Contrôle et jeux sur les graphes

Formation sur les Systèmes à Événements Discrets (SED)

2e édition
Mars 2025
Nantes

Société d'Automatique,
de Génie Industriel & de Productique

Introduction

Introduction

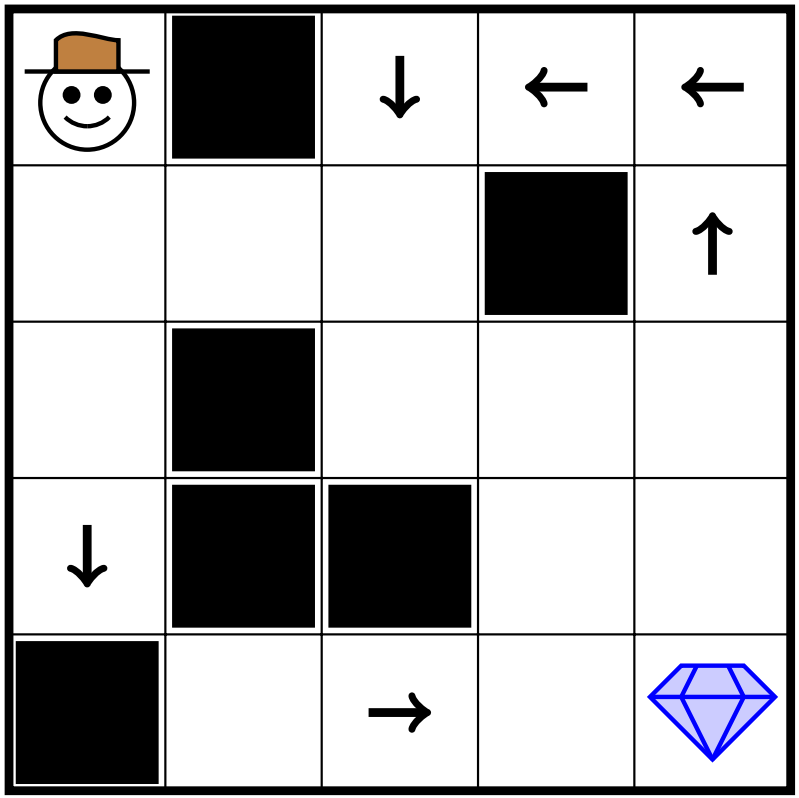
- Supervisory control (Ramadge and Wonham) : control a language ;
- Here, inspiration from game theory and extensive-form games ;
- But our games can be graphs, not only trees !

Introduction

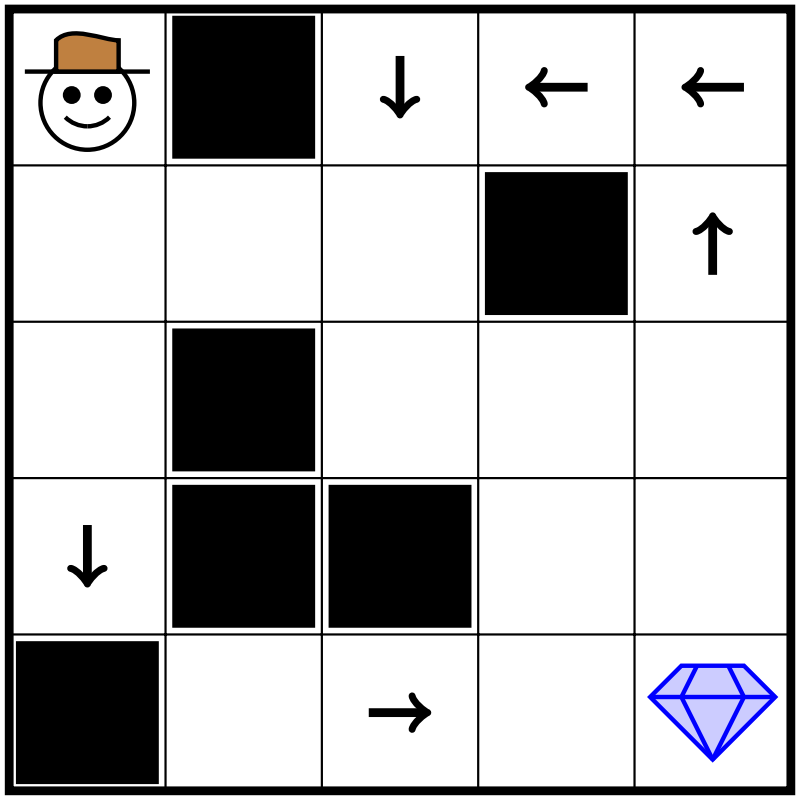
- Supervisory control (Ramadge and Wonham) : control a language ;
- Here, inspiration from game theory and extensive-form games ;
- But our games can be graphs, not only trees !
- Here :
  - One or two-player, win/lose games ;
  - The winning condition is reachability or safety ;
  - Without or with explicit timing constraints.

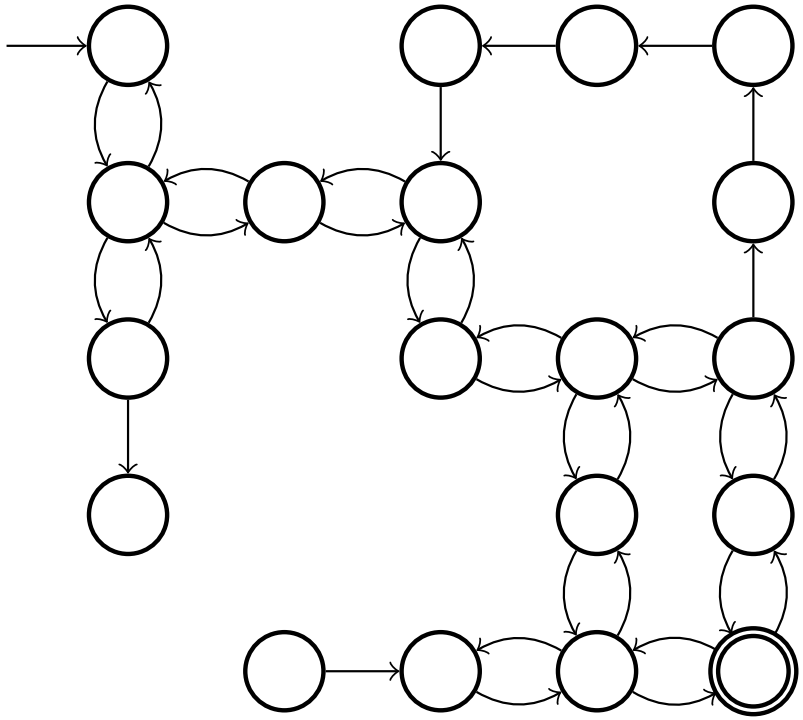
One-player games and verification

Reachability



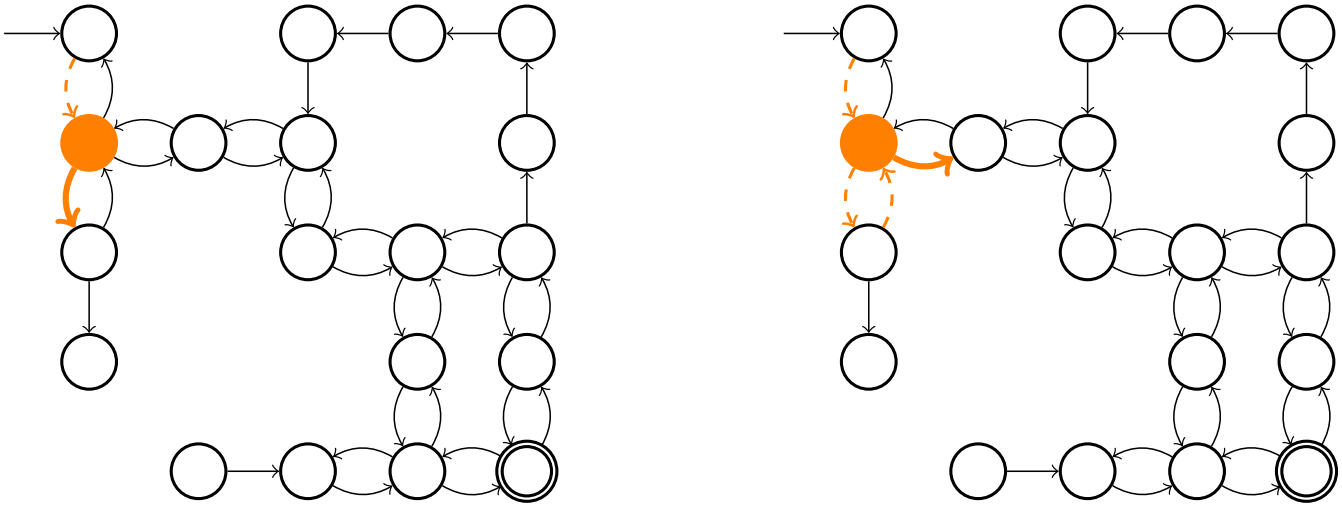
Reachability





Strategies

- A strategy associates an edge to histories of past states



- A positional/memoryless strategy depends only on the last state in the history : the current state.

Playing and winning

- When the unique player plays according to their fixed strategy, we obtain a unique path in the automaton, called a play ;

Playing and winning

- When the unique player plays according to their fixed strategy, we obtain a unique path in the automaton, called a play ;
- A play is maximal if it is infinite or cannot be extended by the controller ;

Playing and winning

- When the unique player plays according to their fixed strategy, we obtain a unique path in the automaton, called a play ;
- A play is maximal if it is infinite or cannot be extended by the controller ;
- A maximal play is winning if it intersects the goal ;

Playing and winning

- When the unique player plays according to their fixed strategy, we obtain a unique path in the automaton, called a play ;
- A play is maximal if it is infinite or cannot be extended by the controller ;
- A maximal play is winning if it intersects the goal ;
- A strategy is winning if the corresponding maximal play is winning ;

Playing and winning

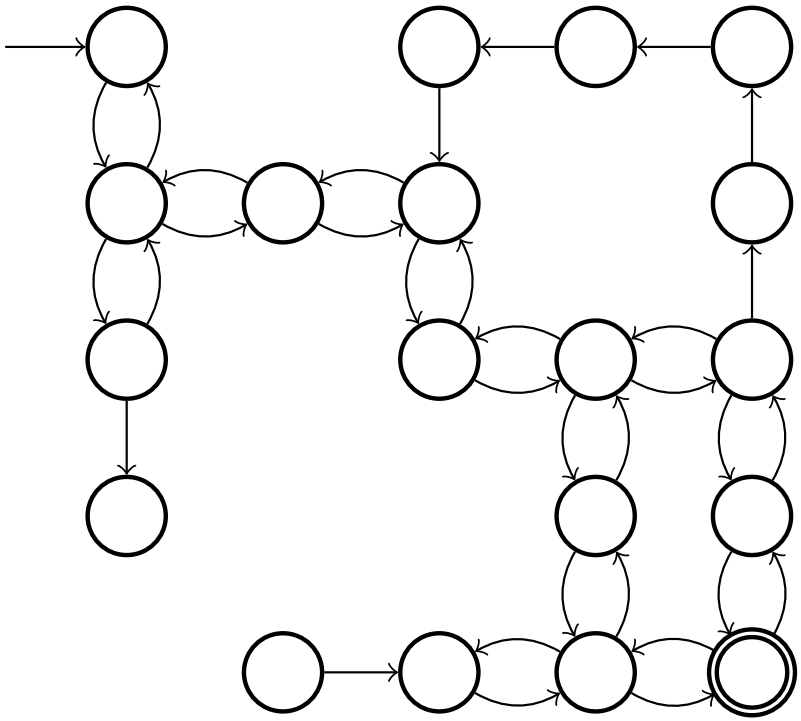
- When the unique player plays according to their fixed strategy, we obtain a unique path in the automaton, called a play ;
- A play is maximal if it is infinite or cannot be extended by the controller ;
- A maximal play is winning if it intersects the goal ;
- A strategy is winning if the corresponding maximal play is winning ;
- From a winning play we can compute a winning strategy : play the edge that allows to stay on the path.

Playing and winning

- When the unique player plays according to their fixed strategy, we obtain a unique path in the automaton, called a play ;
- A play is maximal if it is infinite or cannot be extended by the controller ;
- A maximal play is winning if it intersects the goal ;
- A strategy is winning if the corresponding maximal play is winning ;
- From a winning play we can compute a winning strategy : play the edge that allows to stay on the path.

So we only need to find a winning play : a path to the goal !
Let us use Breadth-first Search (symbolic version).

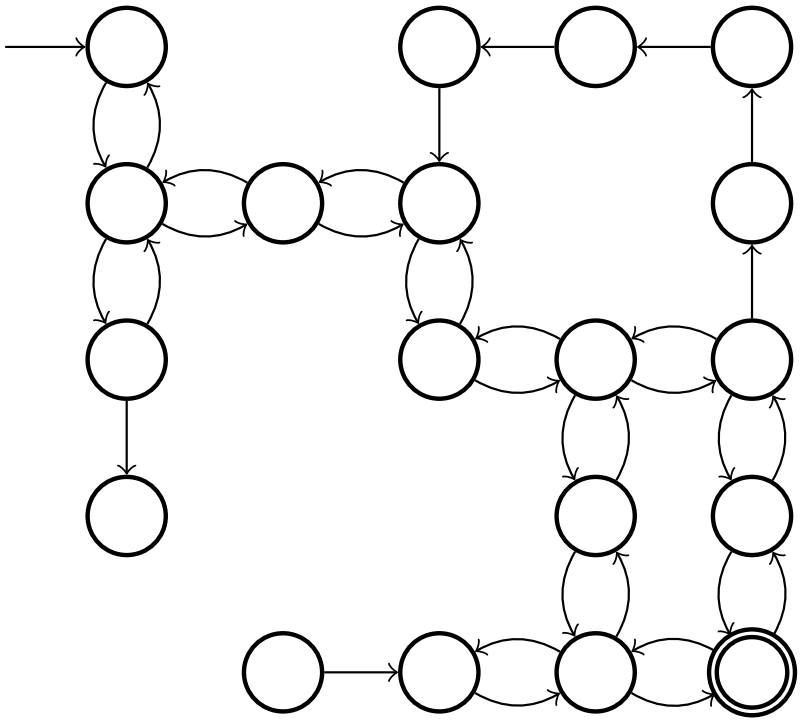
Reachable states



For all $X \subseteq S$ define :

$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

Reachable states

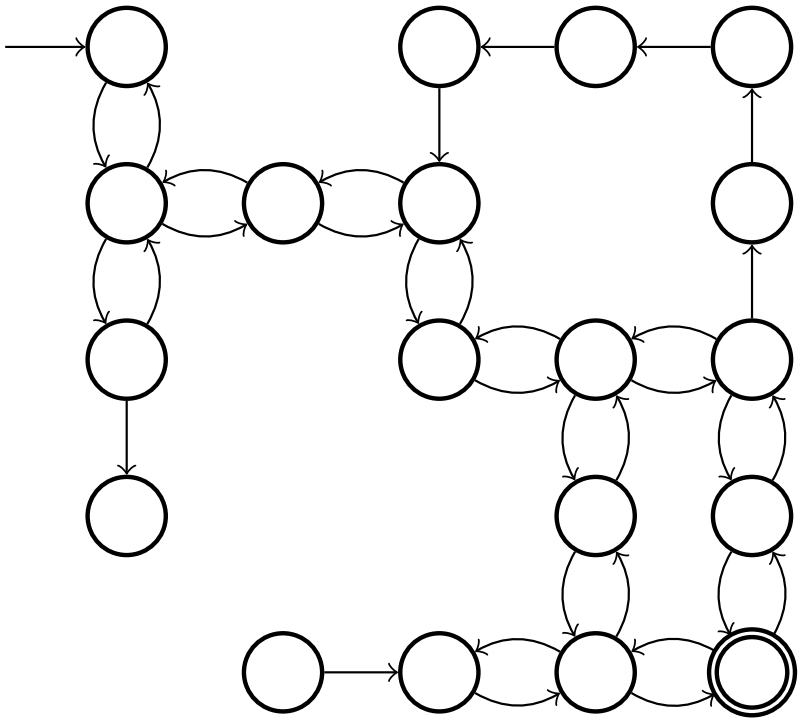


all states

For all $X \subseteq S$ define :

$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

Reachable states



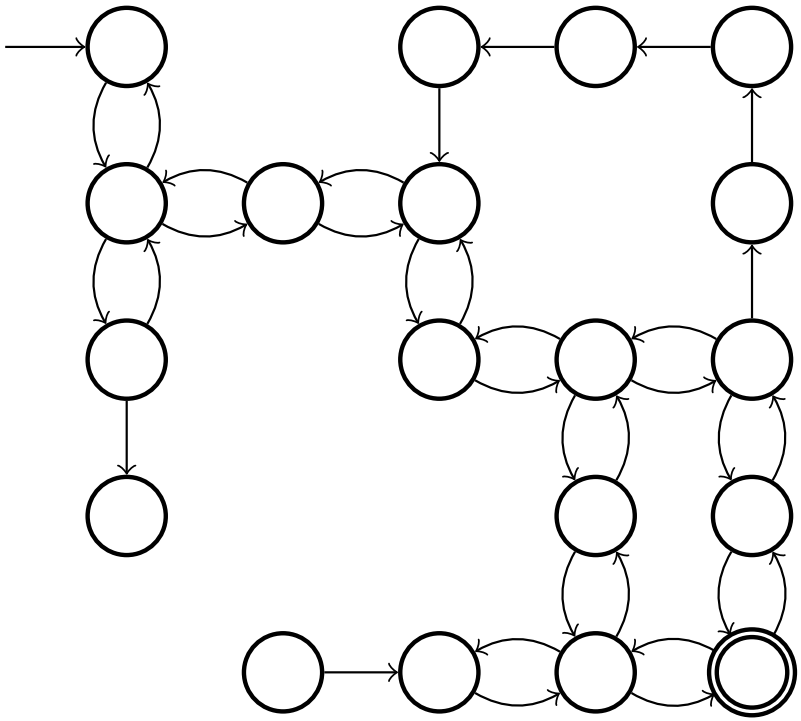
all states

For all $X \subseteq S$ define :

$\text{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

$$X_0 = S_0$$
$$X_{n+1} = X_n \cup \text{Post}(X_n)$$

Reachable states



all states

For all $X \subseteq S$ define :

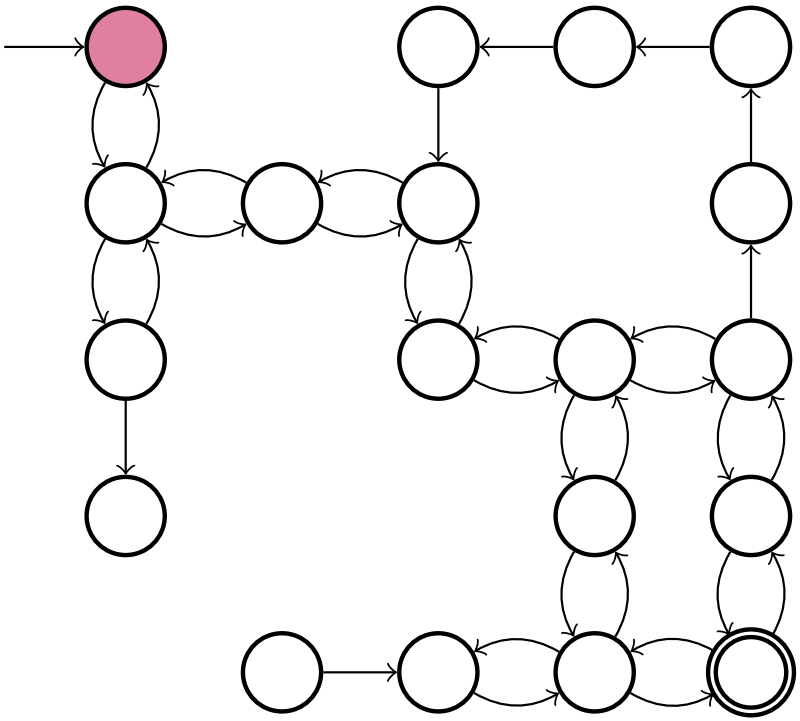
$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

initial states

$X_0 = S_0$

$X_{n+1} = X_n \cup \mathrm{Post}(X_n)$

Reachable states



all states

For all $X \subseteq S$ define :

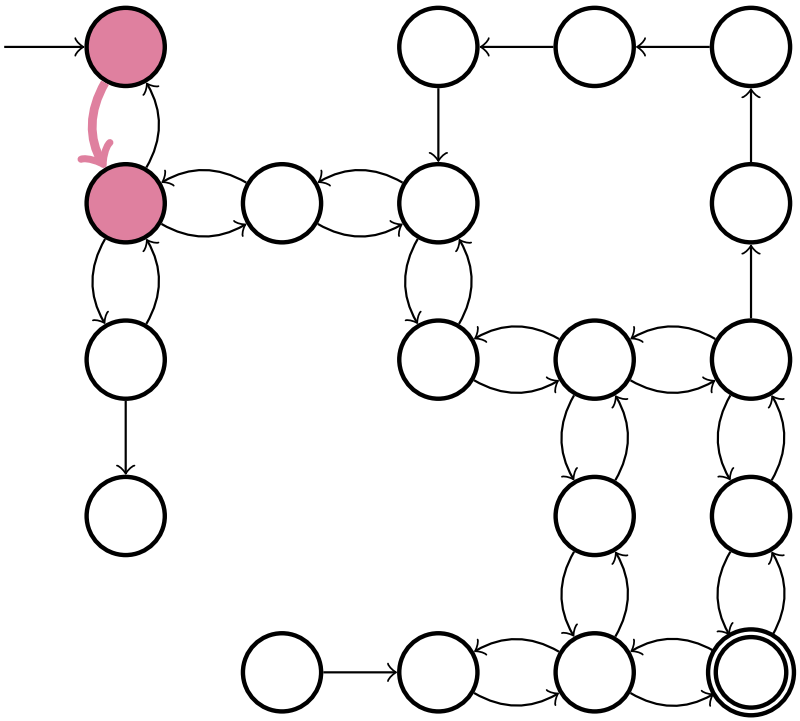
$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

initial states

$X_0 = S_0$

$X_{n+1} = X_n \cup \mathrm{Post}(X_n)$

Reachable states



all states

For all $X \subseteq S$ define :

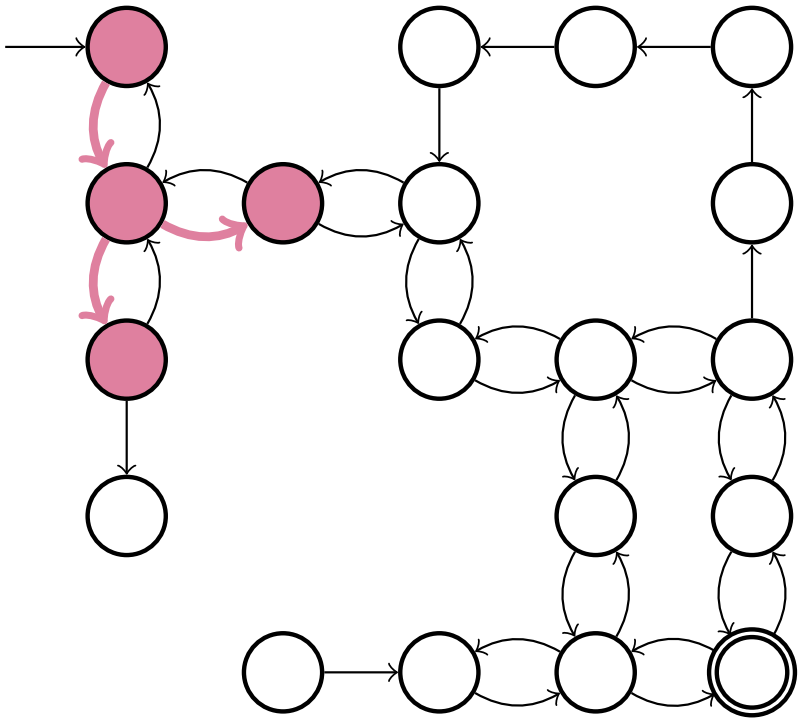
$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

initial states

$X_0 = S_0$

$X_{n+1} = X_n \cup \mathrm{Post}(X_n)$

Reachable states



all states

For all $X \subseteq S$ define :

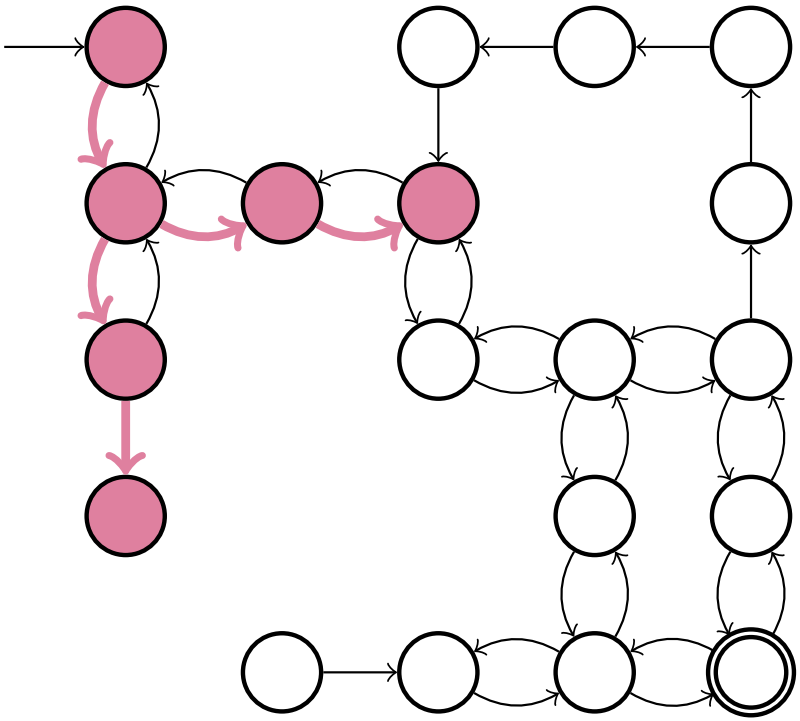
$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

initial states

$X_0 = S_0$

$X_{n+1} = X_n \cup \mathrm{Post}(X_n)$

Reachable states



all states

For all $X \subseteq S$ define :

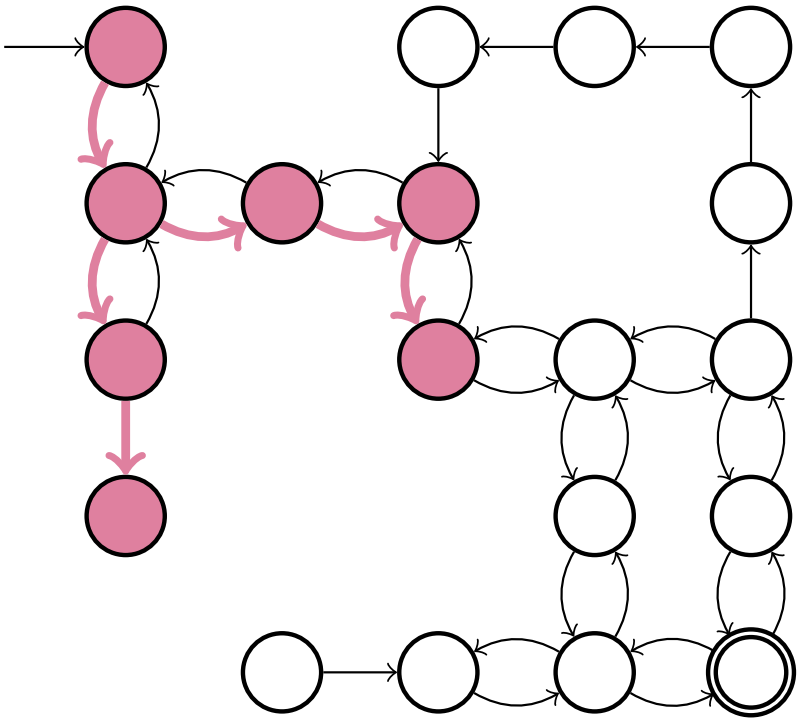
$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

initial states

$X_0 = S_0$

$X_{n+1} = X_n \cup \mathrm{Post}(X_n)$

Reachable states



all states

For all $X \subseteq S$ define :

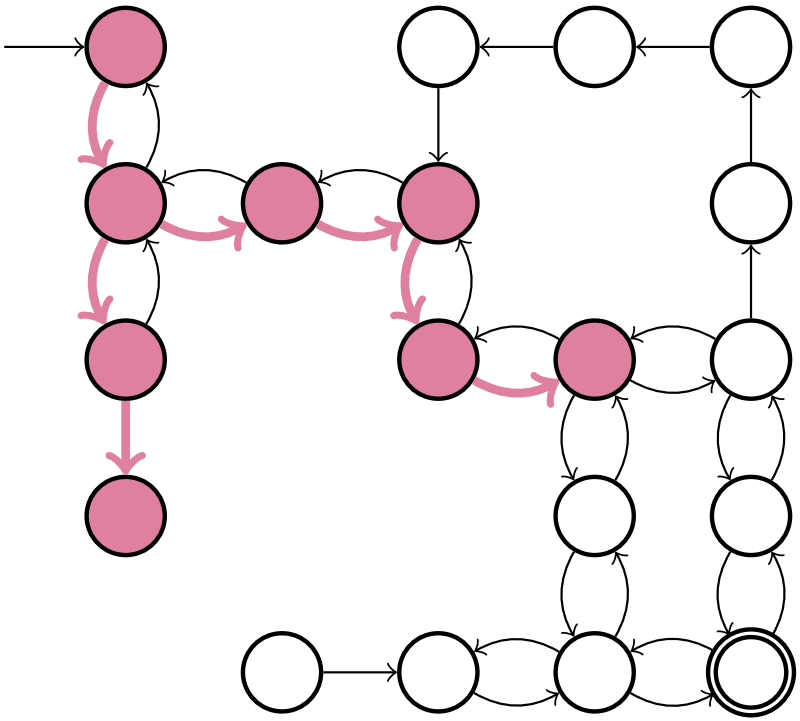
$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

initial states

$X_0 = S_0$

$X_{n+1} = X_n \cup \mathrm{Post}(X_n)$

Reachable states



all states

For all $X \subseteq S$ define :

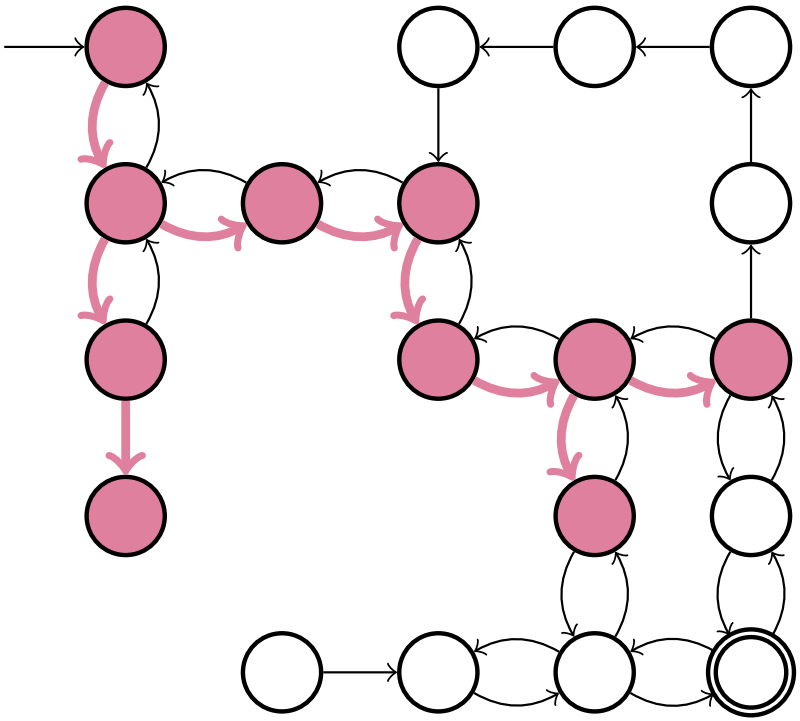
$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

initial states

$X_0 = S_0$

$X_{n+1} = X_n \cup \mathrm{Post}(X_n)$

Reachable states



all states

For all $X \subseteq S$ define :

