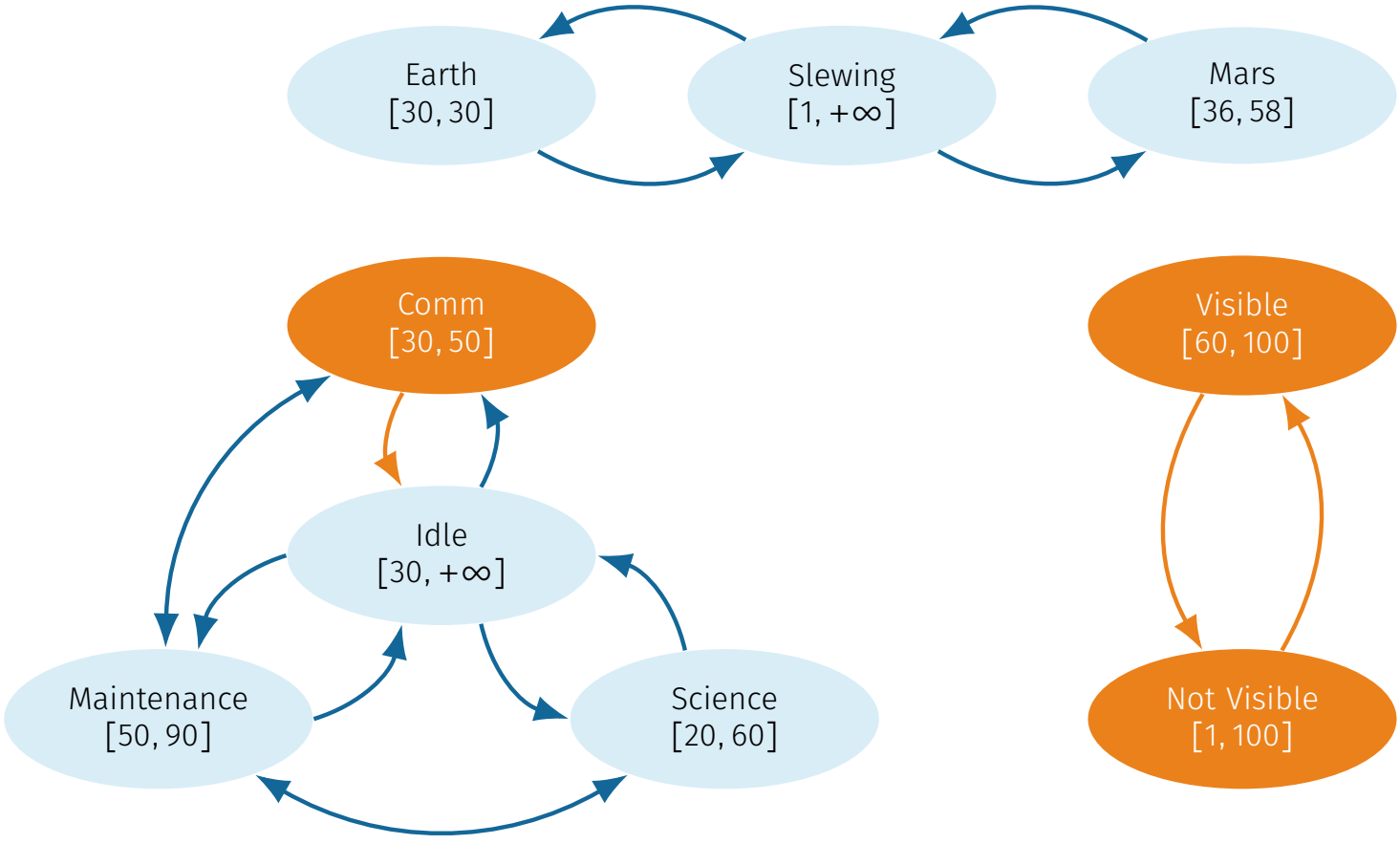


We assume a **discrete** temporal domain.

Timeline-based planning problems

State variables

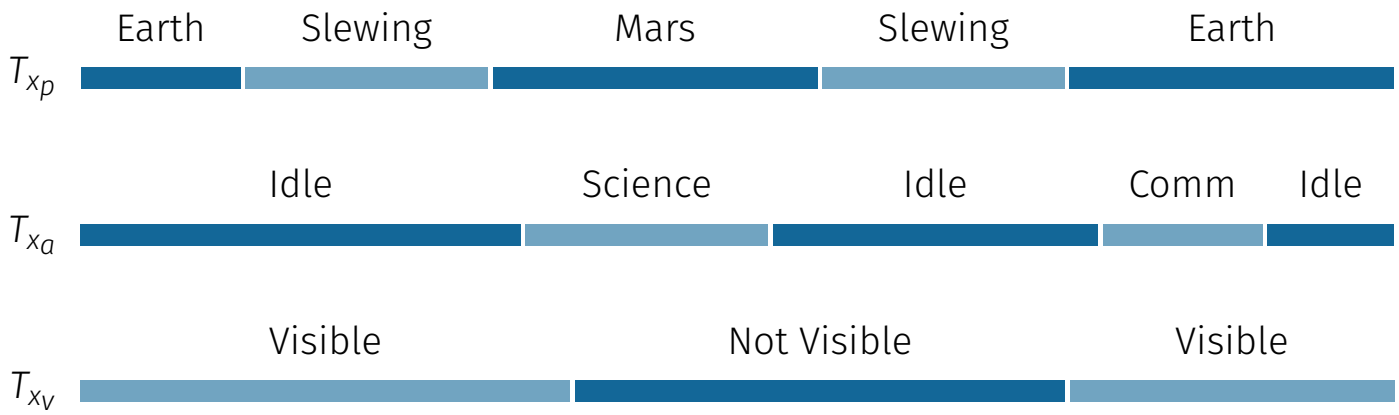
Three variables in our example, x_m (mode), x_a (activity), and x_v (visibility).



The tasks Communication, Visible, and Not Visible are uncontrollable.
The x_v variable is external.

Timeline-based planning problems

Timelines



Timelines are sequences of **tokens**;

- time intervals where the variable holds a single value
- transitions respect T_x, durations respect D_x

Timeline-based planning problems

Synchronisation rules

The interaction of the components is governed by the **synchronization rules**.

Example

Scientific measurements can be done only when pointing to Mars:

$$a[x_a = \mathit{Science}] \rightarrow \exists b[x_p = \mathit{Mars}] \,.\, \mathit{start}(b) \leq \mathit{start}(a) \wedge \mathit{end}(a) \leq \mathit{end}(b)$$

for all tokens a **where** $x_a = \mathit{Science}$,
there is a token b **where** $x_p = \mathit{Mars}$,
such that a **is contained in** b.

Timeline-based planning problems

Synchronisation rules

The interaction of the components is governed by the **synchronization rules**.

Example

Scientific measurements can be done only when pointing to Mars:

$$a[x_a = \mathit{Science}] \rightarrow \exists b[x_p = \mathit{Mars}] \,.\, \mathit{start}(b) \leq \mathit{start}(a) \wedge \mathit{end}(a) \leq \mathit{end}(b)$$

for all tokens a **where** $x_a = \mathit{Science}$,
there is a token b **where** $x_p = \mathit{Mars}$,
such that a **is contained in** b.

Timeline-based planning problems

Synchronisation rules

The interaction of the components is governed by the **synchronization rules**.

Example

Scientific measurements can be done only when pointing to Mars:

$$a[x_a = \mathit{Science}] \rightarrow \exists b[x_p = \mathit{Mars}] \,.\, \mathit{start}(b) \leq \mathit{start}(a) \wedge \mathit{end}(a) \leq \mathit{end}(b)$$

for all tokens a **where** $x_a = \mathit{Science}$,
there is a token b **where** $x_p = \mathit{Mars}$,
such that a **is contained in** b.

Timeline-based planning problems

Synchronisation rules

The interaction of the components is governed by the **synchronization rules**.

Example

Scientific measurements can be done only when pointing to Mars:

$$a[x_a = \mathit{Science}] \rightarrow \exists b[x_p = \mathit{Mars}] \,.\, \mathit{start}(b) \leq \mathit{start}(a) \wedge \mathit{end}(a) \leq \mathit{end}(b)$$

for all tokens a **where** $x_a = \mathit{Science}$,
there is a token b **where** $x_p = \mathit{Mars}$,
such that a is contained in b.

Syntax

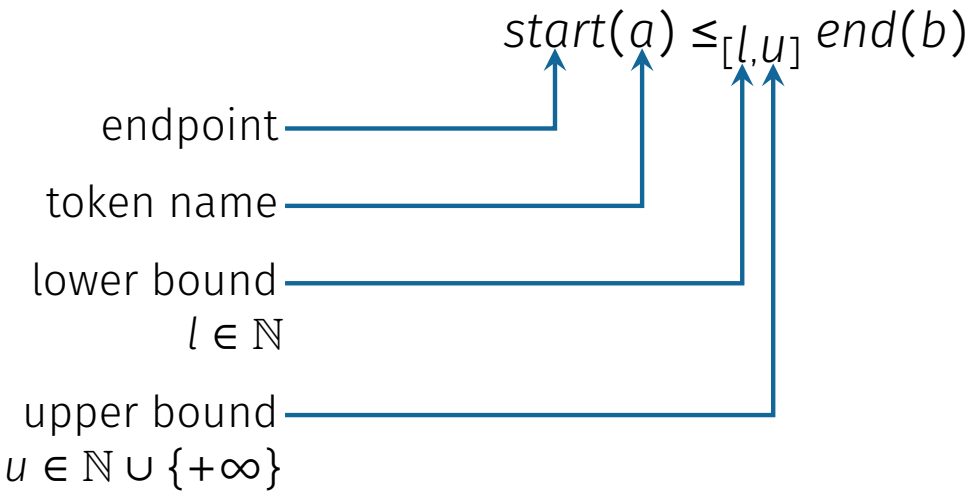
Each rule has a fixed structure:

$$a[x = u] \rightarrow \underbrace{\exists b[y = v] \,.\, \langle body \rangle}_{\text{existential statement}} \vee \exists c[z = w] d[k = r] \,.\, \langle body \rangle \vee \ldots$$

trigger

Syntax (2)

The body is made of a **conjunction** of atomic **temporal relations**:



Other examples

Comm. can only happen when the ground station is visible:

$$a[x_a = \mathit{Comm}] \rightarrow \exists b[x_v = \mathit{Visible}] \,.\, \mathit{start}(b) \leq \mathit{start}(a) \wedge \mathit{end}(a) \leq \mathit{end}(b)$$

The spacecraft can slew only during idle periods:

$$a[x_m = \mathit{Slewing}] \rightarrow \exists b[x_a = \mathit{Idle}] \,.\, \mathit{start}(b) \leq \mathit{start}(a) \wedge \mathit{end}(a) \leq \mathit{end}(b)$$

Maintenance has to happen at most 200 time steps after a measurement:

$$a[x_a = \mathit{Science}] \rightarrow \exists b[x_a = \mathit{Maintenance}] \,.\, \mathit{end}(a) \leq_{[0,200]} \mathit{start}(b)$$

Goal

At least four scientific measurements must be performed:

$$\top \rightarrow \exists b_1[x_a = \mathit{Science}]\, b_2[x_a = \mathit{Science}]\, b_3[x_a = \mathit{Science}]\, b_4[x_a = \mathit{Science}] \,.\, b_1 \neq b_2 \neq b_3 \neq b_4$$

Timeline-based planning problems

Without uncontrollable parts

Timeline-based planning problem

Given problem $P = (SV, S)$, where SV is a set of **state variables**, and S is a set of **rules** over SV, find a set of **timelines** over SV that satisfy the rules in S.

Timeline-based planning problem with bounded horizon

Given problem $P = (SV, S, H)$, where $P = (SV, S)$ is a timeline-based planning problem and $H \geq 0$ is the bound on the solution plan horizon.

A **solution plan** is a set of timelines, one for each $v \in SV$, that satisfy all the synchronization rules in S.

Expressiveness

Expressiveness

We started our study from the expressiveness side.

- How do these problems relate to the action-based counterparts?
- Which problems can be expressed with timelines vs. actions and vice versa?

Temporal planning

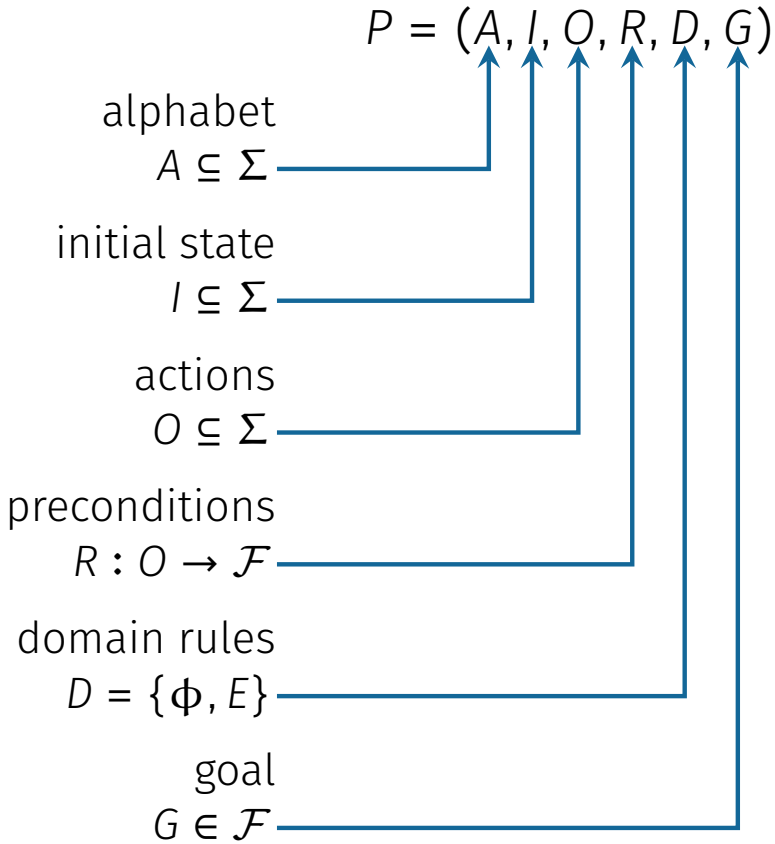
Which is the right action-based counterpart?
Temporal planning, in the form of PDDL with durative actions:

- each action has a duration, possibly dependent on some precondition;
- actions can be executed concurrently and overlap in time.

We did not compare directly to PDDL however, but used a stripped-down temporal planning language introduced by Rintanen (2007) for his computational complexity analysis of temporal planning.

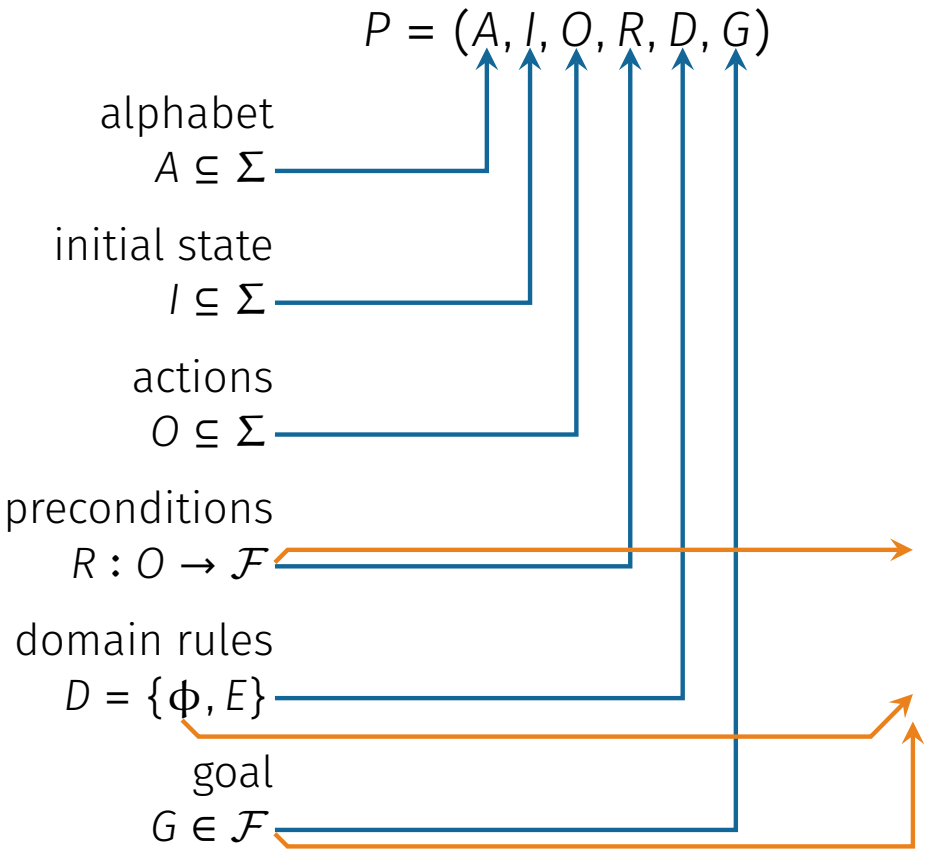
Temporal planning

A temporal planning problem over an alphabet Σ can be defined as:



Temporal planning

A temporal planning problem over an alphabet Σ can be defined as:



Formulas:

- boolean connectives
- $[i \ldots j]\Phi$ temporal operator
- $\langle i \ldots j \rangle \Phi$ temporal operator

Timelines vs. Actions

Assumptions

For this comparison to work, we need to make a specific set of assumptions:

- no uncertainty;
  - news on uncertainty in the second part of the tutorial;
- discrete time domain;
  - stay tuned for recent developments on dense time domains as well.

Hence the problem is:

> *Given a set of state variables SV and a set $\mathcal{S}$ of synchronization rules over SV, is there a set of timelines over SV satisfying $\mathcal{S}$?*

Can we compactly (polynomially) encode a temporal planning problem P into a set $\mathcal{S}$ of rules?

Timelines vs. Actions

Critical syntactic elements

Timeline-based planning is a rich formalism.
Many elements non-trivially affect expressiveness and complexity:

- the bound on the solution **horizon**, accepted as part of the input;
- accepting **unbounded** temporal relations, such as

$$\text{start}(a) \leq_{[0,+\infty]} \text{start}(b)$$

- accepting tokens of **unbounded** length, i.e. variables $x = (V, T, \gamma, D)$ where $D(v) = (0, +\infty)$ for some $v \in V$;
- constrained syntactic structure of the synchronization rules:
  - fixed $\forall\exists$ quantification scheme
  - only top-level disjunctions
  - no negation

$$a_0[x_0 = v_0] \rightarrow \exists a_1[x_1 = v_1] \ldots a_n[x_n = v_n] \,.\, \mathcal{E}_1 \vee \cdots \vee \mathcal{E}_m$$

Timelines vs. Actions

The result

PDDL is simpler. It turns out that timeline-based planning problems can fully **capture** temporal PDDL with:

- no bound on the solution **horizon**;
- only tokens of bounded length;
- only bounded relations:

$$\text{start}(a) <_{[0,+\infty]} \text{start}(b)$$

Note: these restrictions are quite artificial, but they give us the smallest (yet) expressive enough instance of the problem.

TIME 2016

Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer, and Andrea Orlandini (2016). "Timelines are Expressive Enough to Capture Action-Based Temporal Planning." In: *Proc. of the 23rd International Symposium on Temporal Representation and Reasoning*, pp. 100–109

Timelines vs. Actions

The encoding

The main impedance mismatch is the fixed syntactic structure of synchronization rules:

- preconditions in PDDL actions can be arbitrary boolean formulae
- the body of synchronization rules are fixed to disjunctions of conjuncted clauses

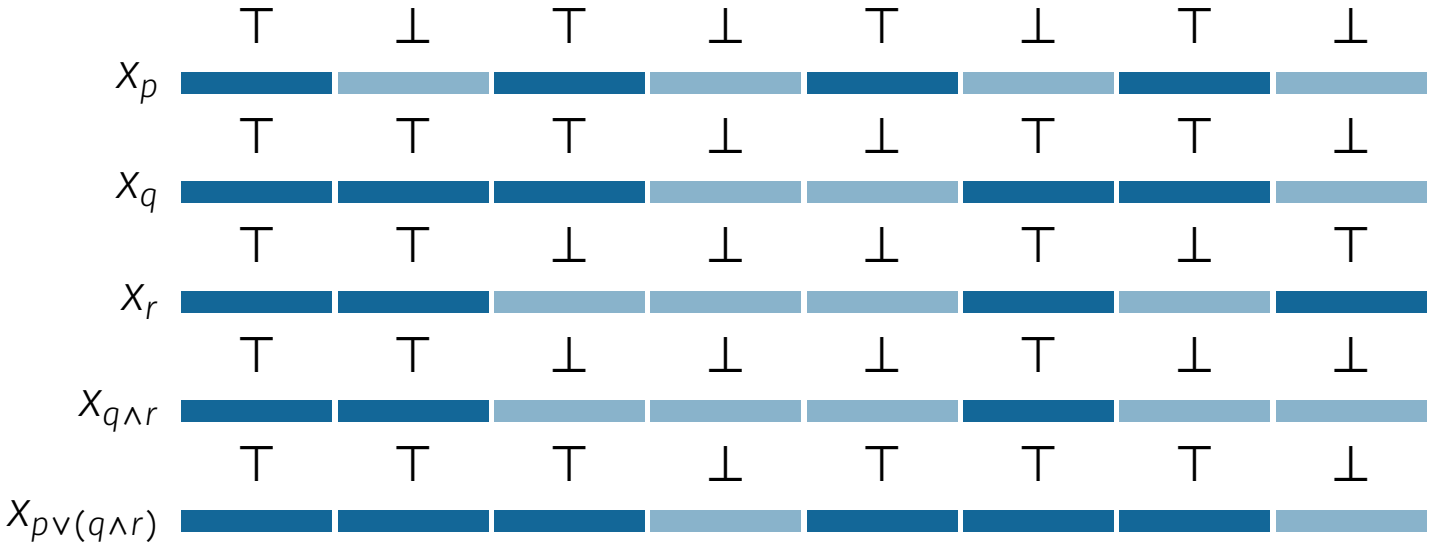
Solution

Have one state variable for each subformula of the precondition formulae

Timelines vs. Actions

Example

Suppose an action has a precondition of $p \vee (q \wedge r)$.
The encoded problem will include the following timelines.



Timelines vs. Actions

The temporal operator

How to encode formulas of the form $[i \ldots j]\phi$?

1. show that the form $[i]\phi$ is sufficient, by implementing a clock for each $[i \ldots j]\phi$ formula;
2. then, formulas $[i]\phi$ can be defined as synchronization rules:

$$a[x_{[i]\phi} = \top] \rightarrow \exists b[x_{\phi} = \top] \,.\, \text{start}(a) \leq_{[i,i]} \text{start}(b)$$

Complexity

Complexity of timeline-based planning

Computational complexity is another important theoretical property that we wanted to study.

- Finding a solution plan for a temporal PDDL problem is EXPSPACE-complete.
- How about finding solution plans for a timeline-based planning problem $P = (SV, S)$?

Complexity of timeline-based planning (2)

The expressiveness results already give us a lower bound:

- finding solution plans for timeline-based planning problems is EXPSPACE-hard.

But is there a matching upper bound?

Complexity

Admitting a bound on the **horizon** affects the complexity:

- if we admit **unbounded** relations, the problem is still **EXPSPACE**-complete;
- with bounds to the solution horizon the complexity drops: **NEXPTIME**-complete;

ICAPS 17

Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer, and Andrea Orlandini (2017). "Complexity of Timeline-based Planning." In: *Proc. of the 27th International Conference on Automated Planning and Scheduling*, pages 116–124

Complexity with unbounded horizon

The core of the complexity proof is a small-model result.

Theorem 1

If a problem P has a solution, then a solution exists of length at most doubly exponential in the size of P.

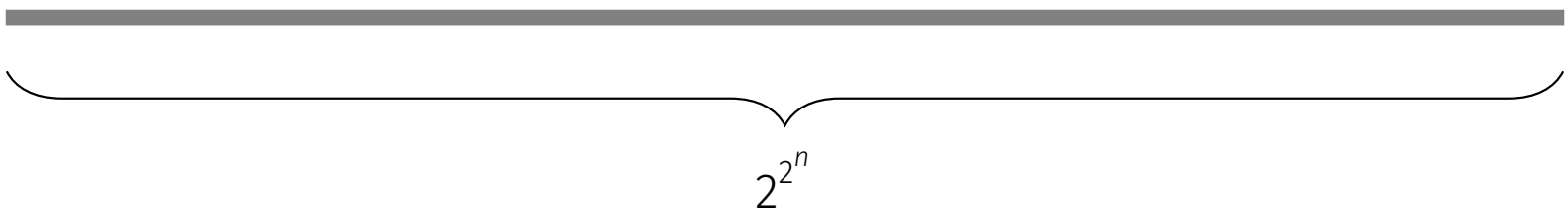
Then, a nondeterministic procedure running in exponential space is used to find the solution.

Complexity with unbounded horizon

The nondeterministic decision procedure works as follows.

Complexity with unbounded horizon

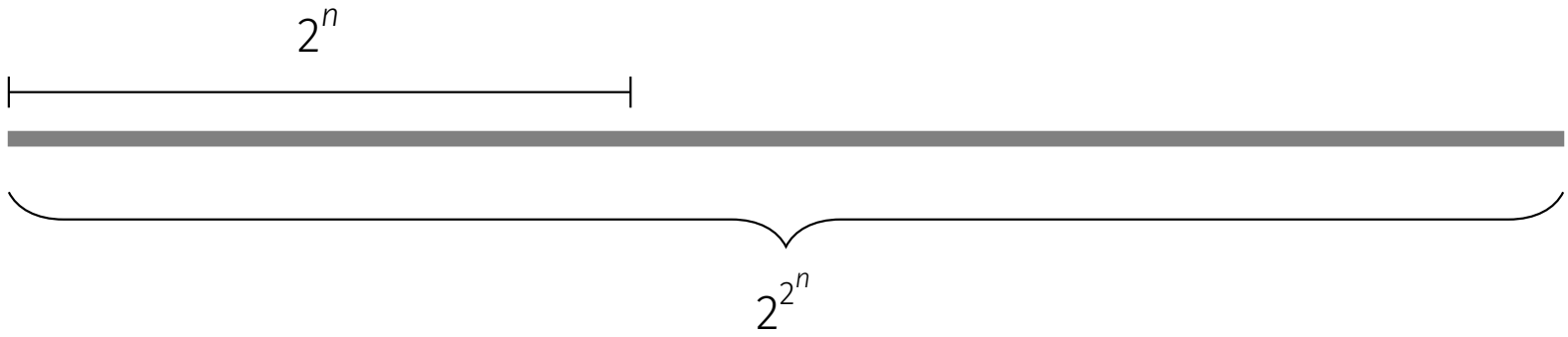
The nondeterministic decision procedure works as follows.



- a solution of length 2^{2^n} is nondeterministically guessed,

Complexity with unbounded horizon

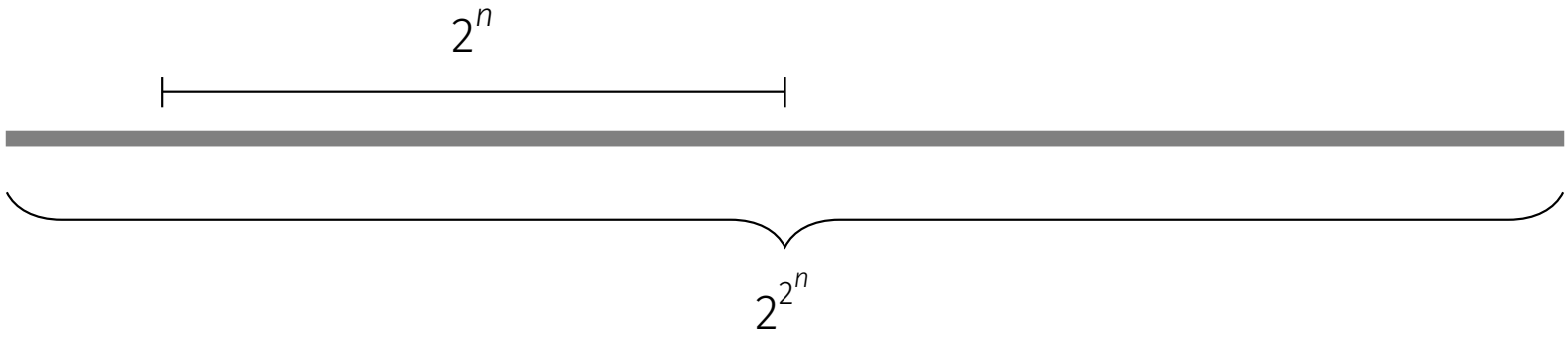
The nondeterministic decision procedure works as follows.



- a solution of length 2^{2^n} is nondeterministically guessed,
- the solution is checked by sliding an exponential window over the solution,

Complexity with unbounded horizon

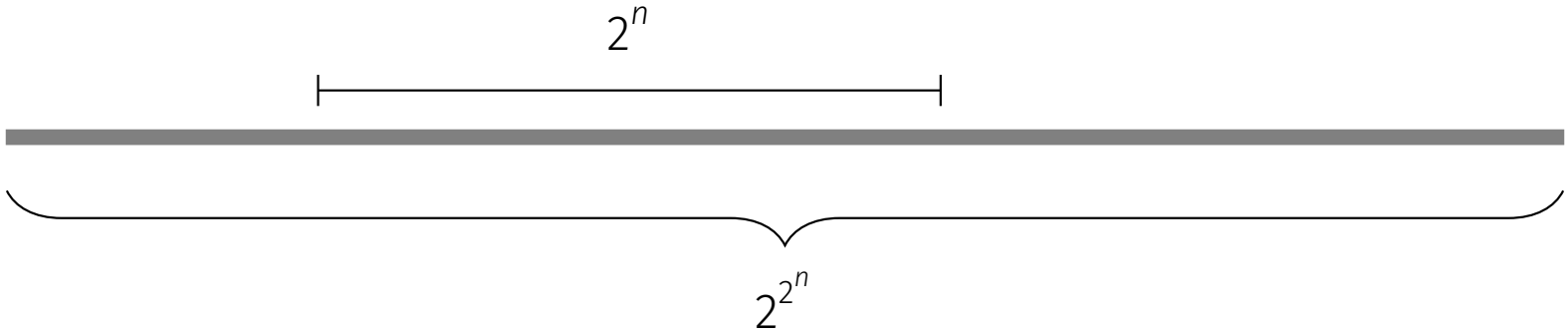
The nondeterministic decision procedure works as follows.



- a solution of length 2^{2^n} is nondeterministically guessed,
- the solution is checked by sliding an exponential window over the solution,

Complexity with unbounded horizon

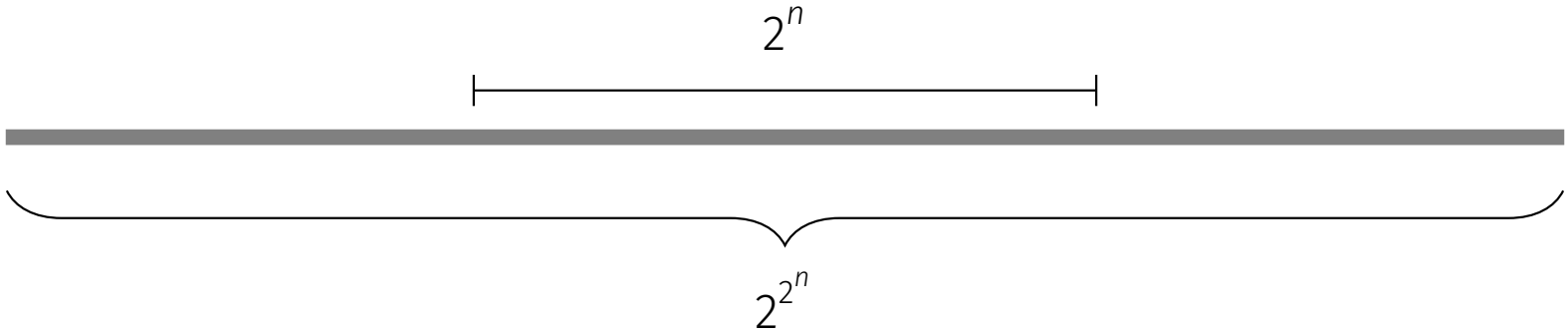
The nondeterministic decision procedure works as follows.



- a solution of length 2^{2^n} is nondeterministically guessed,
- the solution is checked by sliding an exponential window over the solution,

Complexity with unbounded horizon

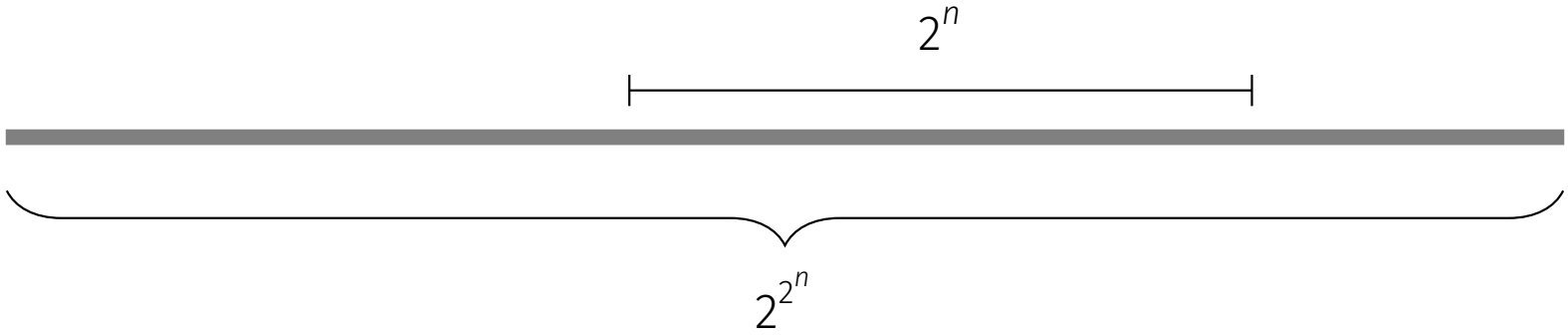
The nondeterministic decision procedure works as follows.



- a solution of length 2^{2^n} is nondeterministically guessed,
- the solution is checked by sliding an exponential window over the solution,

Complexity with unbounded horizon

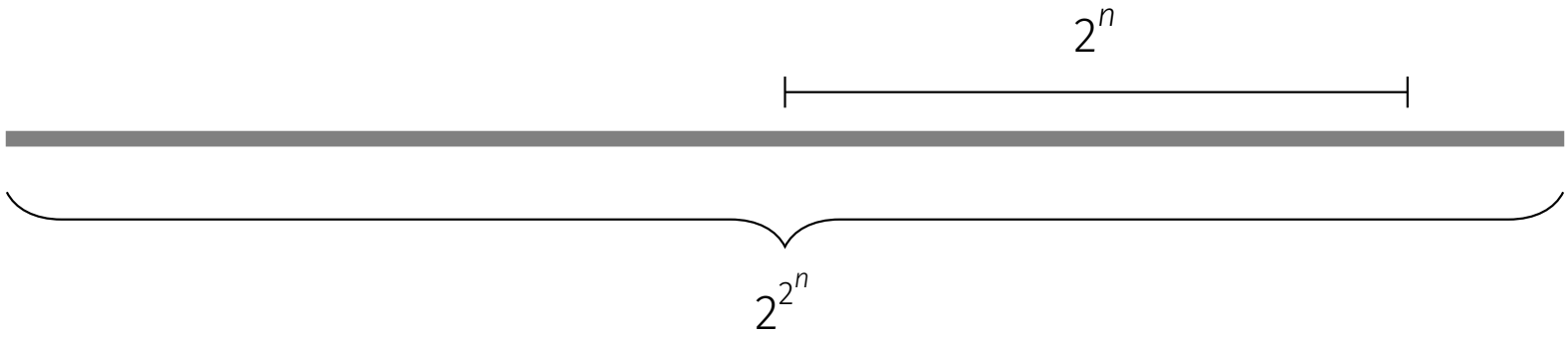
The nondeterministic decision procedure works as follows.



- a solution of length 2^{2^n} is nondeterministically guessed,
- the solution is checked by sliding an exponential window over the solution,

Complexity with unbounded horizon

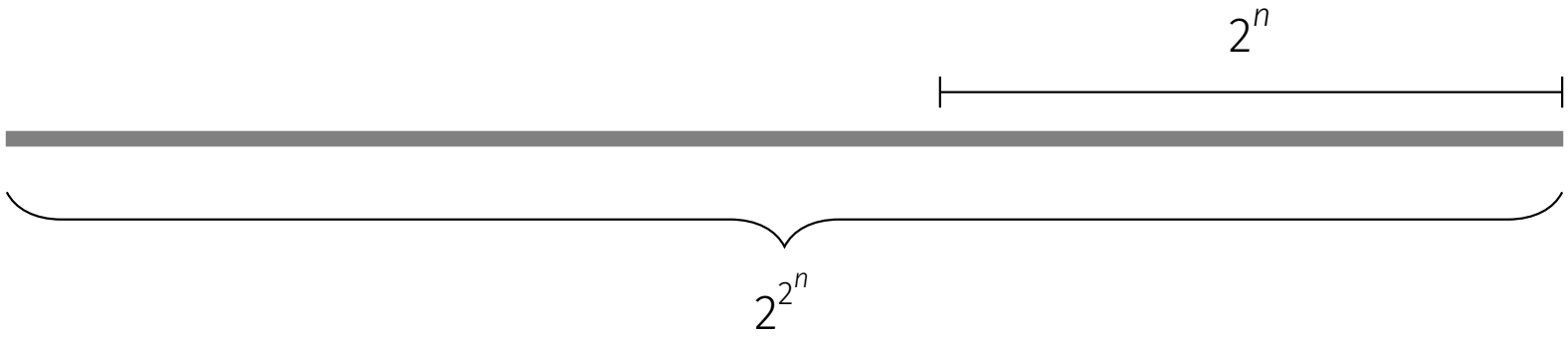
The nondeterministic decision procedure works as follows.



- a solution of length 2^{2^n} is nondeterministically guessed,
- the solution is checked by sliding an exponential window over the solution,

Complexity with unbounded horizon

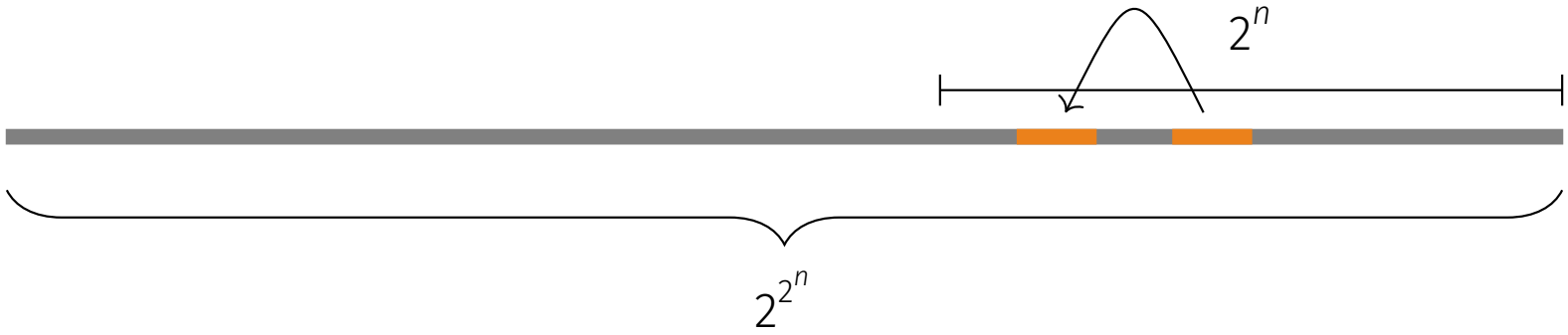
The nondeterministic decision procedure works as follows.



- a solution of length 2^{2^n} is nondeterministically guessed,
- the solution is checked by sliding an exponential window over the solution,

Complexity with unbounded horizon

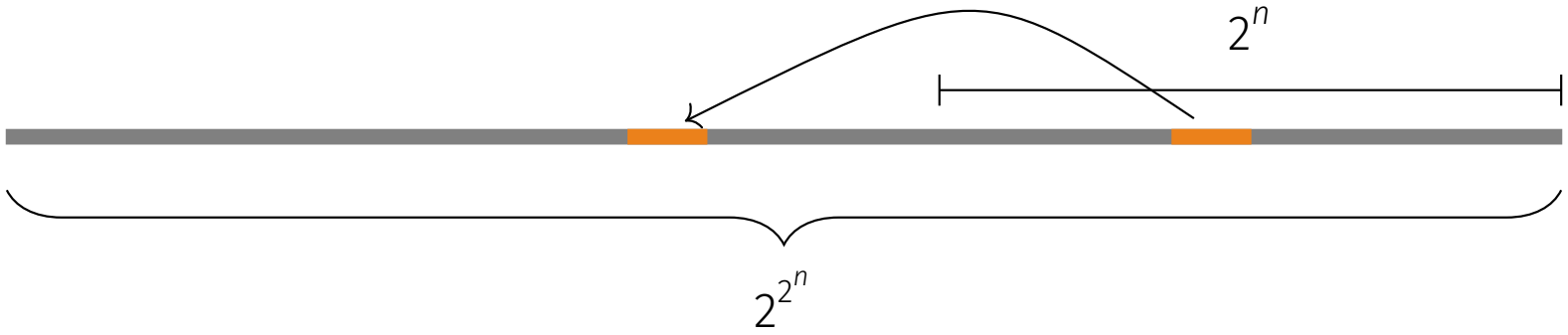
The nondeterministic decision procedure works as follows.



- checking the constraints **inside** of a single window can be done in **nondeterministic exponential time**,

Complexity with unbounded horizon

The nondeterministic decision procedure works as follows.



- checking the constraints **inside** of a single window can be done in **nondeterministic exponential time**,
- outside of the window, the whole history can be represented compactly with **exponential** space
  (Theorem 1 follows)

Decomposition of synchronization rules

Let us take a deeper look at the core result.

Theorem 1

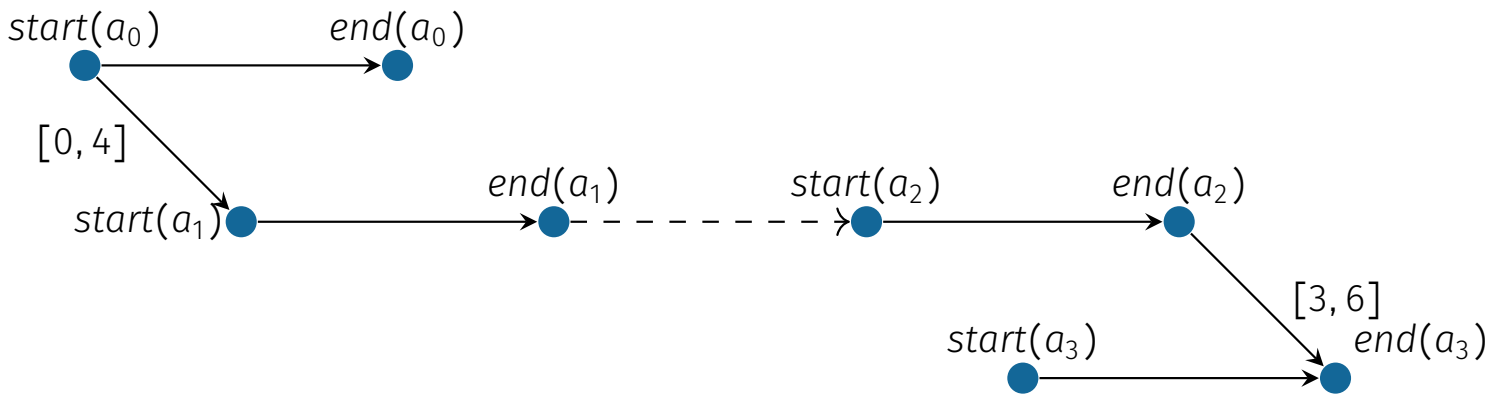
If a problem P has a solution, then a solution exists of length at most **doubly exponential** in the size of P.

Decomposition of synchronization rules (2)

The result comes from the following observation.
Consider a rule:

$$a_0[x_0 = v_0] \rightarrow \exists a_1[x_1 = v_1]a_2[x_2 = v_2]a_3[x_3 = v_3]\ .$$
$$\text{start}(a_0) \leq_{[0,4]} \text{start}(a_1) \wedge \text{end}(a_1) \leq_{[0,+\infty]} \text{start}(a_2)$$
$$\wedge\ \text{end}(a_2) \leq_{[3,6]} \text{end}(a_3)$$

We can see the rule as a graph:

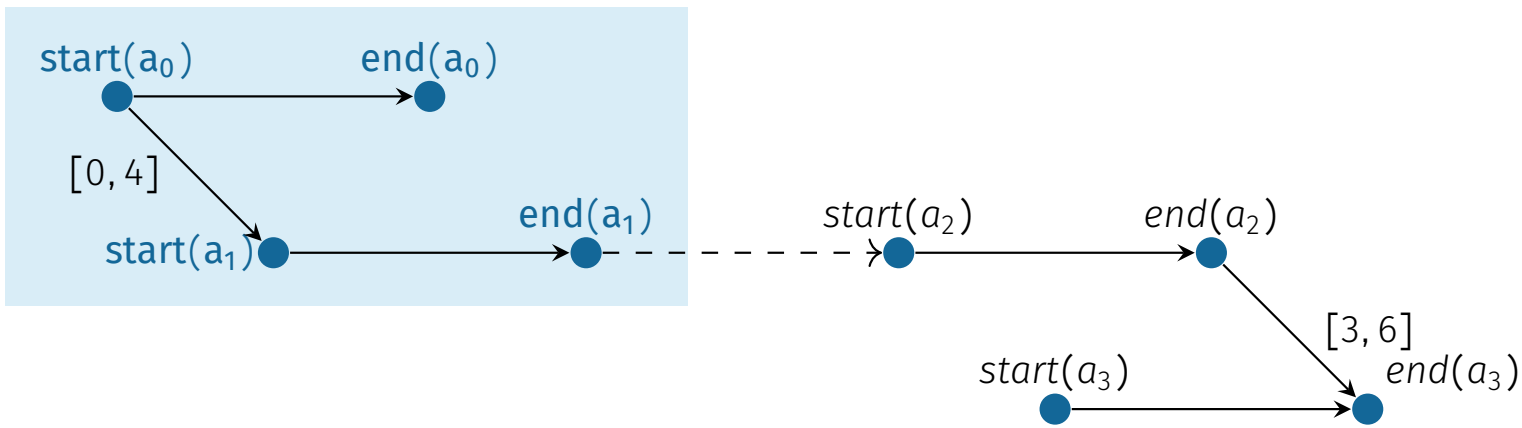


Decomposition of synchronization rules (2)

The result comes from the following observation.
Consider a rule:

$$a_0[x_0 = v_0] \rightarrow \exists a_1[x_1 = v_1]a_2[x_2 = v_2]a_3[x_3 = v_3]\ .$$
$$\text{start}(a_0) \leq_{[0,4]} \text{start}(a_1) \wedge \text{end}(a_1) \leq_{[0,+\infty]} \text{start}(a_2)$$
$$\wedge\ \text{end}(a_2) \leq_{[3,6]} \text{end}(a_3)$$

We can see the rule as a graph:

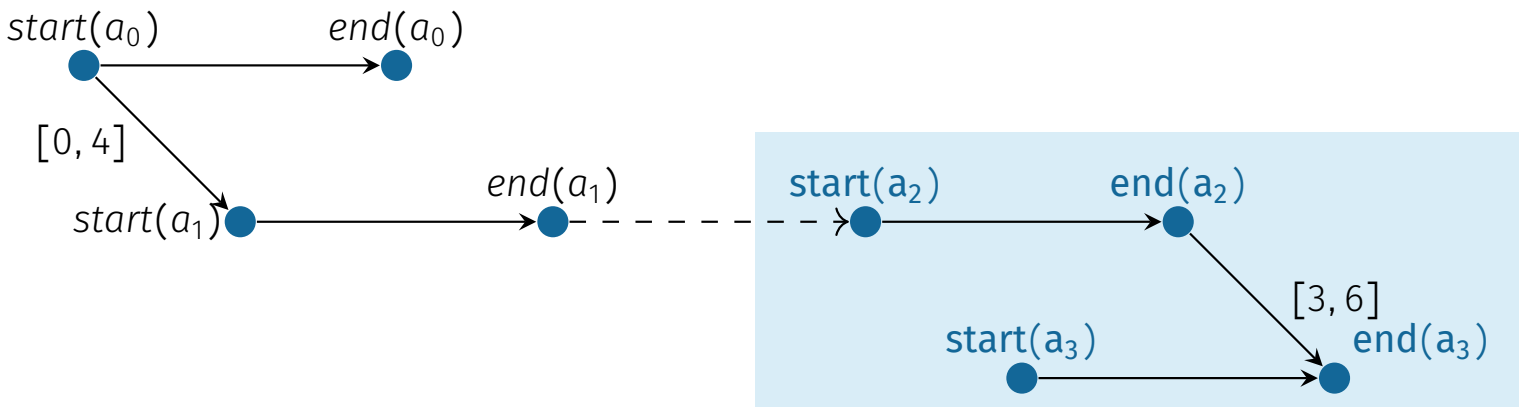


Decomposition of synchronization rules (2)

The result comes from the following observation.
Consider a rule:

$$a_0[x_0 = v_0] \rightarrow \exists a_1[x_1 = v_1]a_2[x_2 = v_2]a_3[x_3 = v_3]\ .$$
$$\text{start}(a_0) \leq_{[0,4]} \text{start}(a_1) \wedge \text{end}(a_1) \leq_{[0,+\infty]} \text{start}(a_2)$$
$$\wedge\ \text{end}(a_2) \leq_{[3,6]} \text{end}(a_3)$$

We can see the rule as a graph:



Rule graphs

Rule graphs give a graph-theoretic representation of rules:

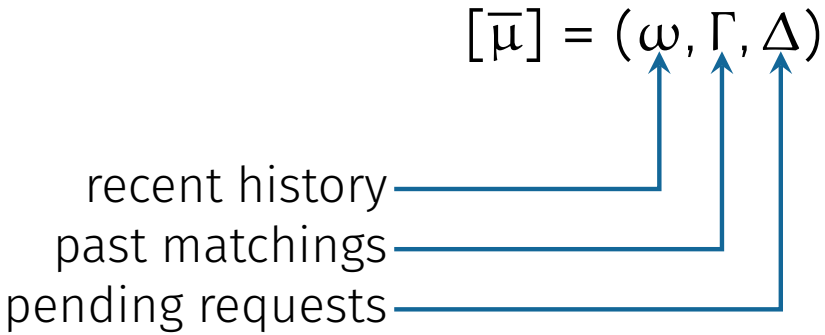
- bounded components identify events that happen “close” to each other
- events belonging to a single component cannot appear further that a certain exponential distance in any solution
- then, we have to keep track of the satisfaction of unbounded relations only between whole components.

Matching records

Matching records use rule graphs to compactly represent all the important information about a plan.

Plans Π are flattened into **event sequences** $\overline{\mu}$.

Given an **event sequence** $\overline{\mu}$ representing a plan π:



Matching records

Properties

Important properties of matching records:

1. the size of $[\overline{\mu}]$ is **exponential** in the size of the problem
2. given $[\overline{\mu}]$, we can decide in **exponential** time whether $\overline{\mu}$ represent a solution plan for the problem
3. from $[\overline{\mu}]$ and an event μ, we can build $[\overline{\mu}\mu]$ in **exponential** time

Hence the small model result:

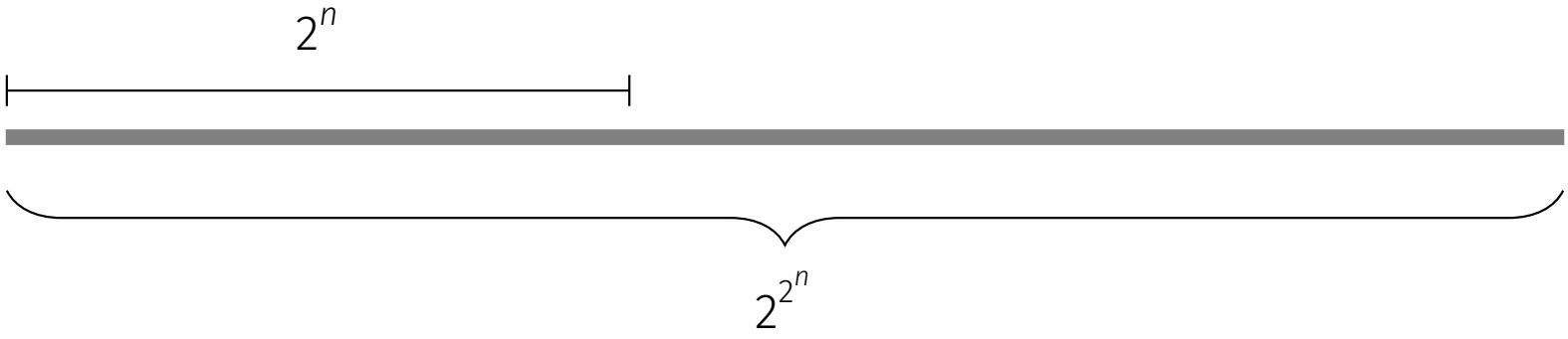
- if a plan is represented by more than a **doubly exponential** amount of events, two matching records must repeat;
- then, a shorter plan is obtained by cutting between the two repetitions.