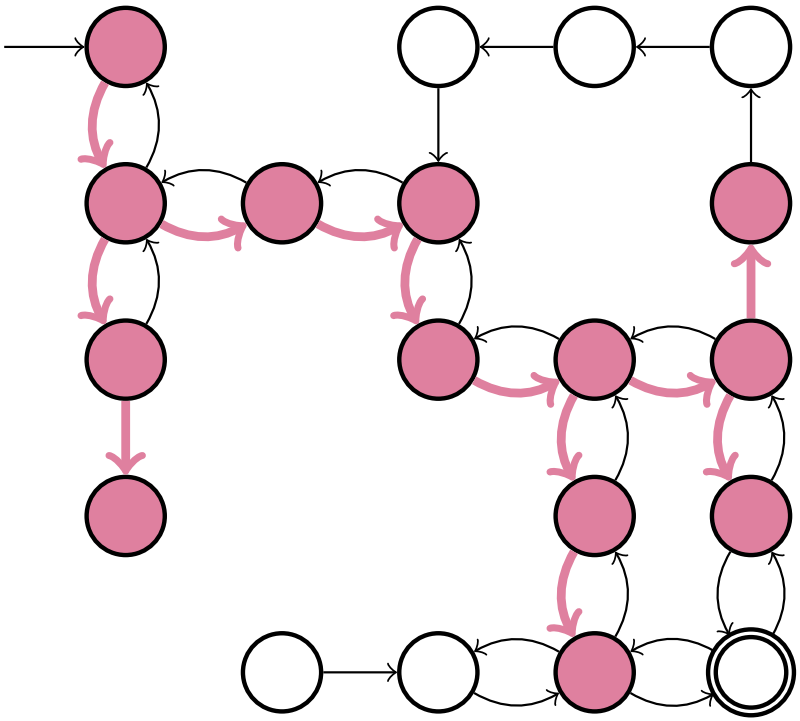
$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

initial states

$X_0 = S_0$

$X_{n+1} = X_n \cup \mathrm{Post}(X_n)$

Reachable states



all states

For all $X \subseteq S$ define :

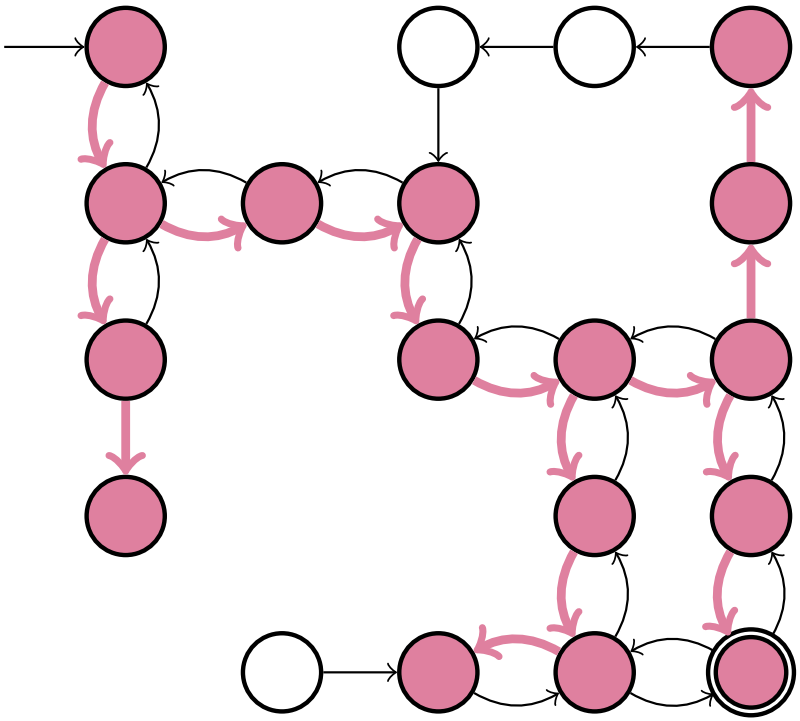
$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

initial states

$X_0 = S_0$

$X_{n+1} = X_n \cup \mathrm{Post}(X_n)$

Reachable states



all states

For all $X \subseteq S$ define :

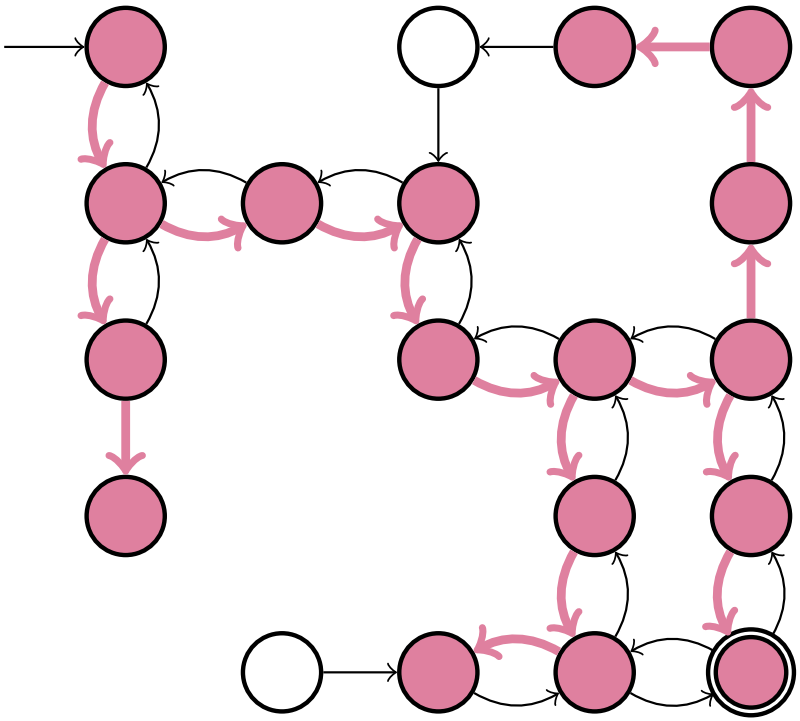
$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

initial states

$X_0 = S_0$

$X_{n+1} = X_n \cup \mathrm{Post}(X_n)$

Reachable states



all states

For all $X \subseteq S$ define :

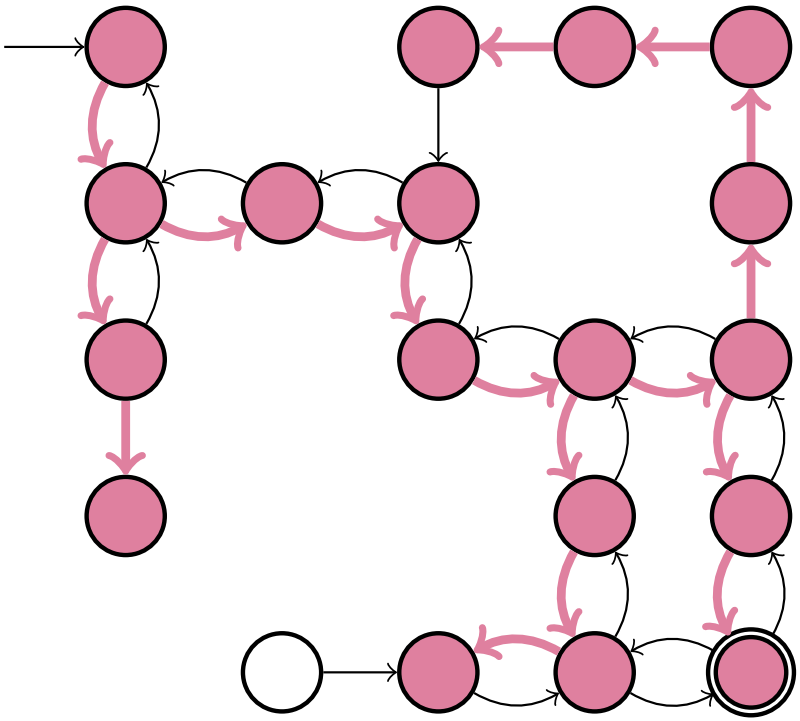
$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

initial states

$X_0 = S_0$

$X_{n+1} = X_n \cup \mathrm{Post}(X_n)$

Reachable states



all states

For all $X \subseteq S$ define :

$\mathrm{Post}(X) = \{s' \in S | \exists s \in X \text{ s. t. } s \to s'\}$

initial states

$X_0 = S_0$

$X_{n+1} = X_n \cup \mathrm{Post}(X_n)$

From reachable states to the winning strategy

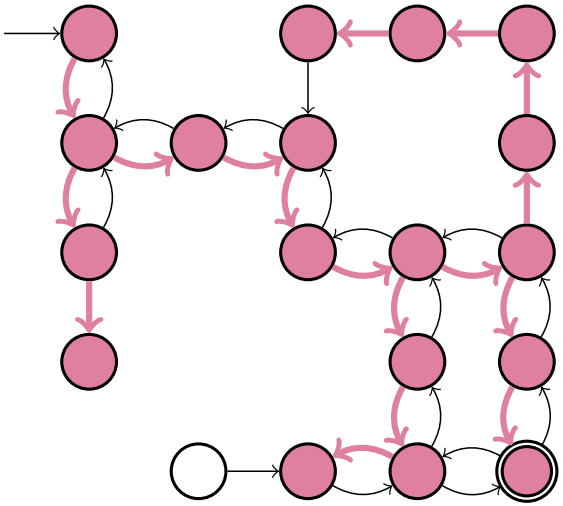
- We have computed states reachable from S_0 ;

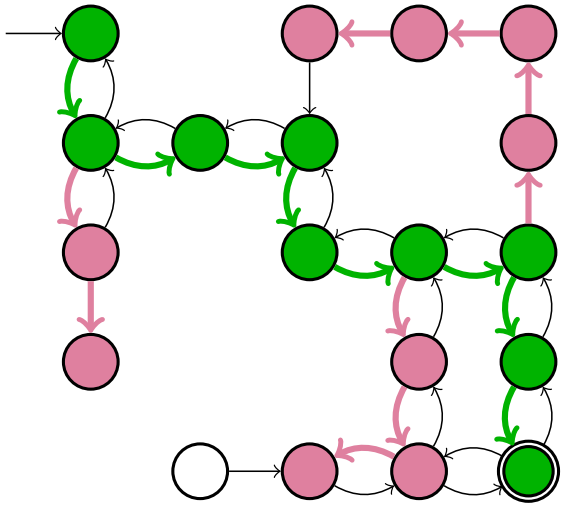
From reachable states to the winning strategy

- We have computed states reachable from S_0 ;
- To get the path, we classically memorize the first predecessor of each reachable state in the exploration ;

From reachable states to the winning strategy

- We have computed states reachable from S_0 ;
- To get the path, we classically memorize the first predecessor of each reachable state in the exploration ;
- If the goal is reachable, compute its predecessor, the predecessor of the predecessor, etc. to get the path in reverse... and the strategy.





Computation in practice

- The given fixpoint formulation is compact but not very efficient ;
- Except if we have a symbolic data structure like *Binary Decision Diagrams* (BDD) ;
- In practice, without BDDs, we could write something like :

$$
\begin{array}{l}
X \leftarrow S_0 \\
Y \leftarrow X \\
\text{while } Y \neq \emptyset : \\
\quad Y \leftarrow \text{Post}(Y) \setminus X \\
\quad X \leftarrow X \cup Y
\end{array}
$$

Reachable states : invariant

Lemma

For all $n \geq 0$, $s \in X_n$ if and only if there exists a path with $\leq n$ edges from an initial state to s.

Reachable states : invariant

Lemma

For all $n \geq 0$, $s \in X_n$ if and only if there exists a path with $\leq n$ edges from an initial state to s.

By induction on n :

Reachable states : invariant

Lemma

For all $n \geq 0$, $s \in X_n$ if and only if there exists a path with $\leq n$ edges from an initial state to s.

By induction on n :

- case $n = 0$ is trivial : $s \in S_0$ iff there exists a path with no edge from S_0 to s.

Reachable states : invariant

Lemma

For all $n \geq 0$, $s \in X_n$ if and only if there exists a path with $\leq n$ edges from an initial state to s.

By induction on n :

- case $n = 0$ is trivial : $s \in S_0$ iff there exists a path with no edge from S_0 to s.
- assume the property holds for $n \geq 0$.

Reachable states : invariant

Lemma

For all $n \geq 0$, $s \in X_n$ if and only if there exists a path with $\leq n$ edges from an initial state to s.

By induction on n :

- case $n = 0$ is trivial : $s \in S_0$ iff there exists a path with no edge from S_0 to s.
- assume the property holds for $n \geq 0$.
  - $\boxed{\Rightarrow}$ if $s \in X_{n+1}$ then either :
    - $s \in X_n$ and the induction hyp. gives a path with $\leq n \leq n+1$ edges ;
    - or $s \in \mathrm{Post}(X_n)$, so there exists $s' \in X_n$ such that $s' \to s$. By the induction hypothesis there is a path $s_0 \rightsquigarrow s'$ with $s_0 \in S_0$ and $\leq n$ edges, and thus a path $s_0 \rightsquigarrow s' \to s$ with $\leq n+1$ edges.

Reachable states : invariant

Lemma

For all $n \geq 0$, $s \in X_n$ if and only if there exists a path with $\leq n$ edges from an initial state to s.

By induction on n :

- case $n = 0$ is trivial : $s \in S_0$ iff there exists a path with no edge from S_0 to s.
- assume the property holds for $n \geq 0$.
  - $\boxed{\Rightarrow}$ if $s \in X_{n+1}$ then either :
    - $s \in X_n$ and the induction hyp. gives a path with $\leq n \leq n+1$ edges ;
    - or $s \in \mathrm{Post}(X_n)$, so there exists $s' \in X_n$ such that $s' \to s$. By the induction hypothesis there is a path $s_0 \rightsquigarrow s'$ with $s_0 \in S_0$ and $\leq n$ edges, and thus a path $s_0 \rightsquigarrow s' \to s$ with $\leq n+1$ edges.
  - $\boxed{\Leftarrow}$ if there is a path $s_0 \rightsquigarrow s$ with $n+1$ or less edges, and $s_0 \in S_0$:
    - either the path has 0 edges and $s \in S_0 \subseteq X_{n+1}$;
    - or the path has at least 1 edge and can be written $s_0 \rightsquigarrow s' \to s$ with n or less edges between s_0 and s'. Then by the induction hypothesis, $s' \in X_n$ and $s \in \mathrm{Post}(X_n) \subseteq X_{n+1}$.

Reachable states : termination, correctness, and soundness

Theorem

There exists $N \geq 0$ such that for all $n \geq N$, $X_n = X_N$.

At each iteration, we can only add states and S is finite.
And we can thus stop the computation at N.

Reachable states : termination, correctness, and soundness

Theorem

There exists $N \geq 0$ such that for all $n \geq N$, $X_n = X_N$.

At each iteration, we can only add states and S is finite.
And we can thus stop the computation at N.

Theorem

Let X^* be the computed fixpoint. State s is reachable if and only if $s \in X^*$.

If s is reachable then there is a finite path with $n \geq 0$ edges from S_0 to s.
And if $s \in X^*$ then there $s \in X_N$, by the above theorem.

Winning states

- A state is winning if there is a winning strategy from it ;
- We have computed (some) winning states by backtracking from the goal using reachable states and their predecessors ;
- We could compute them more directly ;
- What makes a winning state ?

Winning states

- A state is winning if there is a winning strategy from it ;
- We have computed (some) winning states by backtracking from the goal using reachable states and their predecessors ;
- We could compute them more directly ;
- What makes a winning state ? A winning successor !

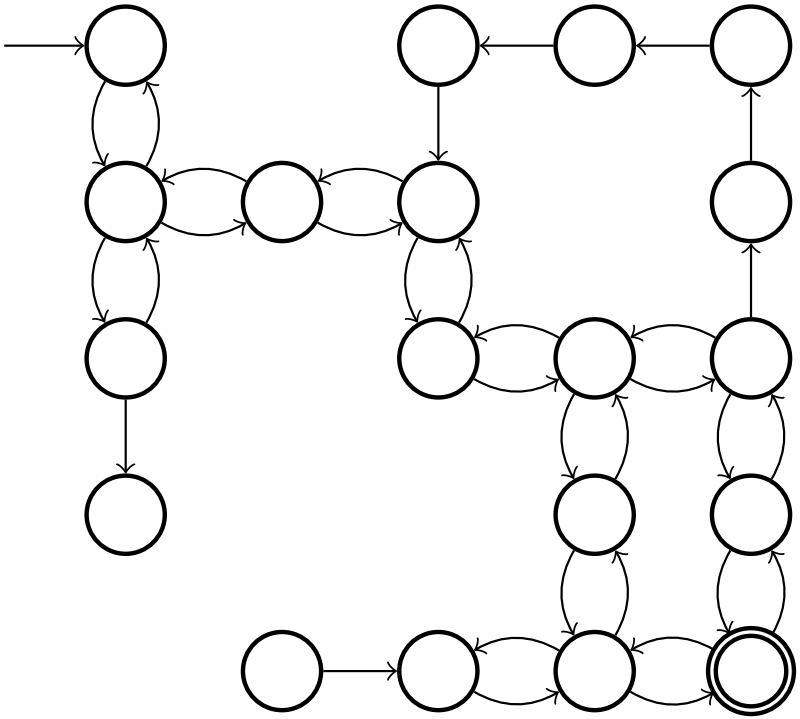
Winning states

- A state is winning if there is a winning strategy from it ;
- We have computed (some) winning states by backtracking from the goal using reachable states and their predecessors ;
- We could compute them more directly ;
- What makes a winning state ? A winning successor !
- What is then a winning strategy ?

Winning states

- A state is winning if there is a winning strategy from it ;
- We have computed (some) winning states by backtracking from the goal using reachable states and their predecessors ;
- We could compute them more directly ;
- What makes a winning state ? A winning successor !
- What is then a winning strategy ? Going to a winning successor !

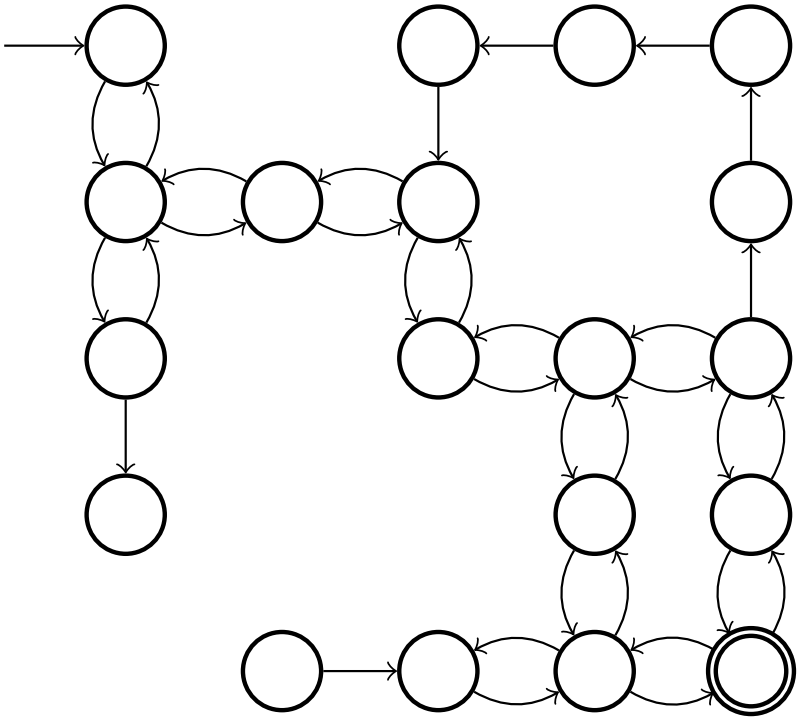
Winning (co-reachable) states



For all $X \subseteq S$ define :

$\mathrm{Pre}(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \to s'\}$

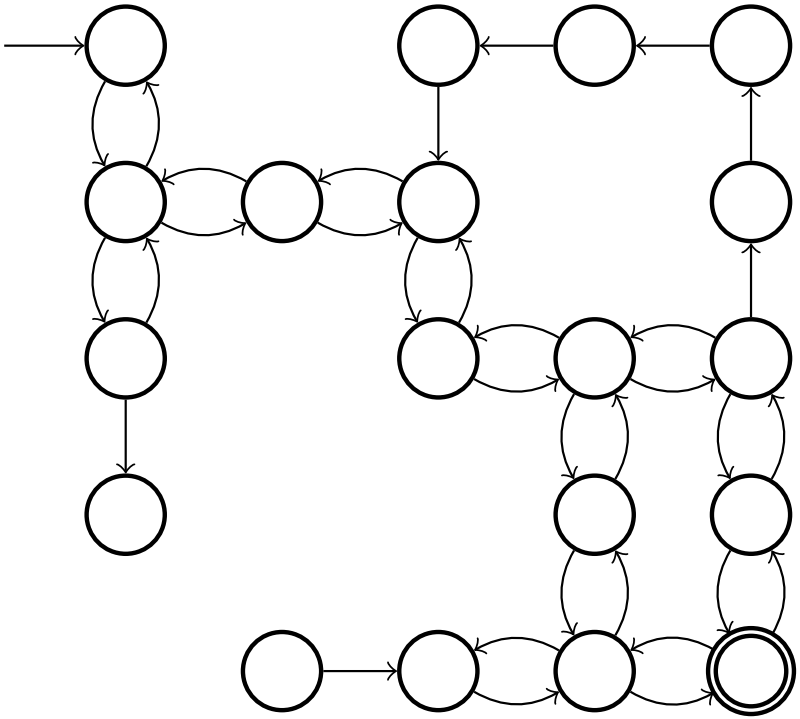
Winning (co-reachable) states



For all $X \subseteq S$ define :
$\mathrm{Pre}(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \to s'\}$

$$X_0 = G$$
$$X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$$

Winning (co-reachable) states



For all $X \subseteq S$ define :

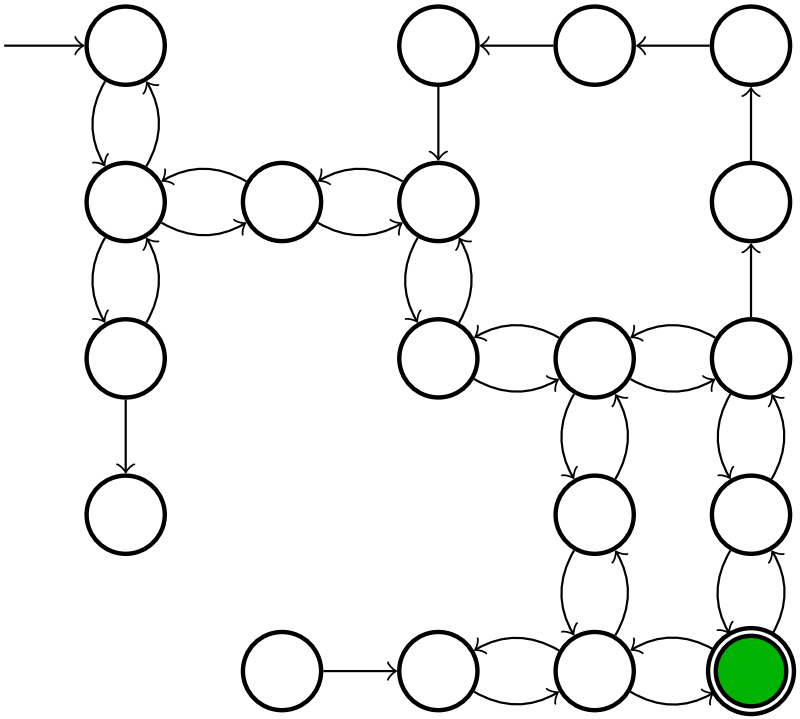
$\mathrm{Pre}(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \to s'\}$

goal states

$X_0 = G$

$X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$

Winning (co-reachable) states



For all $X \subseteq S$ define :

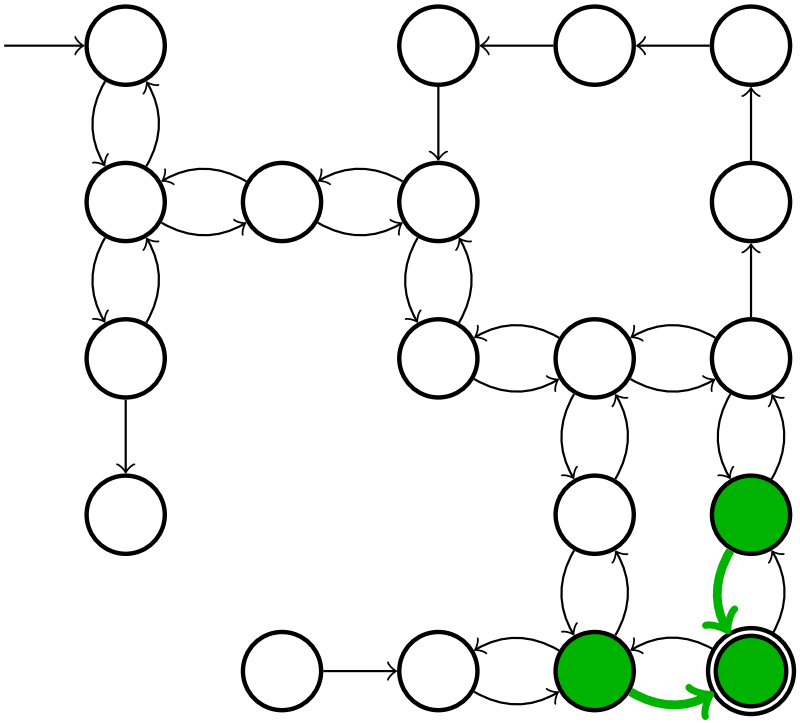
$\mathrm{Pre}(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \to s'\}$

goal states

$X_0 = G$

$X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$

Winning (co-reachable) states



For all $X \subseteq S$ define :

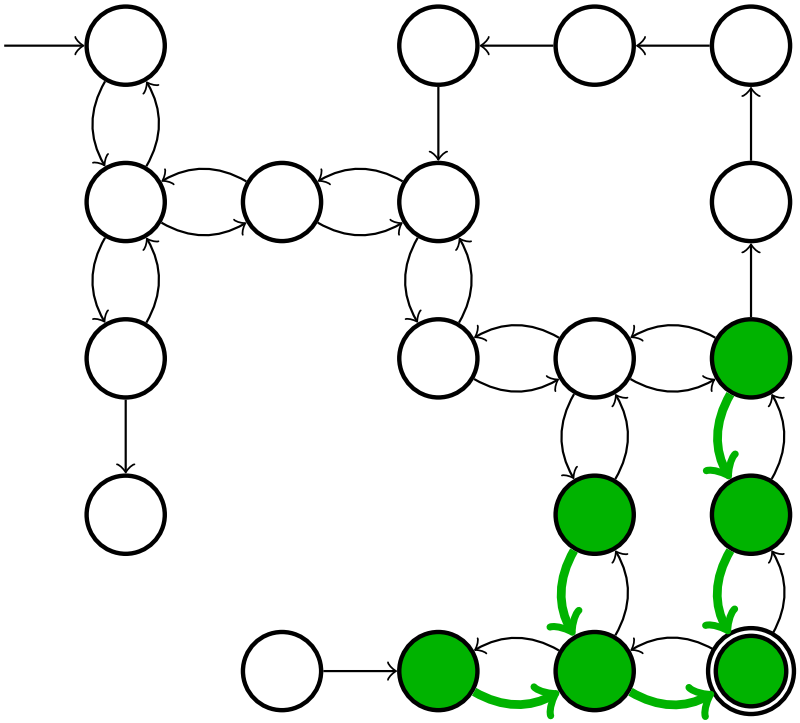
$\mathrm{Pre}(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \to s'\}$

goal states

$X_0 = G$

$X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$

Winning (co-reachable) states



For all $X \subseteq S$ define :

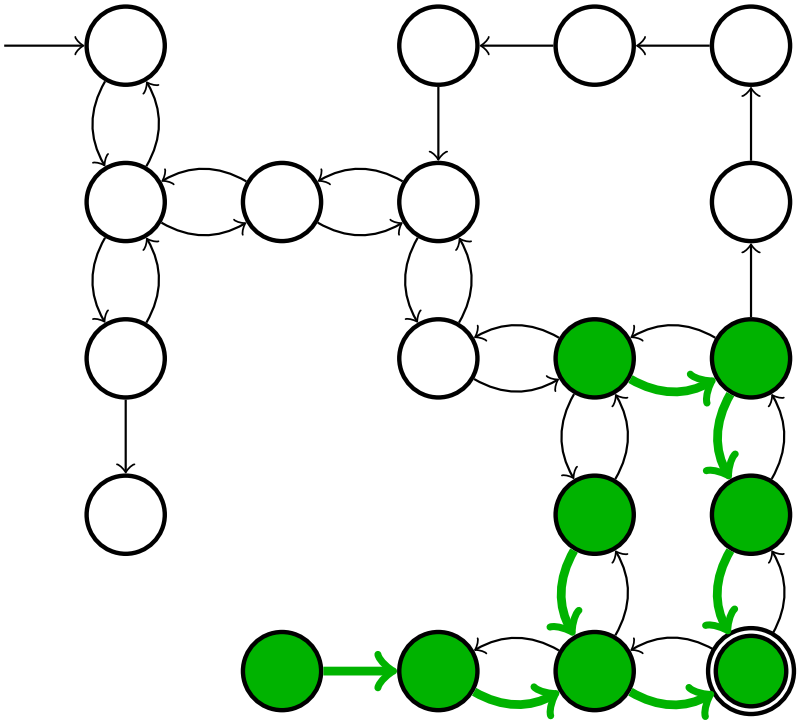
$\mathrm{Pre}(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \to s'\}$

goal states

$X_0 = G$

$X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$

Winning (co-reachable) states



For all $X \subseteq S$ define :

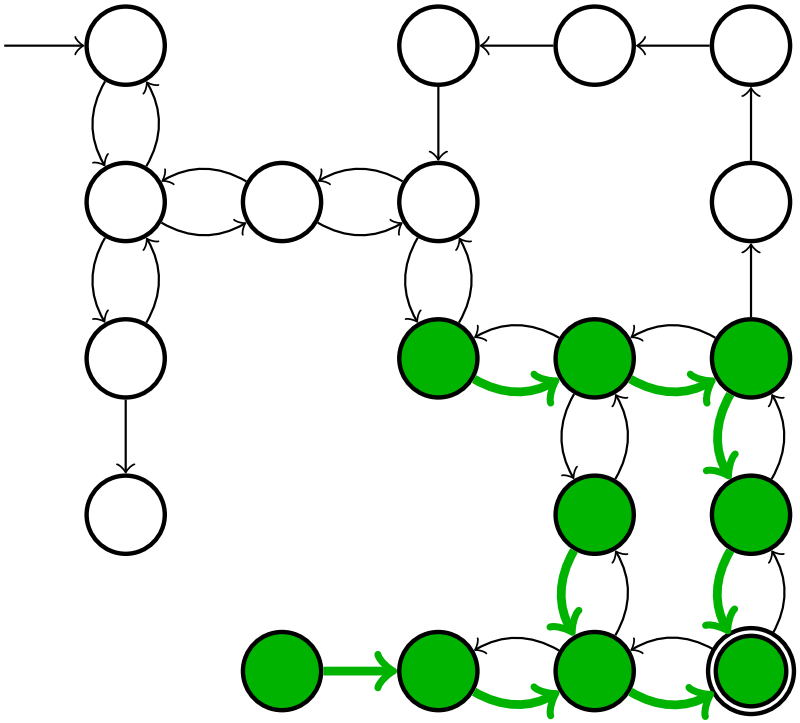
$\mathrm{Pre}(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \to s'\}$

goal states

$X_0 = G$

$X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$

Winning (co-reachable) states



For all $X \subseteq S$ define :

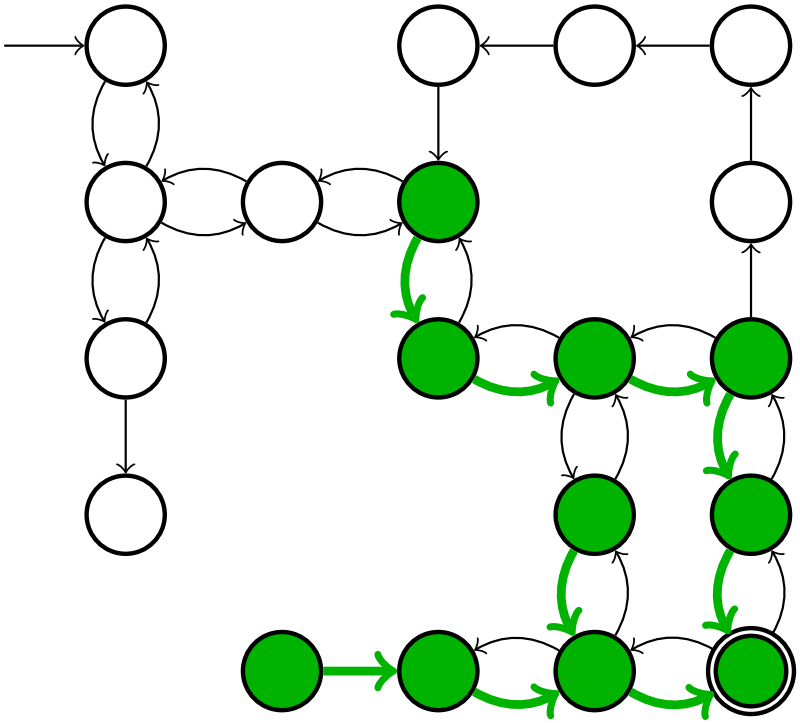
$\mathrm{Pre}(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \to s'\}$

goal states

$X_0 = G$

$X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$

Winning (co-reachable) states



For all $X \subseteq S$ define :