Complexity with unbounded horizon

The complexity of the problem follows:

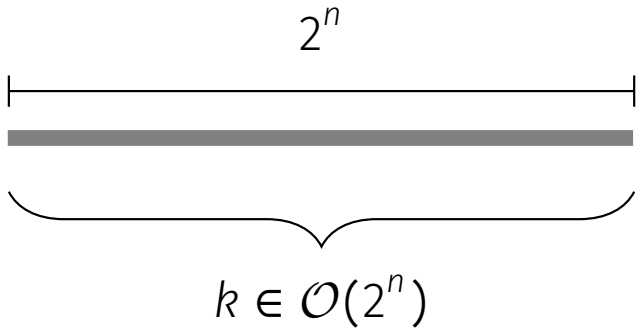
- since EXPSPACE=NEXPSPACE, the entire procedure runs in exponential space;
- as already pointed out, EXPSPACE-hardness comes from the expressiveness results;
- hence the problem is EXPSPACE-complete.

Complexity with bounded horizon



With a bounded horizon, Theorem 1 still holds.

Complexity with bounded horizon



With a bounded horizon, Theorem 1 still holds.

- However, the given horizon bound is only **exponential** in the size of the input.
- Hence exactly the same algorithm runs in **nondeterministic exponential time**, hence the problem belongs to **NEXPTIME**;
- NEXPTIME-hardness proved by a reduction from the **Exponentially Bounded Corridor Tiling** problem.

Timeline-based planning over dense temporal domains

Timeline-based planning over dense temporal domains

Until now we only considered discrete time.

What happens if we consider a dense time model?

Work is being done in this sense. The emerging picture is as follows:

- the general problem is undecidable even with a single state variable
  - by reduction from the halting problem for Minsky two-counter machines;
- decidable but non primitive recursive if:
  - rules only talk about the future w.r.t. the trigger; and
  - non-trigger tokens appear at most once in the body of the rule
- EXPSPACE-complete if, in addition, singular intervals are forbidden;
- PSPACE-complete if we admit only intervals of the form $[0, a]$ and $[b, +\infty]$;
- NP-complete if only trigger-less rules are admitted.

Timeline-based planning over dense temporal domains

Details

Bozzelli, Molinari, Montanari, and Peron (2018b)

Laura Bozzelli, Alberto Molinari, Angelo Montanari, and Adriano Peron (2018b). "Decidability and Complexity of Timeline-Based Planning over Dense Temporal Domains." In: *KR 2018*. Ed. by Michael Thielscher, Francesca Toni, and Frank Wolter. AAAI Press, pp. 627–628. ISBN: 978-1-57735-803-9. URL: https://aaai.org/ocs/index.php/KR/KR18/paper/view/17995

Bozzelli, Molinari, Montanari, Peron, and Woeginger (2018)

Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron, and Gerhard J. Woeginger (2018). "Timeline-Based Planning over Dense Temporal Domains with Trigger-less Rules is NP-Complete." In: *ICTCS 2018*. Ed. by Alessandro Aldini and Marco Bernardo. Vol. 2243. CEUR Workshop Proceedings. CEUR-WS.org, pp. 116–127. URL: http://ceur-ws.org/Vol-2243/paper11.pdf

Bozzelli, Molinari, Montanari, and Peron (2018a)

Laura Bozzelli, Alberto Molinari, Angelo Montanari, and Adriano Peron (2018a). "Complexity of Timeline-Based Planning over Dense Temporal Domains: Exploring the Middle Ground." In: *Proceedings Ninth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2018, Saarbrücken, Germany, 26-28th September 2018*. Ed. by Andrea Orlandini and Martin Zimmermann. Vol. 277. EPTCS, pp. 191–205. DOI: 10.4204/EPTCS.277.14. URL: https://doi.org/10.4204/EPTCS.277.14

Automata-theoretic perspective

Distinctive features of timeline-based planning: a short recap

Four main parameters of the timeline-based planning problem:

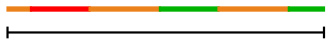
- **token duration**:

bounded	the duration of all tokens is bounded, even when it is not explicitly constrained by synchronization rules
unbounded	the duration of some token may be arbitrarily long ($+\infty$ is taken as the maximum duration of the token)

- **temporal relations**:

bounded	any synchronization rule imposes a bound on the maximum distance between the considered token's endpoints
unbounded	some synchronization rule may allow the maximum distance between the considered token's endpoints to be arbitrarily long

- **horizon**:

bounded	 planning problems commonly specify a bound on the size of the solution plan
unbounded	there is no bound on the size of the solution plan (if any)

- **plan**:

finite	the solution plan is finite
recurrent	the solution plan is infinite

Remark. Note that, by means of synchronization rules, we may force solution plans to be infinite (trigger-less rules: $\exists$; triggered rules: $\forall\exists$).

An automata-theoretic approach to timeline-based planning

We now show how to reduce the finite (resp., recurrent) planning problem with unbounded temporal relations, unbounded token duration, and unbounded horizon to the **emptiness problem** for non-deterministic finite automata (NFA) (resp., non-deterministic Büchi automata (NBA)).

The reduction consists of **two main steps**:

- we first **build** an NFA (resp., NBA) and show that it recognizes exactly those finite (resp., infinite) words that represent solution plans for the given planning problem;
- then, we **check** the NFA (resp., NBA) for non-emptiness by solving a suitable reachability problem.

As it happens with the finite one, the recurrent planning problem is EXPSPACE-complete.

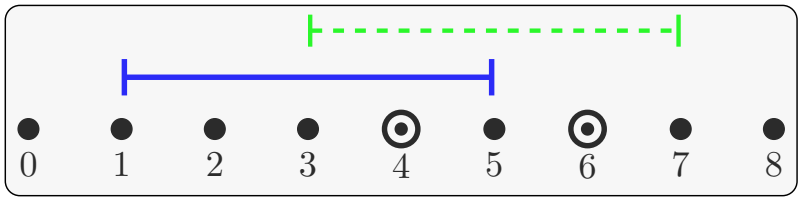
For the sake of simplicity, we restrict our attention to the finite case.

Della Monica, Gigante, Montanari, and Sala (2018)

Dario Della Monica, Nicola Gigante, Angelo Montanari, and Pietro Sala (2018). "A novel automata-theoretic approach to timeline-based planning." In: *KR 2018*

The basic building blocks: blueprints

A **blueprint** is a possible instantiation of a **synchronization rule**, that is, a possible way of satisfying it.



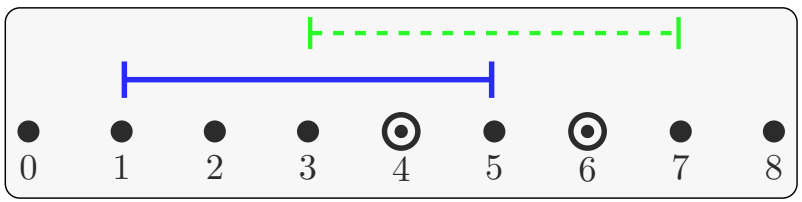
$\models a[y = \bullet] \rightarrow \exists b[x = \bullet].start(a) \leq_{[0,3]} start(b)$

We may have **more than one** instantiation for a single rule:

- different temporal relations between pairs of tokens (the blue token **overlaps** the green one, or the blue token is a prefix of the green one, or the green token is a prefix of the blue one, or . . .);
- different values for the temporal distance between token's endpoints (the green token starts **2 time units after** the blue one, or the blue and the green tokens start at the same time, or . . .).

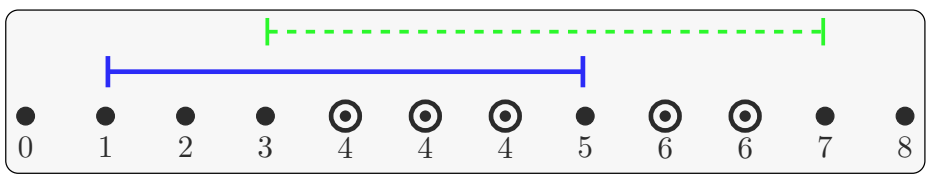
The basic building blocks: blueprints

How do we deal with **unbounded temporal relations** and **tokens with unbounded duration**? In principle, we may have **infinitely many** different instatiations of a given synchronization rule.



$$\models a[y = \bullet] \rightarrow \exists b[x = \bullet].start(a) \leq_{[0,3]} start(b)$$

Blueprints can be equipped with some special points that allow one to "stretch" them. In such a case, a blueprint represents a (possibly infinite) set of models that are equivalent modulo arbitrary finite replications of their "**stretching points**".



$$\models a[y = \bullet] \rightarrow \exists b[x = \bullet].start(a) \leq_{[0,3]} start(b)$$

The basic building blocks: blueprints

To summarize, for any synchronization rule, we have that:

- finitely many different temporal relations may hold between pairs of tokens;
- finitely many different values can be assigned to temporal distances in bounded temporal relations;
- finitely many different values can be assigned to tokens with bounded duration;
- unbounded temporal relations and tokens with unbounded duration are compactly represented, that is, finitely described, by stretching points.

Conclusion

A **finite set** of blueprints (each blueprint being a **finite object**) suffices to capture all possible different models of a synchronization rule.

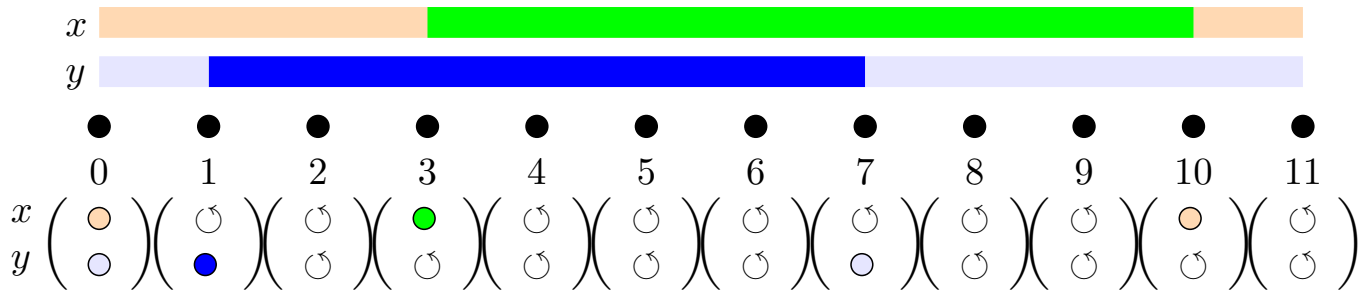
Timelines as words

Timelines as words

n timelines $T_{x_1}, \ldots, T_{x_n}$ are encoded as words on a finite alphabet of n-tuples:

$$\Sigma = (\mathcal{D}(x_1) \cup \{\circlearrowleft\}) \times \ldots \times (\mathcal{D}(x_n) \cup \{\circlearrowleft\})$$

where $\circlearrowleft$ is a fresh special symbol that labels any point which is neither the starting point nor the ending point of a token.



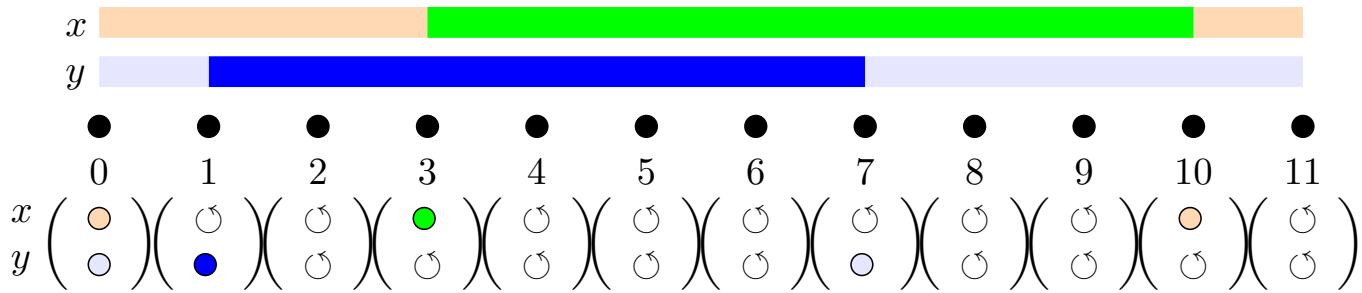
Timelines as words

Timelines as words

n timelines $T_{x_1}, \ldots, T_{x_n}$ are encoded as words on a finite alphabet of n-tuples:

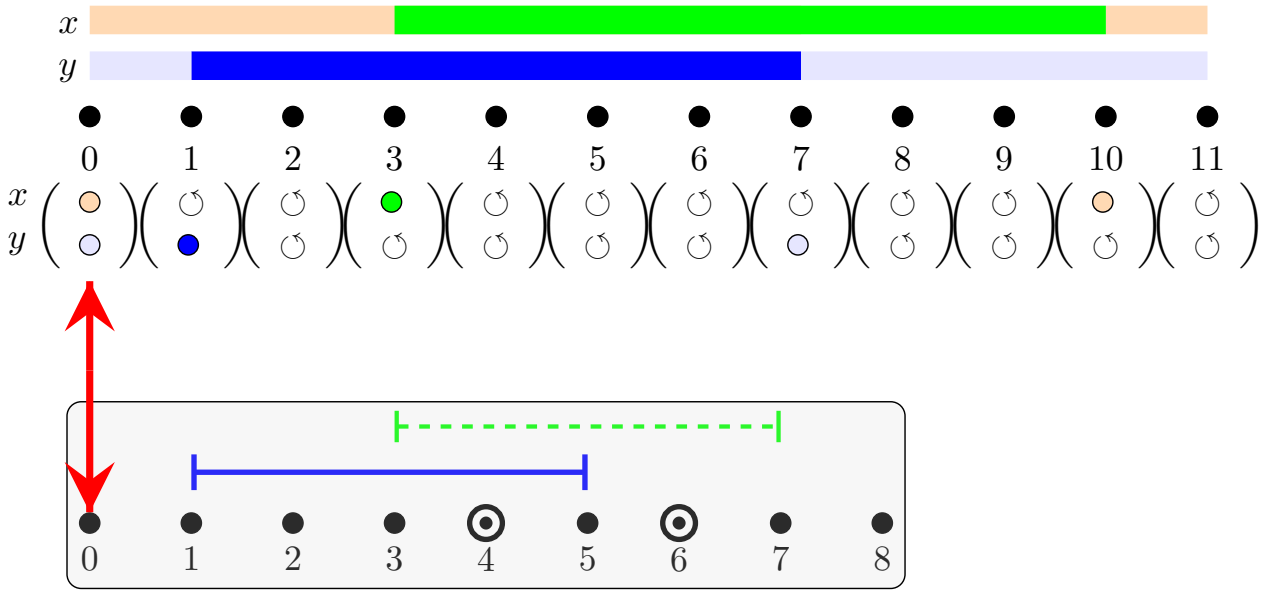
$$\Sigma = (\mathcal{D}(x_1) \cup \{\circlearrowleft\}) \times \ldots \times (\mathcal{D}(x_n) \cup \{\circlearrowleft\})$$

where $\circlearrowleft$ is a fresh special symbol that labels any point which is neither the starting point nor the ending point of a token.



How do we check blueprints against such words?

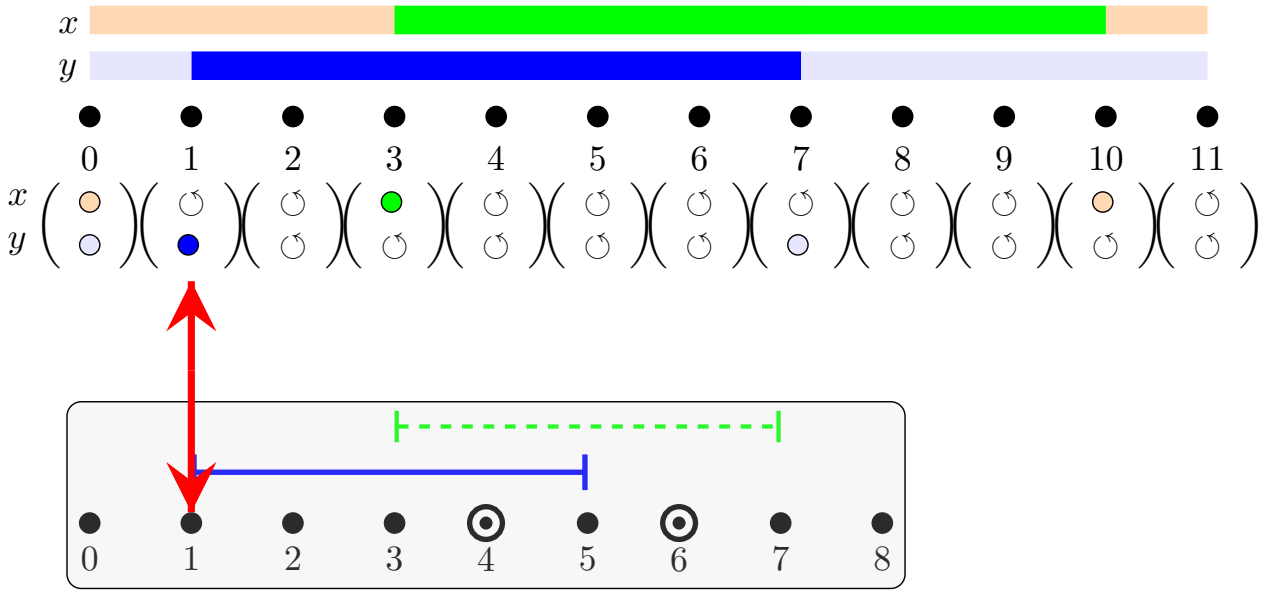
The basic building blocks: viewpoints



Blueprints are equipped with a pointer that identifies its local state, that is, one of its points/positions.

- the pair (blueprint, marked point) is called a **viewpoint**
- viewpoints allow one to check specific portions of the timelines while ignoring the rest
- since the set of blueprints is finite and each blueprint consists of a finite number of points, there is a **finite number** of distinct viewpoints
- viewpoints allows one to check a blueprint against a word.

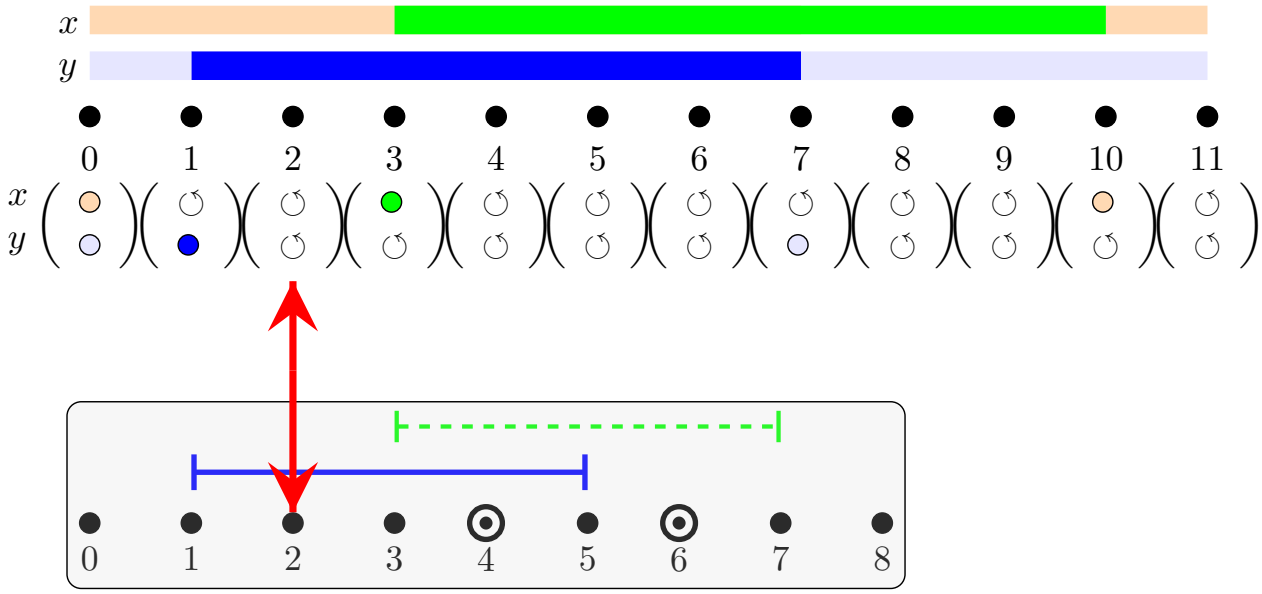
The basic building blocks: viewpoints



Blueprints are equipped with a pointer that identifies its local state, that is, one of its points/positions.

- the pair (blueprint, marked point) is called a **viewpoint**
- viewpoints allow one to check specific portions of the timelines while ignoring the rest
- since the set of blueprints is finite and each blueprint consists of a finite number of points, there is a **finite number** of distinct viewpoints
- viewpoints allows one to check a blueprint against a word.

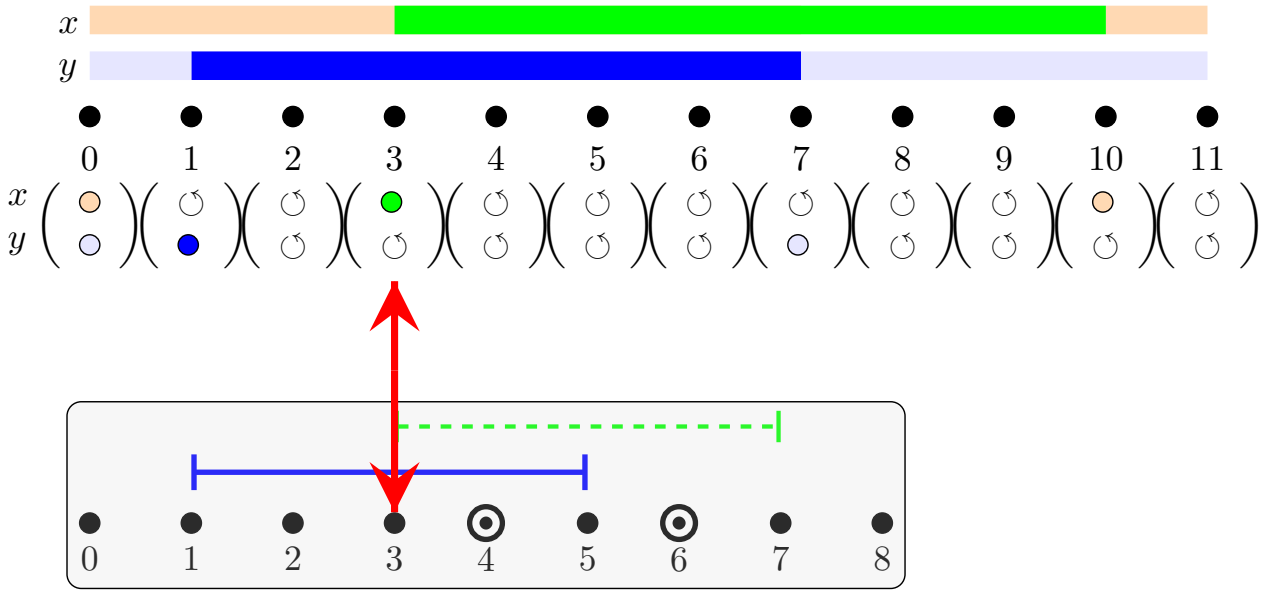
The basic building blocks: viewpoints



Blueprints are equipped with a pointer that identifies its local state, that is, one of its points/positions.

- the pair (blueprint, marked point) is called a **viewpoint**
- viewpoints allow one to check specific portions of the timelines while ignoring the rest
- since the set of blueprints is finite and each blueprint consists of a finite number of points, there is a **finite number** of distinct viewpoints
- viewpoints allows one to check a blueprint against a word.

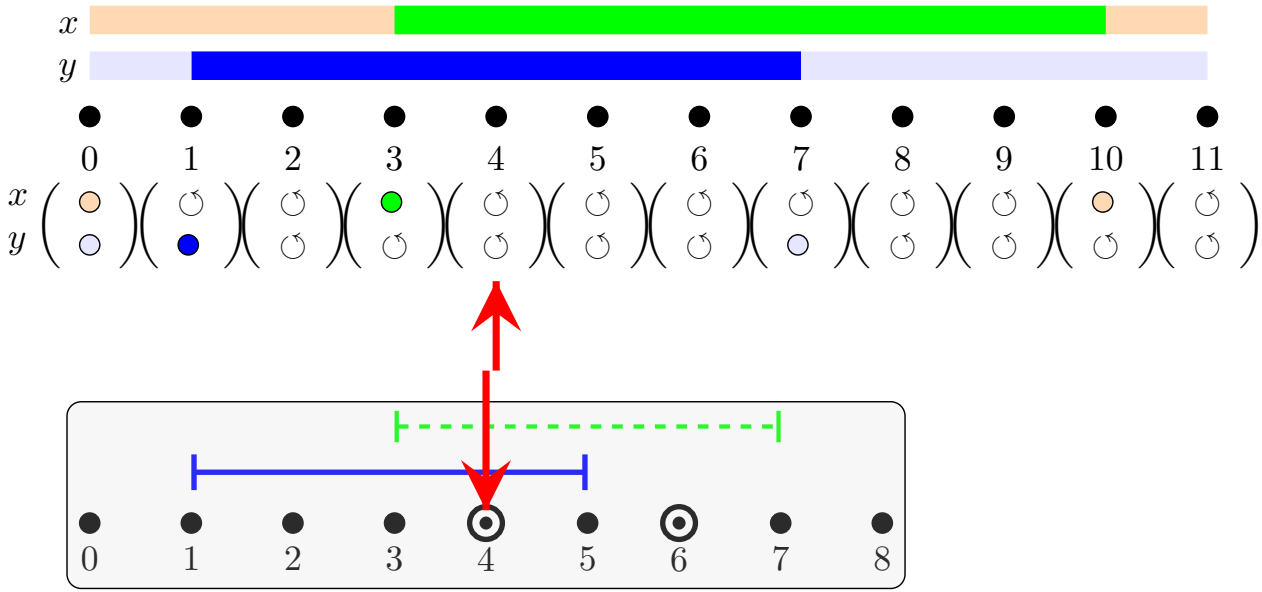
The basic building blocks: viewpoints



Blueprints are equipped with a pointer that identifies its local state, that is, one of its points/positions.

- the pair (blueprint, marked point) is called a **viewpoint**
- viewpoints allow one to check specific portions of the timelines while ignoring the rest
- since the set of blueprints is finite and each blueprint consists of a finite number of points, there is a **finite number** of distinct viewpoints
- viewpoints allows one to check a blueprint against a word.

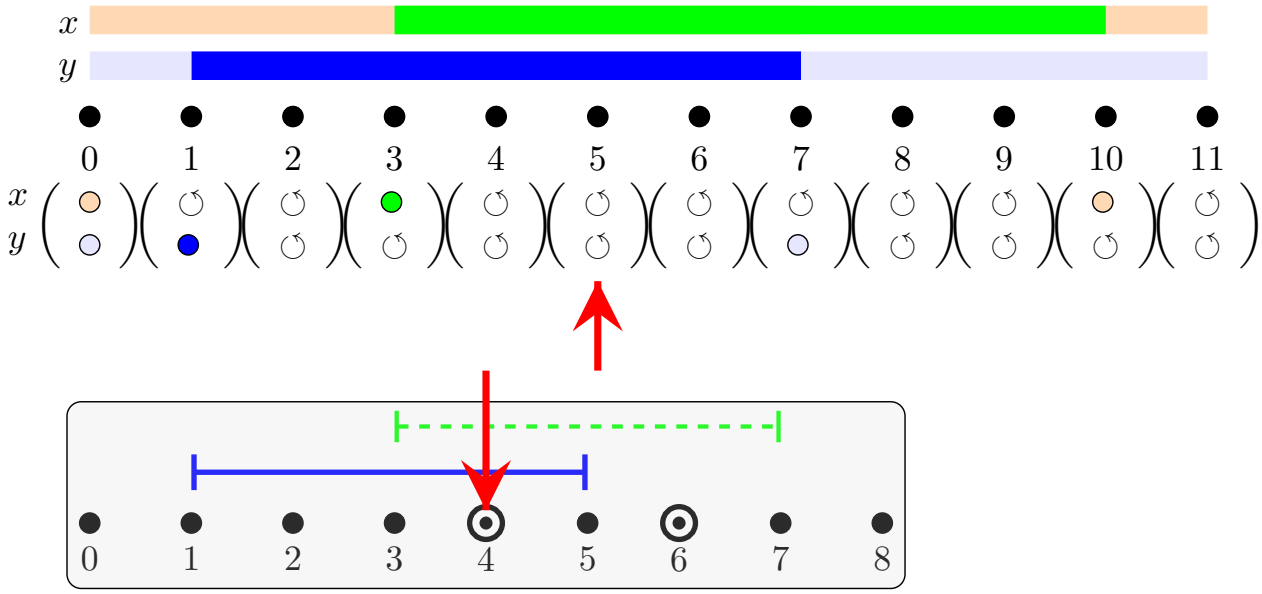
The basic building blocks: viewpoints



Blueprints are equipped with a pointer that identifies its local state, that is, one of its points/positions.

- the pair (blueprint, marked point) is called a **viewpoint**
- viewpoints allow one to check specific portions of the timelines while ignoring the rest
- since the set of blueprints is finite and each blueprint consists of a finite number of points, there is a **finite number** of distinct viewpoints
- viewpoints allows one to check a blueprint against a word.

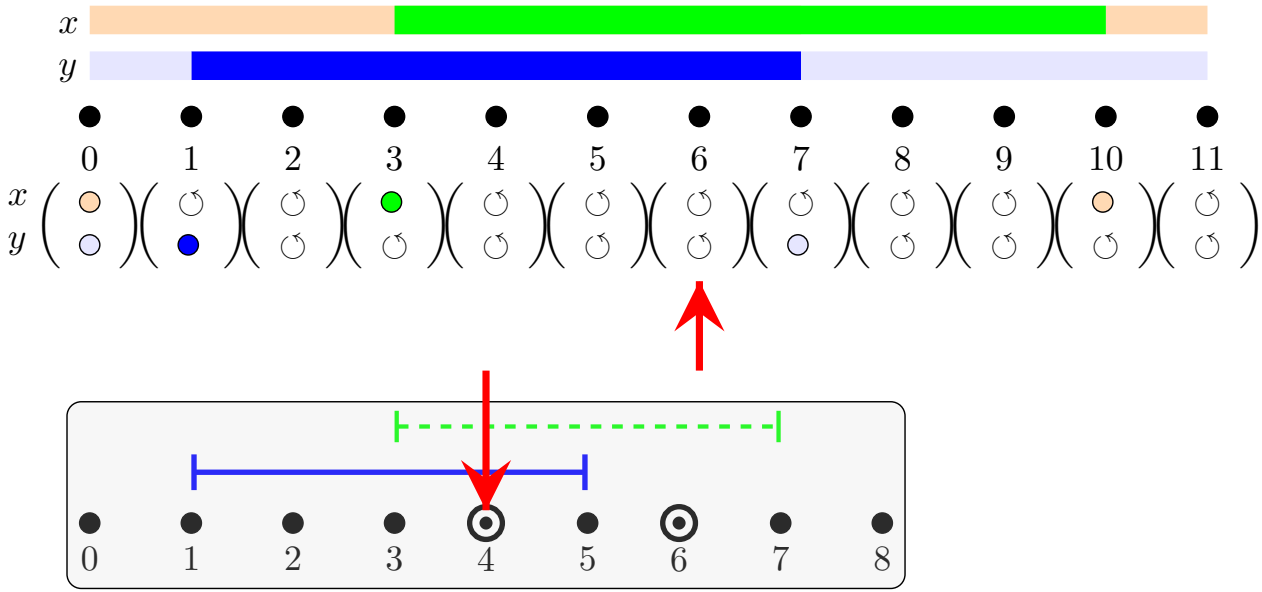
The basic building blocks: viewpoints



Blueprints are equipped with a pointer that identifies its local state, that is, one of its points/positions.

- the pair (blueprint, marked point) is called a **viewpoint**
- viewpoints allow one to check specific portions of the timelines while ignoring the rest
- since the set of blueprints is finite and each blueprint consists of a finite number of points, there is a **finite number** of distinct viewpoints
- viewpoints allows one to check a blueprint against a word.

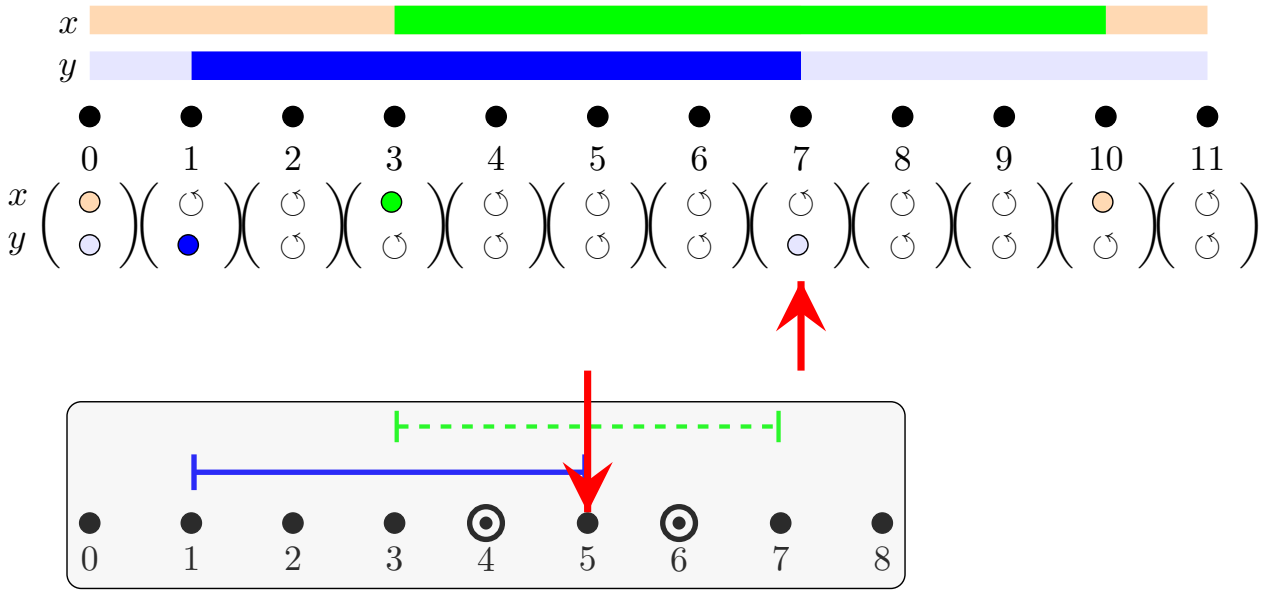
The basic building blocks: viewpoints



Blueprints are equipped with a pointer that identifies its local state, that is, one of its points/positions.

- the pair (blueprint, marked point) is called a **viewpoint**
- viewpoints allow one to check specific portions of the timelines while ignoring the rest
- since the set of blueprints is finite and each blueprint consists of a finite number of points, there is a **finite number** of distinct viewpoints
- viewpoints allows one to check a blueprint against a word.

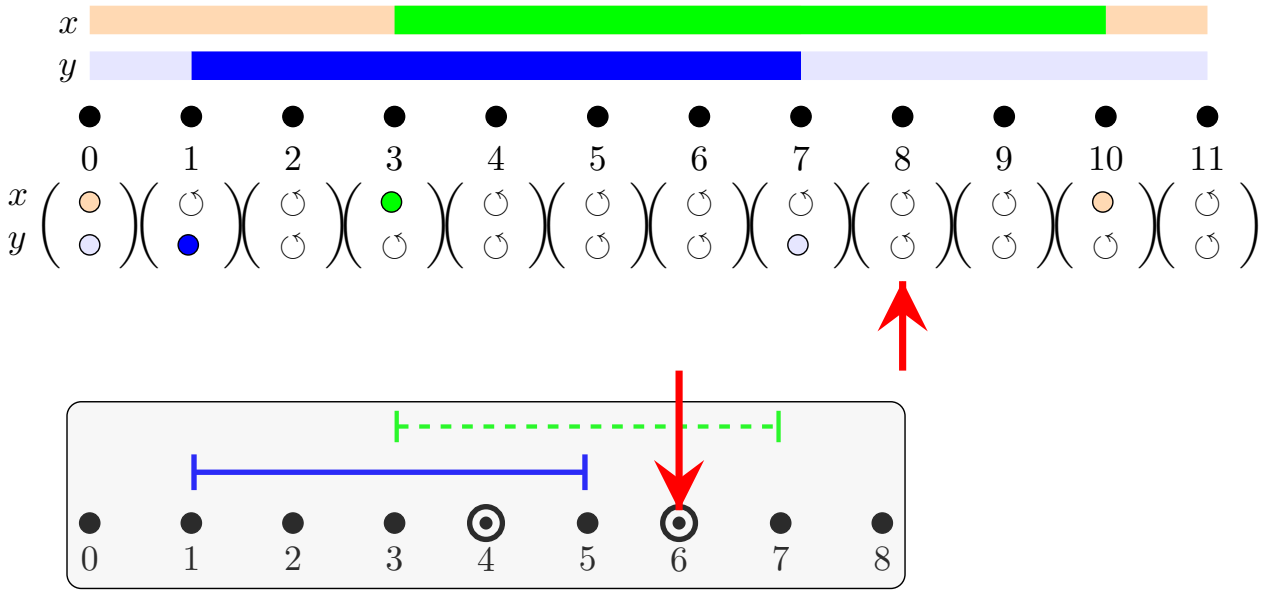
The basic building blocks: viewpoints



Blueprints are equipped with a pointer that identifies its local state, that is, one of its points/positions.

- the pair (blueprint, marked point) is called a **viewpoint**
- viewpoints allow one to check specific portions of the timelines while ignoring the rest
- since the set of blueprints is finite and each blueprint consists of a finite number of points, there is a **finite number** of distinct viewpoints
- viewpoints allows one to check a blueprint against a word.

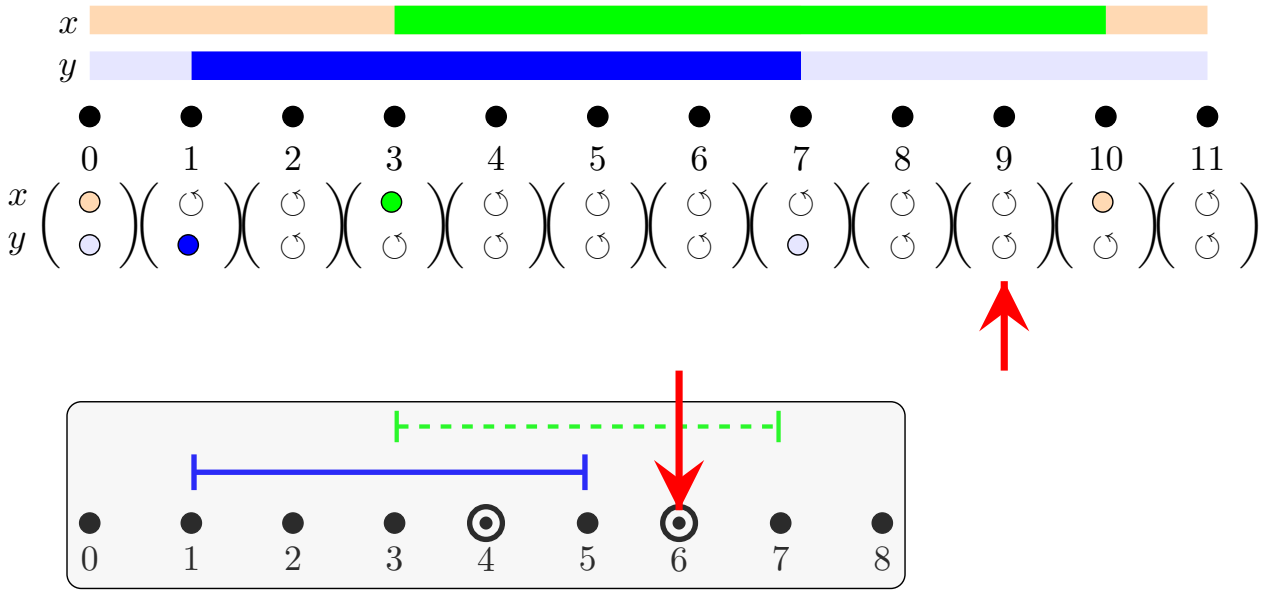
The basic building blocks: viewpoints



Blueprints are equipped with a pointer that identifies its local state, that is, one of its points/positions.

- the pair (blueprint, marked point) is called a **viewpoint**
- viewpoints allow one to check specific portions of the timelines while ignoring the rest
- since the set of blueprints is finite and each blueprint consists of a finite number of points, there is a **finite number** of distinct viewpoints
- viewpoints allows one to check a blueprint against a word.

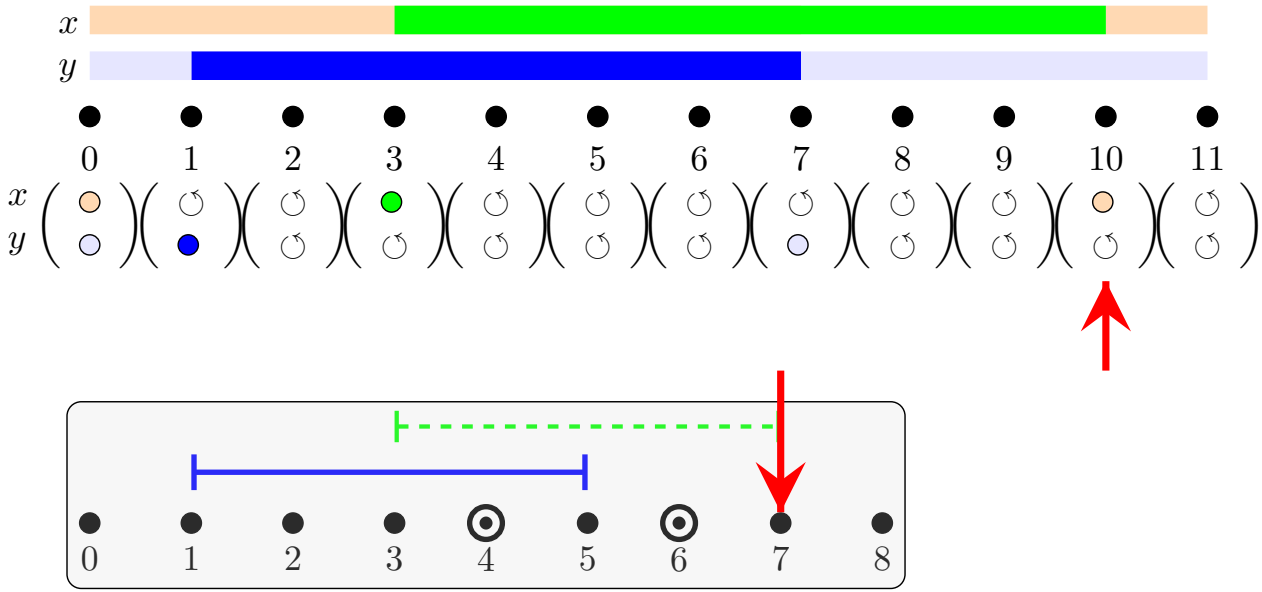
The basic building blocks: viewpoints



Blueprints are equipped with a pointer that identifies its local state, that is, one of its points/positions.

- the pair (blueprint, marked point) is called a **viewpoint**
- viewpoints allow one to check specific portions of the timelines while ignoring the rest
- since the set of blueprints is finite and each blueprint consists of a finite number of points, there is a **finite number** of distinct viewpoints
- viewpoints allows one to check a blueprint against a word.

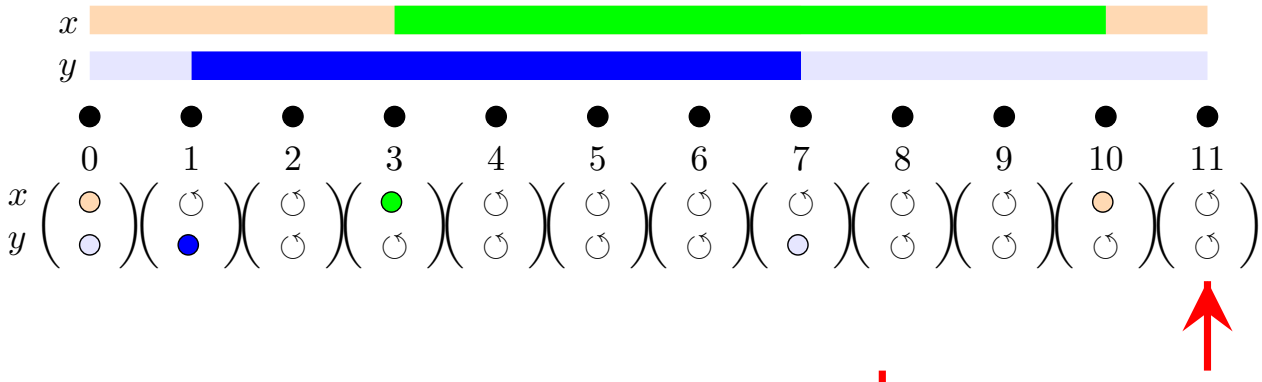
The basic building blocks: viewpoints

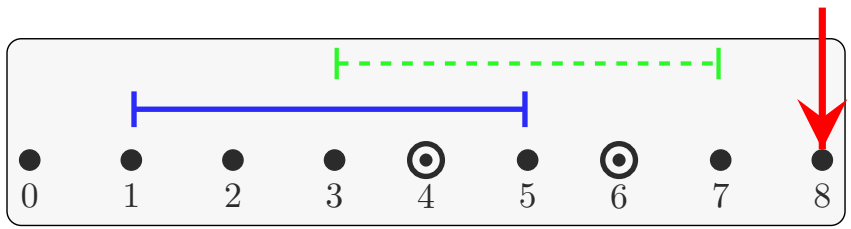


Blueprints are equipped with a pointer that identifies its local state, that is, one of its points/positions.

- the pair (blueprint, marked point) is called a **viewpoint**
- viewpoints allow one to check specific portions of the timelines while ignoring the rest
- since the set of blueprints is finite and each blueprint consists of a finite number of points, there is a **finite number** of distinct viewpoints
- viewpoints allows one to check a blueprint against a word.

The basic building blocks: viewpoints

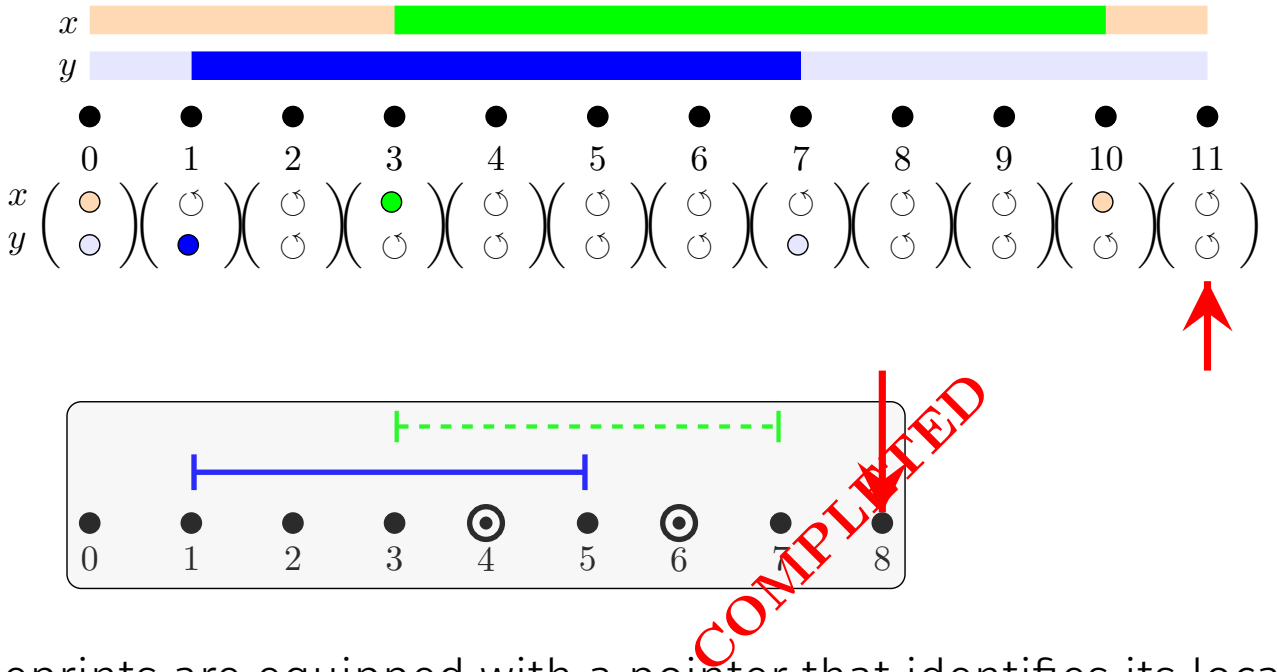




Blueprints are equipped with a pointer that identifies its local state, that is, one of its points/positions.

- the pair (blueprint, marked point) is called a **viewpoint**
- viewpoints allow one to check specific portions of the timelines while ignoring the rest
- since the set of blueprints is finite and each blueprint consists of a finite number of points, there is a **finite number** of distinct viewpoints
- viewpoints allows one to check a blueprint against a word.

The basic building blocks: viewpoints



Blueprints are equipped with a pointer that identifies its local state, that is, one of its points/positions.

- the pair (blueprint, marked point) is called a **viewpoint**
- viewpoints allow one to check specific portions of the timelines while ignoring the rest
- since the set of blueprints is finite and each blueprint consists of a finite number of points, there is a **finite number** of distinct viewpoints
- viewpoints allows one to check a blueprint against a word.

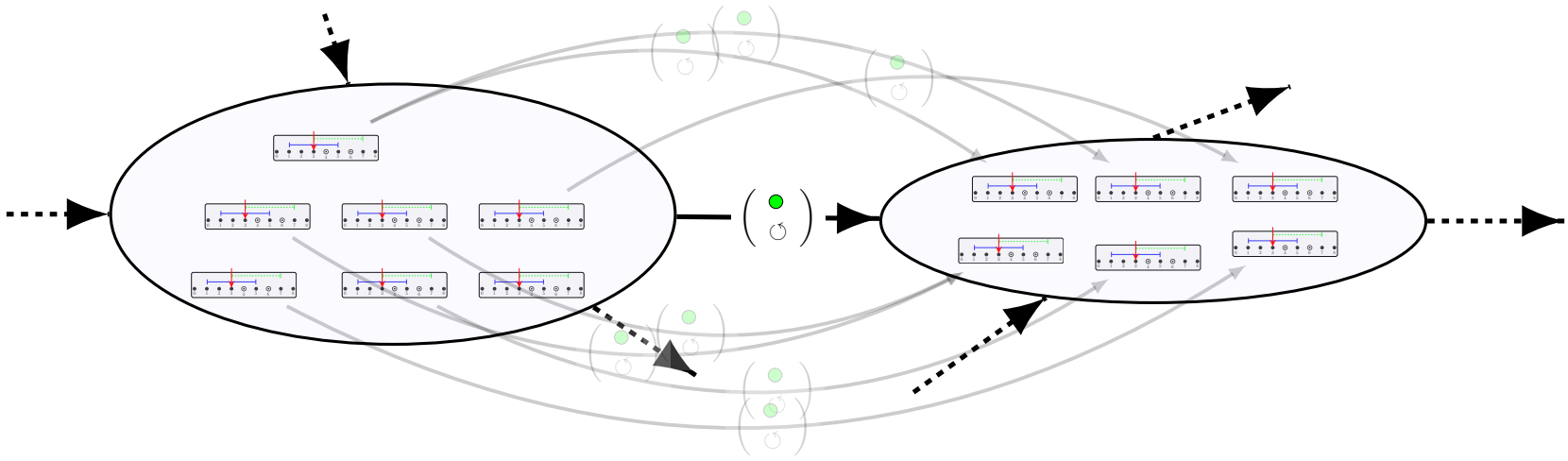
States as sets of viewpoints

The states of the NFA (resp., NBA) are sets of viewpoints.