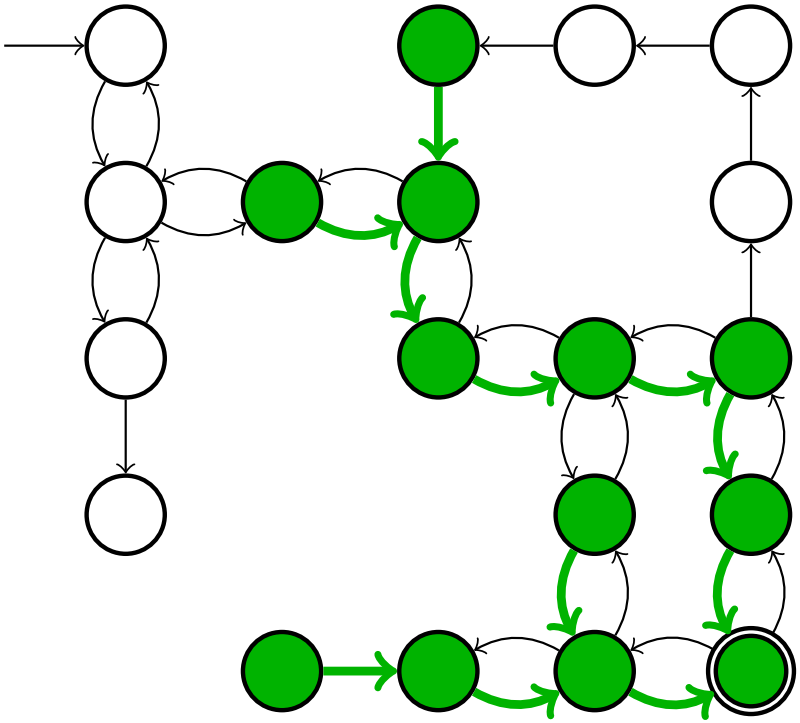
$\mathrm{Pre}(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \to s'\}$

goal states

$X_0 = G$

$X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$

Winning (co-reachable) states



For all $X \subseteq S$ define :

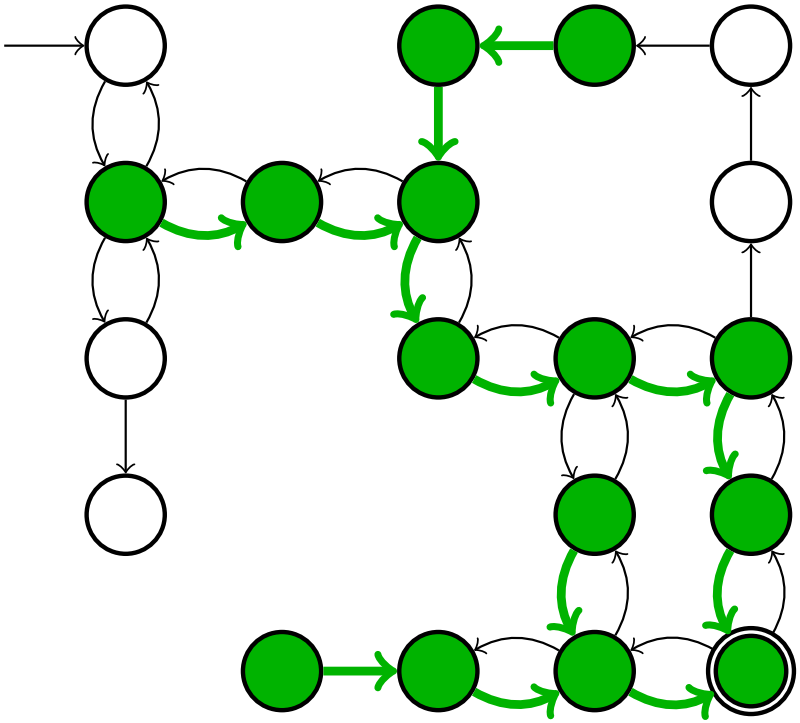
$\mathrm{Pre}(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \to s'\}$

goal states

$X_0 = G$

$X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$

Winning (co-reachable) states



For all $X \subseteq S$ define :

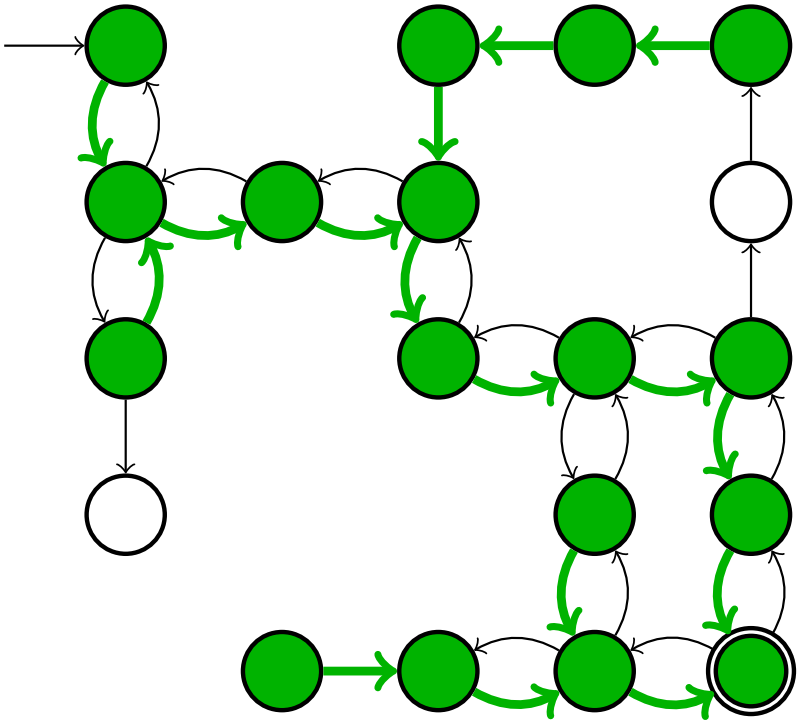
$\mathrm{Pre}(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \to s'\}$

goal states

$X_0 = G$

$X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$

Winning (co-reachable) states



For all $X \subseteq S$ define :

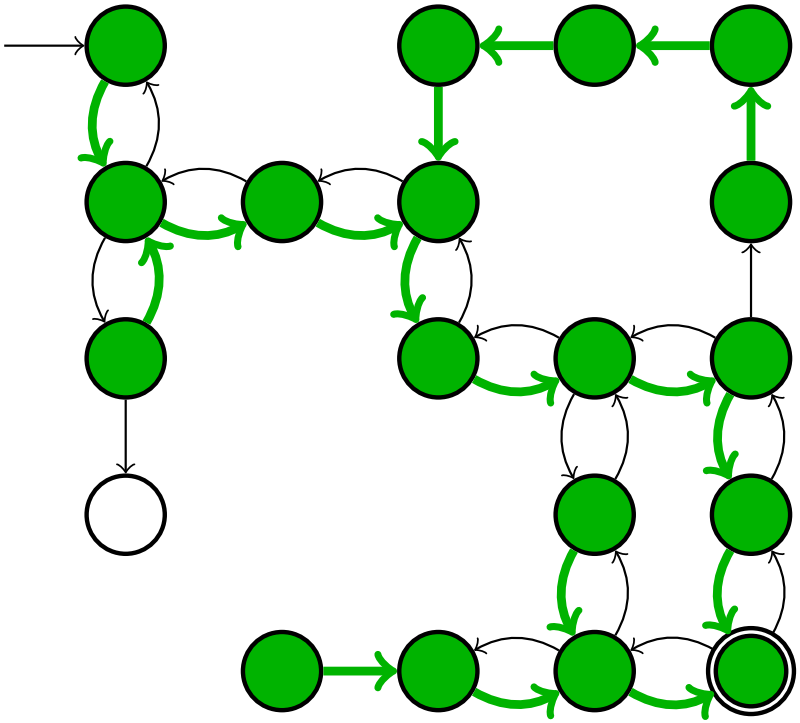
$\mathrm{Pre}(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \to s'\}$

goal states

$X_0 = G$

$X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$

Winning (co-reachable) states



For all $X \subseteq S$ define :

$\mathrm{Pre}(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \to s'\}$

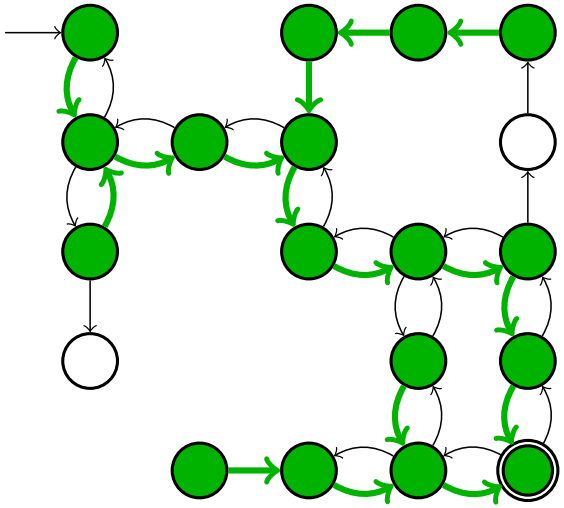
goal states

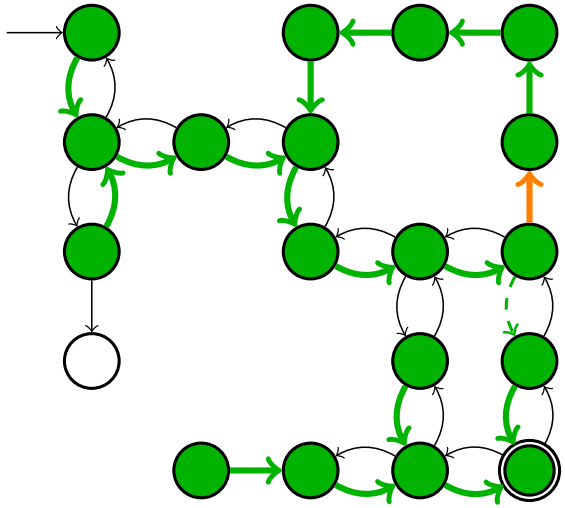
$X_0 = G$

$X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$

Computing strategies

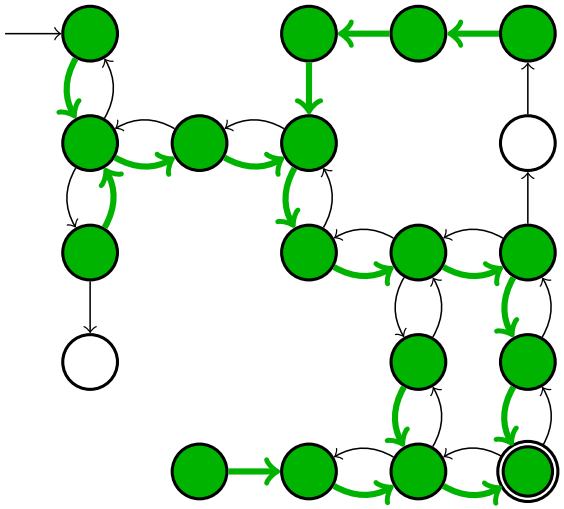
- As for reachability, we need to record one of the first winning successors as strategy ;

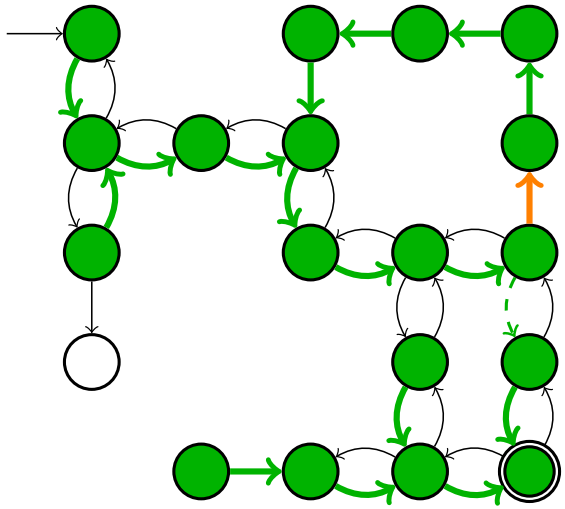




Computing strategies

- As for reachability, we need to record one of the first winning successors as strategy ;





- We need to ensure progress :

Lemma

For all $n \geq 0$, $s \in X_n$ if and only if there exists a path with $\leq n$ edges from s state to a goal state.

Is backward better than forward ?

Is backward better than forward ?

- There might be a lot of non-reachable states ;

Is backward better than forward ?

- There might be a lot of non-reachable states ;
- But also a lot of non-co-reachable state !

Is backward better than forward ?

- There might be a lot of non-reachable states ;
- But also a lot of non-co-reachable state !
- Some problems are more easily (only ?) solved backwards (see 2-player games) ;

Is backward better than forward ?

- There might be a lot of non-reachable states ;
- But also a lot of non-co-reachable state !
- Some problems are more easily (only ?) solved backwards (see 2-player games) ;
- When we have variables, computing possible predecessors is complex ;

Is backward better than forward ?

- There might be a lot of non-reachable states ;
- But also a lot of non-co-reachable state !
- Some problems are more easily (only ?) solved backwards (see 2-player games) ;
- When we have variables, computing possible predecessors is complex ;
- In many cases, combine both !

Safety

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

Safety

- The most basic property of model-checking is reachability :
  Let $G \subseteq S$, can we reach a state in G ?

Safety

- The most basic property of model-checking is reachability :
  Let $G \subseteq S$, can we reach a state in G ?
- Its dual property is safety :

Safety

- The most basic property of model-checking is reachability :
  Let $G \subseteq S$, can we reach a state in G ?
- Its dual property is safety :
  Let $G \subseteq S$, can we stay in G forever ?

Safety

- The most basic property of model-checking is reachability :
  Let $G \subseteq S$, can we reach a state in G ?
- Its dual property is safety :
  Let $G \subseteq S$, can we stay in G forever ?
- The safety property can also be written as :

Safety

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

Safety : A backward symbolic algorithm

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

Safety : A backward symbolic algorithm

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

Safety : A backward symbolic algorithm

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

Safety : A backward symbolic algorithm

- Let Y be the safe states for G and X the coreachable states for $\overline{G}$: we have $Y = \overline{X}$
- Sequence $X_0 = \overline{G}$ et $X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$ converges to X ;
- So $(Y_n)_n = (\overline{X_n})_n$ converges to Y ;
- $Y_0 = G$ and $Y_{n+1} = Y_n \cap \overline{\mathrm{Pre}(\overline{Y_n})}$;

Safety : A backward symbolic algorithm

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

Safety : A backward symbolic algorithm

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

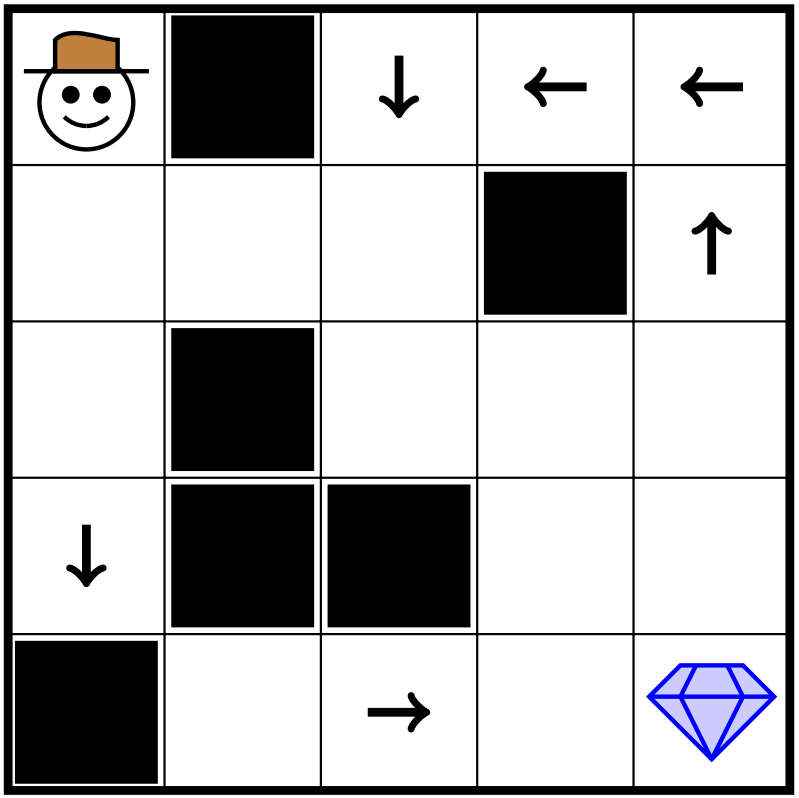
Safety : A backward symbolic algorithm

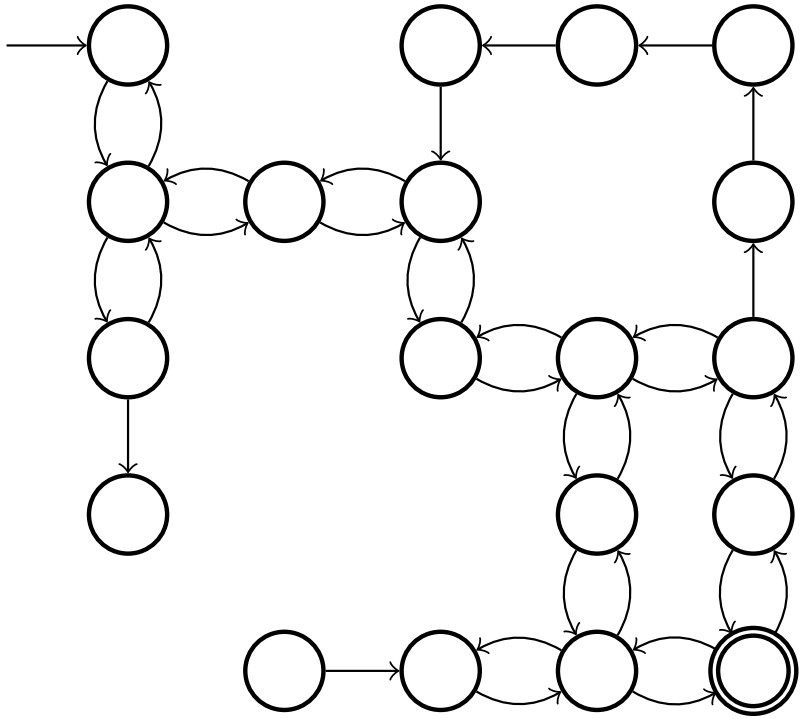
- Let Y be the safe states for G and X the coreachable states for $\overline{G}$: we have $Y = \overline{X}$
- Sequence $X_0 = \overline{G}$ et $X_{n+1} = X_n \cup \mathrm{Pre}(X_n)$ converges to X ;
- So $(Y_n)_n = (\overline{X_n})_n$ converges to Y ;
- $Y_0 = G$ and $Y_{n+1} = Y_n \cap \overline{\mathrm{Pre}(\overline{Y_n})}$;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

$$\begin{cases} Y_0 = G \\ Y_{n+1} = Y_n \cap \widetilde{\mathrm{Pre}}(Y_n) \end{cases}$$

Safety game

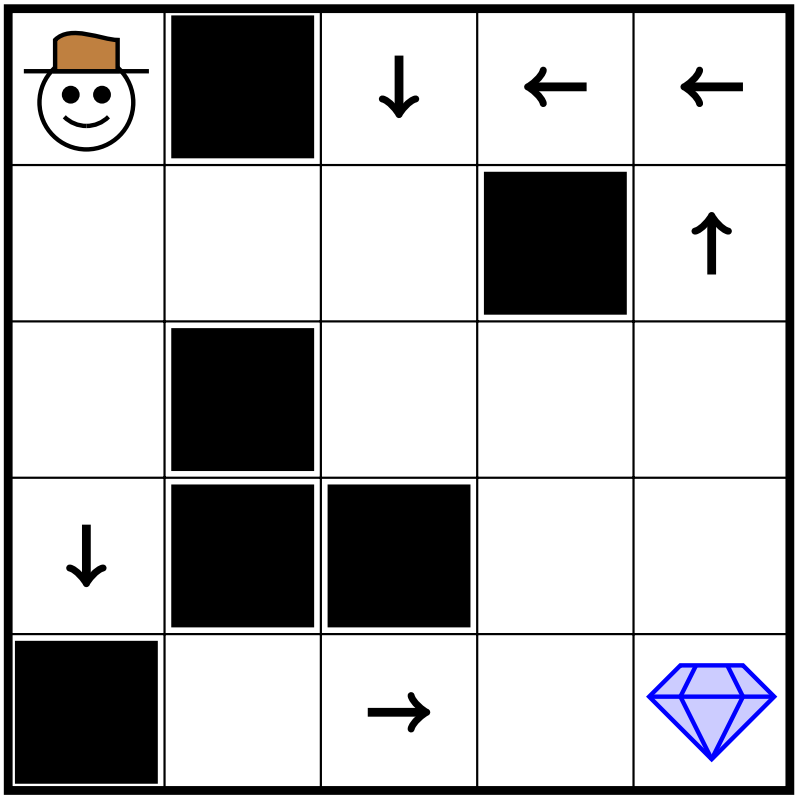
- Is  never stuck, whatever his strategy ?

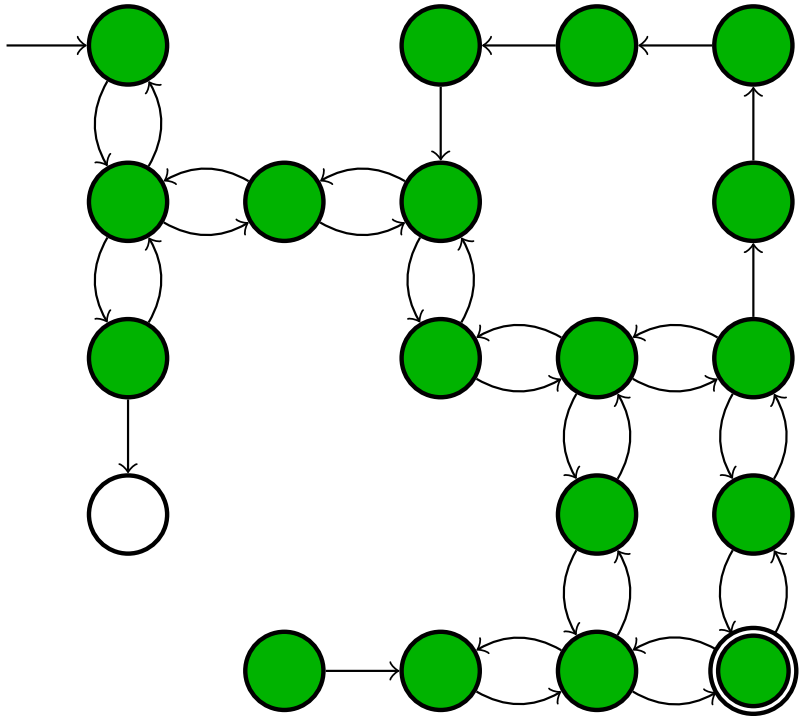




Safety game

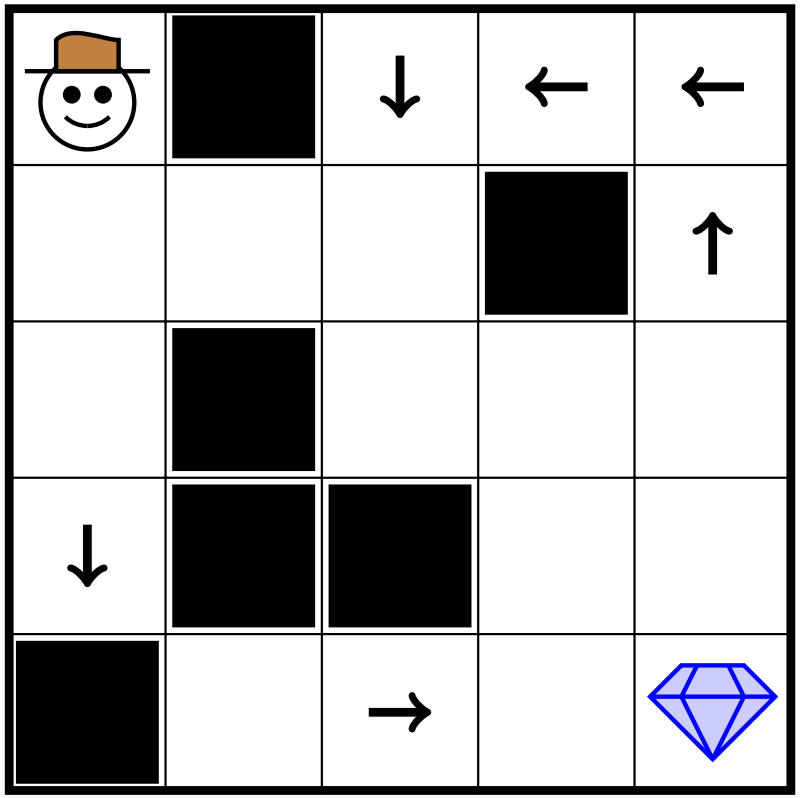
- Is  never stuck, whatever his strategy ?

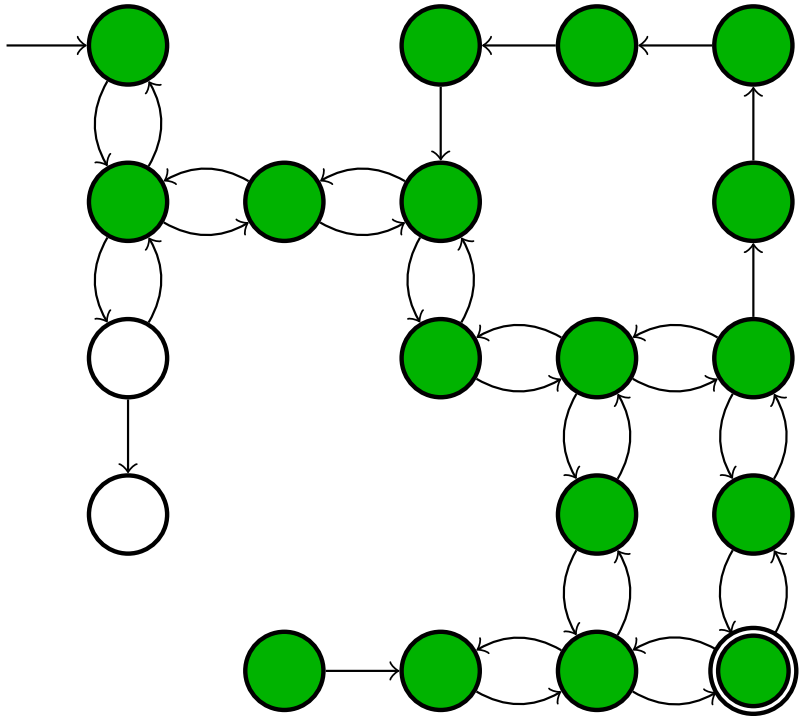




Safety game

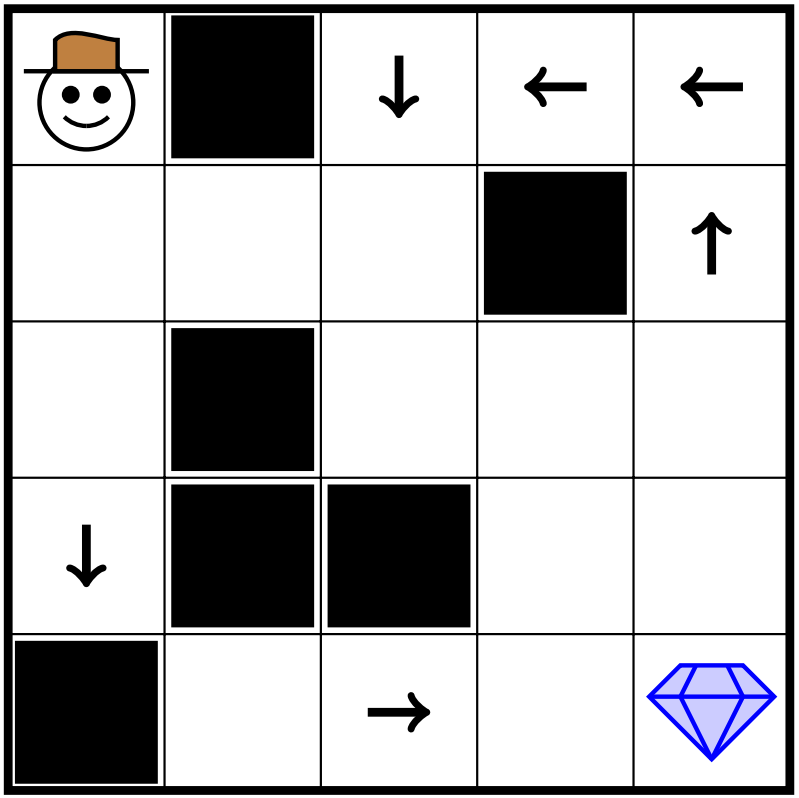
- Is  never stuck, whatever his strategy ?

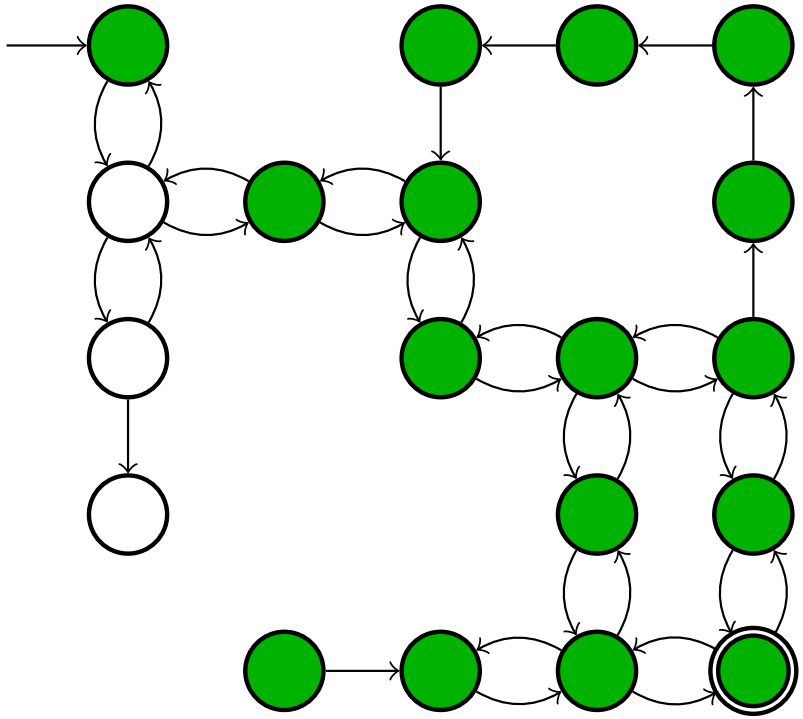




Safety game

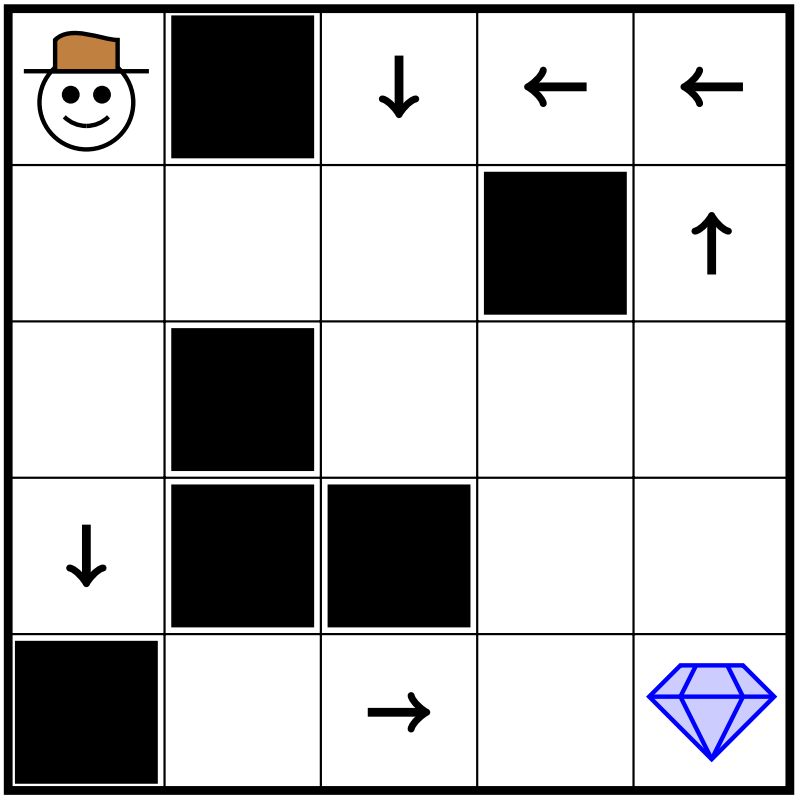
- Is  never stuck, whatever his strategy ?

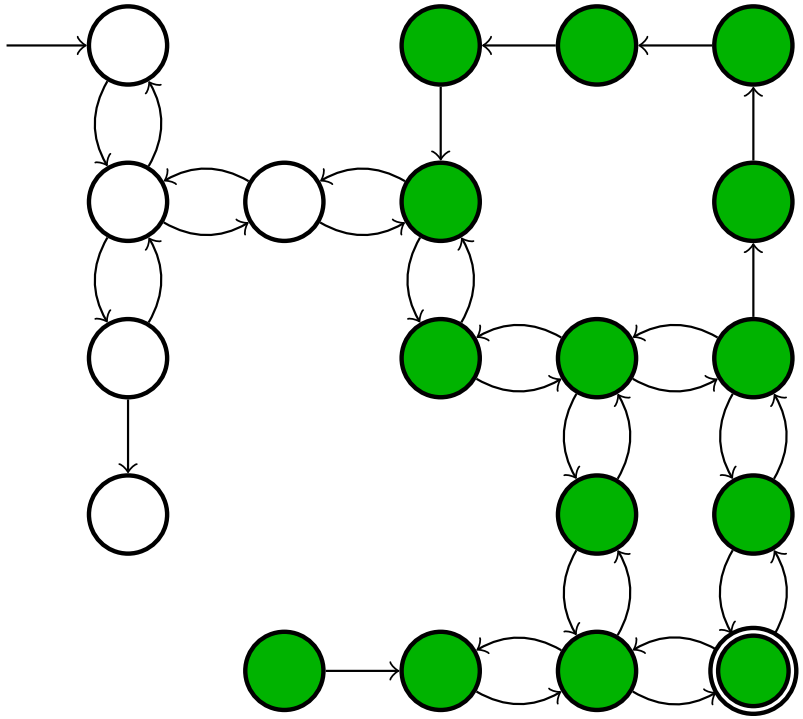




Safety game

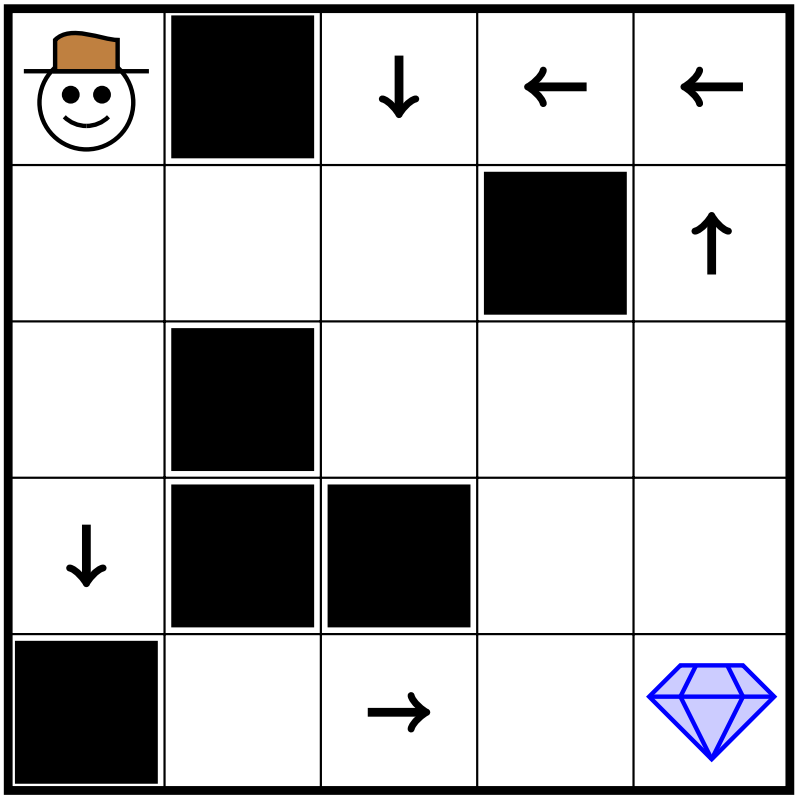
- Is  never stuck, whatever his strategy ?

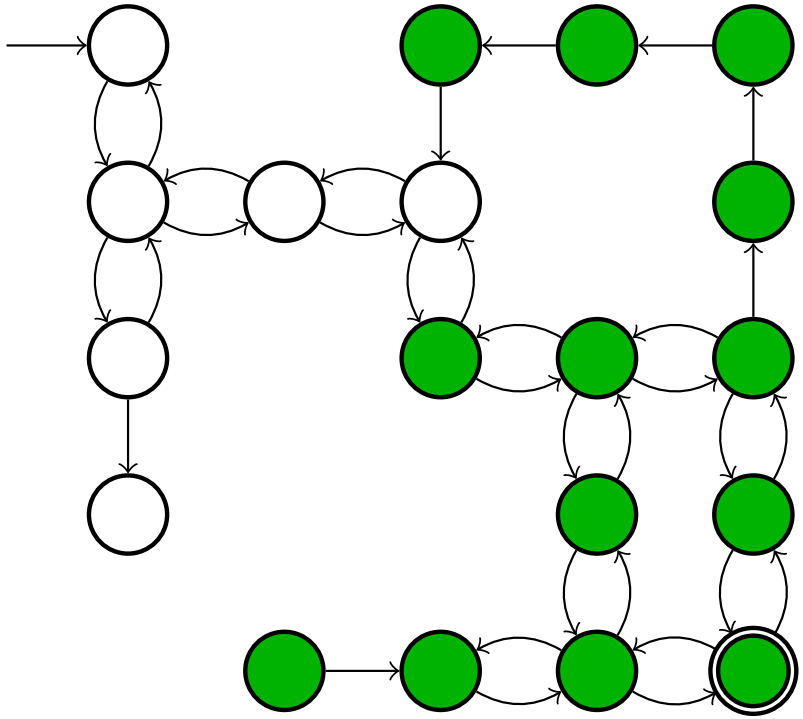




Safety game

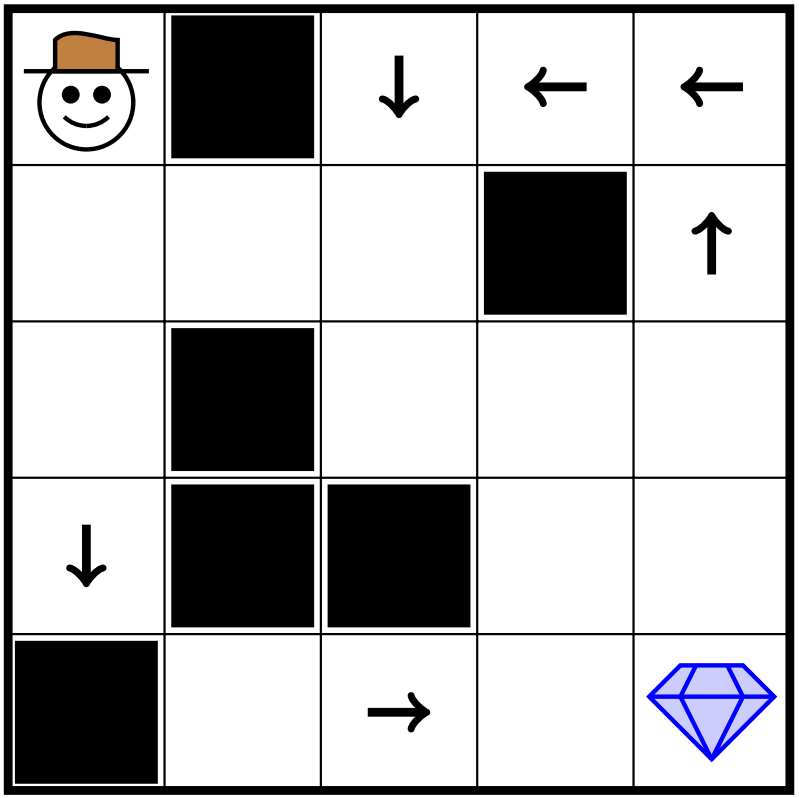
- Is  never stuck, whatever his strategy ?

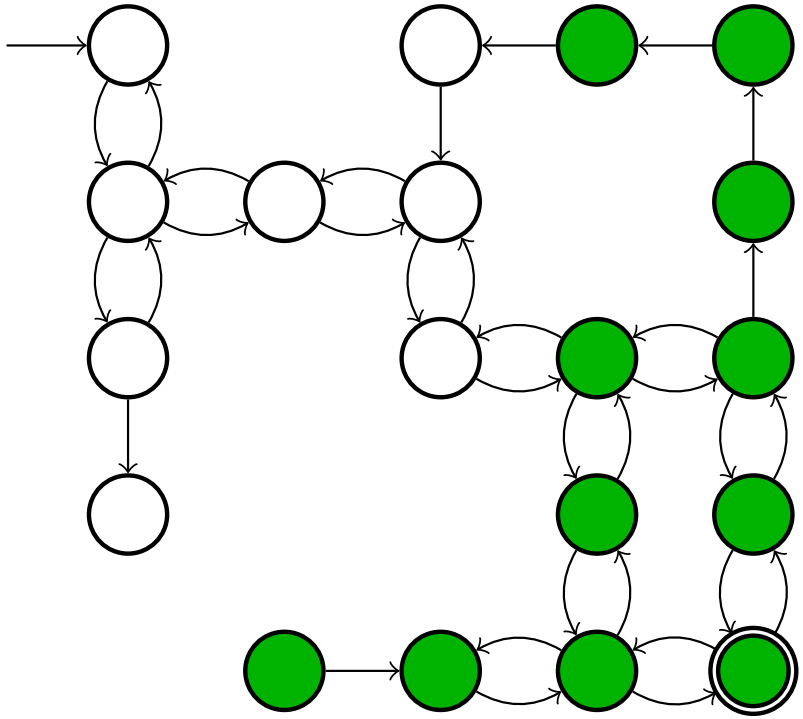




Safety game

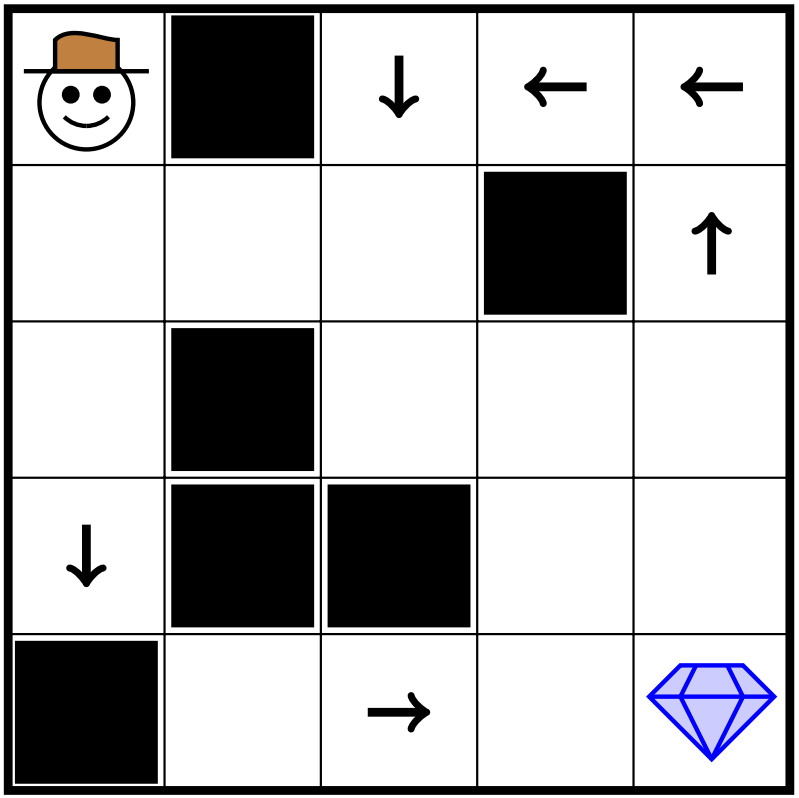
- Is  never stuck, whatever his strategy ?

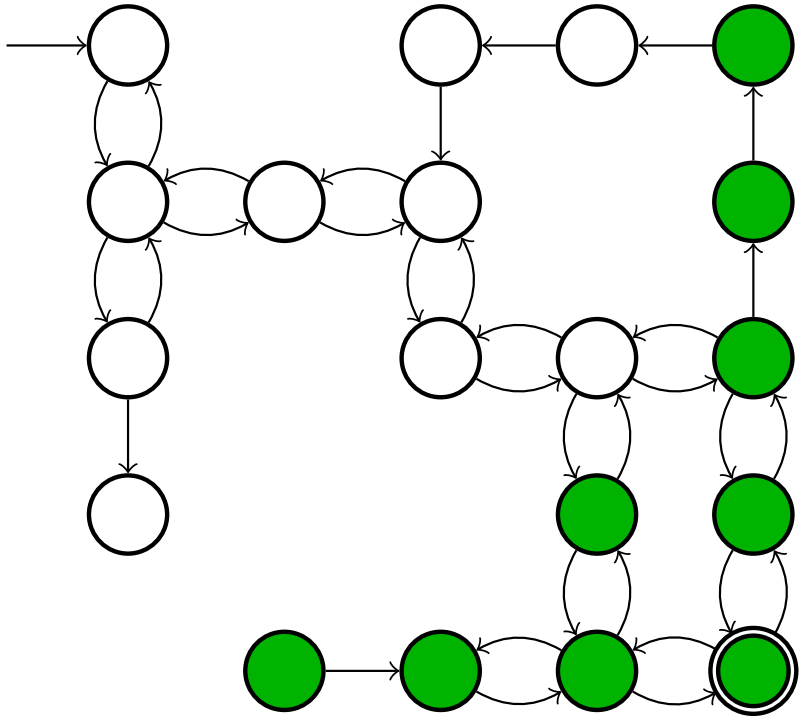




Safety game

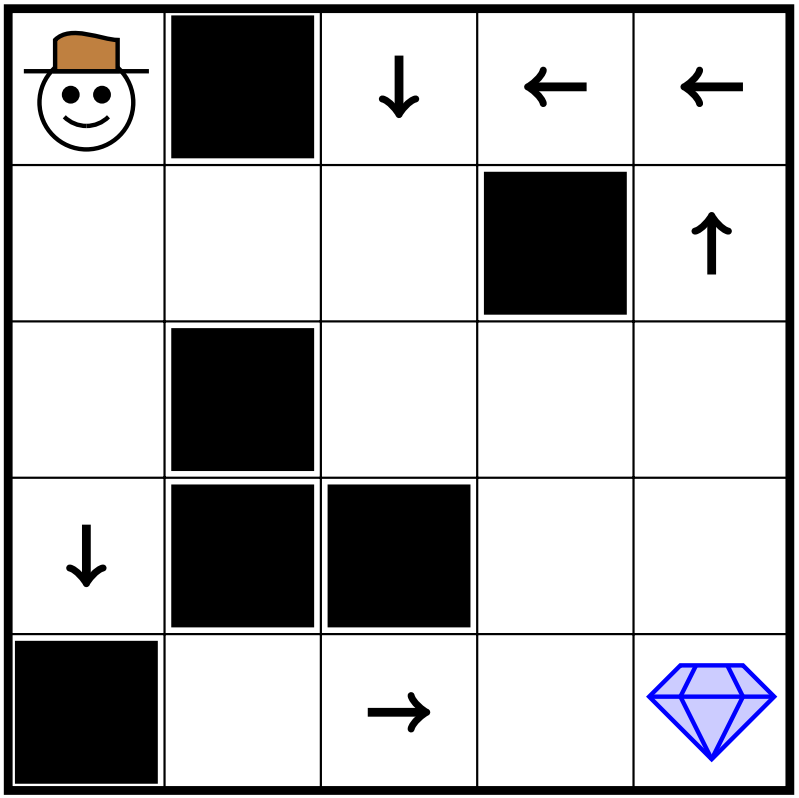
- Is  never stuck, whatever his strategy ?

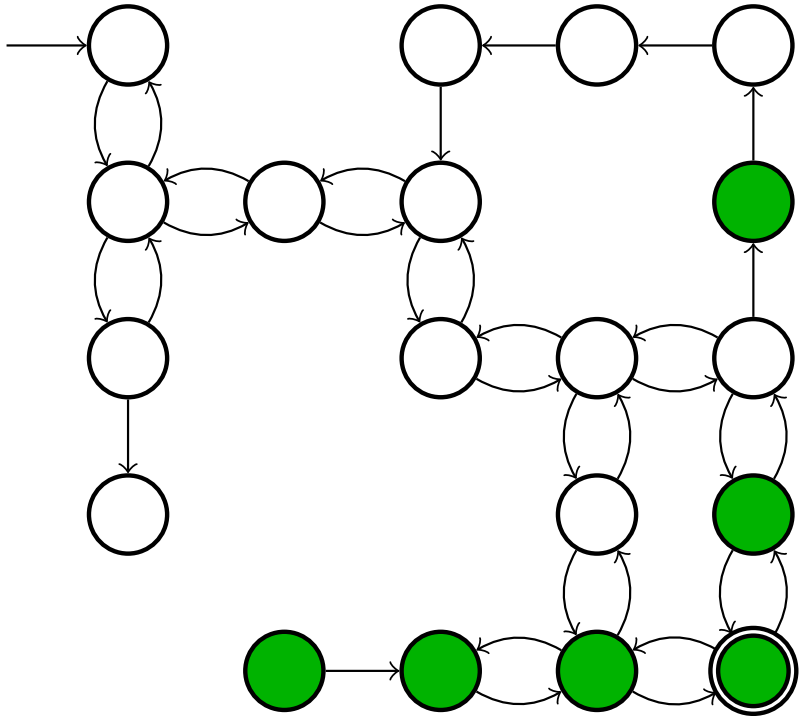




Safety game

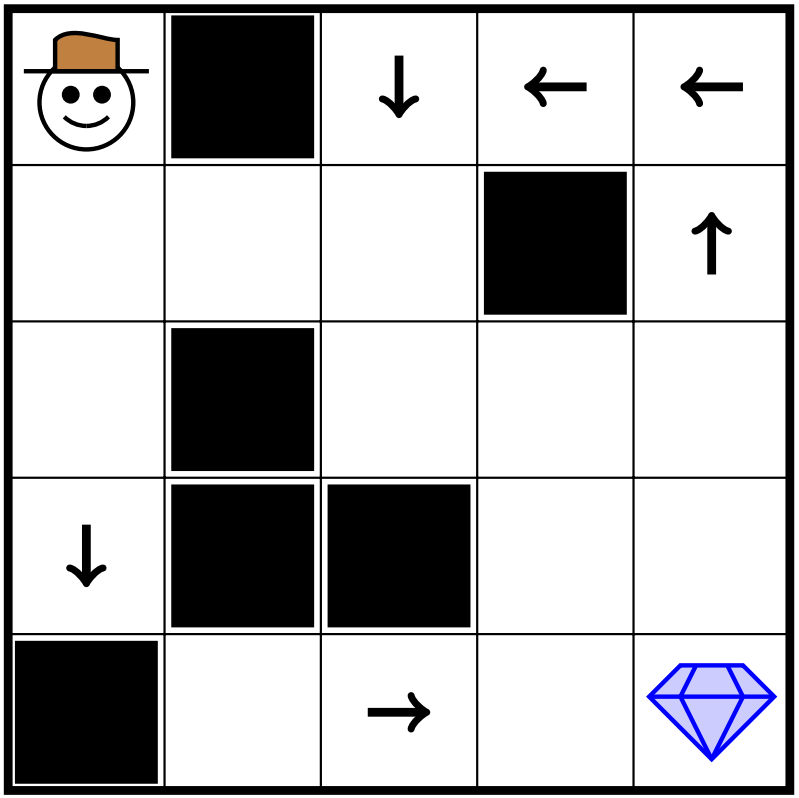
- Is  never stuck, whatever his strategy ?

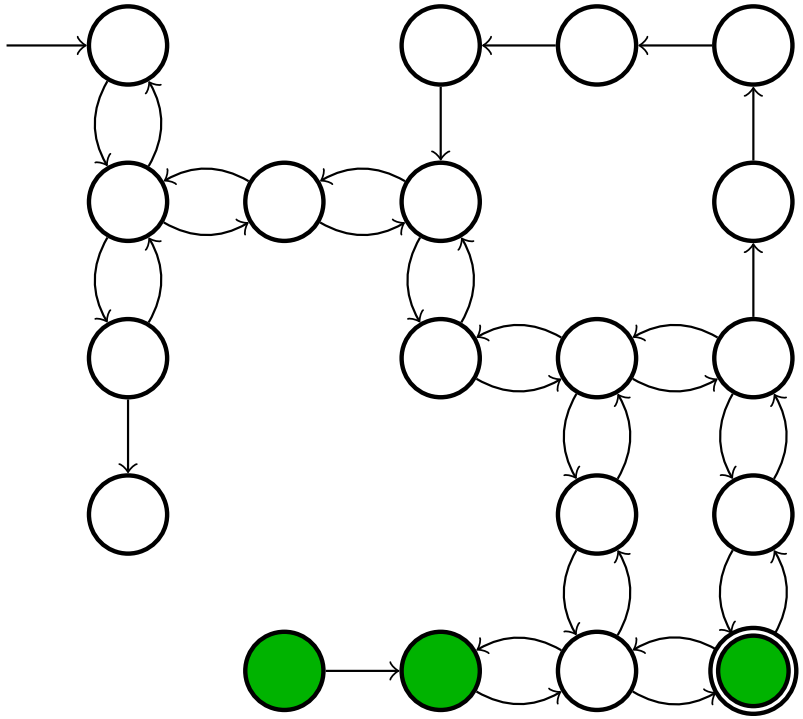




Safety game

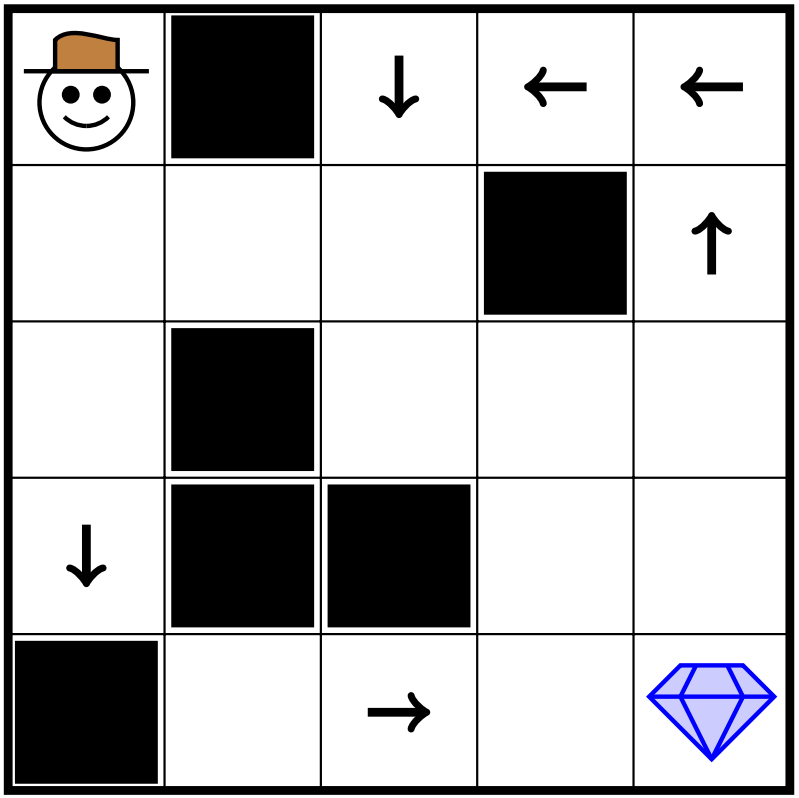
- Is  never stuck, whatever his strategy ?

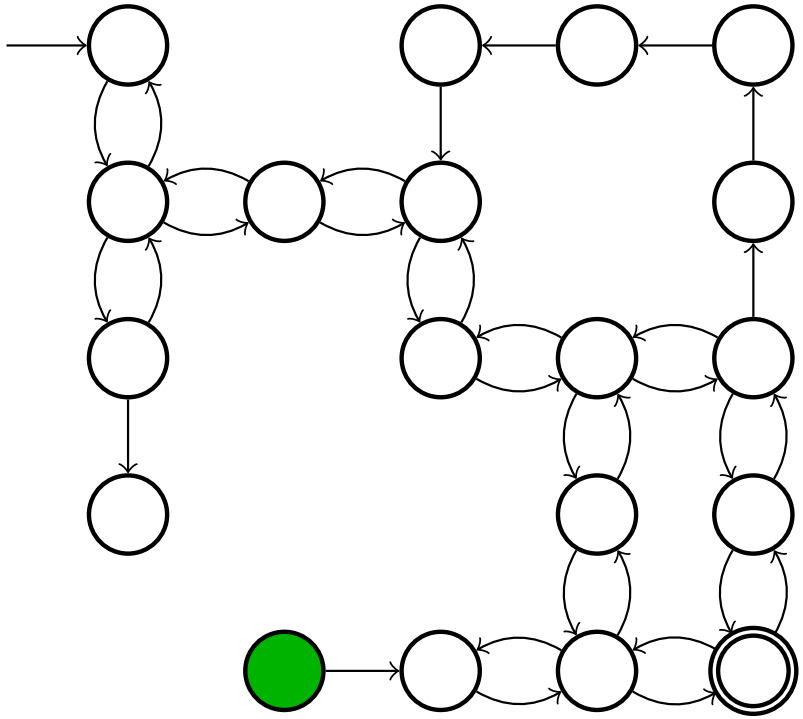




Safety game

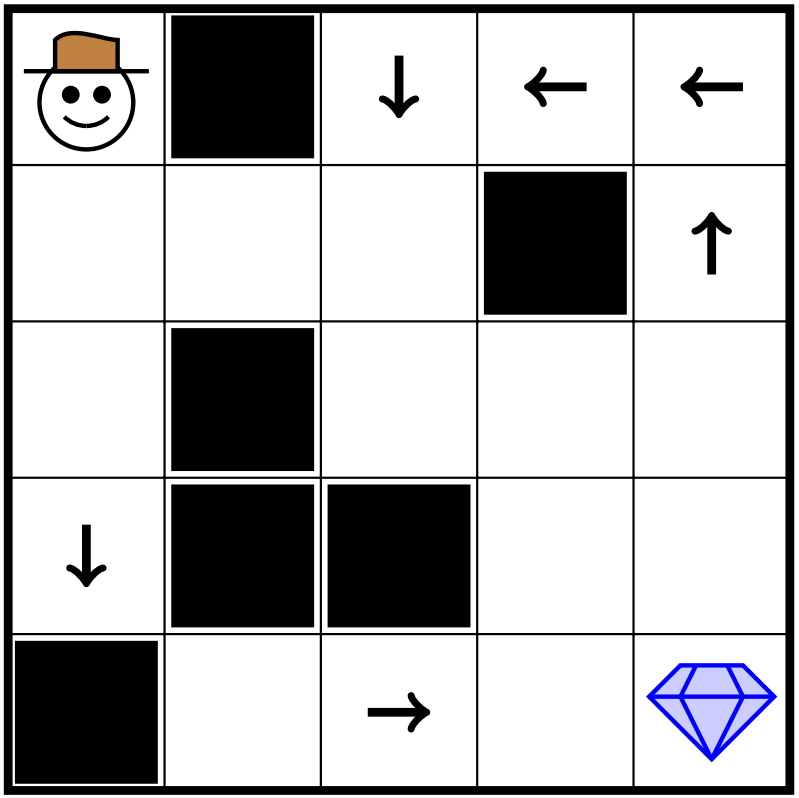
- Is  never stuck, whatever his strategy ?

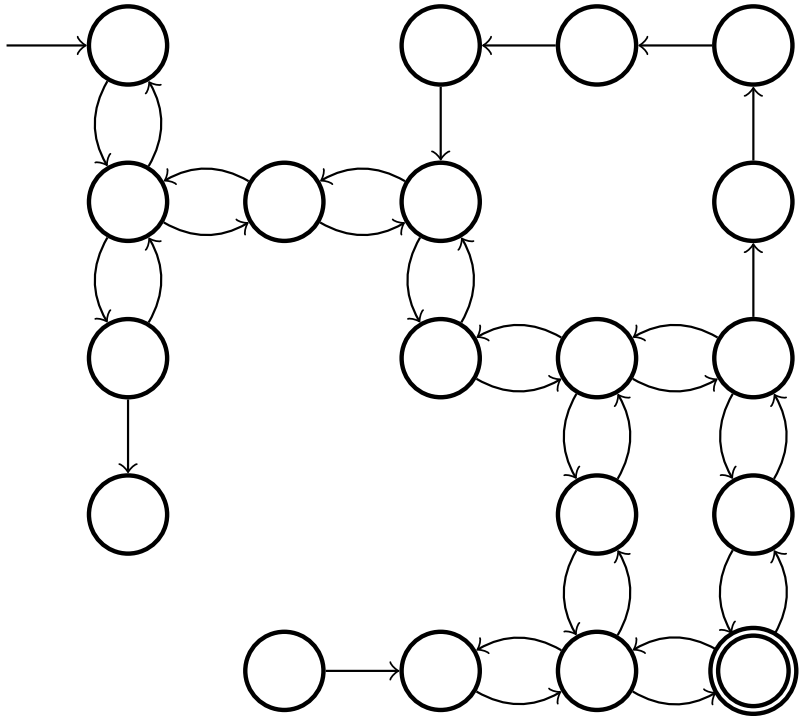




Safety game

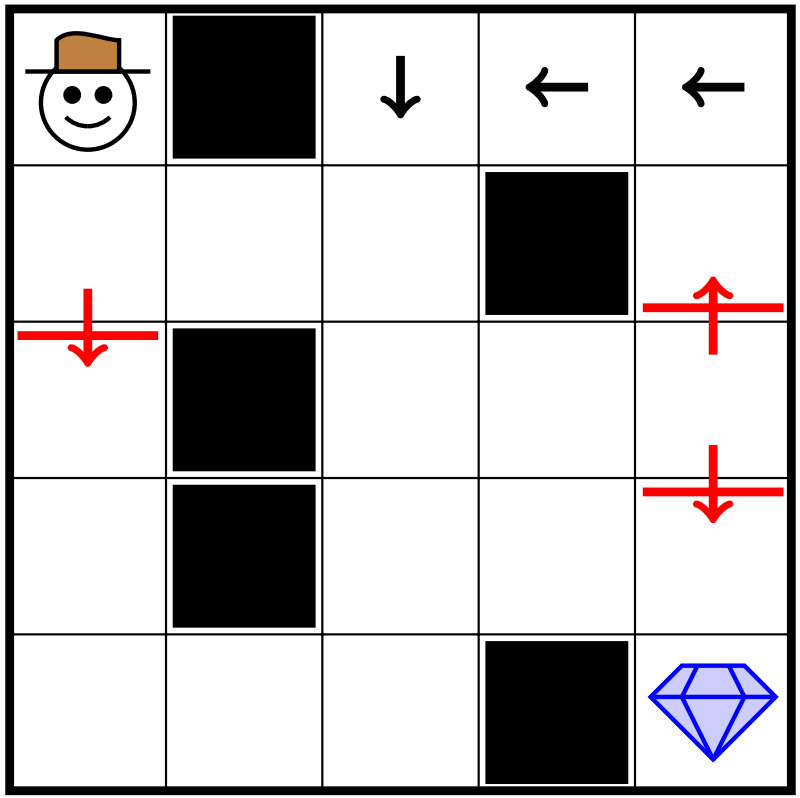
- Is  never stuck, whatever his strategy ?