The transition function guarantees the correct synchronization of viewpoints according to the current symbol.

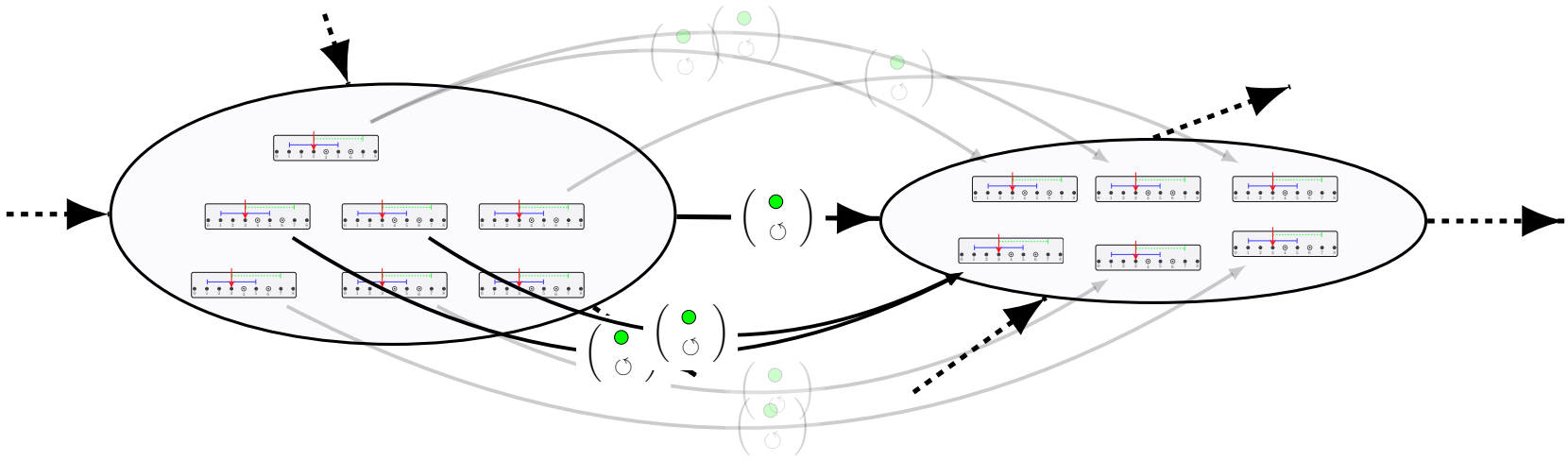
Final states are states in which all the viewpoints are completed.

States as sets of viewpoints (2)



During execution, we may observe two kinds of phenomenon.

States as sets of viewpoints (2)

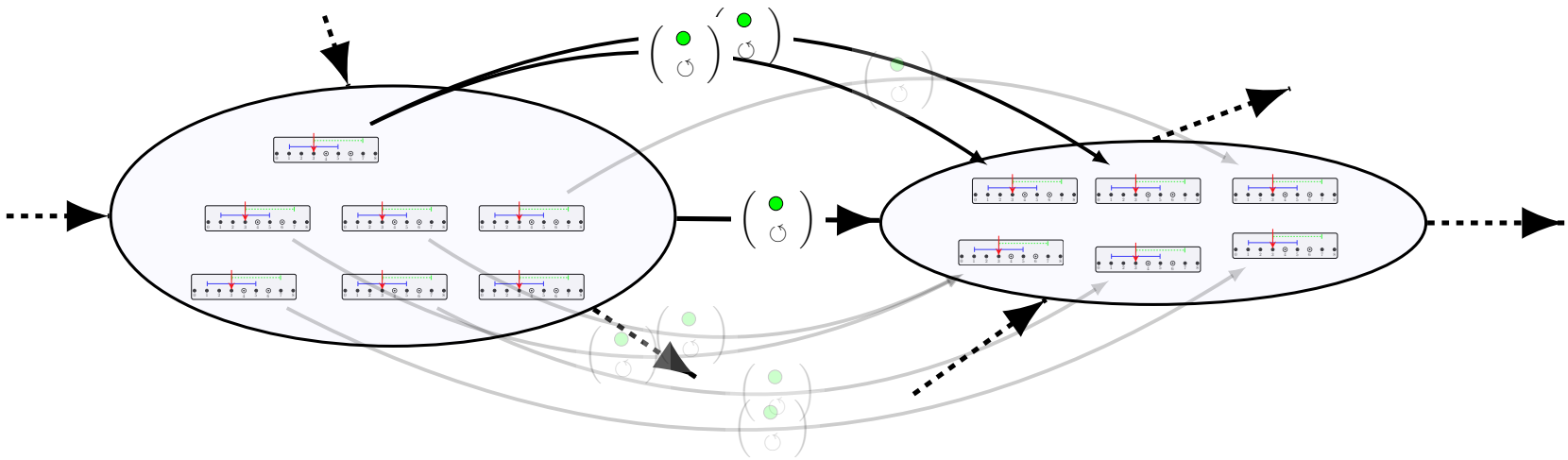


During execution, we may observe two kinds of phenomenon:

- **Confluence**: two or more **distinct** viewpoints on the same blueprint in the source state evolve into the same viewpoint in the target state.

  This is the case when in the source state they were associated with distinct tokens, but, at a certain point, their “future” turns out to be the same and they become indistinguishable.

States as sets of viewpoints (2)

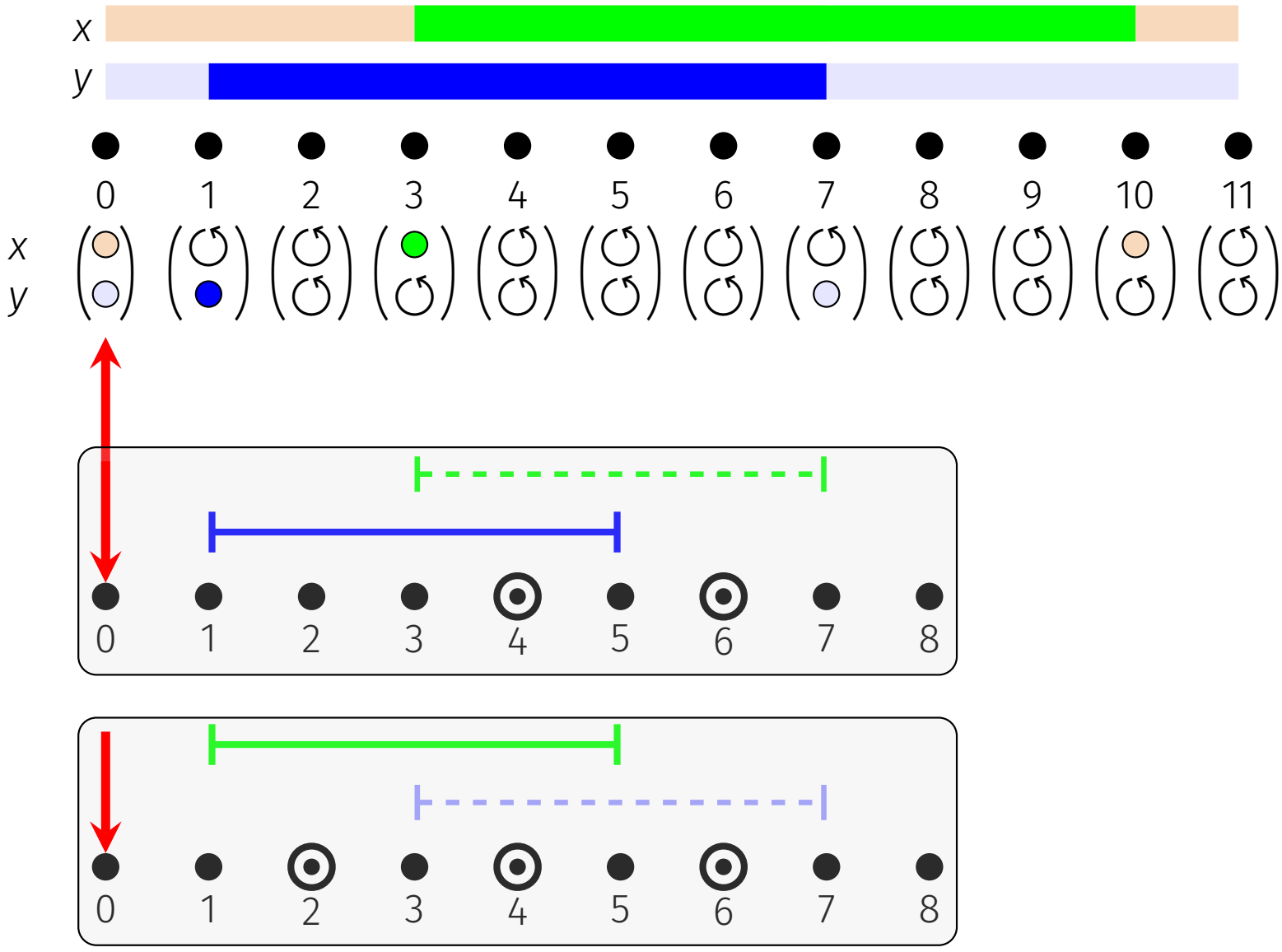


During execution, we may observe two kinds of phenomenon:

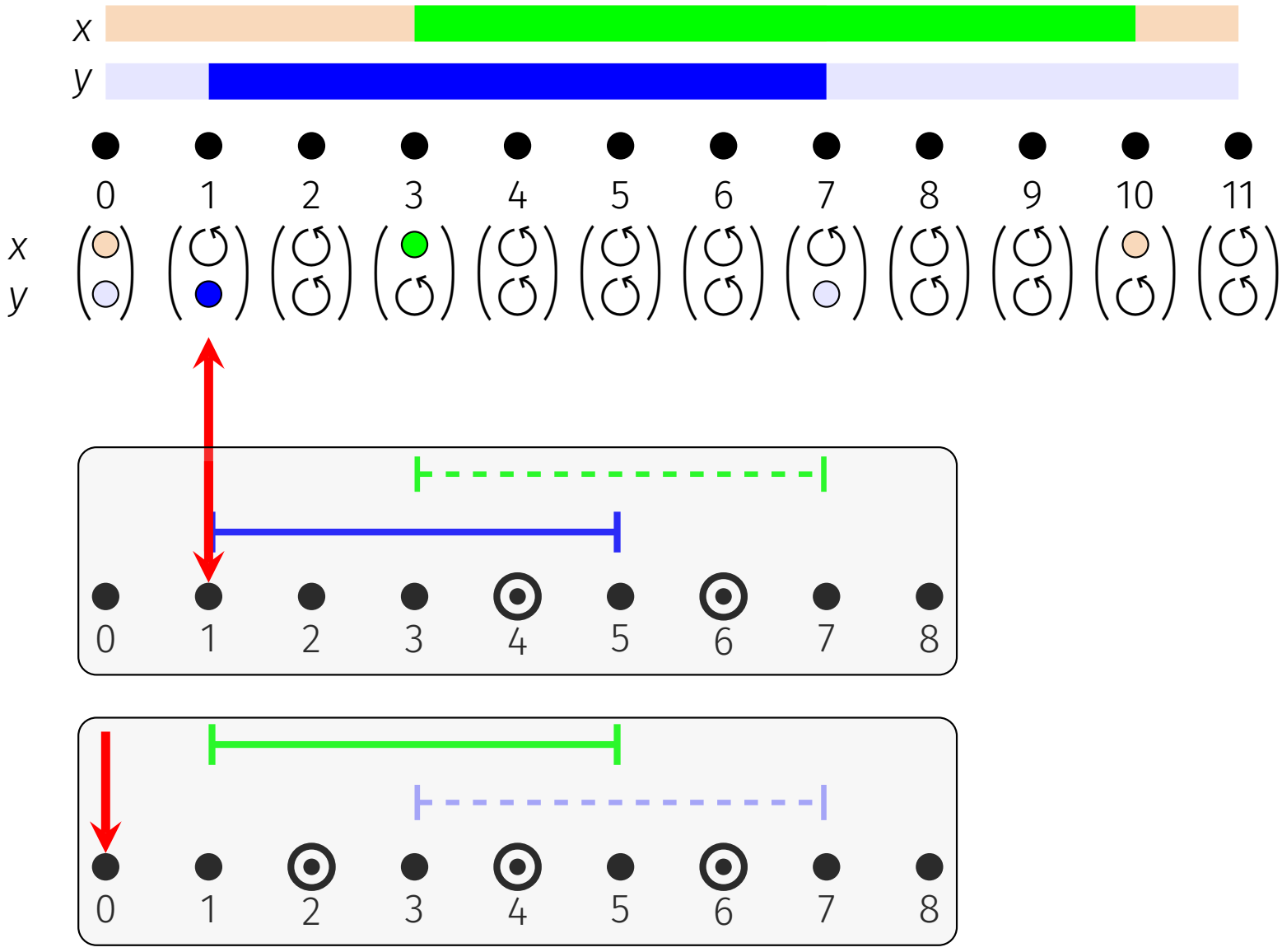
- Splitting: a viewpoint in the source state evolves into two distinct viewpoints in the target state.

  This is the case, for instance, when the same “past history” must be used to fulfill two or more tokens that trigger the same synchronization rule.

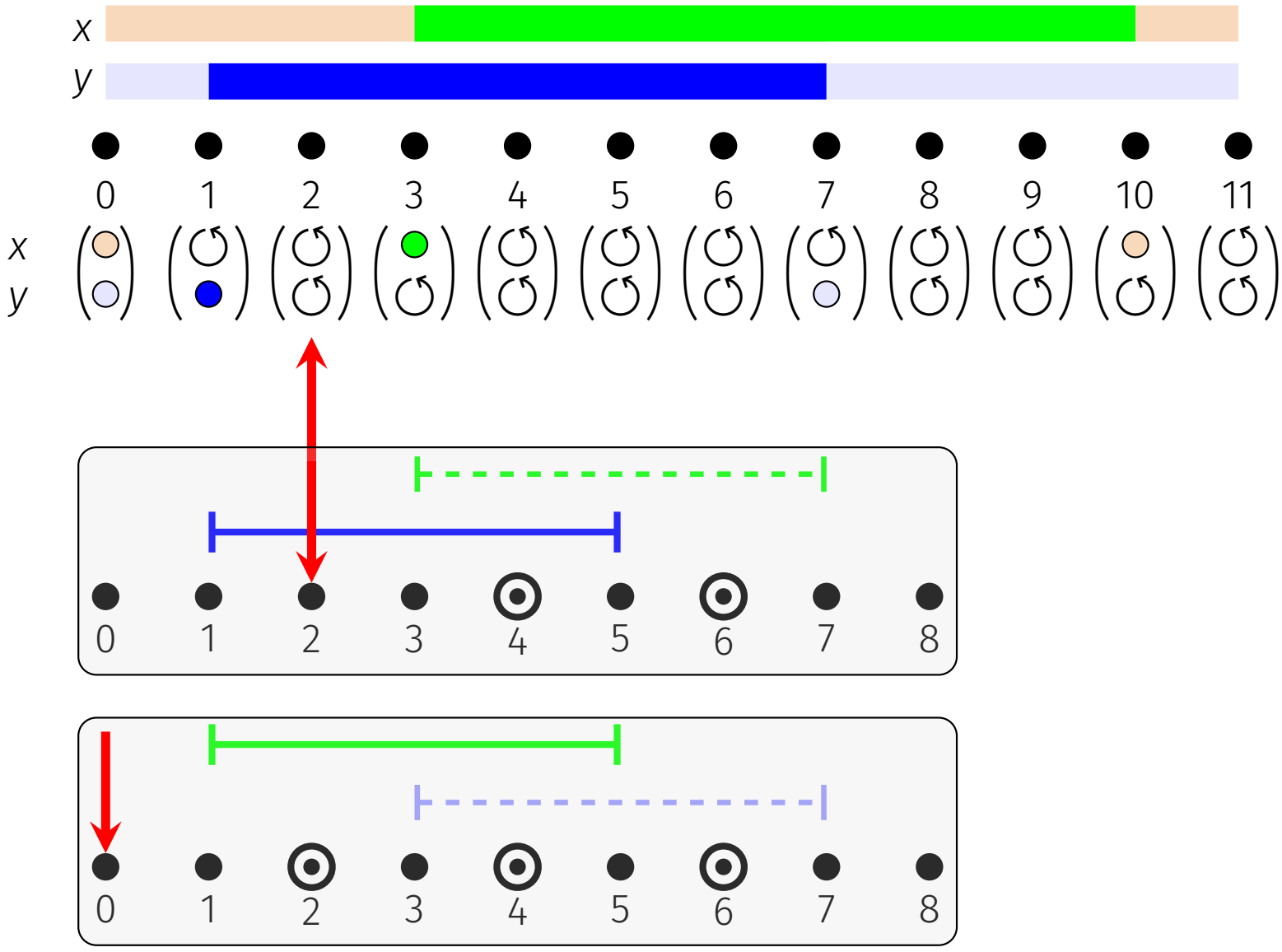
Viewpoints synchronization: success



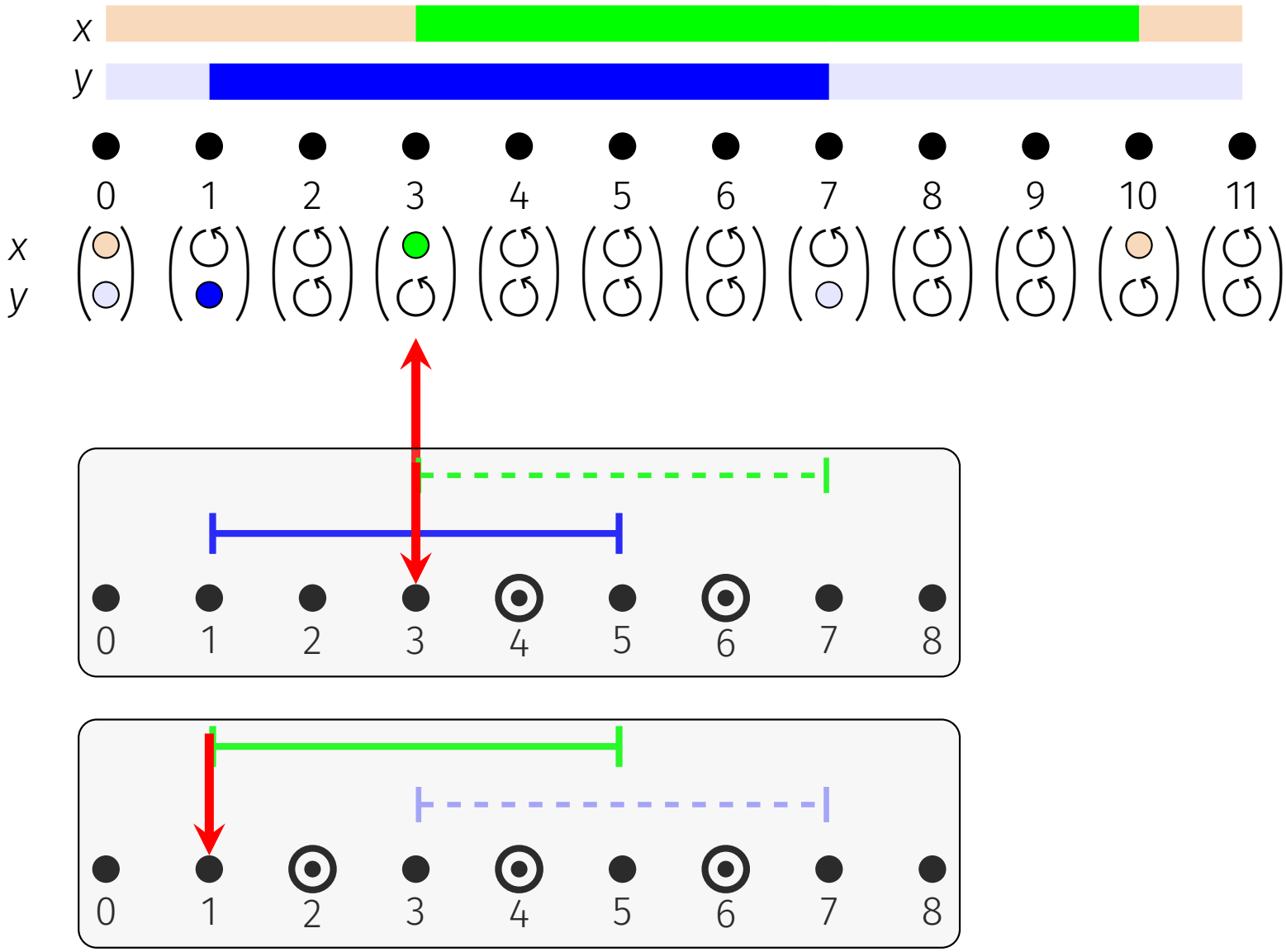
Viewpoints synchronization: success



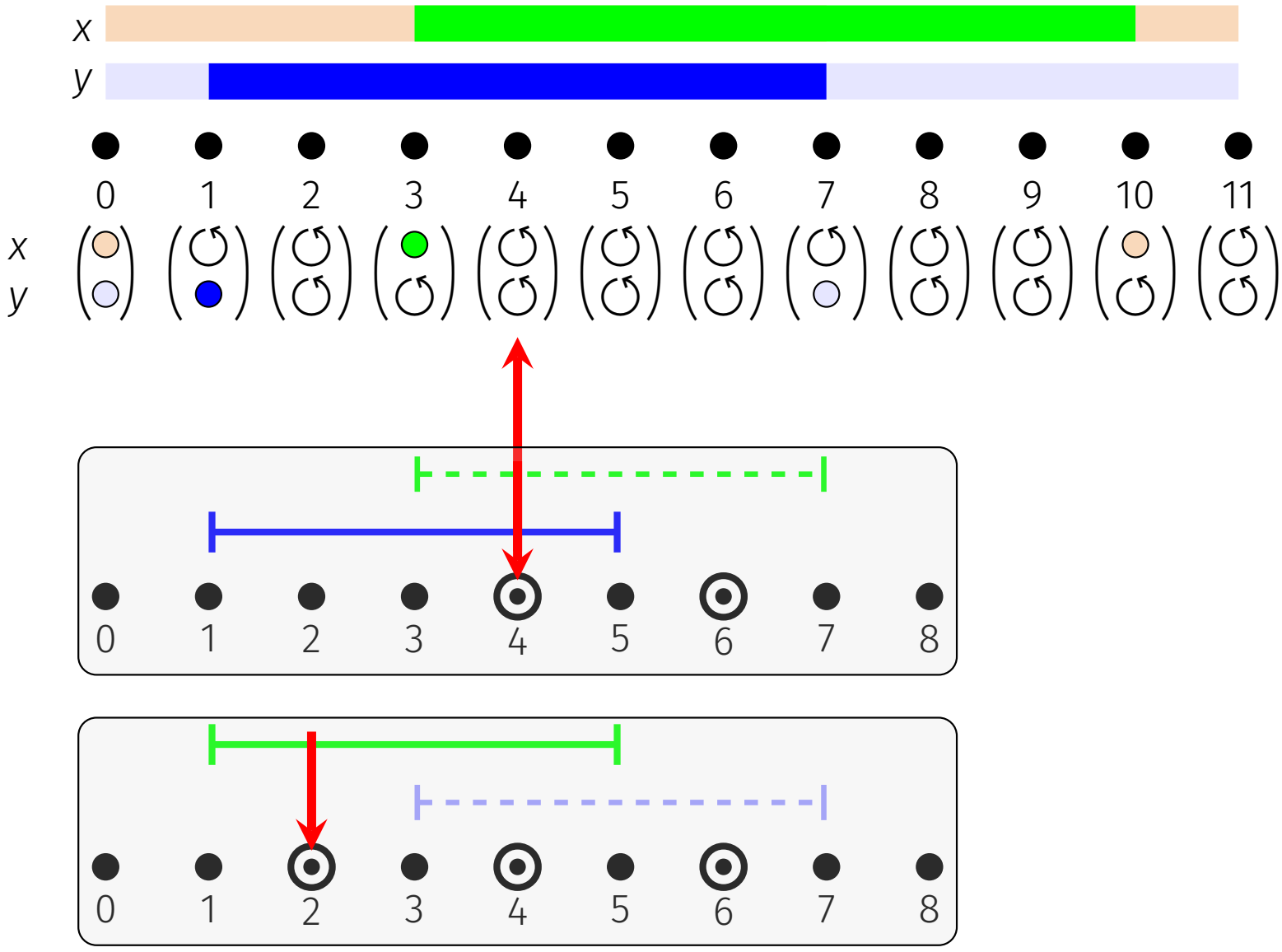
Viewpoints synchronization: success



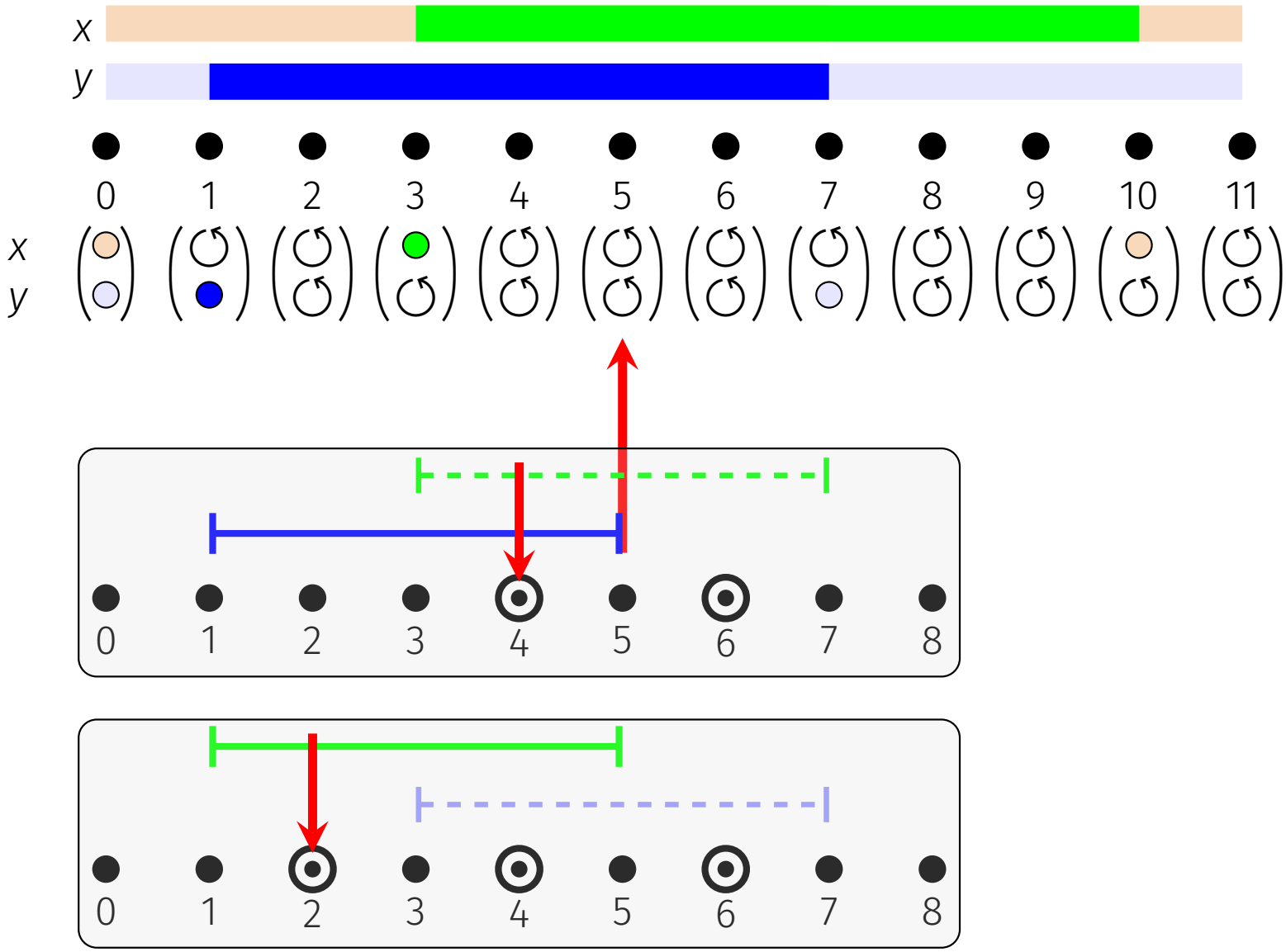
Viewpoints synchronization: success



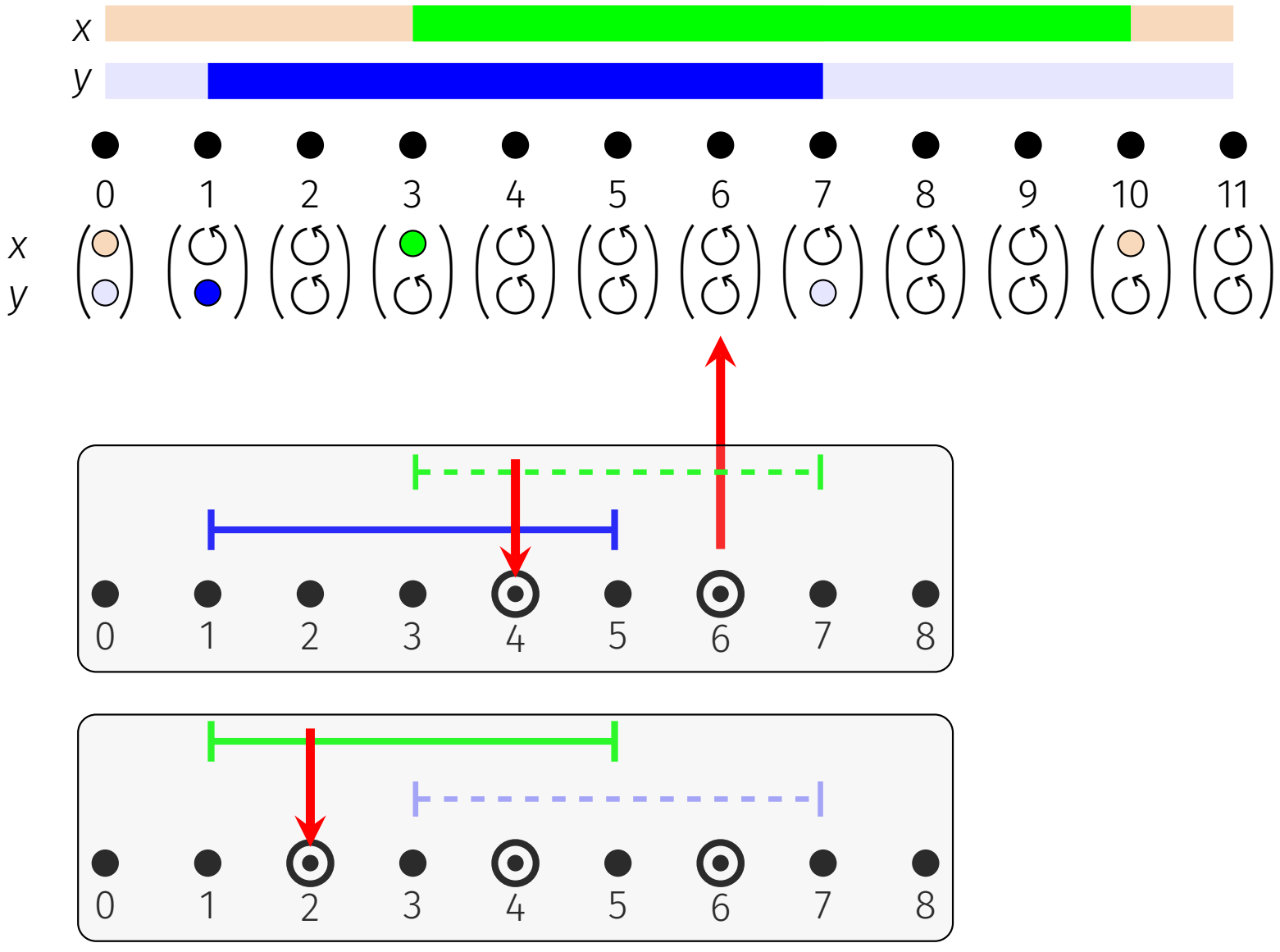
Viewpoints synchronization: success



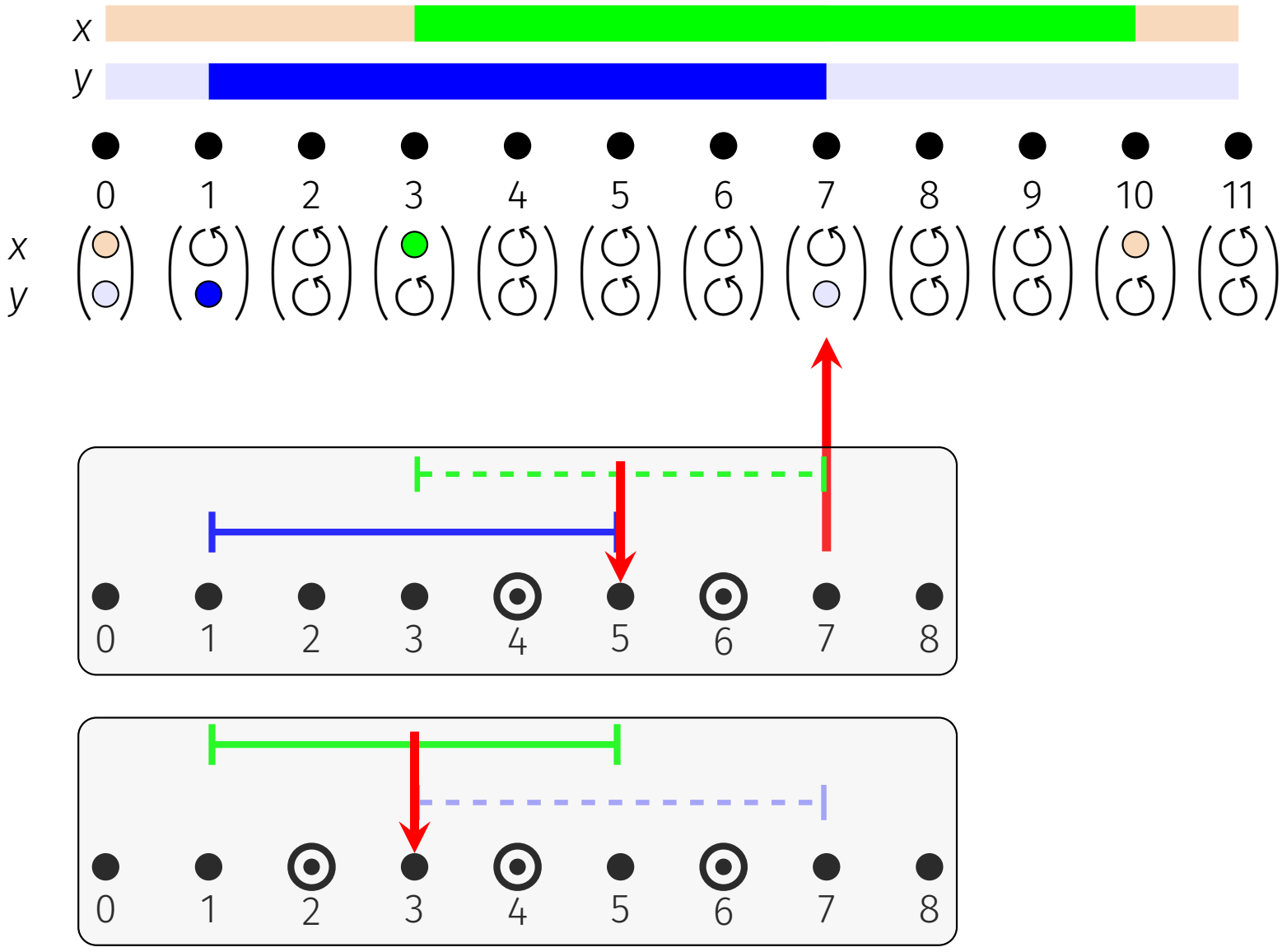
Viewpoints synchronization: success



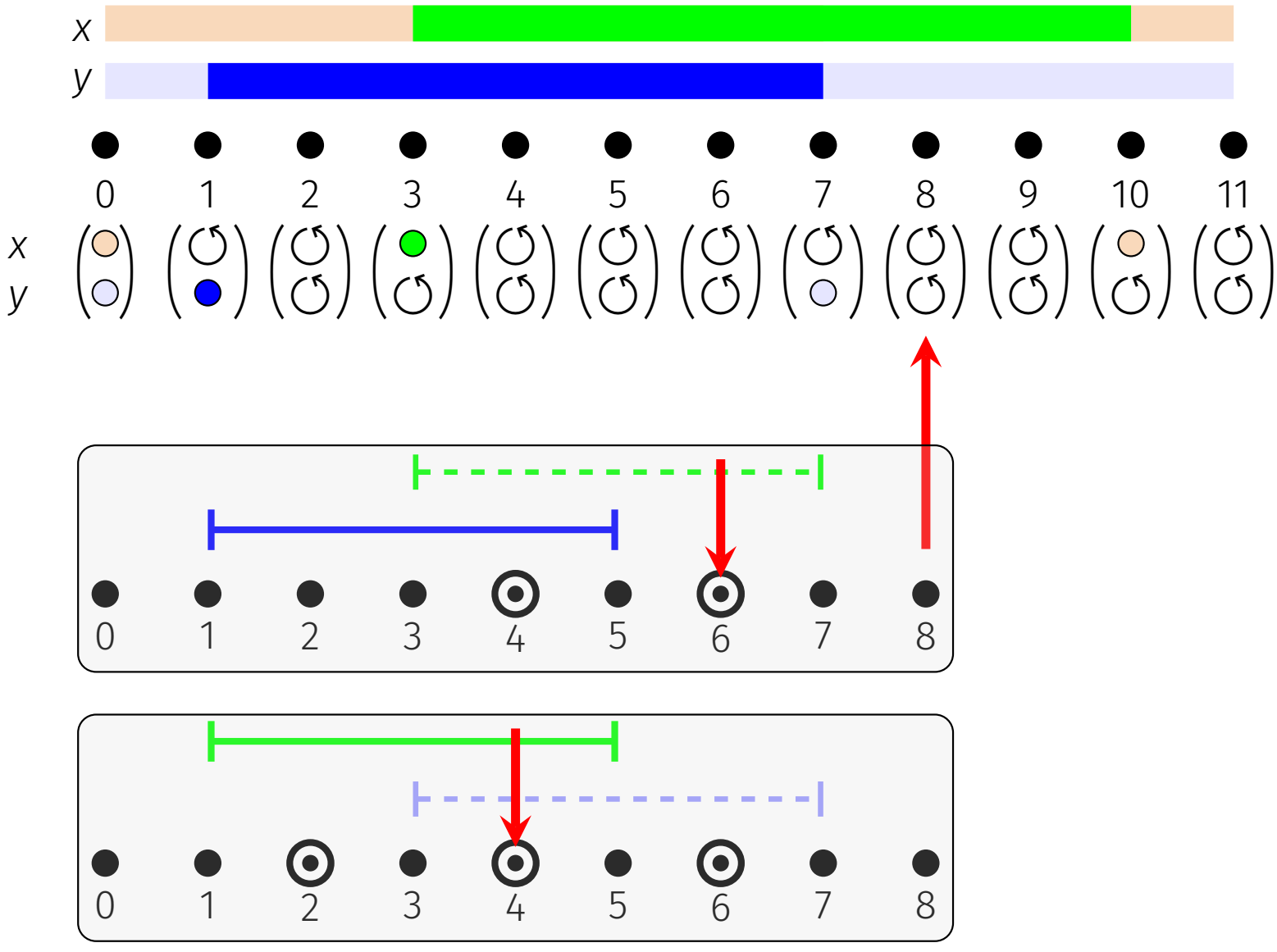
Viewpoints synchronization: success



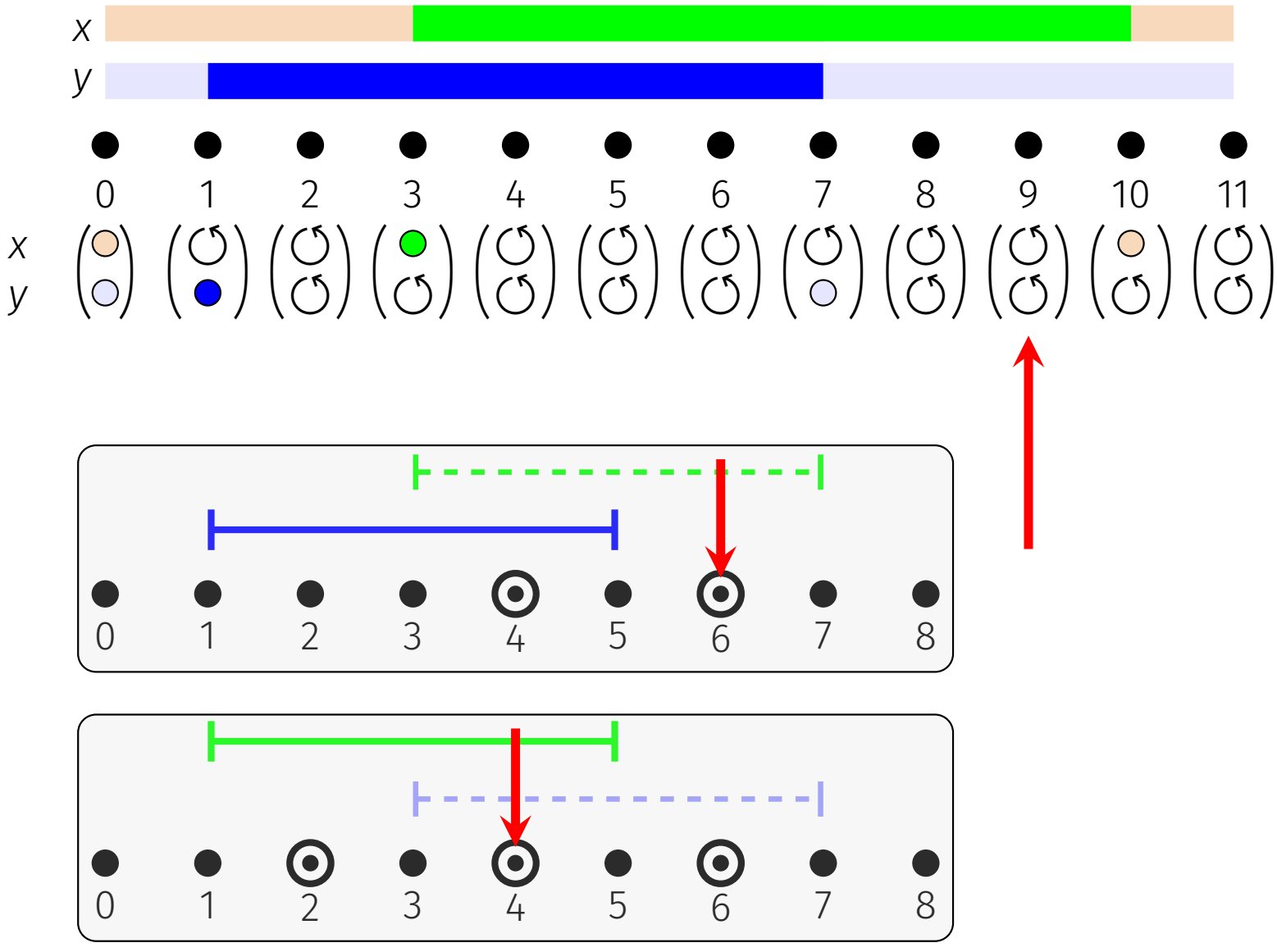
Viewpoints synchronization: success



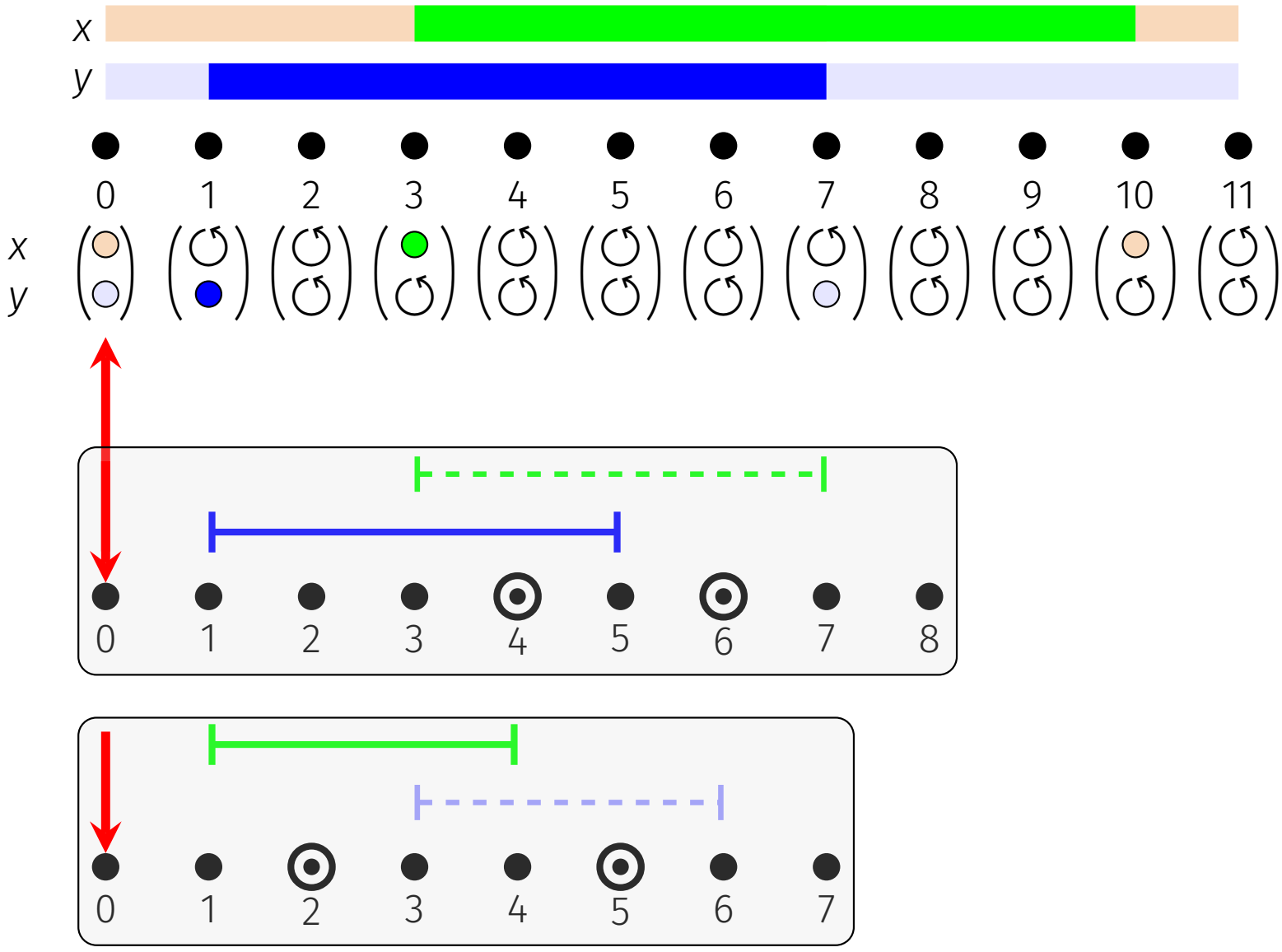
Viewpoints synchronization: success



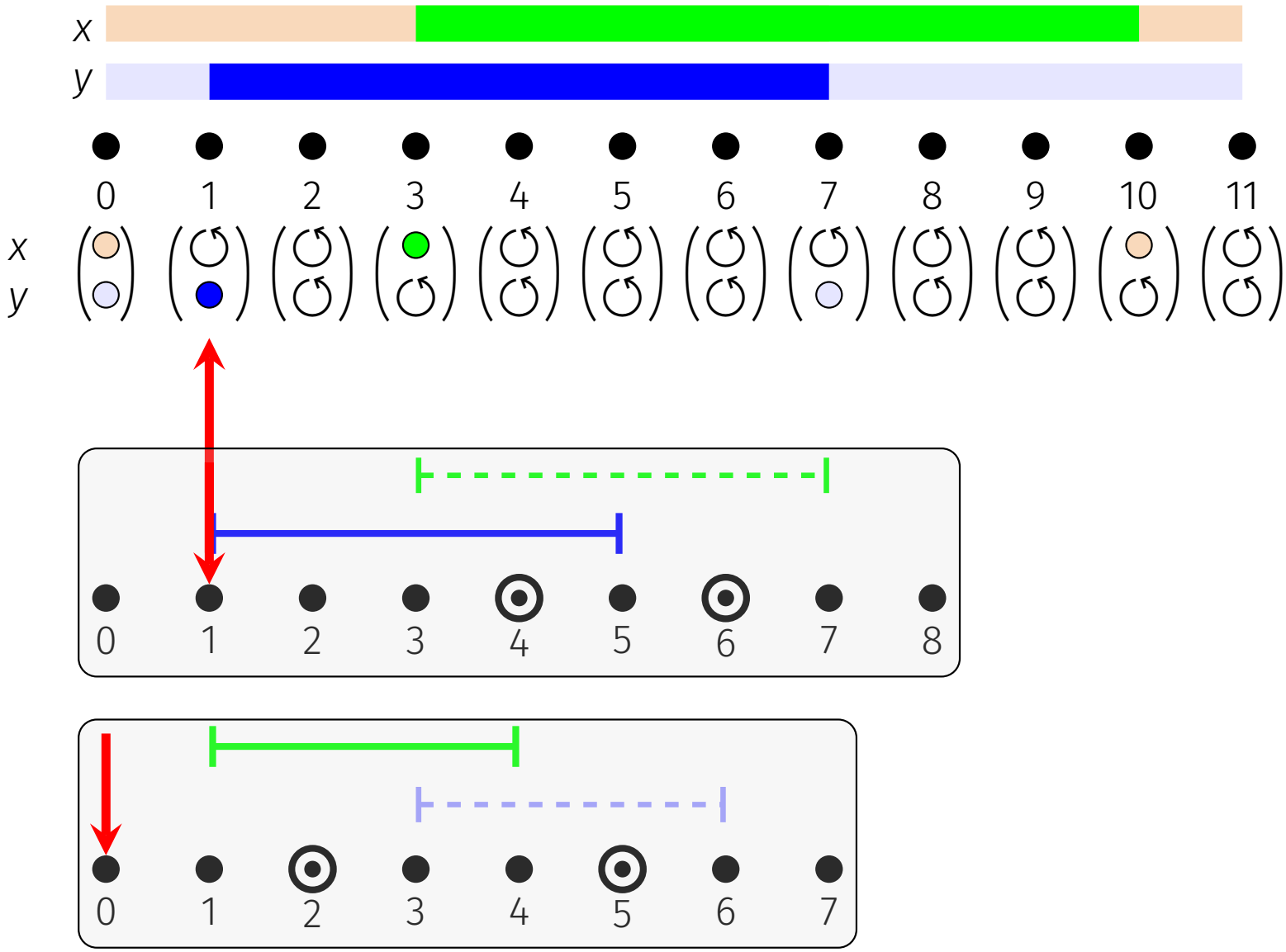
Viewpoints synchronization: success



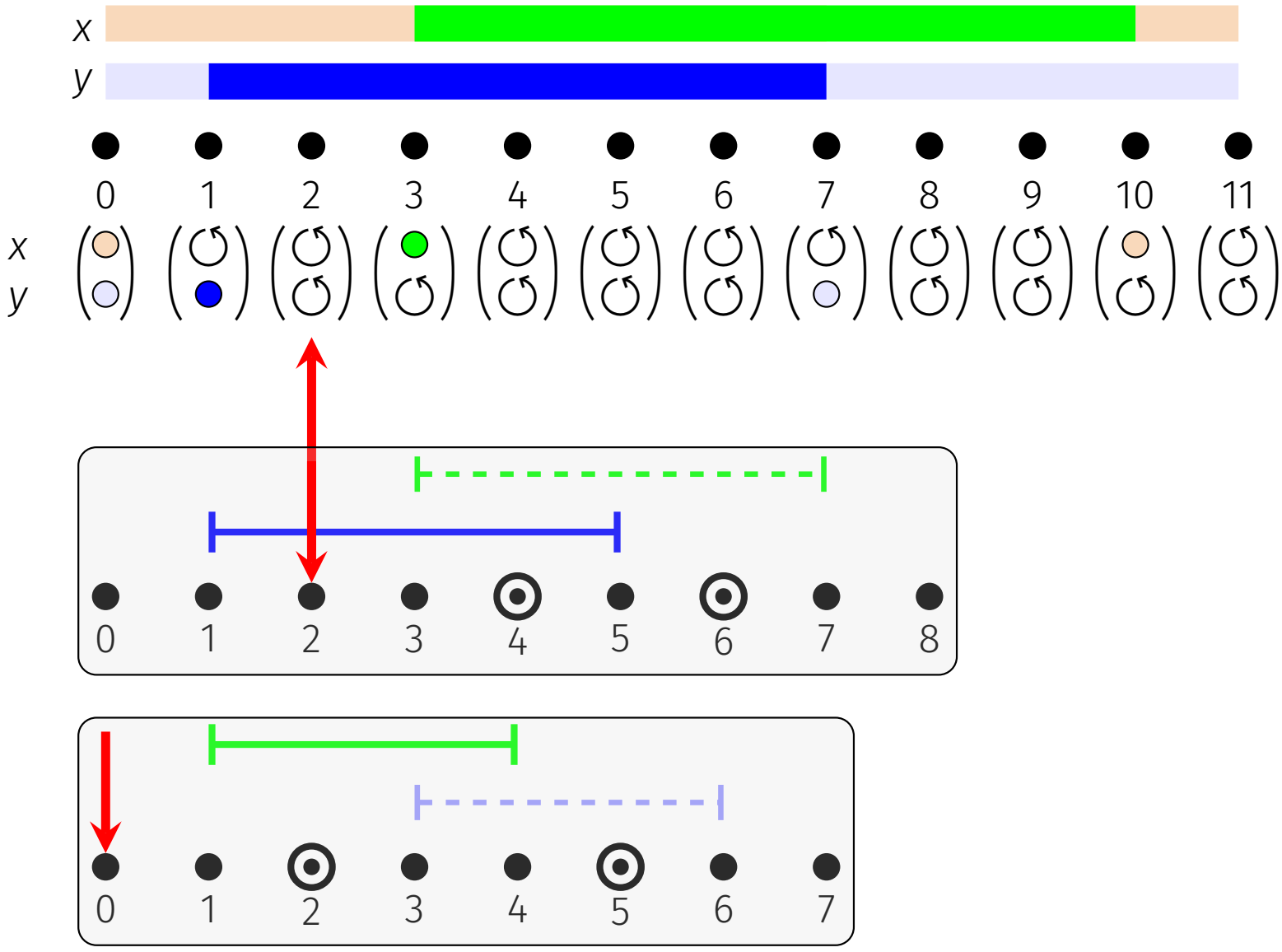
Viewpoints synchronization: failure



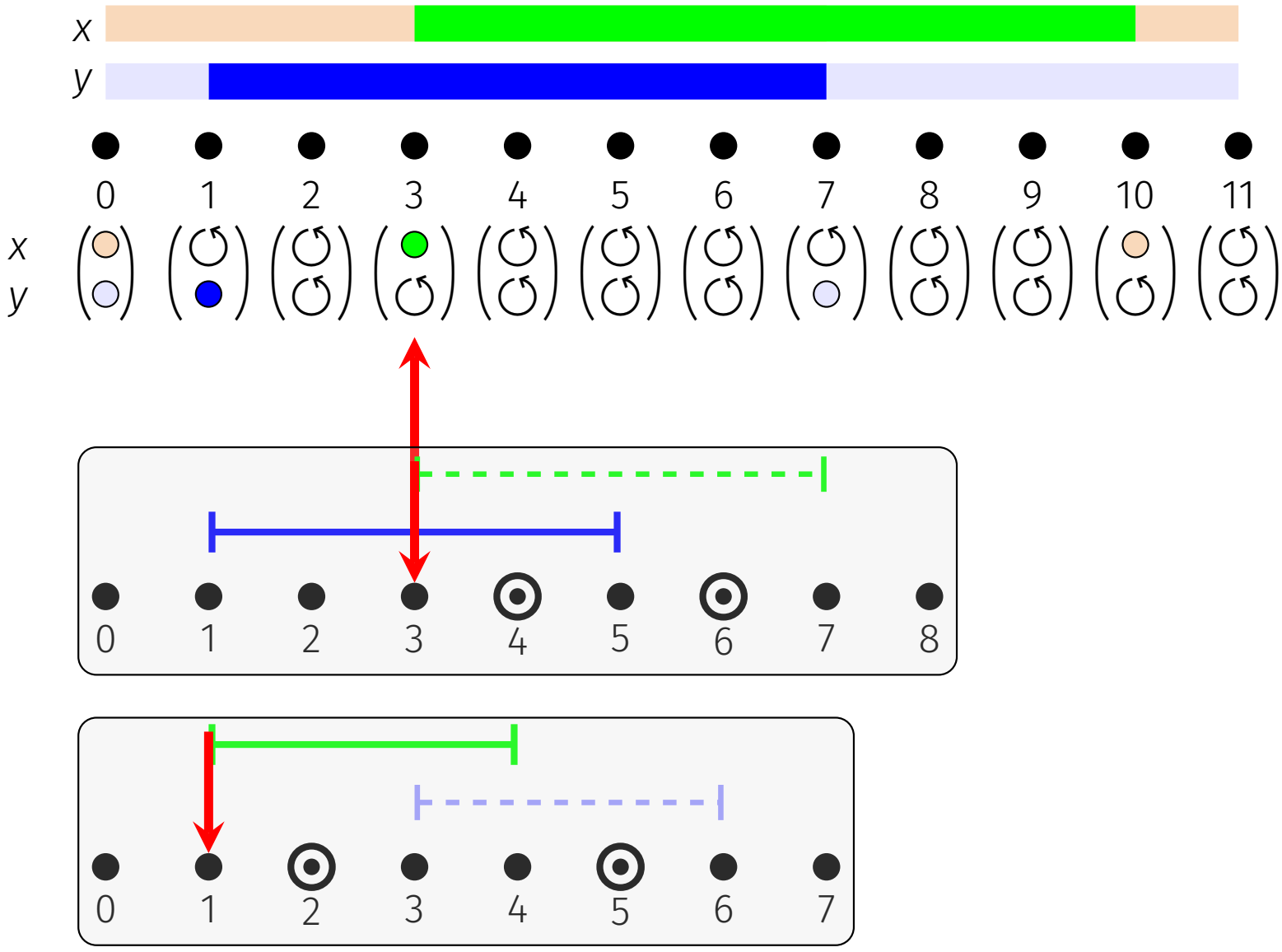
Viewpoints synchronization: failure



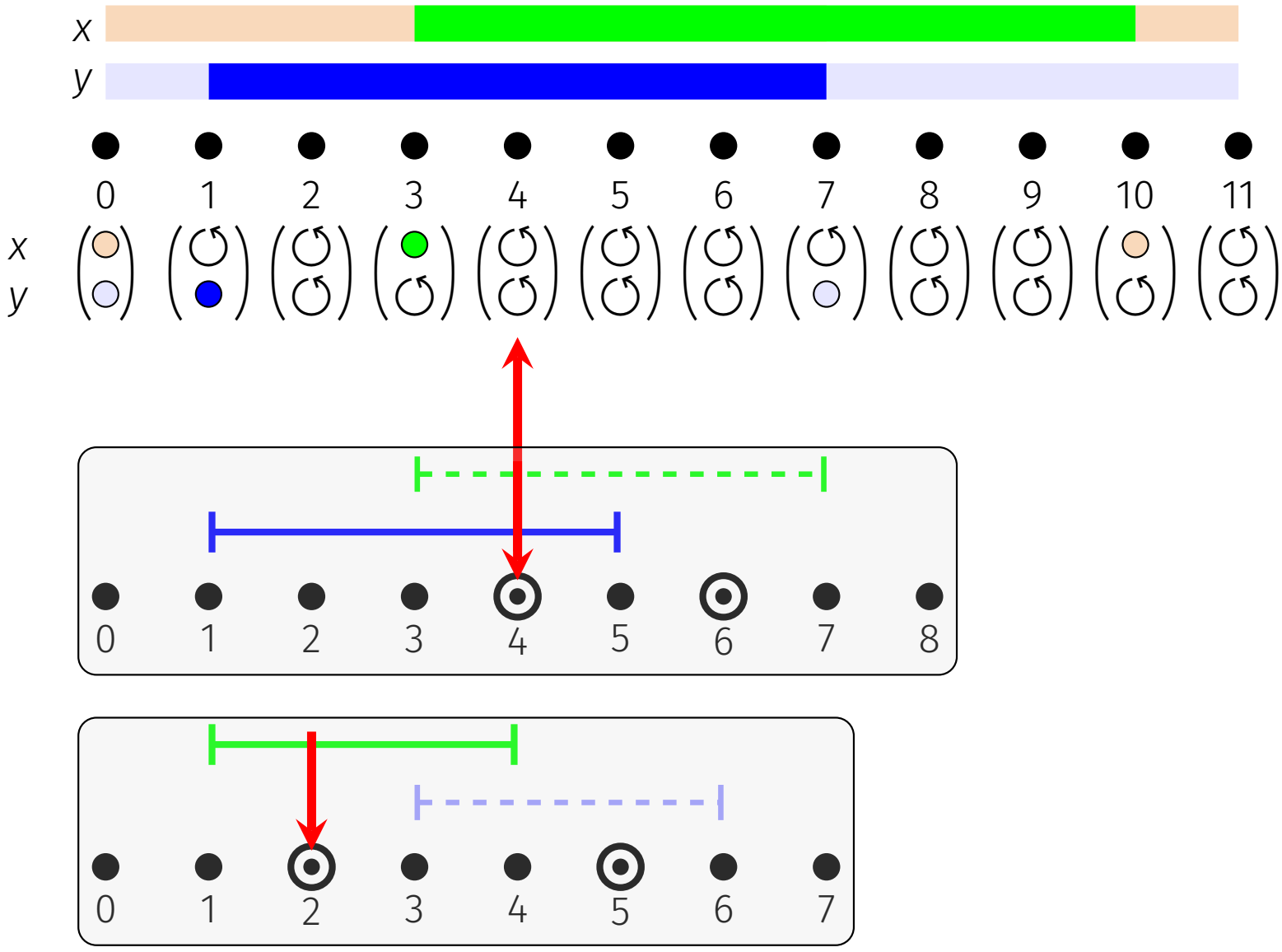
Viewpoints synchronization: failure



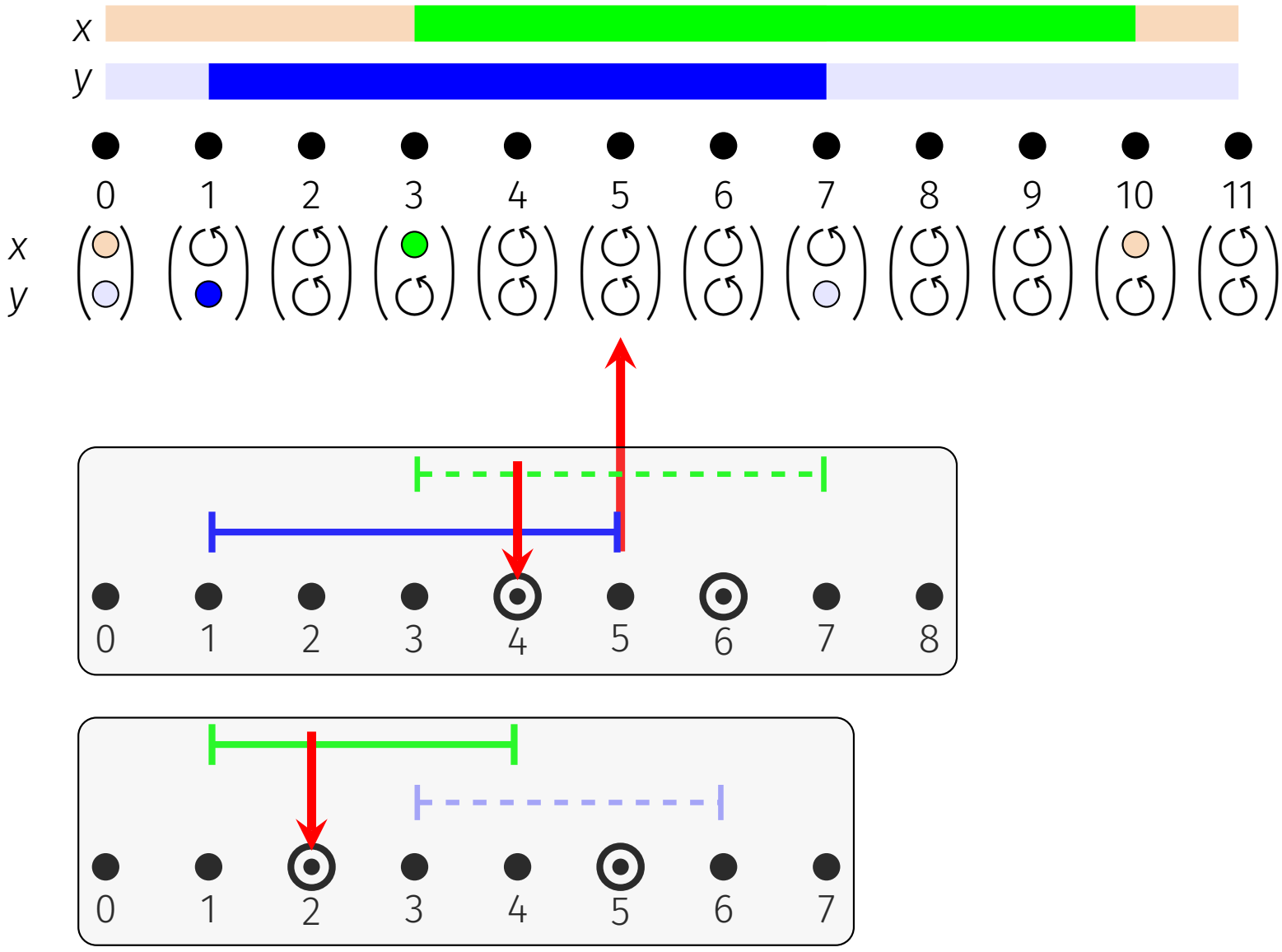
Viewpoints synchronization: failure



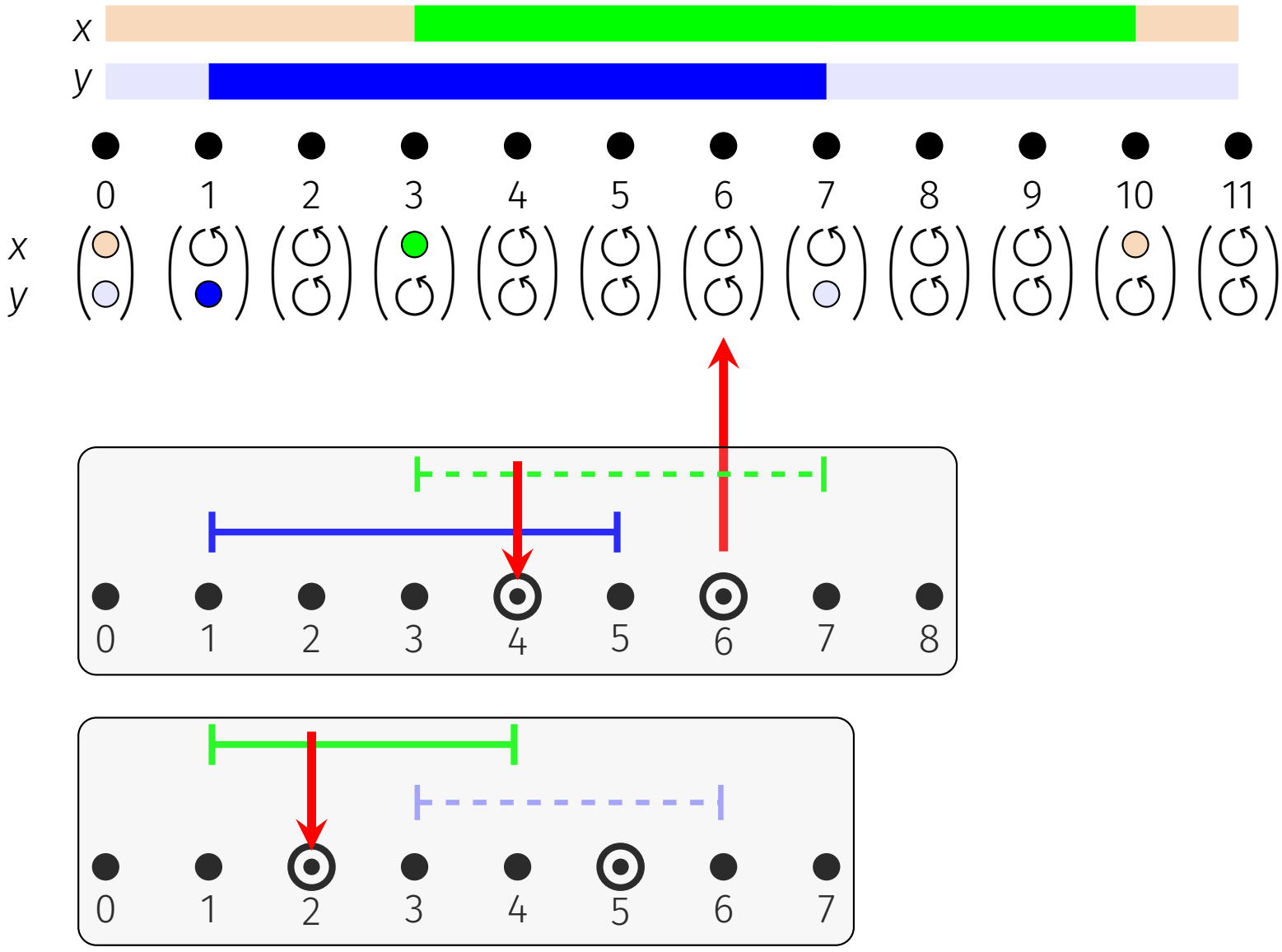
Viewpoints synchronization: failure



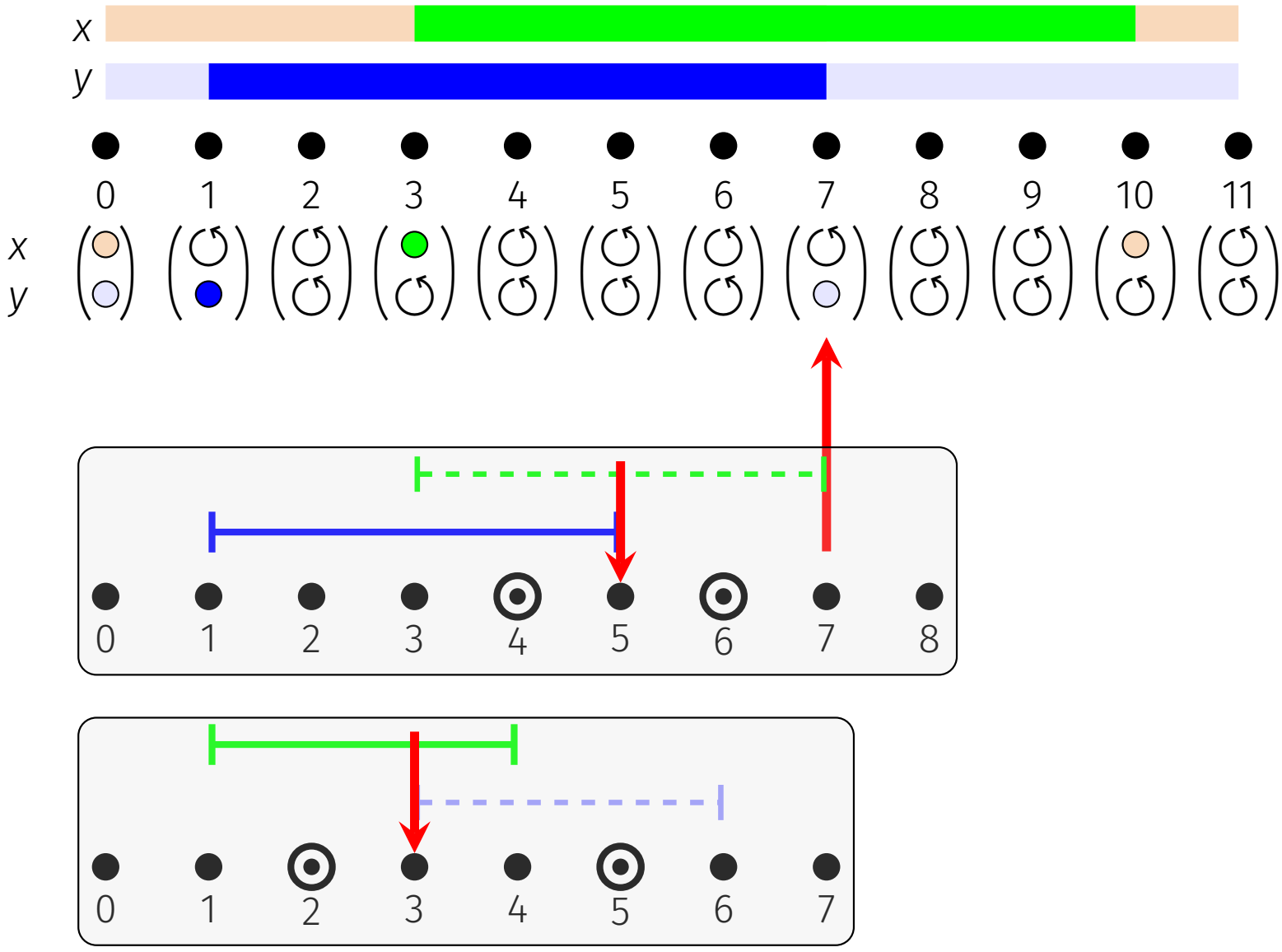
Viewpoints synchronization: failure



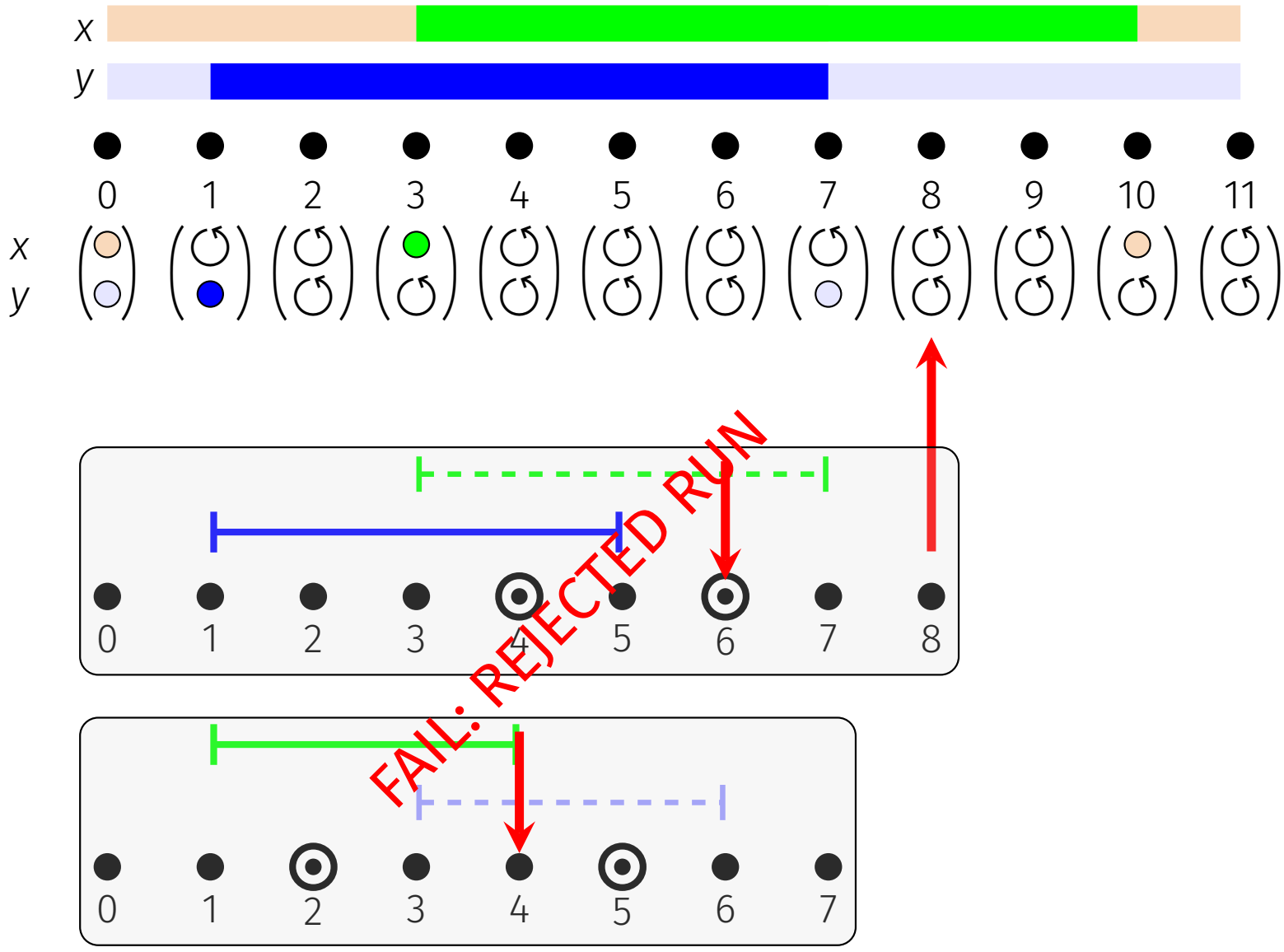
Viewpoints synchronization: failure



Viewpoints synchronization: failure



Viewpoints synchronization: failure



Complexity of the solution algorithm