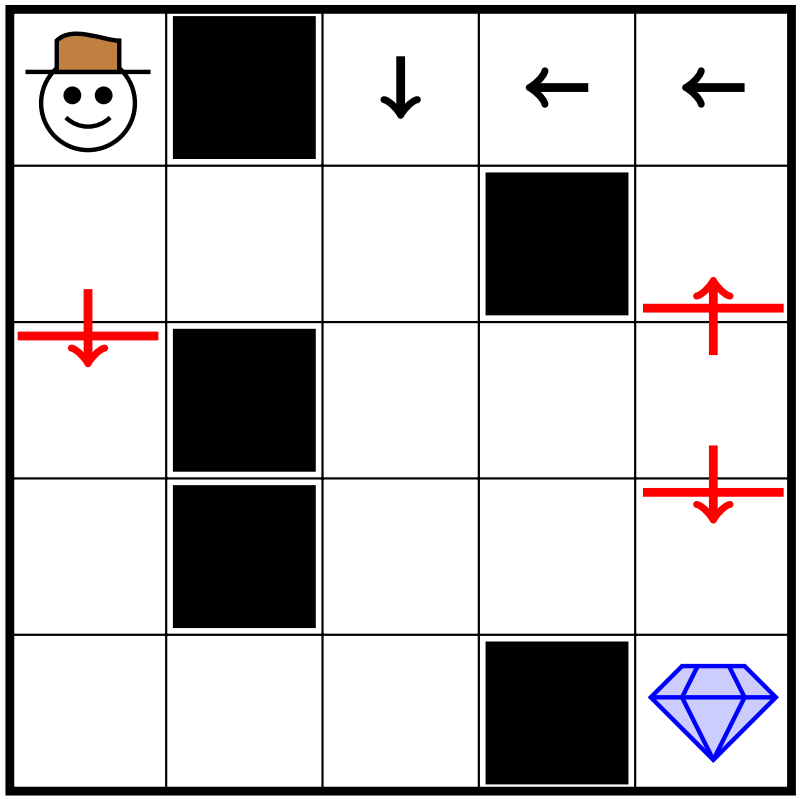


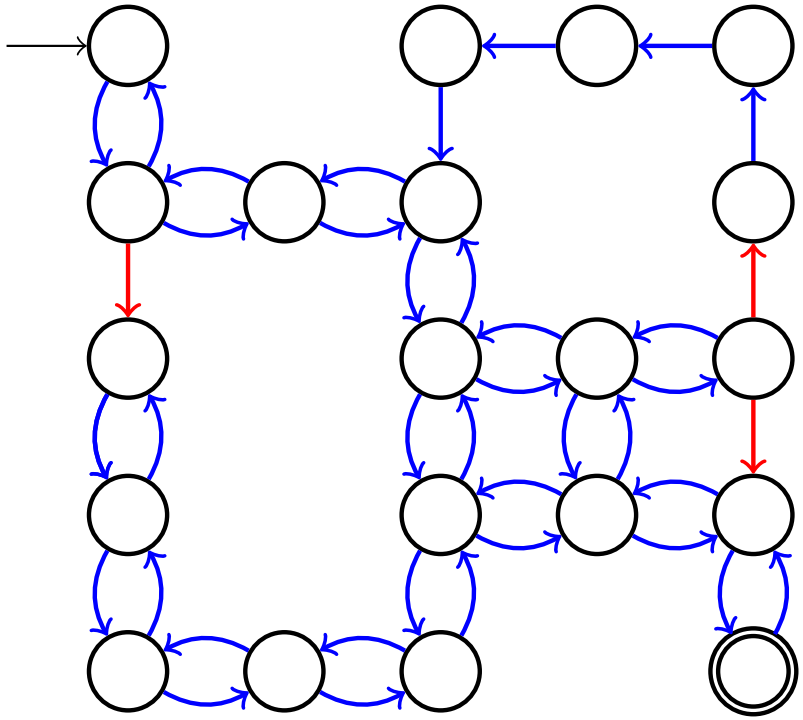
Two-player games

Two-player reachability games

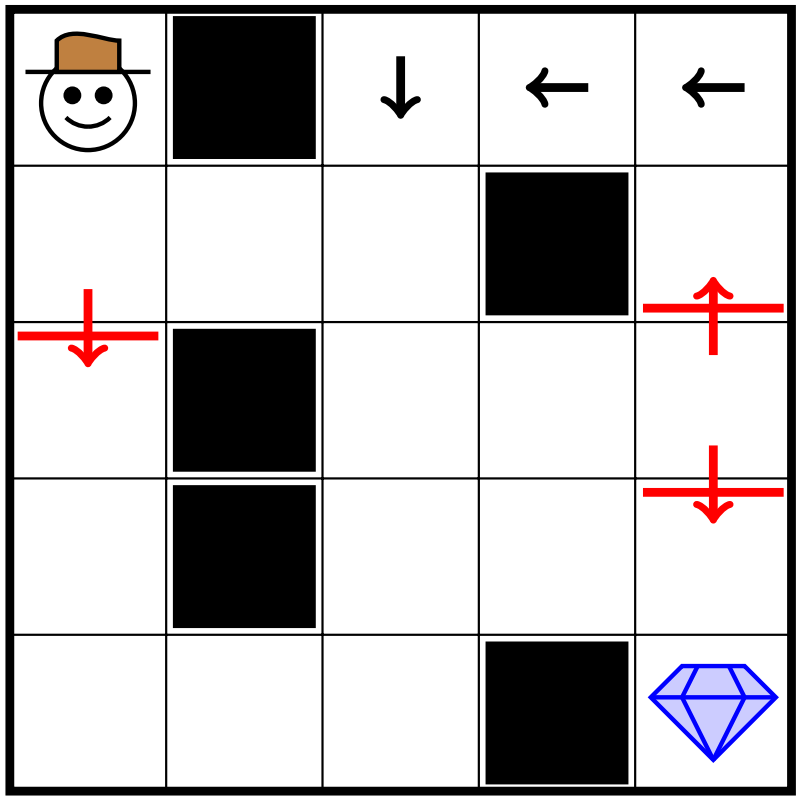


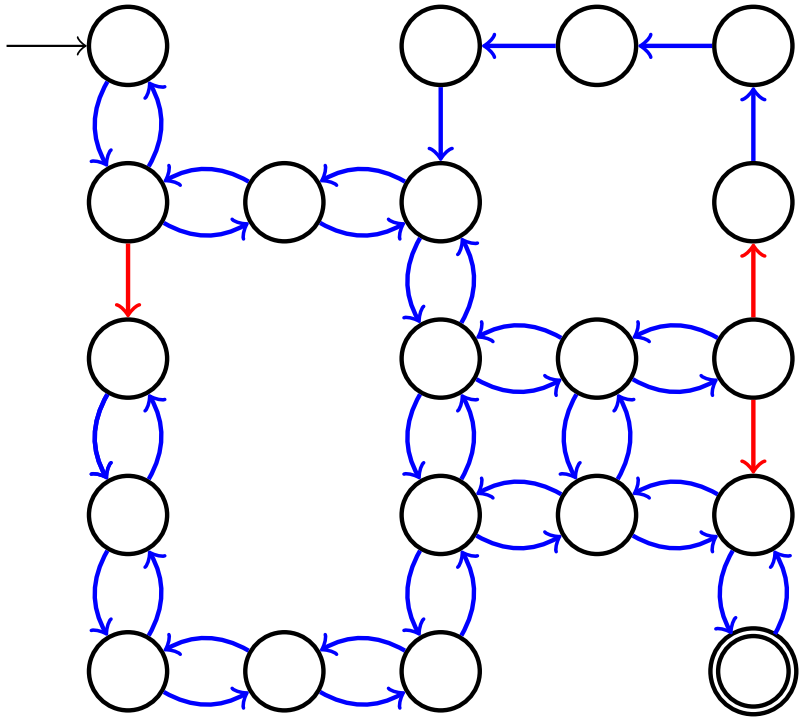
Two-player reachability games





Two-player reachability games





Two players : controller ( ) and environment (traps)

Two-player reachability games : playing and winning

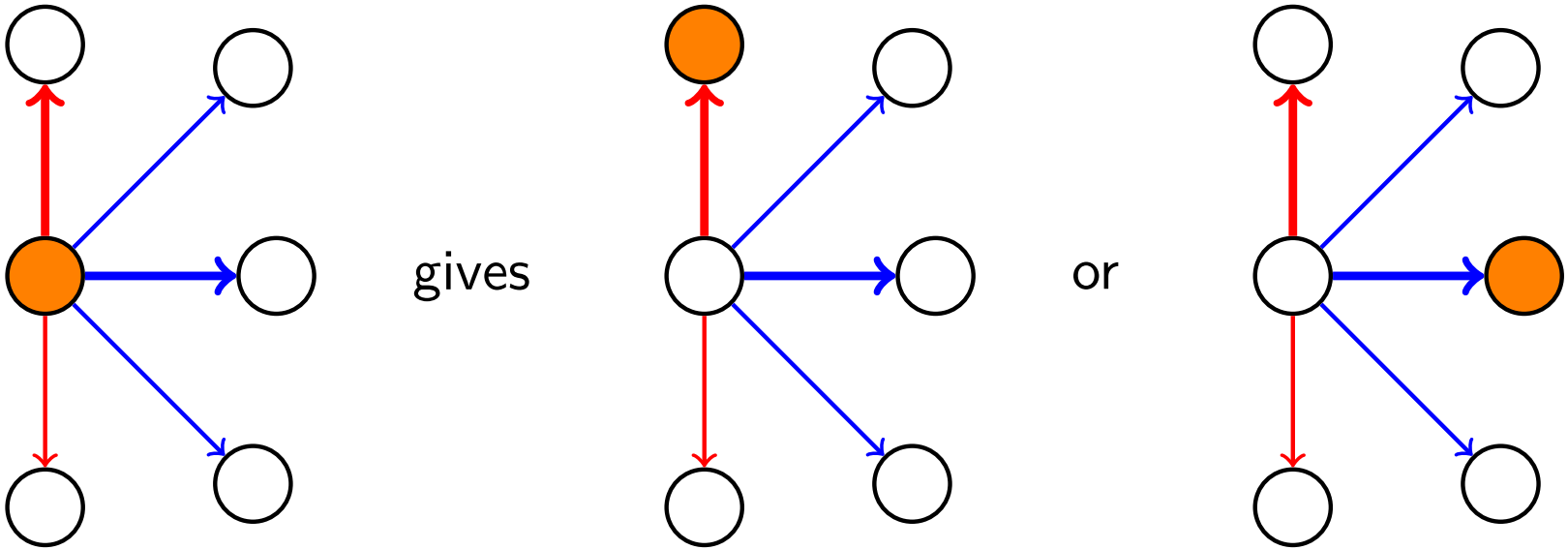
- Strategies for each player allow them only to chose among their own actions ;

Two-player reachability games : playing and winning

- Strategies for each player allow them only to chose among their own actions ;
- When both players fix their strategies, we obtain a set of plays ;

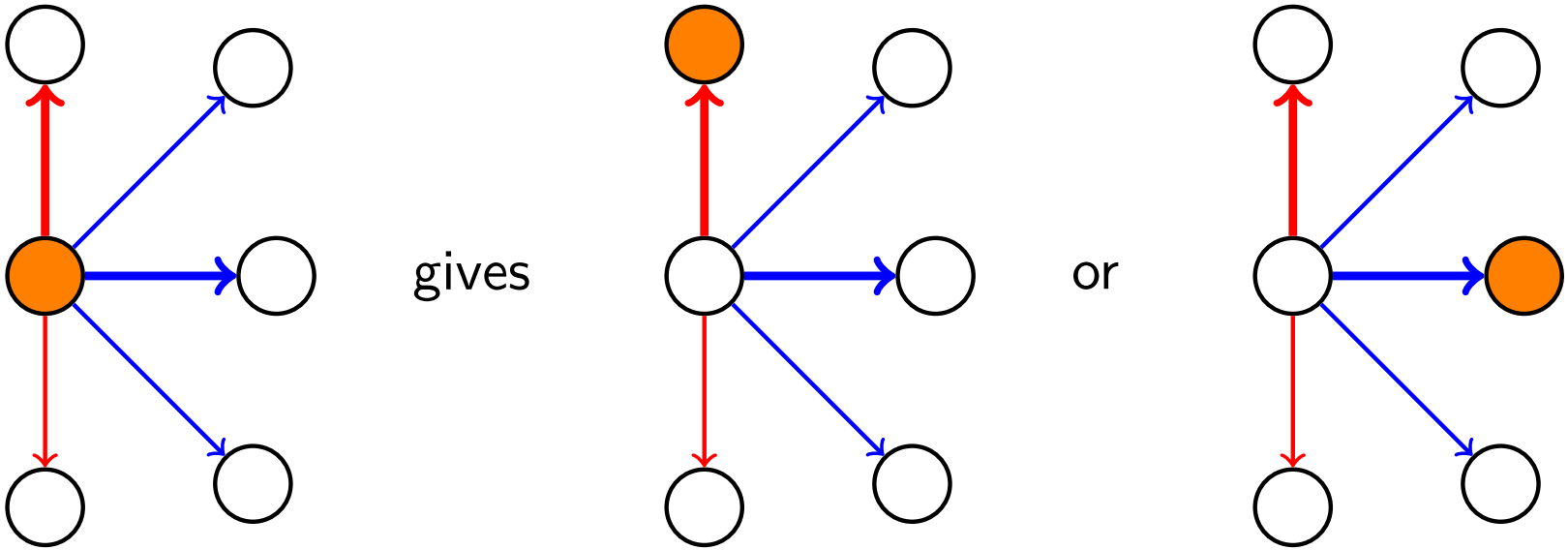
Two-player reachability games : playing and winning

- Strategies for each player allow them only to chose among their own actions ;
- When both players fix their strategies, we obtain a set of plays ;
- This models the outcomes of the race between concurrent actions :



Two-player reachability games : playing and winning

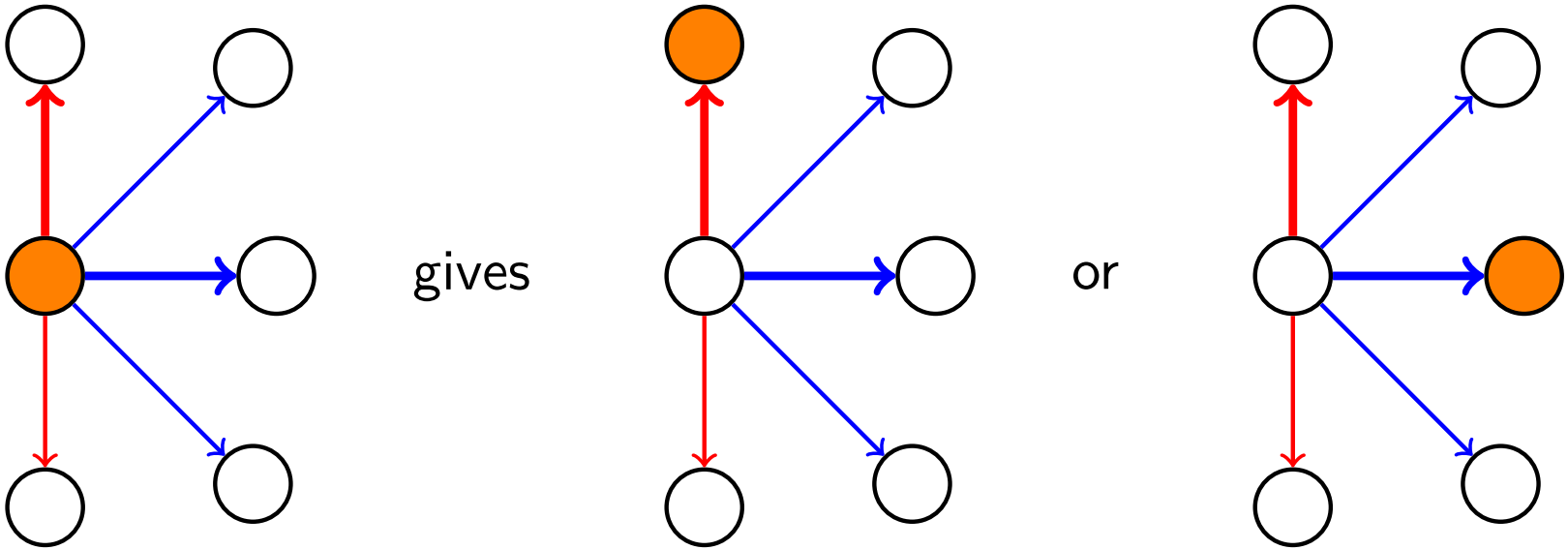
- Strategies for each player allow them only to chose among their own actions ;
- When both players fix their strategies, we obtain a set of plays ;
- This models the outcomes of the race between concurrent actions :



- Worst case : the environment can always be faster if they want to ;

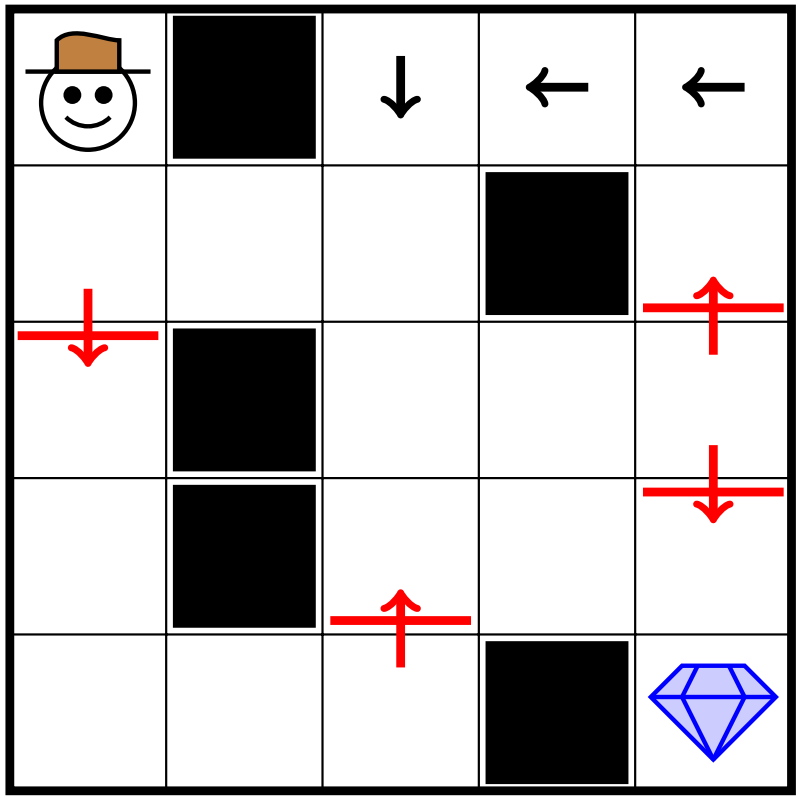
Two-player reachability games : playing and winning

- Strategies for each player allow them only to chose among their own actions ;
- When both players fix their strategies, we obtain a set of plays ;
- This models the outcomes of the race between concurrent actions :



- Worst case : the environment can always be faster if they want to ;
- A (controller) strategy is winning if for all strategies of the environment, all corresponding maximal plays intersect the goal.

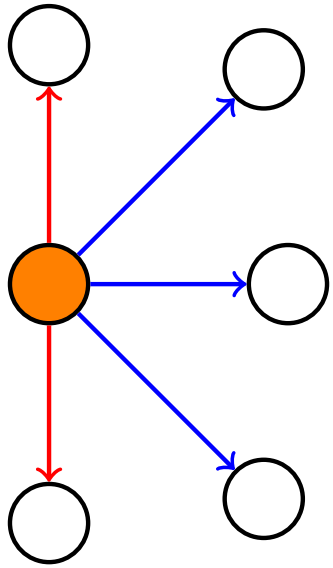
Two-player reachability games



The environment can prevent us from winning :

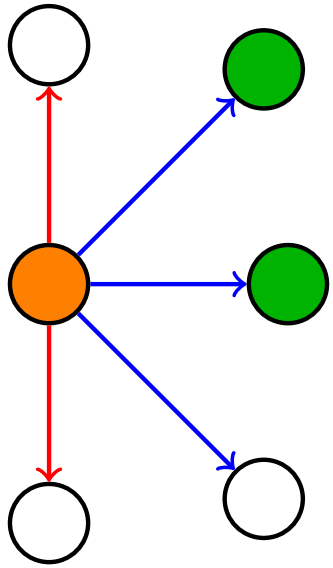
1. By doing something ;
2. By NOT doing something.

Winning states



What makes a winning state ?

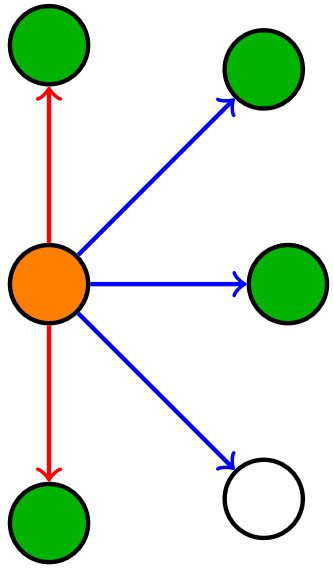
Winning states



What makes a winning state ?

1. at least one controllable winning successor ;

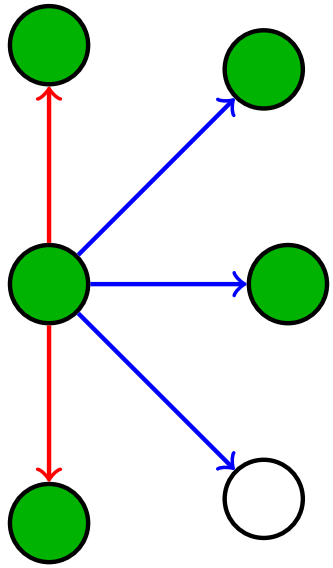
Winning states



What makes a winning state ?

1. at least one controllable winning successor ;
2. only winning uncontrollable successors.

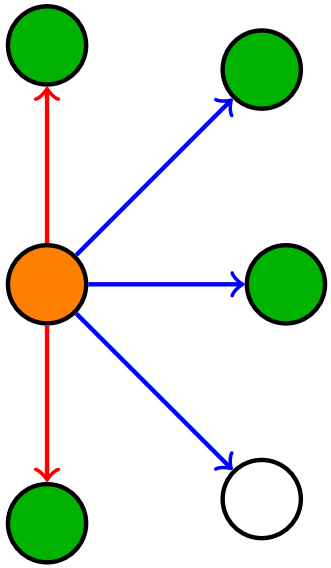
Winning states



What makes a winning state ?

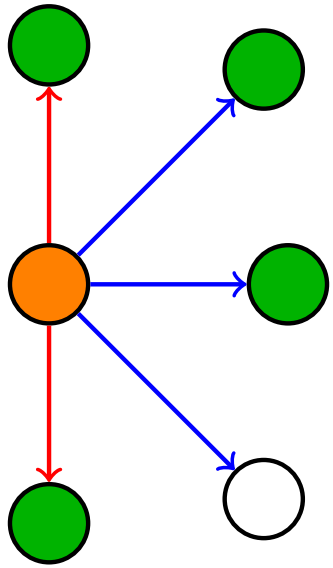
1. at least one controllable winning successor ;
2. only winning uncontrollable successors.

Winning strategy



How to build the winning strategy ?

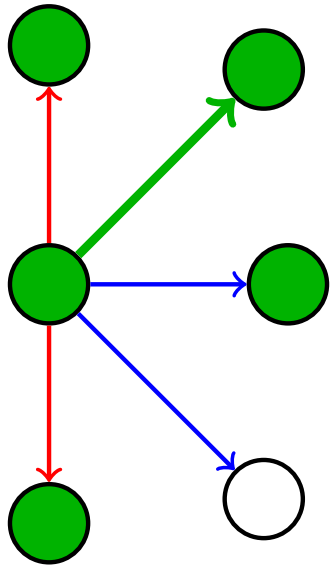
Winning strategy



How to build the winning strategy ?

When a state is first computed to be winning, choose one of its controllable winning successors.

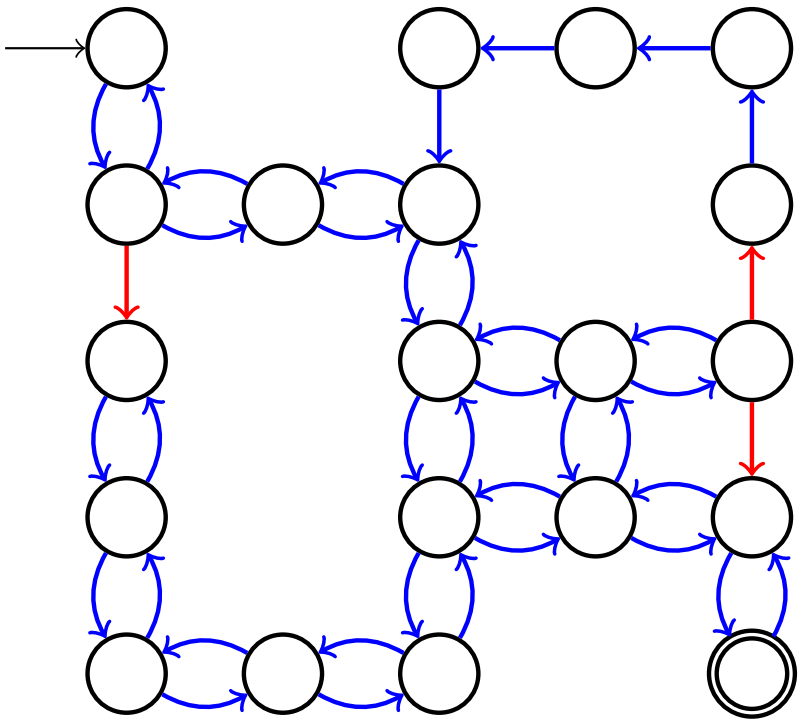
Winning strategy



How to build the winning strategy ?

When a state is first computed to be winning, choose one of its controllable winning successors.

Computing winning states



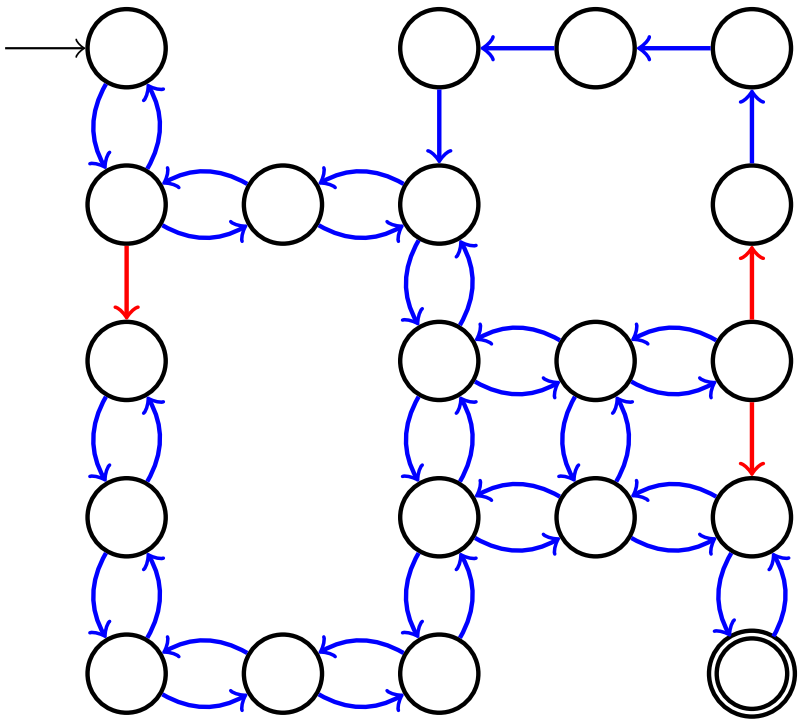
Extend the co-reachability algorithm :

$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

$$\pi(X) =$$

Computing winning states



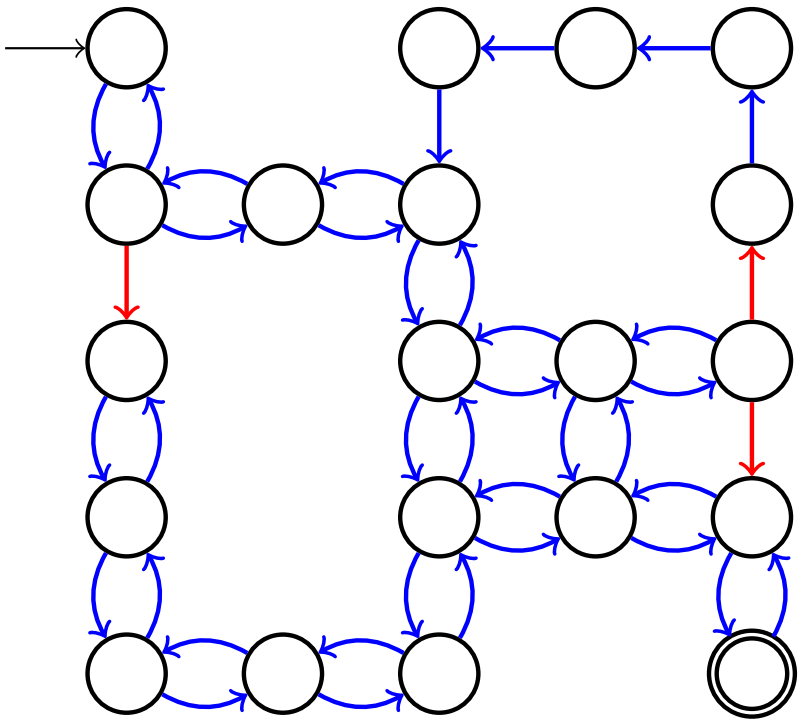
Extend the co-reachability algorithm :

$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

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

Computing winning states



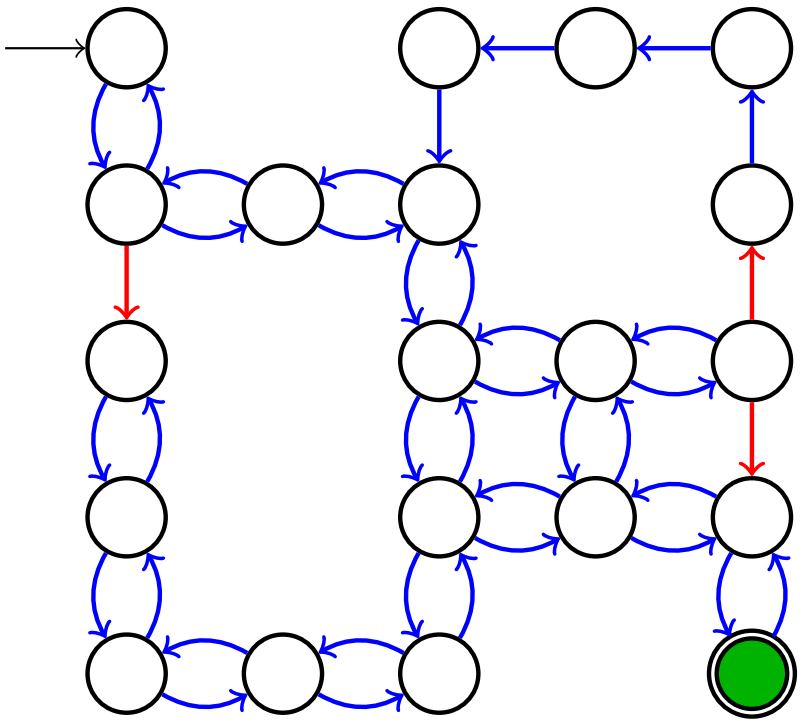
Extend the co-reachability algorithm :

$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

$$\pi(X) = \mathrm{Pre}_c(X) \cap \widetilde{\mathrm{Pre}}_u(X)$$
$$\mathrm{Pre}_c(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \rightarrow s'\}$$
$$\widetilde{\mathrm{Pre}}_u(X) = \{s \in S | s \rightarrow s' \Rightarrow s' \in X\}$$

Computing winning states



Extend the co-reachability algorithm :

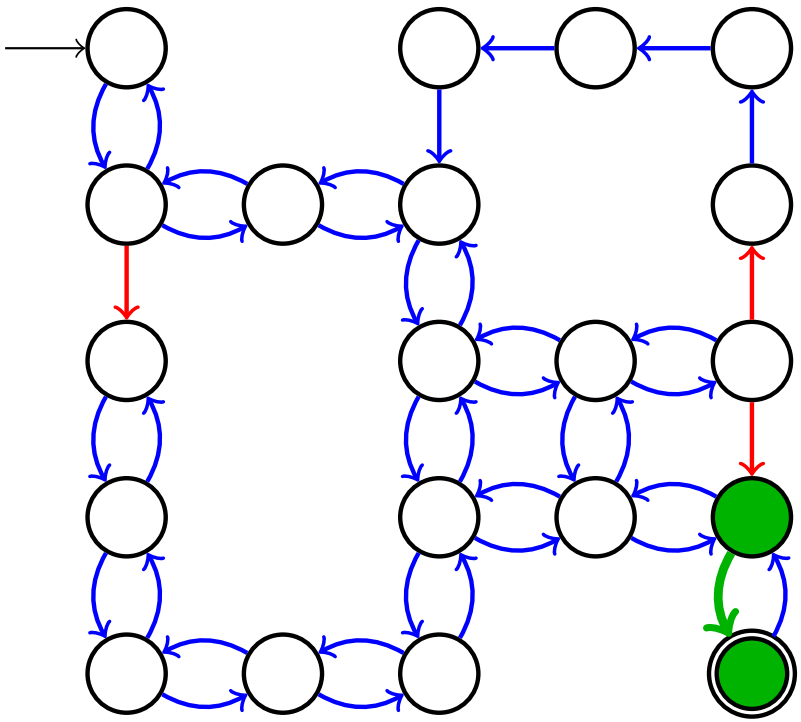
$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

$$\pi(X) = \mathrm{Pre}_c(X) \cap \widetilde{\mathrm{Pre}}_u(X)$$
$$\mathrm{Pre}_c(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \rightarrow s'\}$$
$$\widetilde{\mathrm{Pre}}_u(X) = \{s \in S | s \rightarrow s' \Rightarrow s' \in X\}$$