The finite planning problem is reduced to the non-emptiness problem for an NFA $\mathcal{A}$ with $\mathcal{O}(2^{2^n})$ states.

- final state reachability in $\mathcal{A}$ can be performed on-the-fly and thus the solution algorithm has EXPSPACE complexity
- in a very similar way, the recurrent planning problem can be reduced to the non-emptiness problem for a Büchi automaton, and the solution algorithm has exactly the same complexity (the finite case can actually be viewed as a special case of the recurrent one).
- automata compactly represent all possible execution plans and their modularity makes the addition of constraints easy. Moreover, they allow one to benefit from (theoretical and practical) known results in automata theory.
- Last but not least, thanks to such an encoding into automata, the solution to many planning-related problems, like, for instance, minimality, boundedness, and synthesis / controllability, comes (almost) for free.

Acknowledgement. We would like to thank Pietro Sala, who produced a first version of the slides for his talk at KR 2018.

Timeline-based planning with uncertainty

Uncertainty

Most real-world scenarios demand to consider the unavoidable uncertainty that comes from the interaction with the environment.

- timeline-based planning systems handle **temporal uncertainty** very well by means of **flexible plans**;
- once a flexible plan for a problem is found, different (weak/strong/dynamic) control strategies may be employed to handle its execution;
- we focus here on **dynamically controllable** flexible plans.