We also have :
$\pi(X) = \mathrm{Pre}_c(X) \setminus \mathrm{Pre}_u(\overline{X})$

Computing winning states



Extend the co-reachability algorithm :

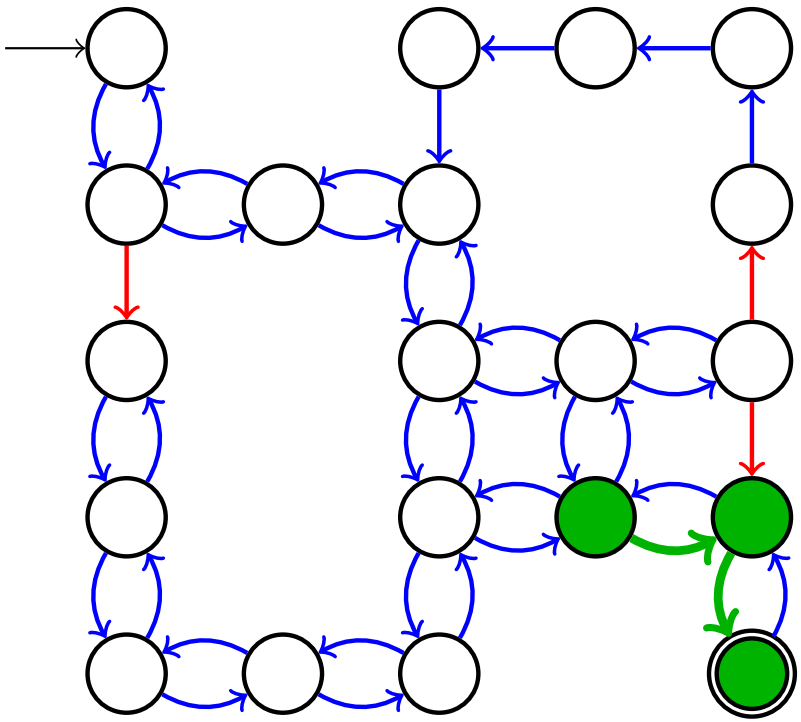
$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

$$\pi(X) = \mathrm{Pre}_c(X) \cap \widetilde{\mathrm{Pre}}_u(X)$$
$$\mathrm{Pre}_c(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \rightarrow s'\}$$
$$\widetilde{\mathrm{Pre}}_u(X) = \{s \in S | s \rightarrow s' \Rightarrow s' \in X\}$$

We also have :
$\pi(X) = \mathrm{Pre}_c(X) \setminus \mathrm{Pre}_u(\overline{X})$

Computing winning states



Extend the co-reachability algorithm :

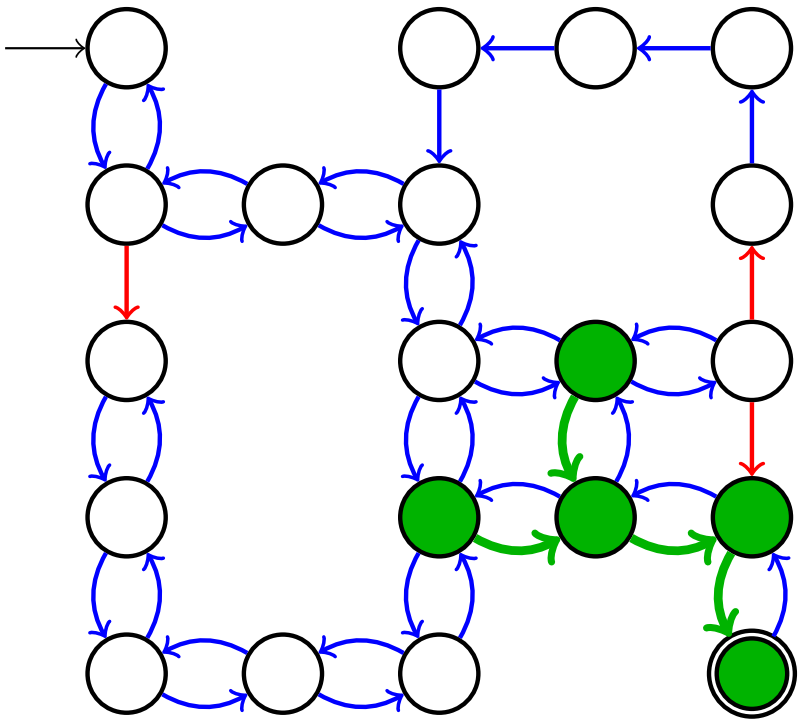
$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

$$\pi(X) = \mathrm{Pre}_c(X) \cap \widetilde{\mathrm{Pre}}_u(X)$$
$$\mathrm{Pre}_c(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \rightarrow s'\}$$
$$\widetilde{\mathrm{Pre}}_u(X) = \{s \in S | s \rightarrow s' \Rightarrow s' \in X\}$$

We also have :
$\pi(X) = \mathrm{Pre}_c(X) \setminus \mathrm{Pre}_u(\overline{X})$

Computing winning states



Extend the co-reachability algorithm :

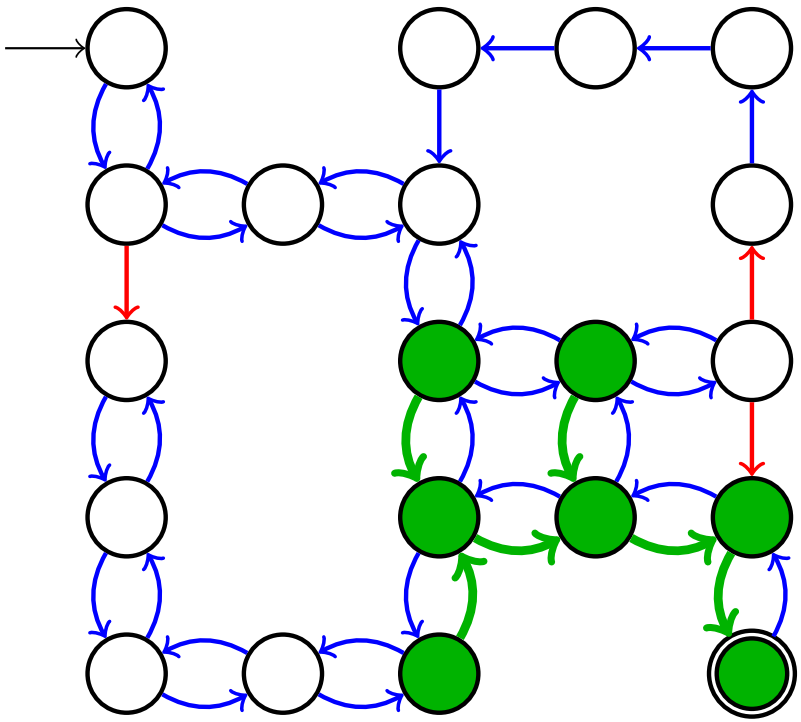
$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

$$\pi(X) = \mathrm{Pre}_c(X) \cap \widetilde{\mathrm{Pre}}_u(X)$$
$$\mathrm{Pre}_c(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \rightarrow s'\}$$
$$\widetilde{\mathrm{Pre}}_u(X) = \{s \in S | s \rightarrow s' \Rightarrow s' \in X\}$$

We also have :
$\pi(X) = \mathrm{Pre}_c(X) \setminus \mathrm{Pre}_u(\overline{X})$

Computing winning states



Extend the co-reachability algorithm :

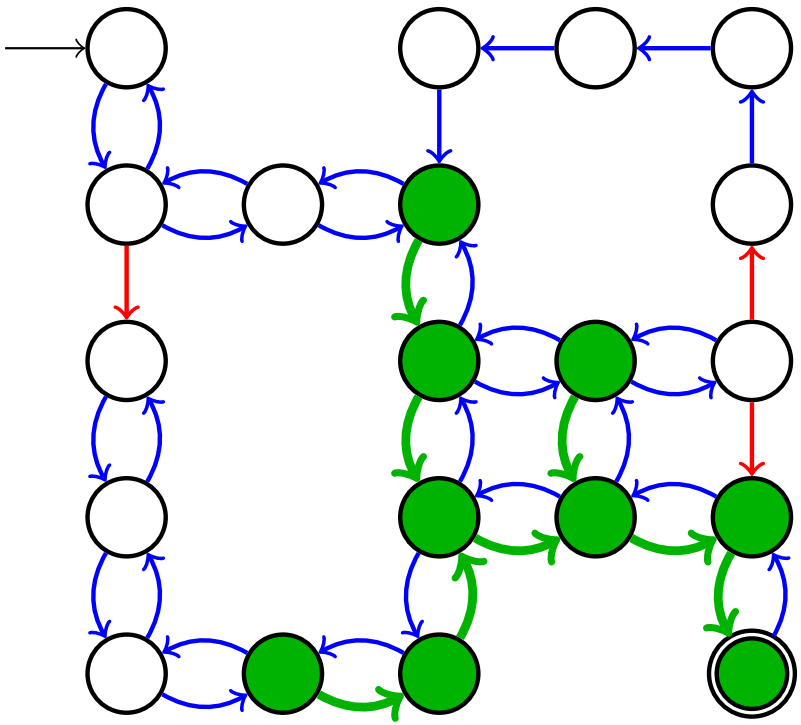
$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

$$\pi(X) = \mathrm{Pre}_c(X) \cap \widetilde{\mathrm{Pre}}_u(X)$$
$$\mathrm{Pre}_c(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \rightarrow s'\}$$
$$\widetilde{\mathrm{Pre}}_u(X) = \{s \in S | s \rightarrow s' \Rightarrow s' \in X\}$$

We also have :
$\pi(X) = \mathrm{Pre}_c(X) \setminus \mathrm{Pre}_u(\overline{X})$

Computing winning states



Extend the co-reachability algorithm :

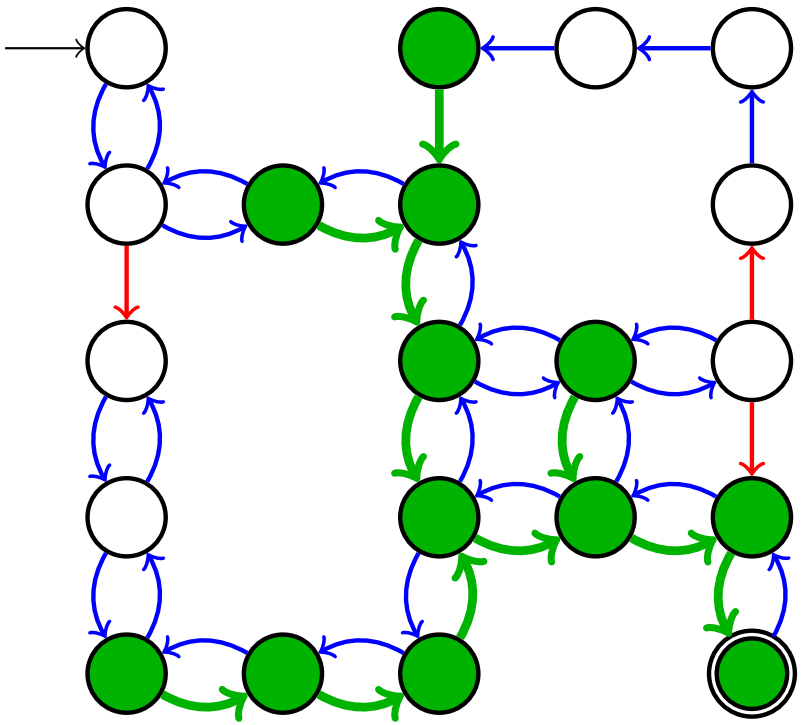
$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

$$\pi(X) = \mathrm{Pre}_c(X) \cap \widetilde{\mathrm{Pre}}_u(X)$$
$$\mathrm{Pre}_c(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \rightarrow s'\}$$
$$\widetilde{\mathrm{Pre}}_u(X) = \{s \in S | s \rightarrow s' \Rightarrow s' \in X\}$$

We also have :
$\pi(X) = \mathrm{Pre}_c(X) \setminus \mathrm{Pre}_u(\overline{X})$

Computing winning states



Extend the co-reachability algorithm :

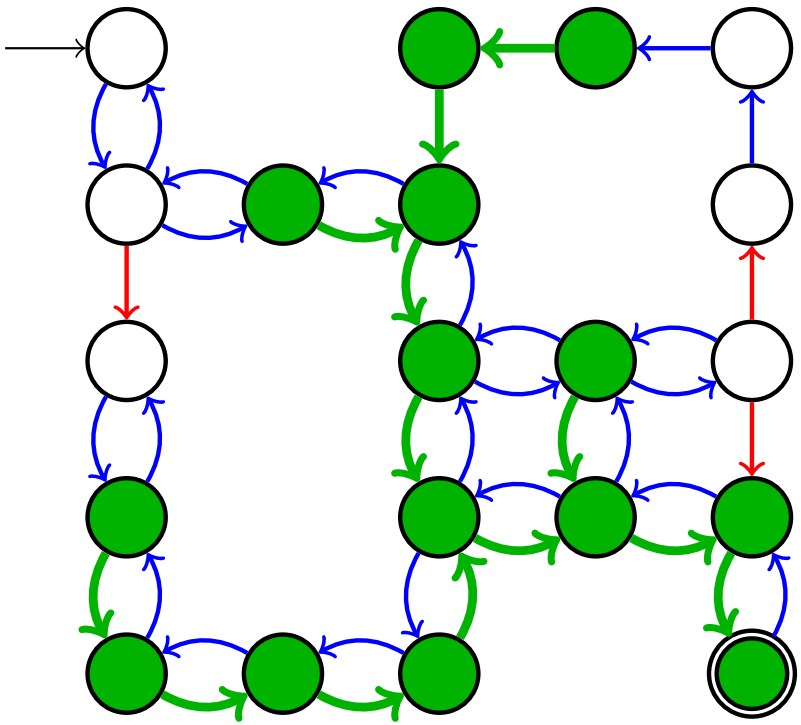
$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

$$\pi(X) = \mathrm{Pre}_c(X) \cap \widetilde{\mathrm{Pre}}_u(X)$$
$$\mathrm{Pre}_c(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \rightarrow s'\}$$
$$\widetilde{\mathrm{Pre}}_u(X) = \{s \in S | s \rightarrow s' \Rightarrow s' \in X\}$$

We also have :
$\pi(X) = \mathrm{Pre}_c(X) \setminus \mathrm{Pre}_u(\overline{X})$

Computing winning states



Extend the co-reachability algorithm :

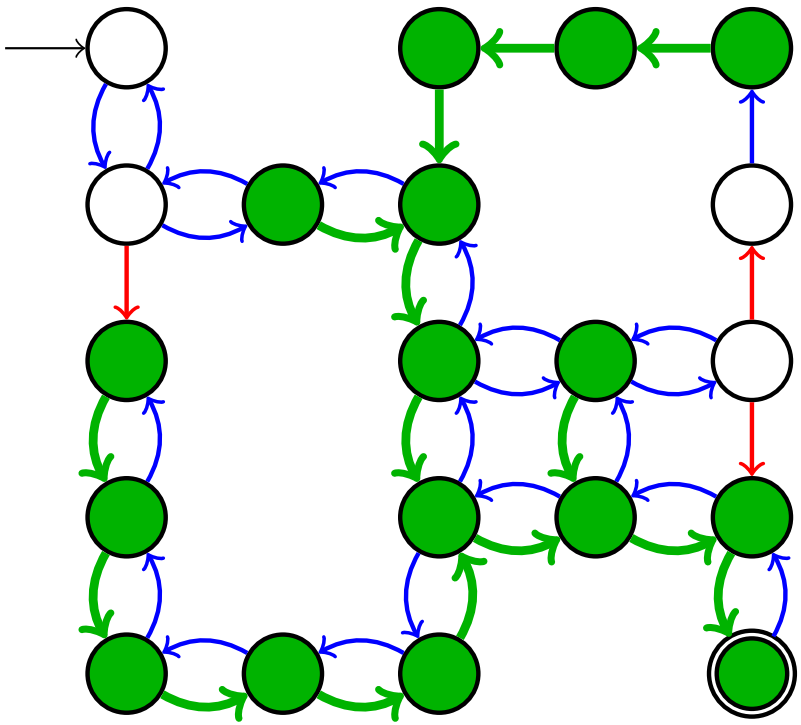
$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

$$\pi(X) = \mathrm{Pre}_c(X) \cap \widetilde{\mathrm{Pre}}_u(X)$$
$$\mathrm{Pre}_c(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \rightarrow s'\}$$
$$\widetilde{\mathrm{Pre}}_u(X) = \{s \in S | s \rightarrow s' \Rightarrow s' \in X\}$$

We also have :
$\pi(X) = \mathrm{Pre}_c(X) \setminus \mathrm{Pre}_u(\overline{X})$

Computing winning states



Extend the co-reachability algorithm :

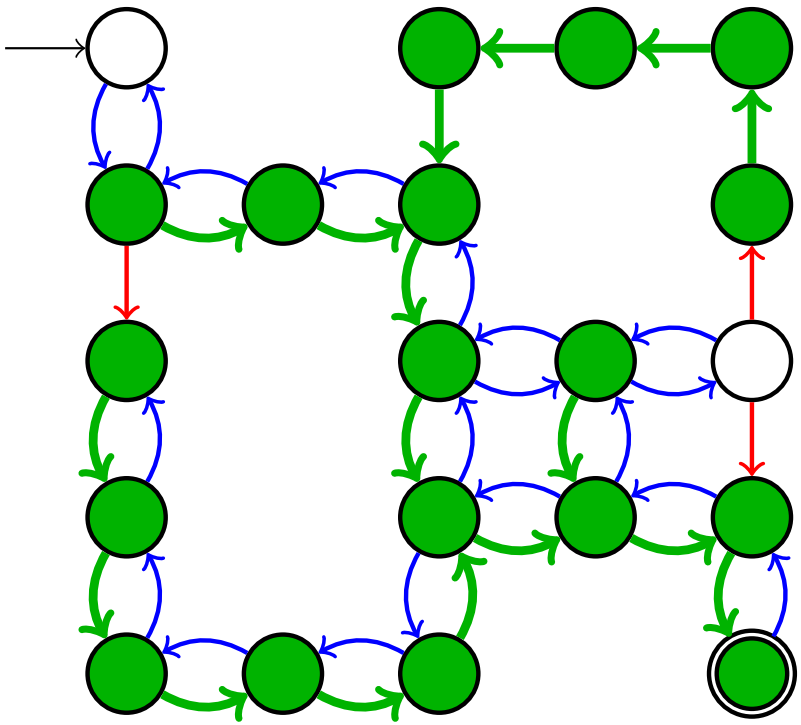
$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

$$\pi(X) = \mathrm{Pre}_c(X) \cap \widetilde{\mathrm{Pre}}_u(X)$$
$$\mathrm{Pre}_c(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \rightarrow s'\}$$
$$\widetilde{\mathrm{Pre}}_u(X) = \{s \in S | s \rightarrow s' \Rightarrow s' \in X\}$$

We also have :
$\pi(X) = \mathrm{Pre}_c(X) \setminus \mathrm{Pre}_u(\overline{X})$

Computing winning states



Extend the co-reachability algorithm :

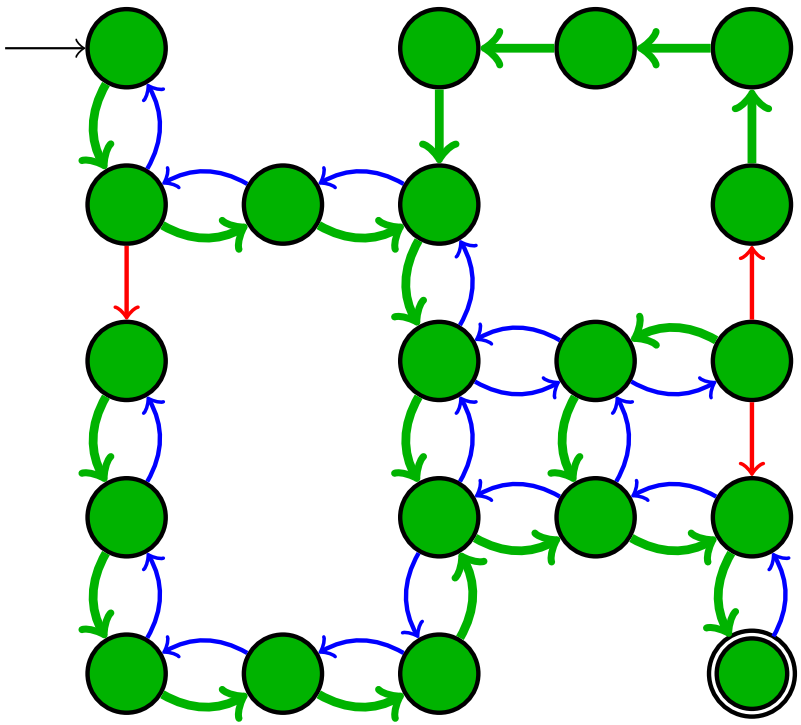
$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

$$\pi(X) = \mathrm{Pre}_c(X) \cap \widetilde{\mathrm{Pre}}_u(X)$$
$$\mathrm{Pre}_c(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \rightarrow s'\}$$
$$\widetilde{\mathrm{Pre}}_u(X) = \{s \in S | s \rightarrow s' \Rightarrow s' \in X\}$$

We also have :
$\pi(X) = \mathrm{Pre}_c(X) \setminus \mathrm{Pre}_u(\overline{X})$

Computing winning states



Extend the co-reachability algorithm :

$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

$$\pi(X) = \mathrm{Pre}_c(X) \cap \widetilde{\mathrm{Pre}}_u(X)$$
$$\mathrm{Pre}_c(X) = \{s \in S | \exists s' \in X \text{ s. t. } s \rightarrow s'\}$$
$$\widetilde{\mathrm{Pre}}_u(X) = \{s \in S | s \rightarrow s' \Rightarrow s' \in X\}$$

We also have :
$\pi(X) = \mathrm{Pre}_c(X) \setminus \mathrm{Pre}_u(\overline{X})$

Games Vs Verification

- How can we check (classical) reachability with a two-player game ?

Games Vs Verification

- How can we check (classical) reachability with a two-player game ?
  Make all transitions controllable

Games Vs Verification

- How can we check (classical) reachability with a two-player game ?
  Make all transitions controllable
- How can we check inevitability (AF) with a two-player game ?

Games Vs Verification

- How can we check (classical) reachability with a two-player game ?
  Make all transitions controllable
- How can we check inevitability (AF) with a two-player game ?
  Make all transitions uncontrollable

Games Vs Verification

- How can we check (classical) reachability with a two-player game ?
  Make all transitions controllable
- How can we check inevitability (AF) with a two-player game ?
  Make all transitions uncontrollable
  ...and add a controllable copy (because the environment could just refuse to play.)

Two-player safety game

- The reachability game is : there exists a strategy for the controller such that all maximal plays contain a goal state ;

Two-player safety game

- The reachability game is : there exists a strategy for the controller such that all maximal plays contain a goal state ;
- We could define the safety game in several ways :

Two-player safety game

- The reachability game is : there exists a strategy for the controller such that all maximal plays contain a goal state ;
- We could define the safety game in several ways :
  1. for all strategies of the controller, all maximal plays contain only safe states ;

Two-player safety game

- The reachability game is : there exists a strategy for the controller such that all maximal plays contain a goal state ;
- We could define the safety game in several ways :
  1. for all strategies of the controller, all maximal plays contain only safe states ;
  2. there exists a strategy for the controller such that all maximal plays contain only safe states ;

Two-player safety game

- The reachability game is : there exists a strategy for the controller such that all maximal plays contain a goal state ;
- We could define the safety game in several ways :
  1. for all strategies of the controller, all maximal plays contain only safe states ;
  2. there exists a strategy for the controller such that all maximal plays contain only safe states ;
- The first one consists in negating the fix point for the reachability game ;

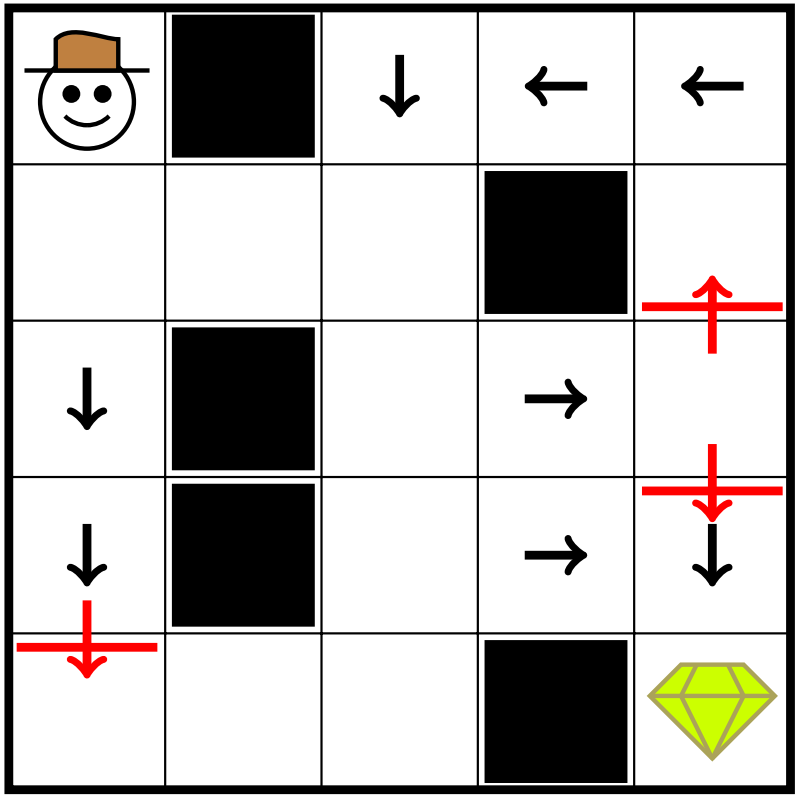
Two-player safety game

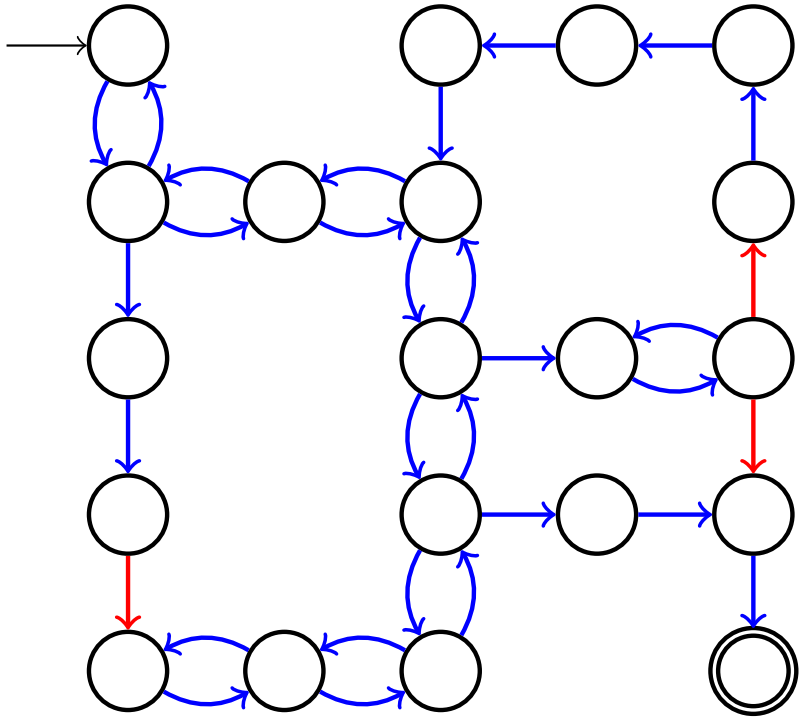
- The reachability game is : there exists a strategy for the controller such that all maximal plays contain a goal state ;
- We could define the safety game in several ways :
  1. for all strategies of the controller, all maximal plays contain only safe states ;
  2. there exists a strategy for the controller such that all maximal plays contain only safe states ;
- The first one consists in negating the fix point for the reachability game ;
- The second one can be computed by :

$$X_0 = G$$
$$X_{n+1} = X_n \cap \pi(X_n)$$

Two-player safety game

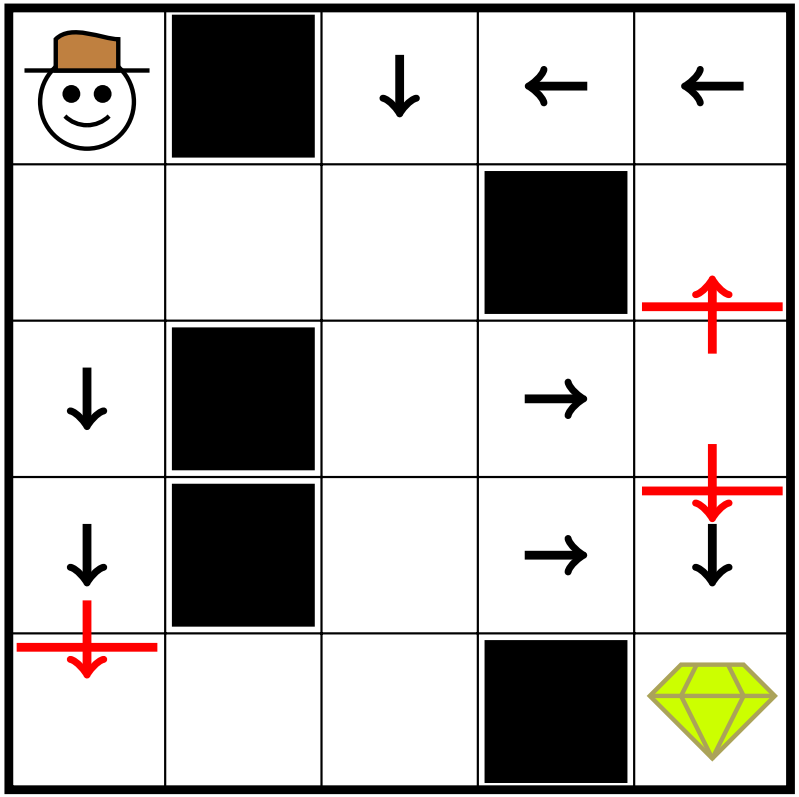
- Does  have a strategy to avoid  (it's a trap !) and being unable to move ?

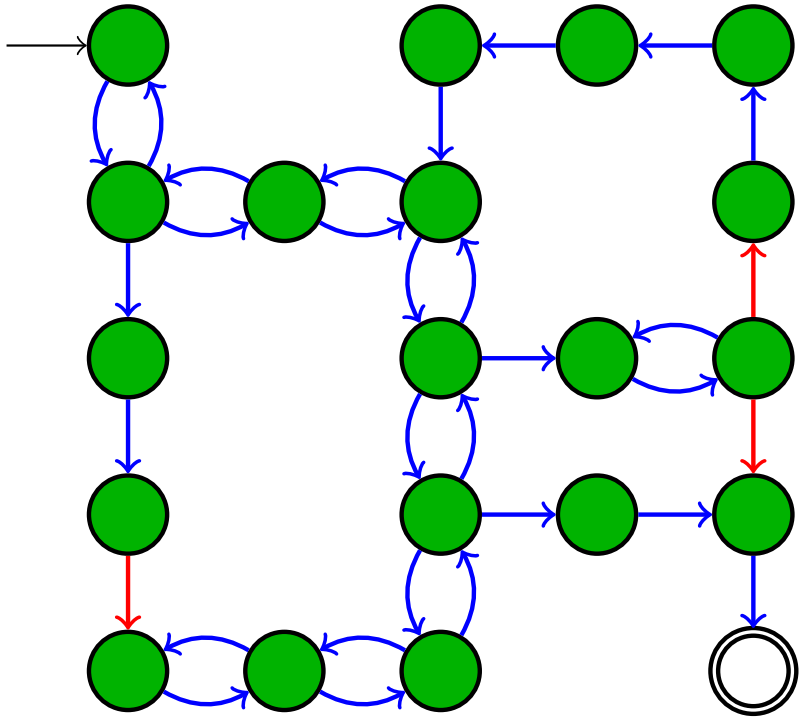




Two-player safety game

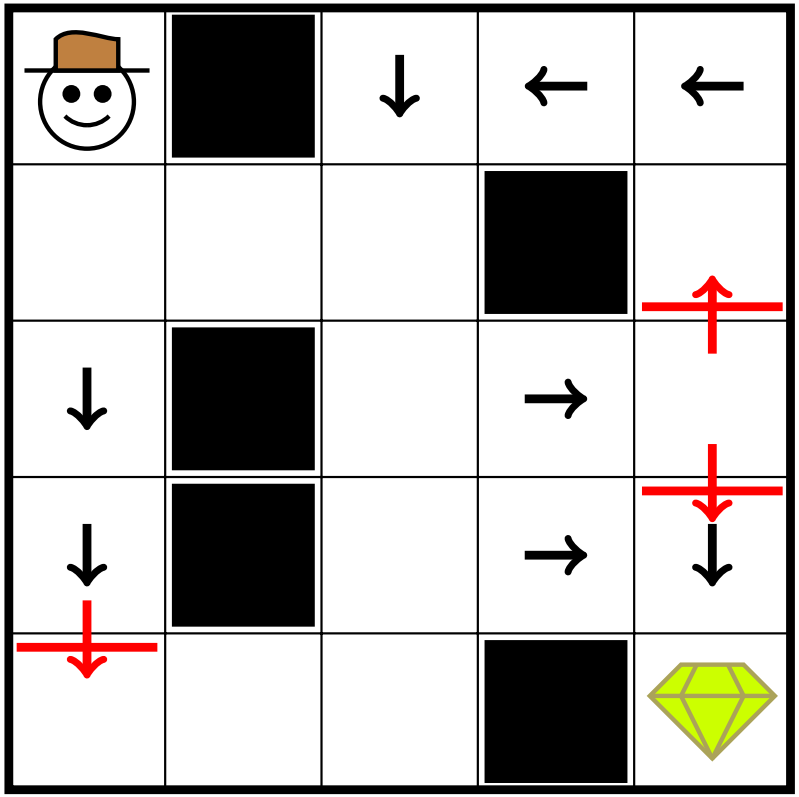
- Does  have a strategy to avoid  (it's a trap !) and being unable to move ?

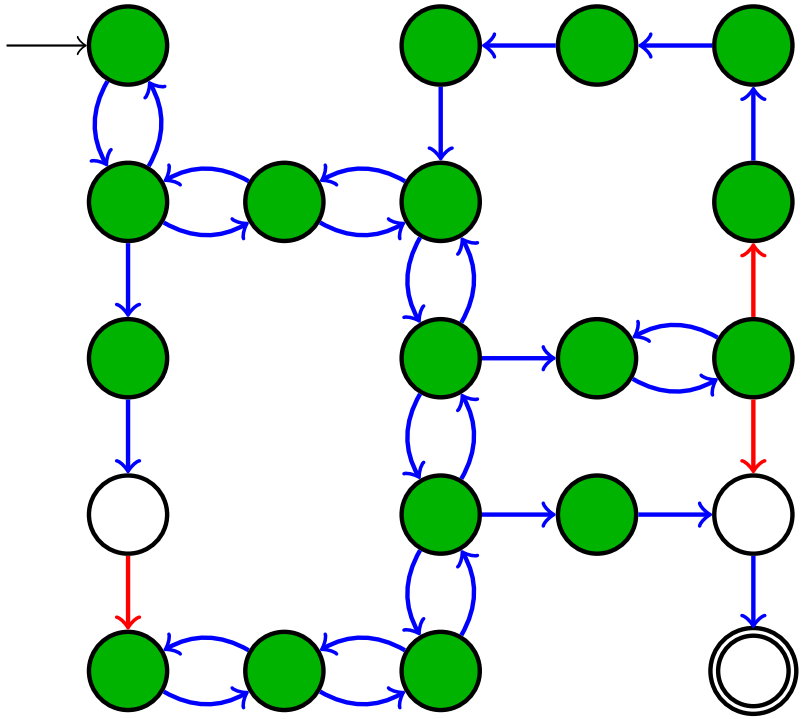




Two-player safety game

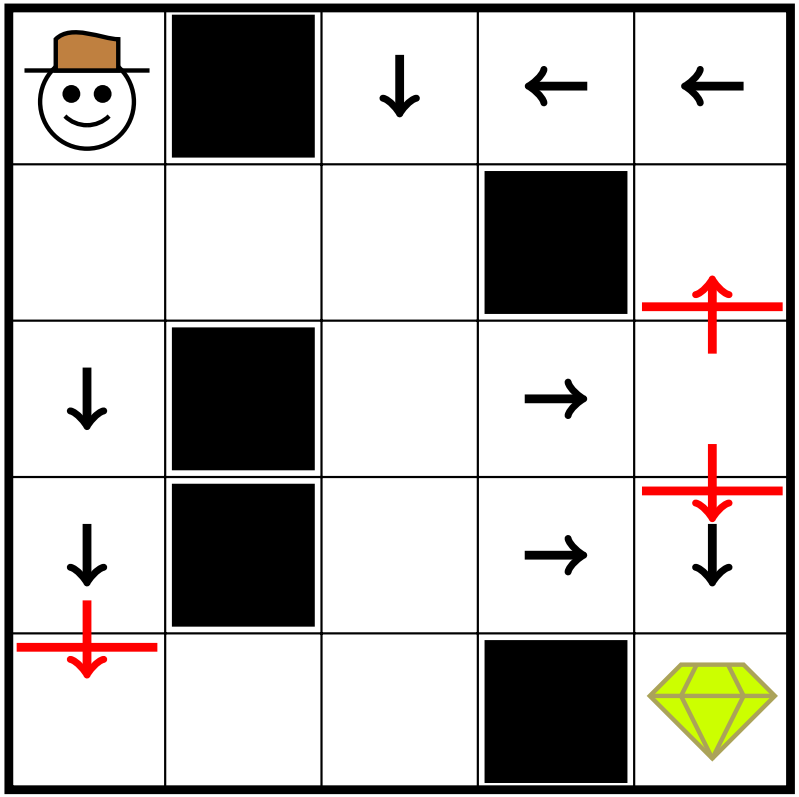
- Does  have a strategy to avoid  (it's a trap !) and being unable to move ?

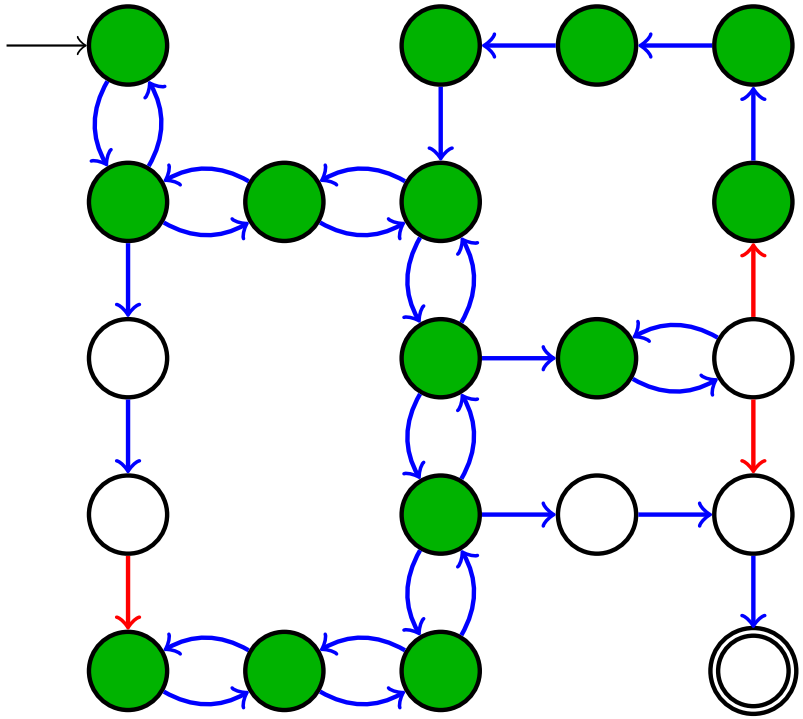




Two-player safety game

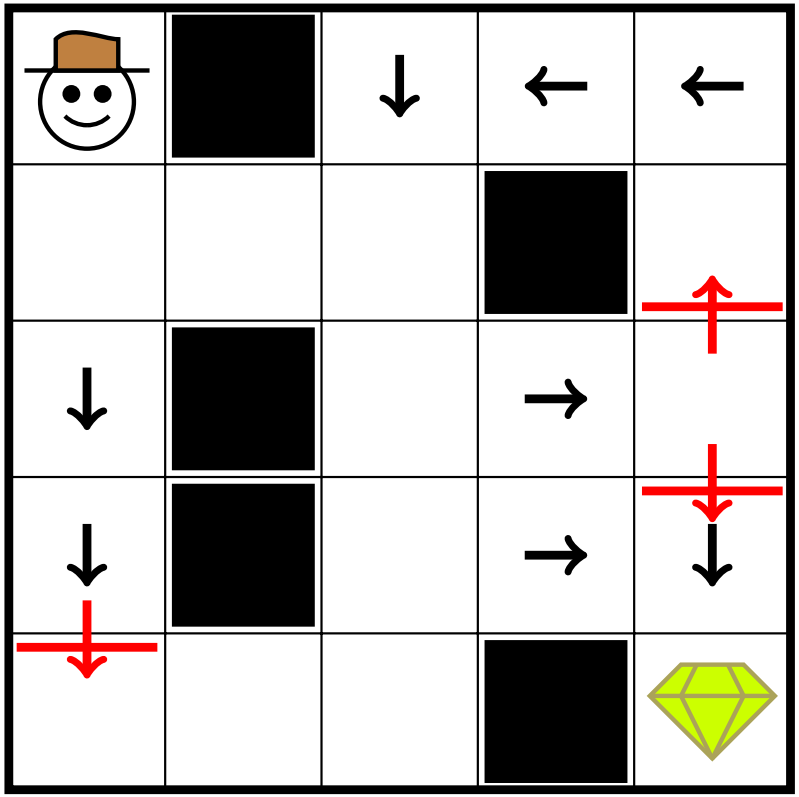
- Does  have a strategy to avoid  (it's a trap !) and being unable to move ?

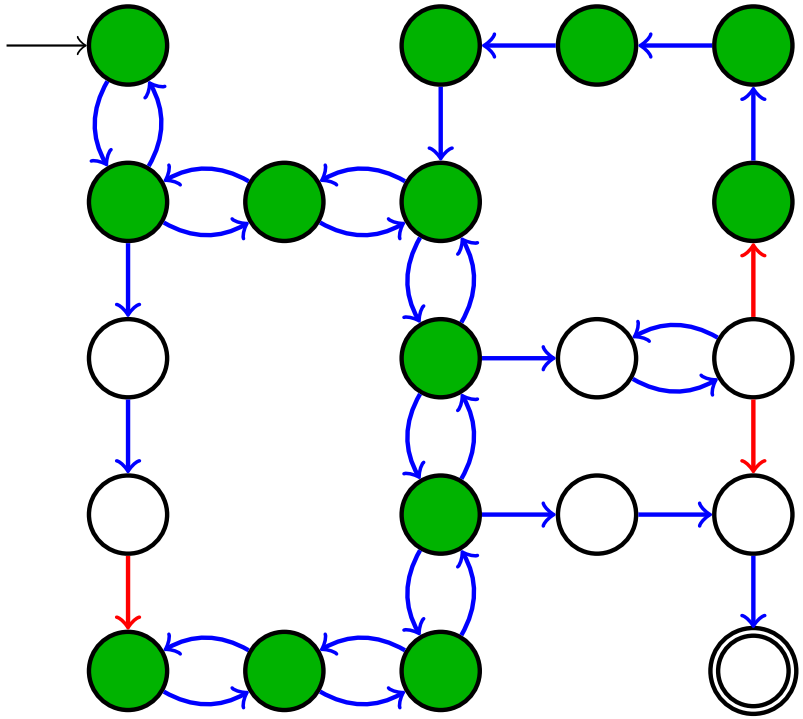




Two-player safety game

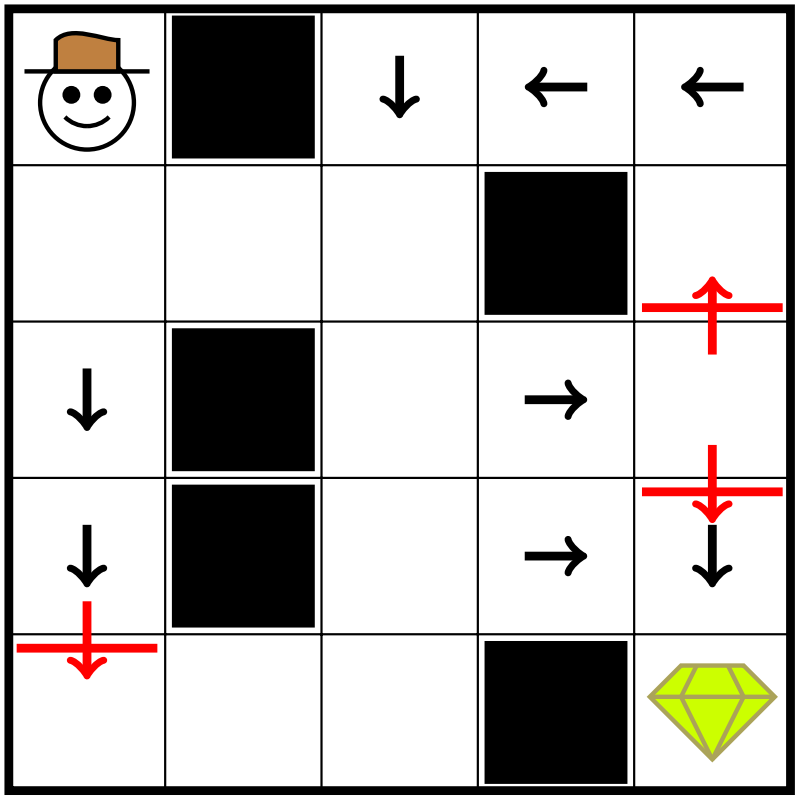
- Does  have a strategy to avoid  (it's a trap !) and being unable to move ?

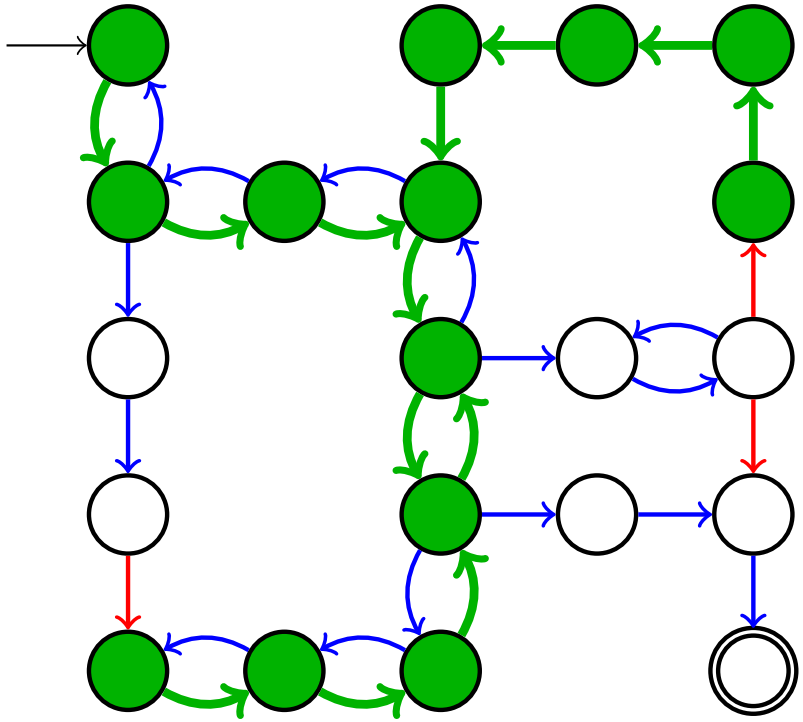




Two-player safety game

- Does  have a strategy to avoid  (it's a trap !) and being unable to move ?

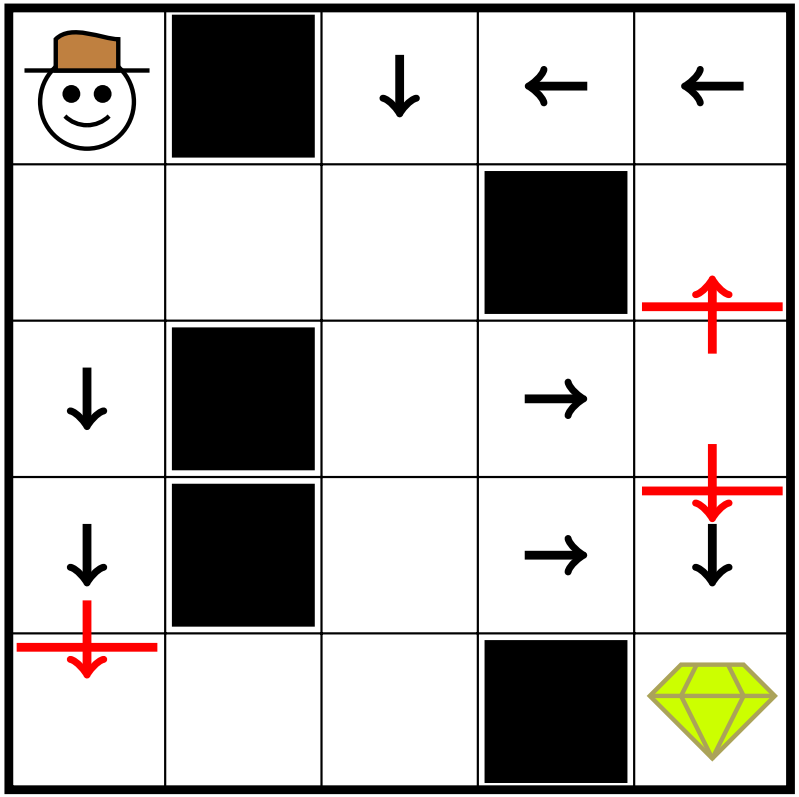


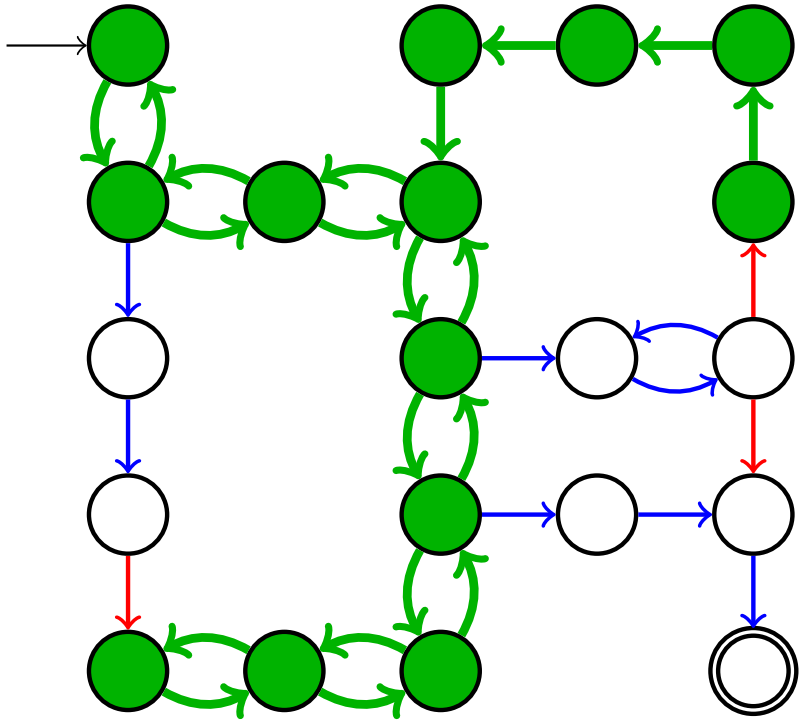


In winning states, a winning strategy is any controllable action to a winning state.

Two-player safety game

- Does  have a strategy to avoid  (it's a trap !) and being unable to move ?

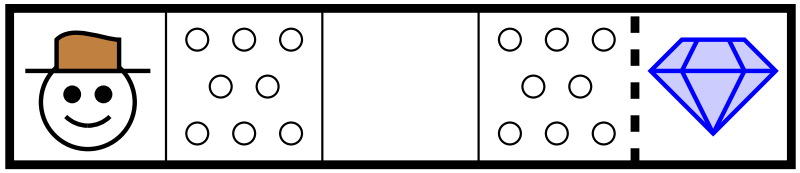




Any strategy that always goes to winning states is winning, memoryless or not ! We can define a most permissive strategy (which is not a strategy...)

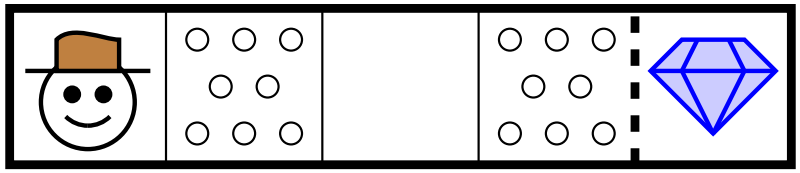
Two-player Timed Games

Timed games

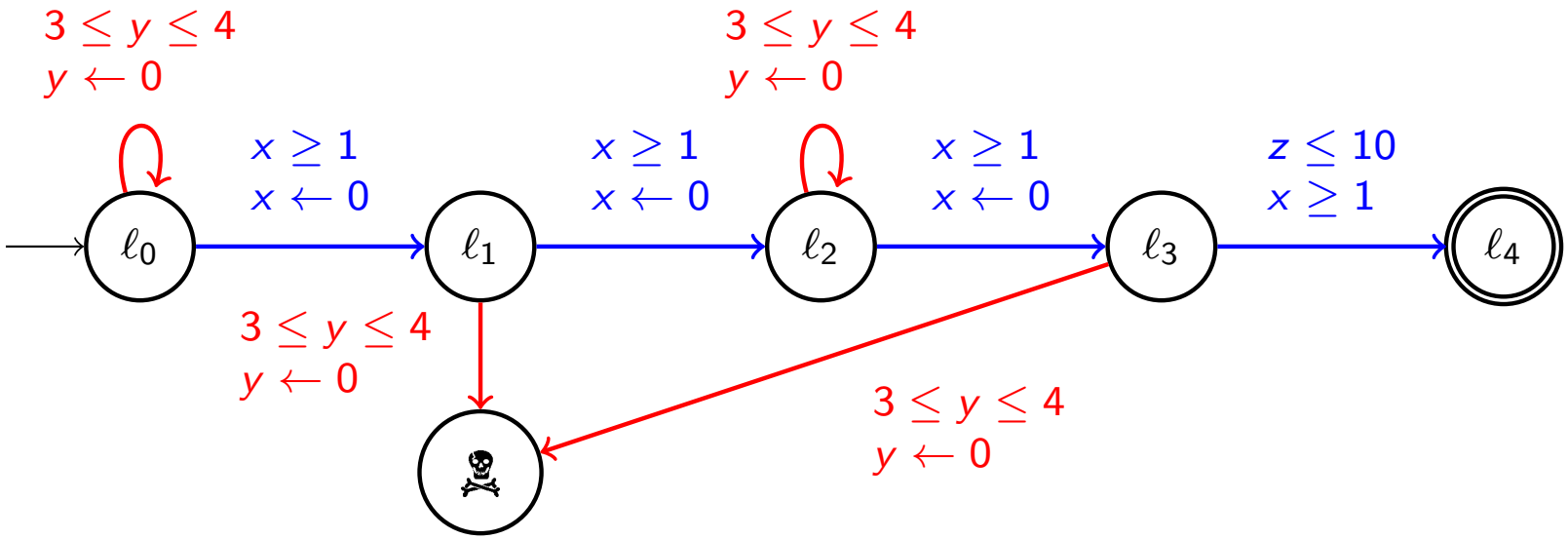


1. The door to the  closes after 10 t.u. ;
2. Spikes () thrust upward with a period between 3 and 4 t.u. ;
3.  needs at least 1 t.u. to move to the next tile.

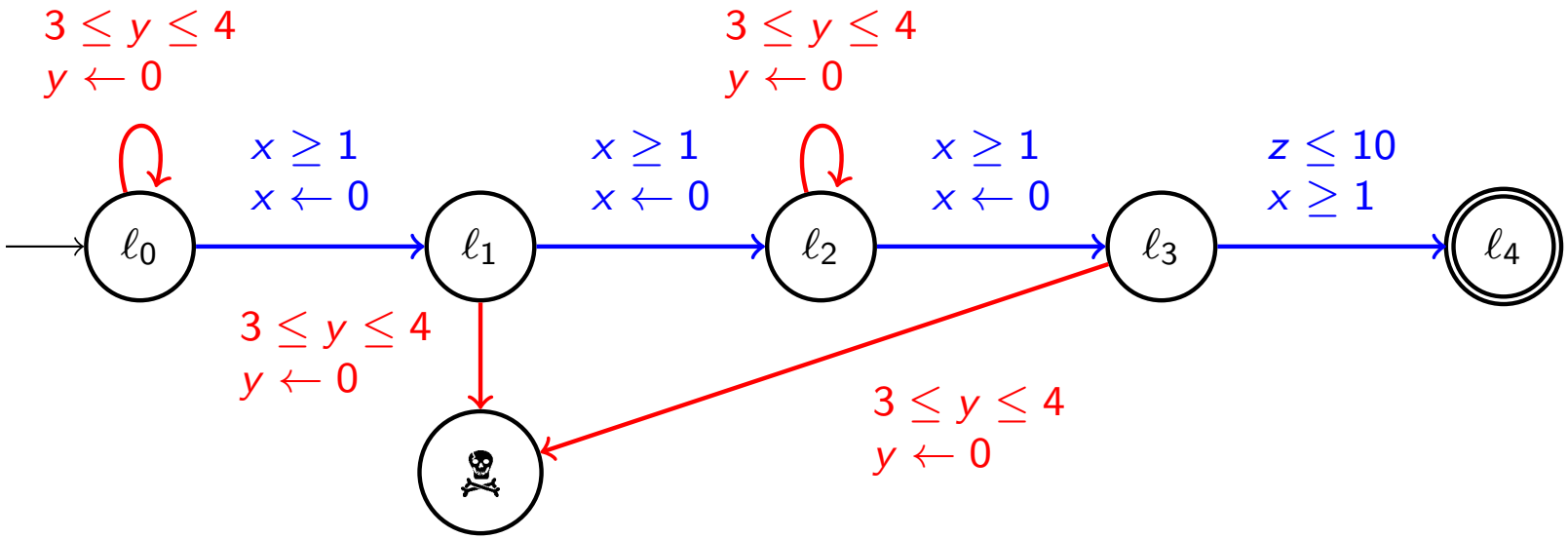
Timed games



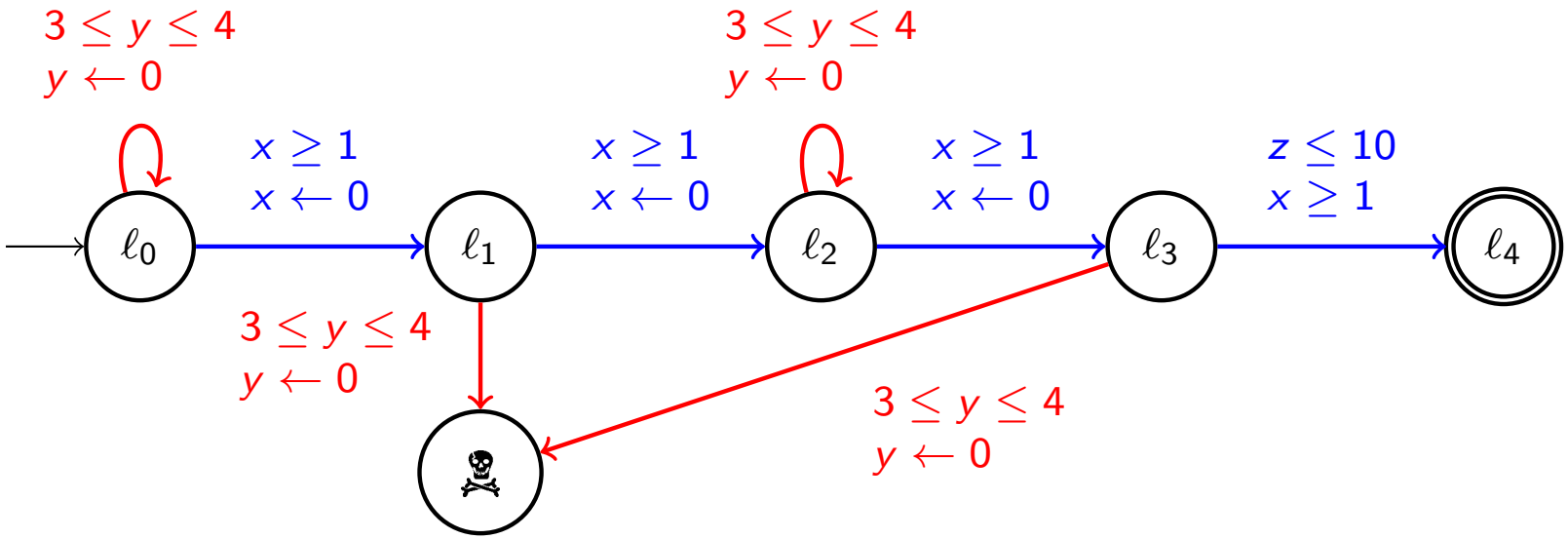
1. The door to the 💎 closes after 10 t.u. ;
2. Spikes ( ) thrust upward with a period between 3 and 4 t.u. ;
3. 🤠 needs at least 1 t.u. to move to the next tile.