Uncertainty (2)

We recently extended our results to timeline-based planning with uncertainty.

In doing that we addressed two issues of flexible plans:

1. general nondeterminism;
2. unnecessary replanning.

General nondeterminsm

The focus on temporal uncertainty means flexible plans cannot represent strategies involving non-temporal choices.

- However, general nondeterminism may arise from problems that apparently involve only temporal uncertainty.

General nondeterminism (2)

Suppose $x = (V, T, \gamma, D)$ with $V = \{0, 1, 2\}$ and $\gamma(0) = u$, and $D(0) = [0, 10]$, and consider:

$$a[x = 0] \rightarrow \exists b[x = 1] \,.\, \text{start}(a) \leq_{[1,5]} \text{end}(a) \wedge \text{end}(a) \leq_{[0,0]} \text{start}(b)$$
$$\vee \exists c[x = 2] \,.\, \text{start}(a) \leq_{[6,10]} \text{end}(a) \wedge \text{end}(a) \leq_{[0,0]} \text{start}(c)$$

In words: if $x = 0$ lasts between 1 and 5 units, $x = 1$ must follow, otherwise, $x = 2$ must follow.

- no flexible plan can represent the strategy needed to satisfy this rule
- hence this kind of problems have no way to represent their solutions

Unnecessary replanning

Flexible plans are sequential in nature:

- the flexible plans and their control strategy rely on some information about the behavior of external variables, up to temporal flexibility;
- what if, during execution, the observation turns out to be wrong?
- some systems detect the mismatch and perform a re-planning phase;
- however, re-planning is may be unfeasible in real-time scenarios.

Uncertainty

Hence a few questions arise:

- can timeline-based systems handle general nondeterminism?
- can we avoid the re-planning phase?
- what is the complexity of finding a strategy for these problems?

Game-theoretic approach

We propose to approach timeline-based planning with uncertainty in game-theoretic terms.

- We define the timeline-based planning game as a two-player game;
- the controller tries to satisfy the given set of synchronization rules;
- the environment plays arbitrarily.

Timeline-based games

A timeline-based game is a tuple $G = (SV_C, SV_E, S, D)$.

- Two players, Charlie (the controller) and Eve (the environment);
- players play by starting and ending tokens, building a plan;
- Charlie can start tokens for variables in SV_C, Eve those for variable in SV_E;
- Charlie decides when to stop controllable tokens, while Eve decides when to stop uncontrollable ones;
- Charlie tries to satisfy the set S of system rules, whatever the behavior of Eve;
- both players are assumed to play as to satisfy the set D of domain rules.

Strategies

We want to guarantee the existence of a winning strategy for Charlie.

- a strategy is a function σ that given a partial plan gives the next move of the player (i.e. which token to start/end, if any).
- a strategy σ is **admissible** if any play played according to σ will eventually satisfy $\mathcal{D}$.
- a strategy σ_C for Charlie is **winning** if, for any admissible strategy σ_E for Eve, any play played according to σ_C and σ_E is going to satisfy $S \cup D$.

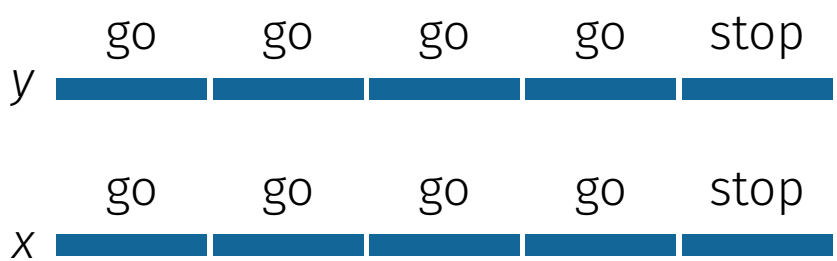
Example

Suppose two variables x and y can take the values *go* and *stop*.

- $x \in SV_C$, $y \in SV_E$
- Charlie can play **stop** only when Eve plays stop as well.
- We can win because **Eve must play stop sooner or later**

$$\mathcal{S} = \left\{ \begin{array}{r} a[x = stop] \rightarrow \exists b[y = stop] \,.\, end(b) = start(a) \\ \top \rightarrow \exists a[x = stop] \,.\, \top \end{array} \right\}$$

$$\mathcal{D} = \left\{ \top \rightarrow \exists a[y = stop] \,.\, \top \right\}$$



Advantages

Charlie has a winning strategy if he can play to satisfy the rules no matter what Eve does, supposing rules in D are satisfied.

- a general form of nondeterminism is handled in this way, not only temporal uncertainty
- no need for re-planning: the winning strategy already handles any scenario
- greater modeling flexibility: domain rules allow to describe complex interactions between the agent and the environment
- strictly more general than the approach based on dynamically controllable flexible plans
- but how hard is it to find such a strategy?

Finding a winning strategy

The decision procedure is based on ATL* model-checking over concurrent game structures (Rajeev Alur et al. 2002):

- concurrent game structures (CGS) are a general formalism to represent multi-agent concurrent systems.
- Alternating-time Temporal Logic (ATL) and its generalization ATL*, are interpreted over CGSs;
- ATL and ATL* are similar to CTL and CTL*, but branching modalities quantify over paths played according to specific strategies of a specific set of players;

ATL and ATL*

The Alternating Temporal Logic (ATL) and its extension ATL* are **strategic** temporal logics:

- interpreted over **game structures**
- can quantify over paths of the structures played according to some player's strategy

ATL* syntax

An ATL* formula is a **state formula** as define below:

$$\phi := p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \langle\langle A \rangle\rangle \psi \qquad \text{state formulas}$$

$$\psi := \phi \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \mathsf{X}\,\psi_1 \mid \psi_1 \,\mathcal{U}\, \psi_2 \qquad \text{path formulas}$$

where A is a set of **players**.

ATL and ATL*

Strategy quantifier

$$\langle\!\langle A \rangle\!\rangle \psi$$

There exists a strategy for players in A such that ψ holds in **all** paths played according to said strategy.

Example

$$\langle\!\langle 1 \rangle\!\rangle (\mathsf{G}\,\mathsf{F}\, req \rightarrow \mathsf{G}\,\mathsf{F}\, grant)$$

Finding a winning strategy (2)

A doubly exponential sized (turn-based synchronous) CGS can be built to represent the game;

- nodes are partial plans, edges labeled by players moves;
- particular attention to guarantee a finite state space;
- states where D and S are satisfied are labelled, respectively, by proposition letters d and w;
- The winning condition is then encoded in ATL* as follows:

$$\phi \equiv \langle\langle 1 \rangle\rangle \big(Fd \rightarrow Fw \big)$$

- Model-checking a fixed-size ATL* formula over a CGS can be done in polynomial time, hence the 2EXPTIME complexity.

Making the state space finite

Abstractly, the state space of the game is infinite:

- each synchronization rule can in principle be affected by anything happening arbitrarily far in the past or in the future
- but we do not really need to keep track of all the history
- matching records allow us to represent compactly the game history
  - we can decide if $\overline{\mu}$ satisfies the rules looking only at $[\overline{\mu}]$;
  - given a round ρ of the game we can build $[\rho(\overline{\mu})]$ from $[\overline{\mu}]$;
- then, states of the game are all the possible $[\overline{\mu}]$
- size of $[\overline{\mu}]$ is exponential, hence the state space is doubly exponential.

Hardness

The 2EXPTIME upper bound is strict:

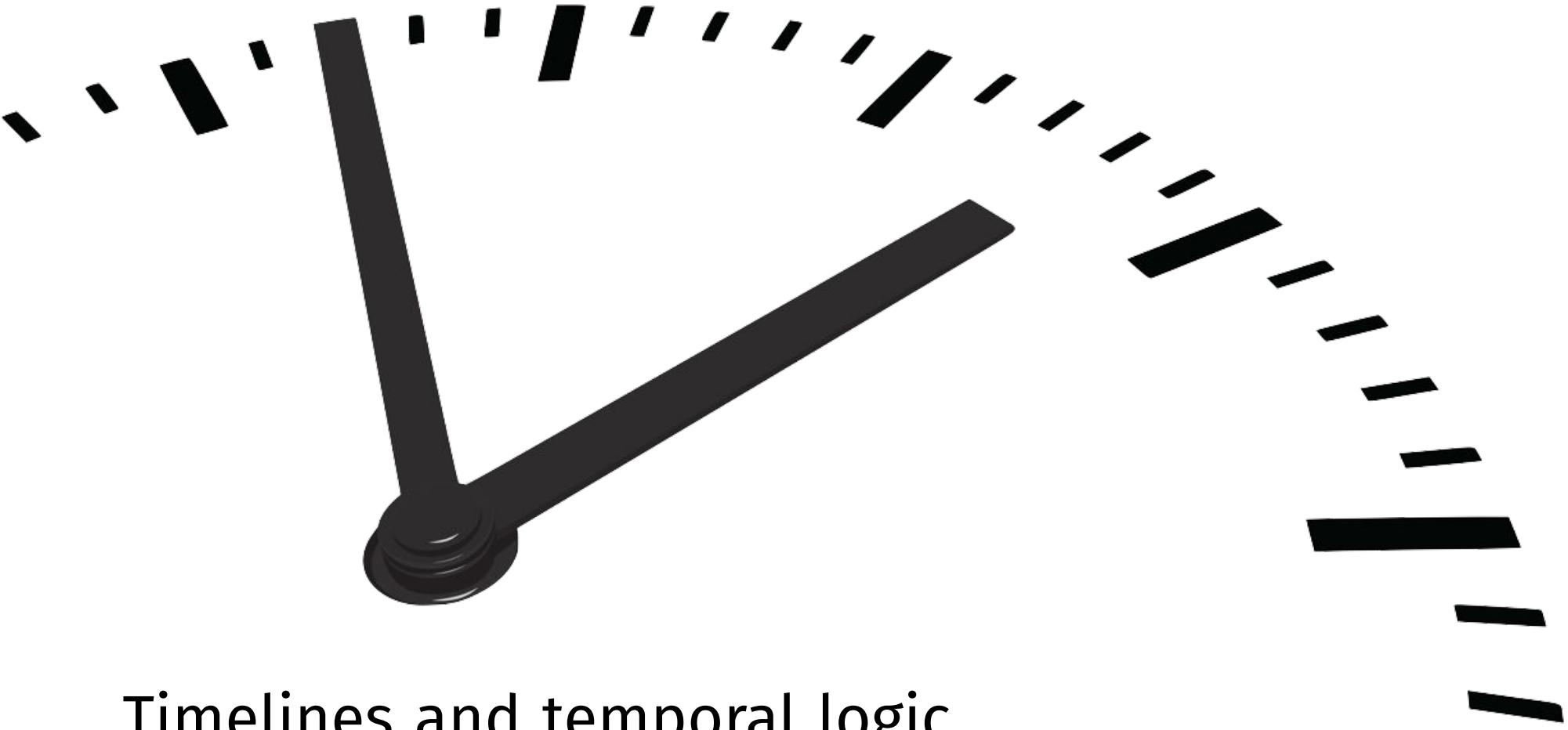
- deciding the existence of winning strategies is 2EXPTIME-complete!
- hardness proved by reduction from **domino tiling games** (Chlebus 1986)
- same idea of the reduction of the plan existence problem from coridor tiling games, adapted to the two-player setting

This is not the end

We know how to decide if a winning strategy exists.

- How to concretely implement such a strategy?

The synthesis of controllers for these games is still an open problem.



Timelines and temporal logic

Logical characterization of planning problems

In the next step we wanted to capture timeline-based planning problems with a well-behaved logical language.
Why?

- Logical characterizations are available both for STRIPS-like planning (Cialdea Mayer et al. 2007) and for temporal planning (Cimatti et al. 2017).
- Easier to compare the expressiveness of different languages if they are reduced to commonly known logics.
- Easier to think of the synthesis of controllers if the specification language is a well-defined logical formalism.

The result

We devised a new logic, that we called **TPTL_b+P**, showing that:

- its satisfiability problem is **EXPSPACE-complete**, and
- it can **capture** the restricted kind of timeline-based planning problem studied in (Gigante et al. 2016):
  - given a problem P, there is a TPTL_b+Pformula Φ_P such that Φ_P is satisfiable iff there is a solution plan for P.

Della Monica, Gigante, Montanari, Sala, and Sciavicco (2017)

Dario Della Monica, Nicola Gigante, Angelo Montanari, Pietro Sala, and Guido Sciavicco (2017). "Bounded Timed Propositional Temporal Logic with Past Captures Timeline-based Planning with Bounded Constraints." In: *Proc. of the 26th International Joint Conference on Artificial Intelligence.* pages 1008–1014

Timed Propositional Temporal Logic

The TPTL logic was originally introduced for the verification of real-time systems (R. Alur and Henzinger 1994).

$$\phi := p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \overbrace{x.\phi_1}^{\text{freeze quantifier}} \mid \overbrace{x \leq y + c \mid x \leq c}^{\text{timed constraints } c \in \mathbb{Z}} \mid \underbrace{\mathrm{X}\,\phi_1 \mid \phi_1 \,\mathcal{U}\, \phi_2}_{\text{Linear Temporal Logic temporal operators}}$$

Formulae are interpreted over **timed words** (σ, τ), *i.e.*, each state σ_i is associated with a **timestamp** τ_i.

Timed Propositional Temporal Logic

The TPTL logic was originally introduced for the verification of real-time systems (R. Alur and Henzinger 1994).

$$\phi := p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \overbrace{x.\phi_1}^{\text{freeze quantifier}} \mid \overbrace{x \leq y + c \mid x \leq c}^{\text{timed constraints } c \in \mathbb{Z}} \mid \underbrace{\mathrm{X}\,\phi_1 \mid \phi_1\,\mathcal{U}\,\phi_2}_{\text{Linear Temporal Logic temporal operators}}$$

The **freeze quantifier** binds the timestamp of the current state to a variable, used in the evaluation of the **timed constraints**.

Timed Propositional Temporal Logic

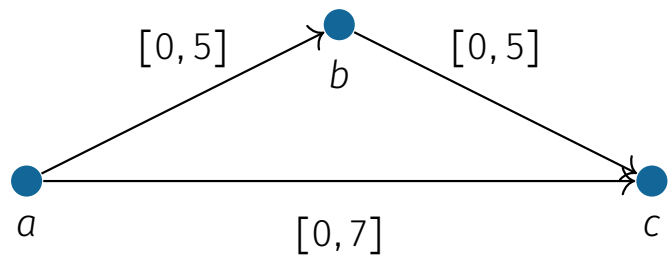
The TPTL logic was originally introduced for the verification of real-time systems (R. Alur and Henzinger 1994).

$$\phi := p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \overbrace{x.\phi_1}^{\text{freeze quantifier}} \mid \overbrace{x \leq y + c \mid x \leq c}^{\text{timed constraints } c \in \mathbb{Z}} \mid \underbrace{\mathrm{X}\,\phi_1 \mid \phi_1\,\mathcal{U}\,\phi_2}_{\text{Linear Temporal Logic temporal operators}}$$

The satisfiability problem for TPTL is EXPSPACE-complete.

Why TPTL?

The freeze quantifier and timed constraints allow one to **compactly** express constraints of this kind:

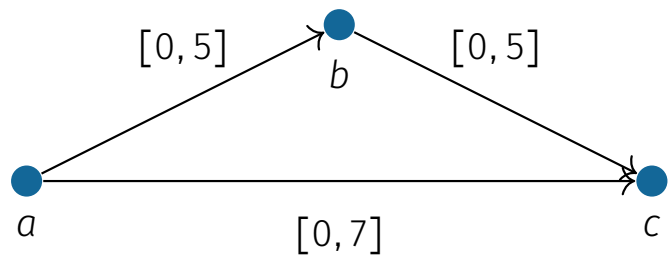


$$a[\ldots] \rightarrow \exists b[\ldots]c[\ldots]\,.\ \mathrm{start}(a) \leq_{[0,5]} \mathrm{start}(b) \wedge \mathrm{start}(b) \leq_{[0,5]} \mathrm{start}(c) \wedge \mathrm{start}(a) \leq_{[0,7]} \mathrm{start}(c)$$

$$\mathsf{G}\, t_a.(p_a \rightarrow \mathsf{F}\, t_b.(p_b \wedge t_b \leq t_a + 5 \wedge \mathsf{F}\, t_c.(p_c \wedge t_c \leq t_b + 5 \wedge t_c \leq t_a + 7)))$$

Why TPTL?

The freeze quantifier and timed constraints allow one to **compactly** express constraints of this kind:



$$b[\ldots] \rightarrow \exists a[\ldots]c[\ldots]\,.\ \mathrm{start}(a) \leq_{[0,5]} \mathrm{start}(b) \wedge \mathrm{start}(b) \leq_{[0,5]} \mathrm{start}(c) \wedge \mathrm{start}(a) \leq_{[0,7]} \mathrm{start}(c)$$

But what if the trigger was token b?

TPTL with Past

We need **past operators** to encode synchronization rules,
which can talk about future and past interchangeably.

freeze quantifier ↓ timed constraints $c \in \mathbb{Z}$ ↓

$$\phi := p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \overbrace{x.\phi_1}^{\text{freeze quantifier}} \mid \overbrace{x \leq y + c \mid x \leq c}^{\text{timed constraints } c \in \mathbb{Z}}$$

$$\underbrace{\mid \mathsf{X}\,\phi_1 \mid \phi_1\,\mathcal{U}\,\phi_2 \mid \mathsf{Y}\,\phi_1 \mid \phi_1\,\mathcal{S}\,\phi_2}_{\text{future (X, } \mathcal{U}\text{) and past (Y, } \mathcal{S}\text{) temporal operators}}$$

Unfortunately, past operators make the satisfiability problem become **non elementary**.

TPTL with Past

We need **past operators** to encode synchronization rules, which can talk about future and past interchangeably.

$$\phi := p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \overbrace{x.\phi_1}^{\text{freeze quantifier}} \mid \overbrace{x \leq y + c \mid x \leq c}^{\text{timed constraints } c\in\mathbb{Z}}$$
$$\mid \underbrace{\mathsf{X}\,\phi_1 \mid \phi_1\,\mathcal{U}\,\phi_2 \mid \mathsf{Y}\,\phi_1 \mid \phi_1\,\mathcal{S}\,\phi_2}_{\text{future (X, } \mathcal{U}\text{) and past (Y, } \mathcal{S}\text{) temporal operators}}$$

Why? Together with the freeze quantifier, we can simulate first-order existential quantification.

$$\exists x \phi(x) \equiv y.\, \mathsf{F}\, x.\, \mathsf{P}\, z.(y = z \wedge \phi(x))$$

Bounded TPTL with Past

To recover an acceptable complexity while being still able to use past operators, we **restricted** the temporal operators with a bound w.

$$\phi := p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \overbrace{x.\phi_1}^{\text{freeze quantifier}} \mid \overbrace{x \leq y + c \mid x \leq c}^{\text{timed constraints } c\in\mathbb{Z}}$$
$$\underbrace{\mid \mathsf{X}_w\,\phi_1 \mid \phi_1\,\mathcal{U}_w\,\phi_2 \mid \mathsf{Y}_w\,\phi_1 \mid \phi_1\,\mathcal{S}_w\,\phi_2}_{\text{MTL-like temporal operators } w\in\mathbb{N}\cup\{\infty\}}$$

The bound limits the **timestamp** of the states,
e.g., $X_5\phi$ holds at state σ_i iff ϕ holds at σ_{i+1} and $\tau_{i+1} - \tau_i \leq 5$.

Bounded TPTL with Past

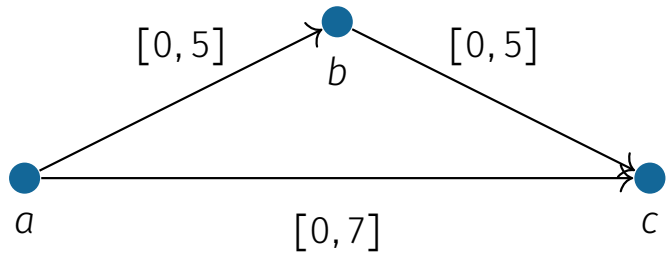
To recover an acceptable complexity while being still able to use past operators, we **restricted** the temporal operators with a bound w.

$$\phi := p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \overbrace{x.\phi_1}^{\text{freeze quantifier}} \mid \overbrace{x \leq y + c \mid x \leq c}^{\text{timed constraints } c \in \mathbb{Z}}$$
$$\underbrace{\mid \mathsf{X}_w\, \phi_1 \mid \phi_1\, \mathcal{U}_w\, \phi_2 \mid \mathsf{Y}_w\, \phi_1 \mid \phi_1\, \mathcal{S}_w\, \phi_2}_{\text{MTL-like temporal operators } w \in \mathbb{N} \cup \{\infty\}}$$

The bound can be **omitted** (*i.e.*, $w = \infty$) only if the underlying formulae are **closed**, *i.e.*, they do not refer to variables quantified outside.

$\mathrm{TPTL_b}$+PExample

Now the previous example can be encoded in both cases:



$$a[\ldots] \rightarrow \exists b[\ldots]c[\ldots]\,.\ \mathrm{start}(a) \leq_{[0,5]} \mathrm{start}(b) \wedge \mathrm{start}(b) \leq_{[0,5]} \mathrm{start}(c) \wedge \mathrm{start}(a) \leq_{[0,7]} \mathrm{start}(c)$$

$$\mathsf{G}\, t_a.(p_a \rightarrow \mathsf{F}_{10}\, \mathsf{P}_{10}\, t_b.(p_b \wedge \mathsf{F}_{10}\, \mathsf{P}_{10}\, t_c.(p_c \wedge t_b \leq t_a + 5 \wedge t_c \leq t_b + 5 \wedge t_c \leq t_a + 7)))$$

The encoding

Thus, the encoding generally works this way. Given a rule:

$$a[x = v] \rightarrow \exists b[y = v']c[z = v''] \,.\, \mathcal{C}$$

- Globally, whenever the head of some rule is satisfied, we look for the needed events across the entire model, with **bounded** temporal operators.
  - It is essential here to disallow **unbounded** atoms in rules.
- Once the correct events have been identified, we constrain their position and distances using timed constraints.

Complexity of TPTL_b+P

The complexity of TPTL_b+Pis proved by adapting the original **tableau method** provided by R. Alur and Henzinger for TPTL.

- In turn, a clever adaptation of the classic graph-shaped tableau for LTL;
- simply adding past operators does not work, as the tableau becomes potentially infinite;
- the bounds on TPTL_b+Ptemporal operators allow to adapt the idea and support past operators.

Thank you

Questions?

Bibliography

Alur, R. and T. A. Henzinger (1994). "A Really Temporal Logic." In: *Journal of the ACM* 41.1, pp. 181–204. DOI: 10.1145/174644.174651. URL: http://doi.acm.org/10.1145/174644.174651.

Alur, Rajeev, Thomas A. Henzinger, and Orna Kupferman (2002). "Alternating-time Temporal Logic." In: *Journal of the ACM* 49.5, pp. 672–713.

Bozzelli, Laura, Alberto Molinari, Angelo Montanari, and Adriano Peron (2018a). "Complexity of Timeline-Based Planning over Dense Temporal Domains: Exploring the Middle Ground." In: *Proceedings Ninth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2018, Saarbrücken, Germany, 26-28th September 2018*. Ed. by Andrea Orlandini and Martin Zimmermann. Vol. 277. EPTCS, pp. 191–205. DOI: 10.4204/EPTCS.277.14. URL: https://doi.org/10.4204/EPTCS.277.14.

— (2018b). "Decidability and Complexity of Timeline-Based Planning over Dense Temporal Domains." In: *KR 2018*. Ed. by Michael Thielscher, Francesca Toni, and Frank Wolter. AAAI Press, pp. 627–628. ISBN: 978-1-57735-803-9. URL: https://aaai.org/ocs/index.php/KR/KR18/paper/view/17995.

Bozzelli, Laura, Alberto Molinari, Angelo Montanari, Adriano Peron, and Gerhard J. Woeginger (2018). "Timeline-Based Planning over Dense Temporal Domains with Trigger-less Rules is NP-Complete." In: *ICTCS 2018*. Ed. by Alessandro Aldini and Marco Bernardo. Vol. 2243. CEUR Workshop Proceedings. CEUR-WS.org, pp. 116–127. URL: http://ceur-ws.org/Vol-2243/paper11.pdf.

Chlebus, Bogdan S. (1986). "Domino-Tiling Games." In: *Journal of Computer and System Sciences* 32.3, pp. 374–392. DOI: 10.1016/0022-0000(86)90036-X.

Cialdea Mayer, Marta, Carla Limongelli, Andrea Orlandini, and Valentina Poggioni (2007). "Linear Temporal Logic as an Executable Semantics for Planning Languages." In: *Journal of Logic, Language and Information* 16.1, pp. 63–89.

Cialdea Mayer, Marta, Andrea Orlandini, and Alessandro Umbrico (2016). "Planning and Execution with Flexible Timelines: a Formal Account." In: *Acta Informatica* 53.6-8, pp. 649–680.

Cimatti, A., A. Micheli, and M. Roveri (2017). "Validating Domains and Plans for Temporal Planning via Encoding into Infinite-State Linear Temporal Logic." In: *Proc. of the 31st AAAI Conference on Artificial Intelligence*, pp. 3547–3554.

Della Monica, Dario, Nicola Gigante, Angelo Montanari, and Pietro Sala (2018). "A novel automata-theoretic approach to timeline-based planning." In: *KR 2018*.

Della Monica, Dario, Nicola Gigante, Angelo Montanari, Pietro Sala, and Guido Sciavicco (2017). "Bounded Timed Propositional Temporal Logic with Past Captures Timeline-based Planning with Bounded Constraints." In: *Proc. of the 26th International Joint Conference on Artificial Intelligence*. pages 1008–1014.

Bibliography (2)

Gigante, Nicola, Angelo Montanari, Marta Cialdea Mayer, and Andrea Orlandini (2016). "Timelines are Expressive Enough to Capture Action-Based Temporal Planning." In: *Proc. of the 23rd International Symposium on Temporal Representation and Reasoning*, pp. 100–109.

— (2017). "Complexity of Timeline-based Planning." In: *Proc. of the 27th International Conference on Automated Planning and Scheduling*, pages 116–124.

Rintanen, Jussi (2007). "Complexity of Concurrent Temporal Planning." In: *Proc. of the 17th International Conference on Automated Planning and Scheduling*, pp. 280–287.

Smith, David E., Jeremy Frank, and William Cushing (2008). "The ANML Language." In: *Proceedinsgs of the ICAPS 2008 Workshop on Knowledge Engineering for Planning and Scheduling*.