Strategies in timed games

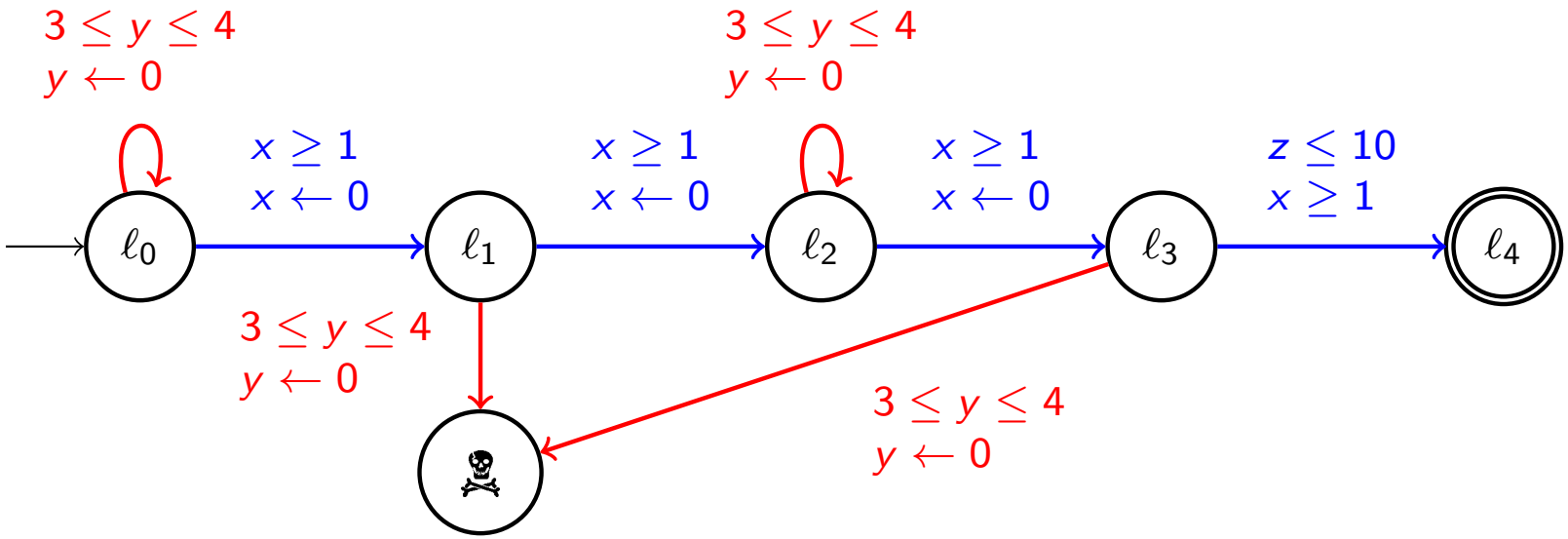


Strategies in timed games



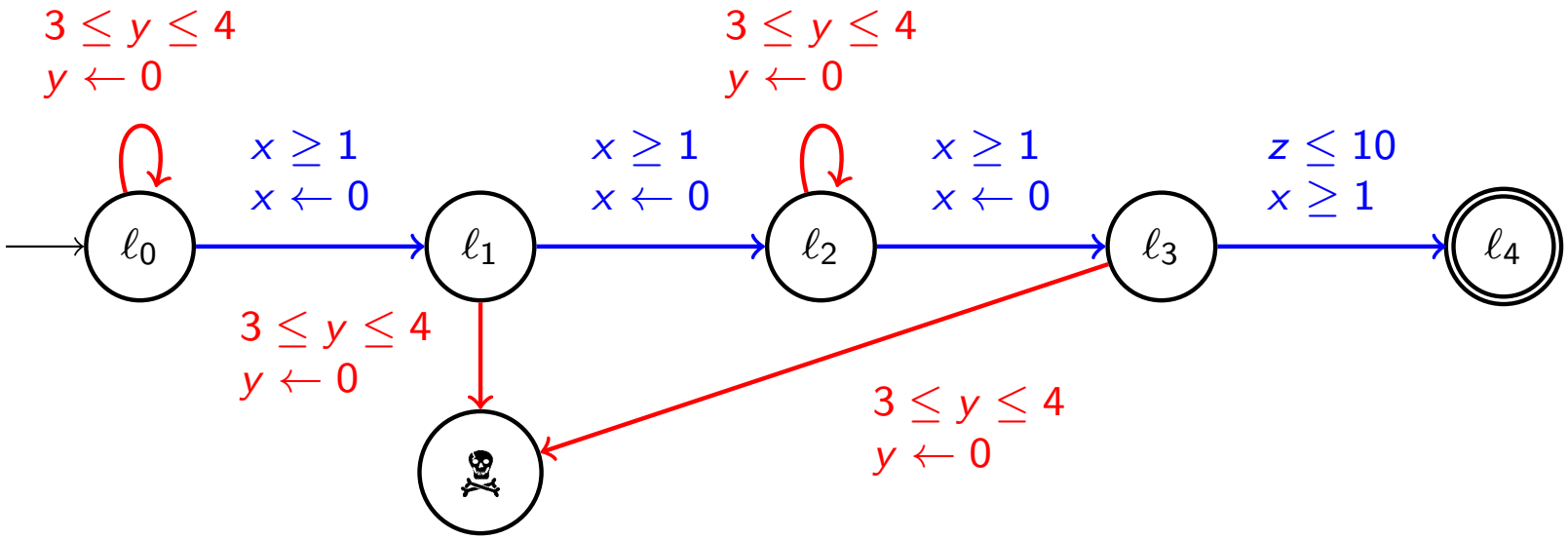
- States : (ℓ, x, y, z) ;

Strategies in timed games



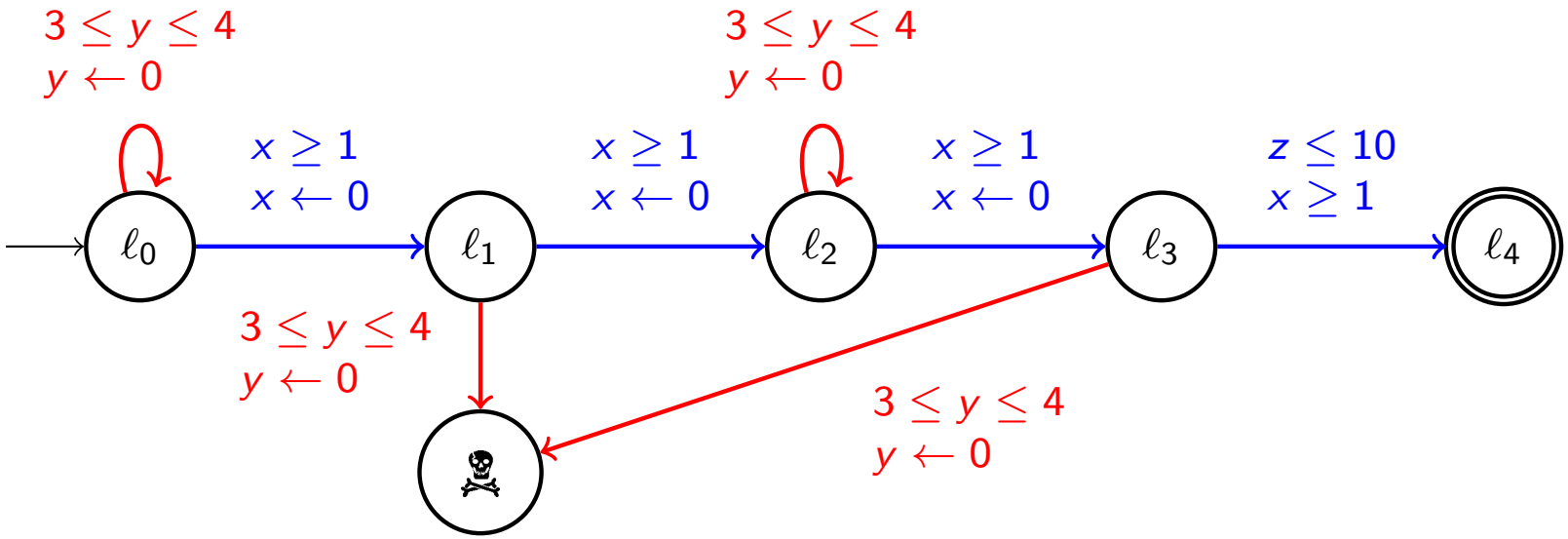
- States : (ℓ, x, y, z) ;
- Strategies assign to histories/states either an action or delay (δ) ;

Strategies in timed games



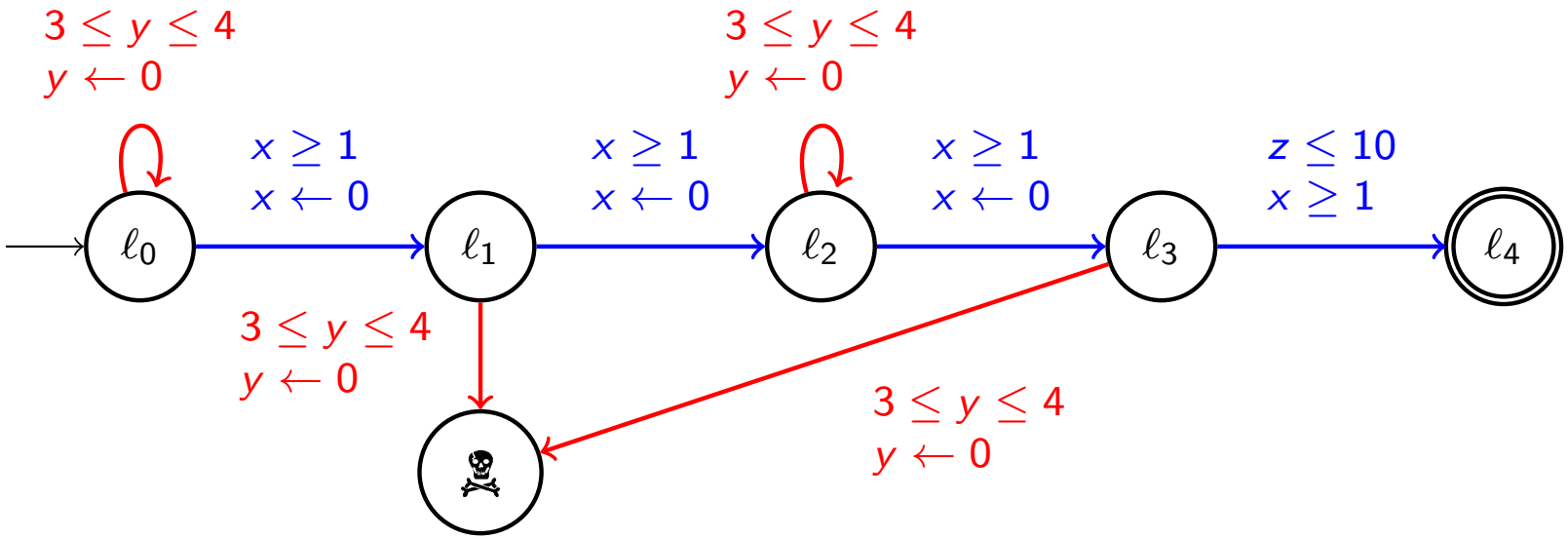
- States : (ℓ, x, y, z) ;
- Strategies assign to histories/states either an action or delay (δ) ;
- If one plays δ and the other an action, the action is played ;

Strategies in timed games



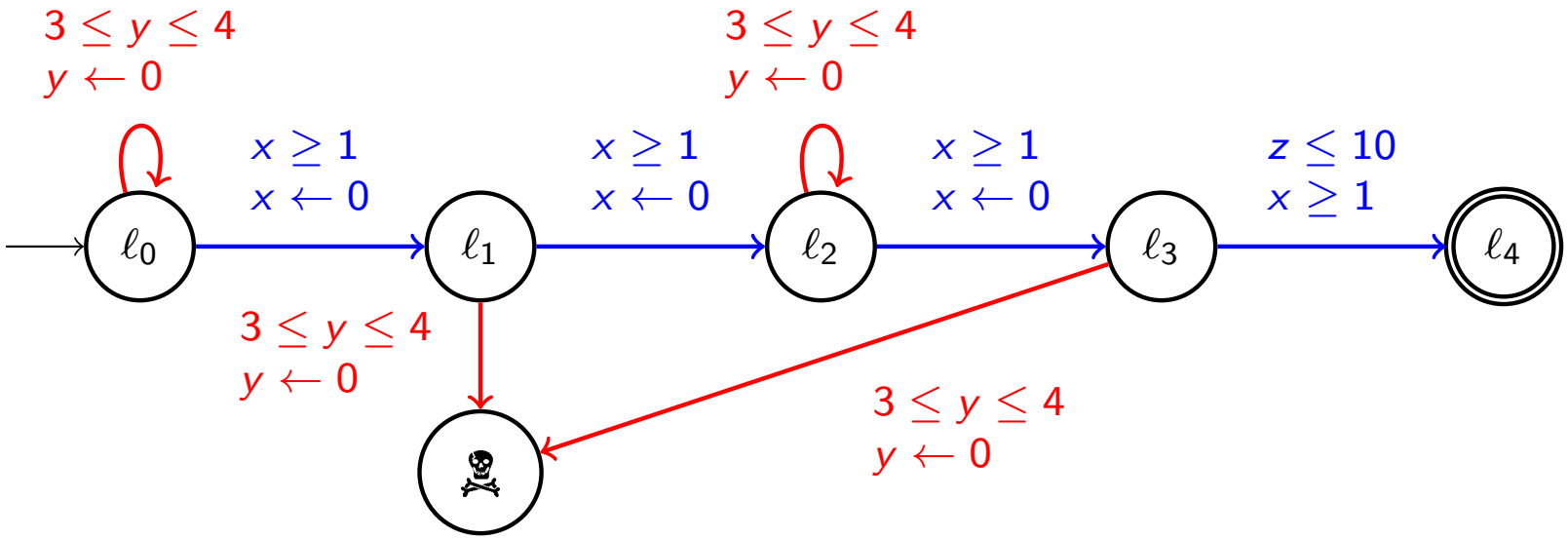
- States : (ℓ, x, y, z) ;
- Strategies assign to histories/states either an action or delay (δ) ;
- If one plays δ and the other an action, the action is played ;
- If both play δ, we wait until one of the strategy changes ;

Strategies in timed games



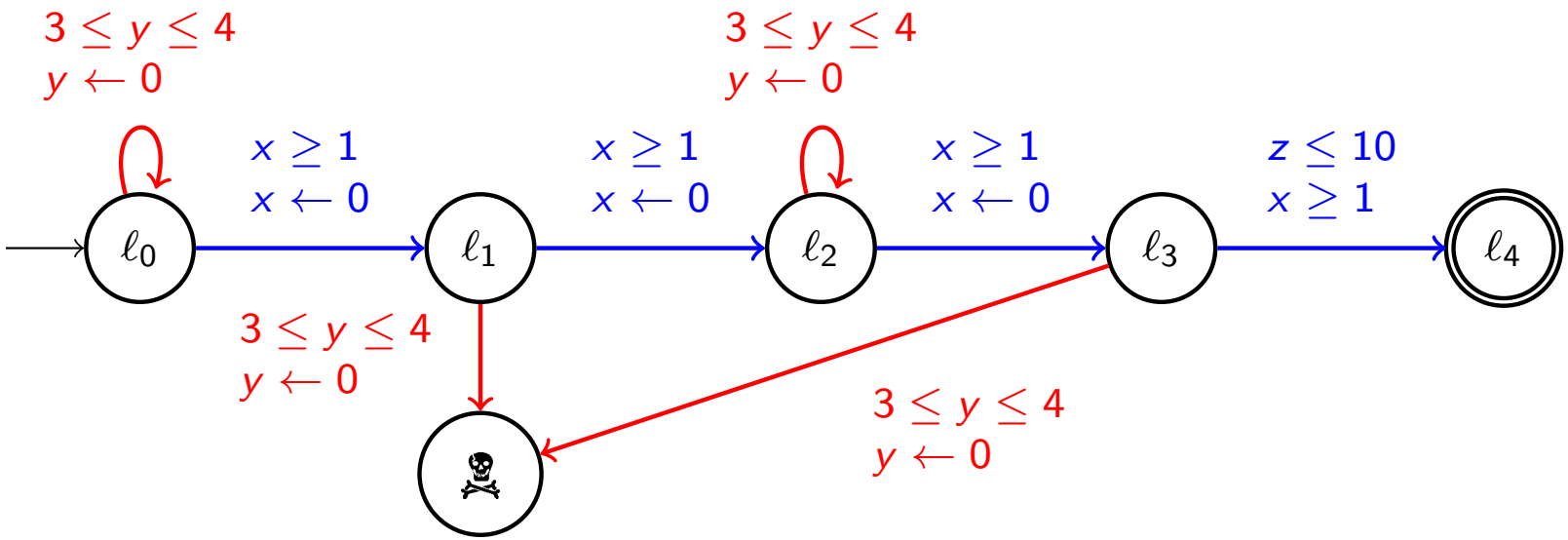
- States : (ℓ, x, y, z) ;
- Strategies assign to histories/states either an action or delay (δ) ;
- If one plays δ and the other an action, the action is played ;
- If both play δ, we wait until one of the strategy changes ;
- If both play an action, one is chosen non-deterministically (as before) ;

Strategies in timed games



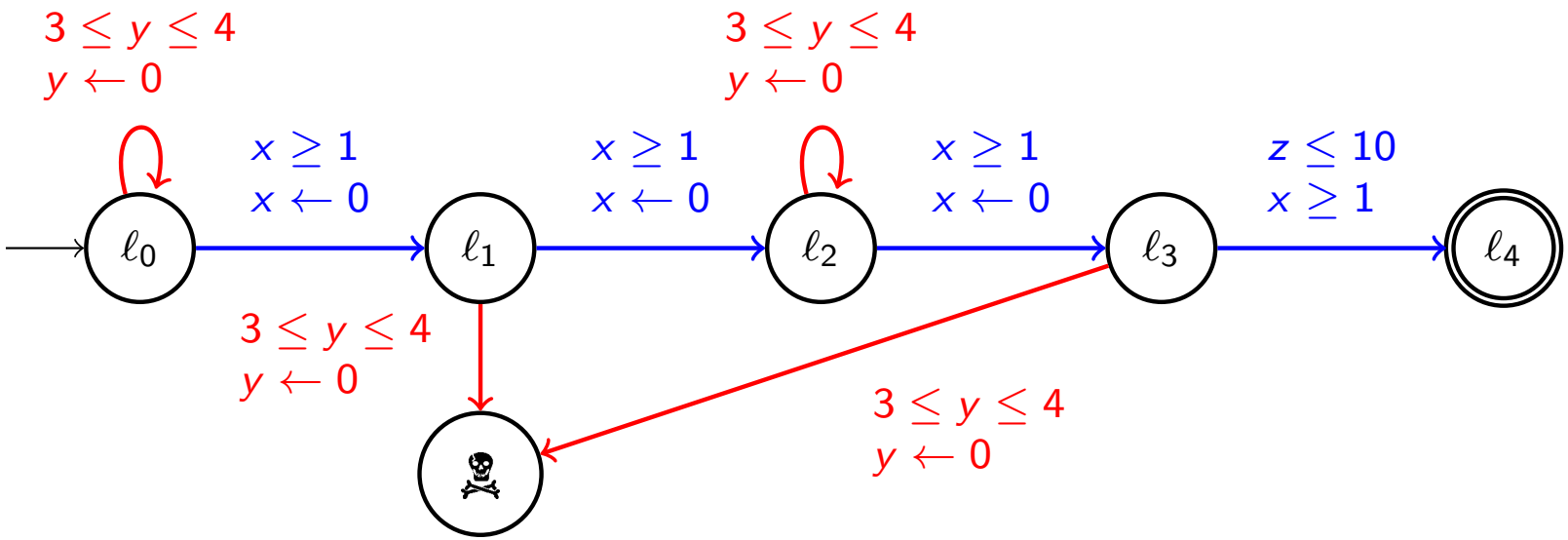
- States : (ℓ, x, y, z) ;
- Strategies assign to histories/states either an action or delay (δ) ;
- If one plays δ and the other an action, the action is played ;
- If both play δ, we wait until one of the strategy changes ;
- If both play an action, one is chosen non-deterministically (as before) ;
- We assume strategies are constant on regions.

Plays in timed games



- We write :
  $(\ell_0, 0, 0, 0) \xrightarrow{@3.2} (\ell_1, 0, 3.2, 3.2)$ for $(\ell_0, 0, 0, 0) \xrightarrow{3.2} (\ell_0, 3.2, 3.2, 3.2) \rightarrow (\ell_1, 0, 3.2, 3.2)$
- Example plays :

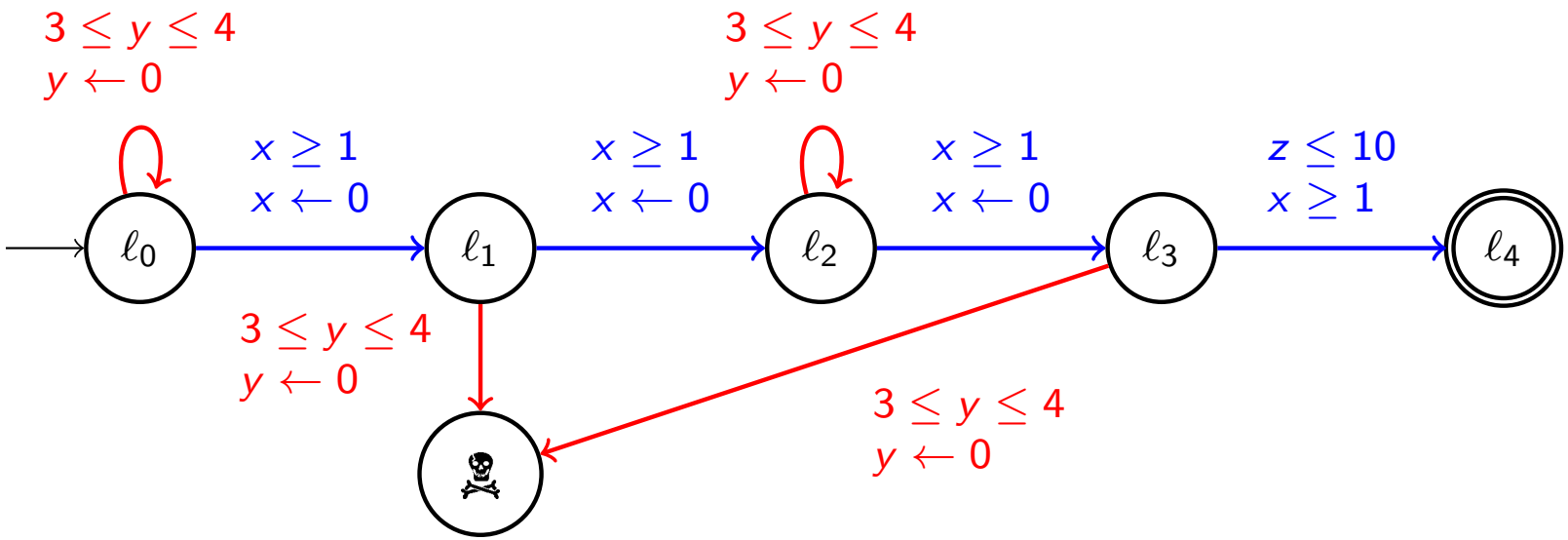
Plays in timed games



- We write :
  $(\ell_0, 0, 0, 0) \xrightarrow{@3.2} (\ell_1, 0, 3.2, 3.2)$ for $(\ell_0, 0, 0, 0) \xrightarrow{3.2} (\ell_0, 3.2, 3.2, 3.2) \rightarrow (\ell_1, 0, 3.2, 3.2)$
- Example plays :

$$(\ell_0, 0, 0, 0) \xrightarrow{@3.2} (\ell_1, 0, 3.2, 3.2) \xrightarrow{@1} (\ell_2, 0, 4.2, 4.2) \xrightarrow{@1.3} (\ell_3, 0, 5.5, 5.5) \xrightarrow{@1.4} (\ell_4, 1.4, 6.9, 6.9)$$

Plays in timed games

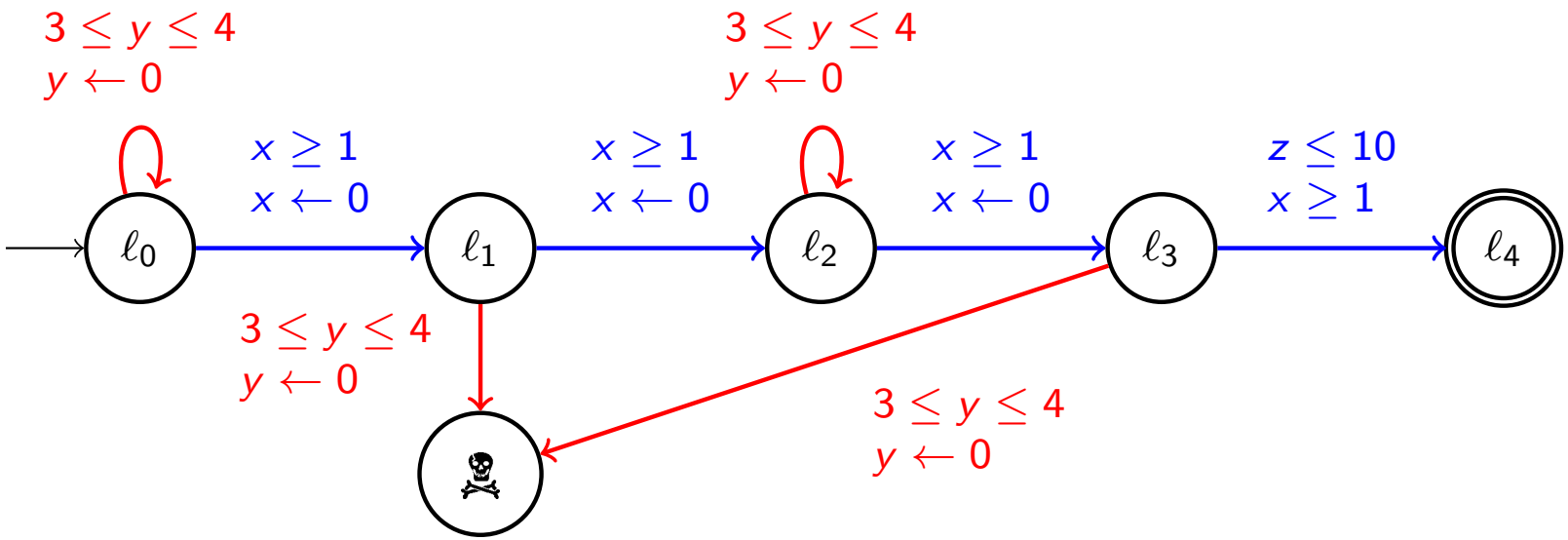


- We write :
  $(\ell_0, 0, 0, 0) \xrightarrow{@3.2} (\ell_1, 0, 3.2, 3.2)$ for $(\ell_0, 0, 0, 0) \xrightarrow{3.2} (\ell_0, 3.2, 3.2, 3.2) \rightarrow (\ell_1, 0, 3.2, 3.2)$
- Example plays :

$(\ell_0, 0, 0, 0) \xrightarrow{@3.2} (\ell_1, 0, 3.2, 3.2) \xrightarrow{@1} (\ell_2, 0, 4.2, 4.2) \xrightarrow{@1.3} (\ell_3, 0, 5.5, 5.5) \xrightarrow{@1.4} (\ell_4, 1.4, 6.9, 6.9)$

$(\ell_0, 0, 0, 0) \xrightarrow{@3.2} (\ell_1, 0, 3.2, 3.2) \xrightarrow{@0.5} (☠, 0.5, 0, 3.7) \xrightarrow{10.7} (☠, 11.2, 10.7, 14.4)$

Plays in timed games



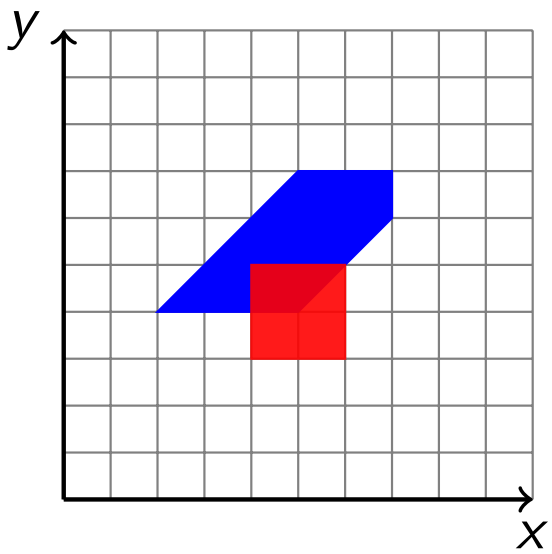
- We write :
  $(\ell_0, 0, 0, 0) \xrightarrow{@3.2} (\ell_1, 0, 3.2, 3.2)$ for $(\ell_0, 0, 0, 0) \xrightarrow{3.2} (\ell_0, 3.2, 3.2, 3.2) \rightarrow (\ell_1, 0, 3.2, 3.2)$
- Example plays :

$(\ell_0, 0, 0, 0) \xrightarrow{@3.2} (\ell_1, 0, 3.2, 3.2) \xrightarrow{@1} (\ell_2, 0, 4.2, 4.2) \xrightarrow{@1.3} (\ell_3, 0, 5.5, 5.5) \xrightarrow{@1.4} (\ell_4, 1.4, 6.9, 6.9)$

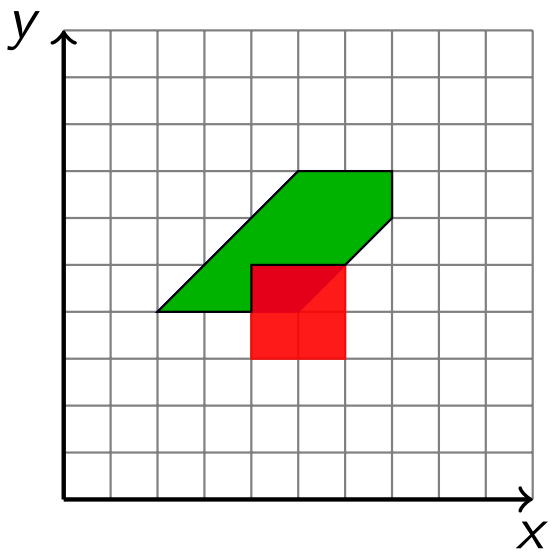
$(\ell_0, 0, 0, 0) \xrightarrow{@3.2} (\ell_1, 0, 3.2, 3.2) \xrightarrow{@0.5} (☠, 0.5, 0, 3.7) \xrightarrow{10.7} (☠, 11.2, 10.7, 14.4)$

$(\ell_0, 0, 0, 0) \xrightarrow{@1.9} (\ell_1, 0, 1.9, 1.9) \xrightarrow{@1} (\ell_2, 0, 2.9, 2.9) \xrightarrow{@0.1} (\ell_2, 0.1, 0, 3) \xrightarrow{@1} (\ell_3, 0, 1, 4) \xrightarrow{@1} (\ell_4, 1, 2, 5)$

Computing winning states

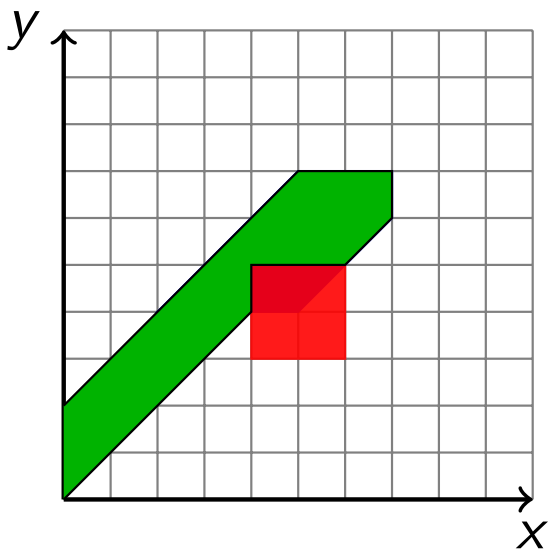


Computing winning states



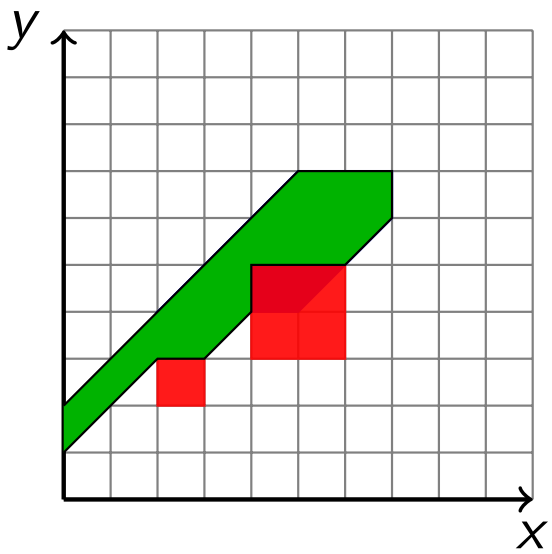
- As before, if W are winning states, states in $\mathrm{Pre}_c(W) \setminus \mathrm{Pre}_u(\overline{W})$ are winning too.

Computing winning states



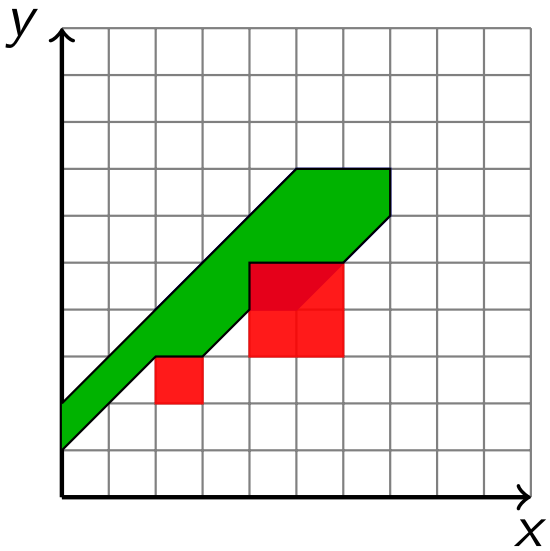
- As before, if W are winning states, states in $\mathrm{Pre}_c(W) \setminus \mathrm{Pre}_u(\overline{W})$ are winning too.
- But also states that can reach those by delay ;

Computing winning states



- As before, if W are winning states, states in $\mathrm{Pre}_c(W) \setminus \mathrm{Pre}_u(\overline{W})$ are winning too.
- But also states that can reach those by delay ;
- Provided they avoid losing/uncertain states on the way.

Computing winning states



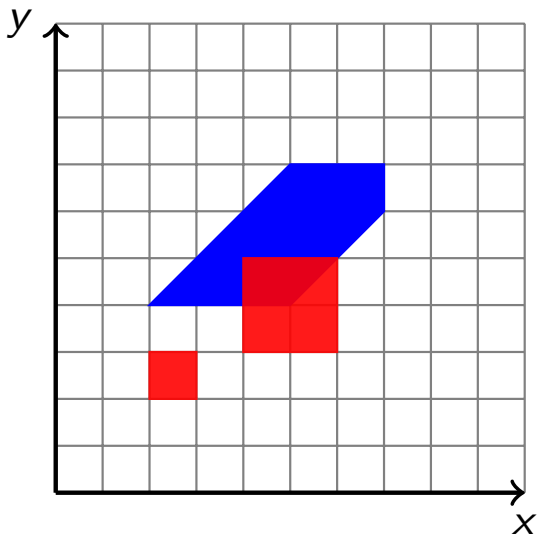
$$X_0 = G$$
$$X_{n+1} = X_n \cup \pi(X_n)$$

With :

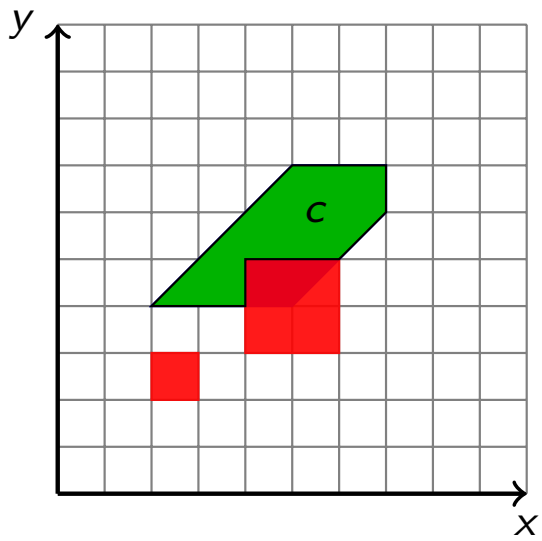
$$\pi(X) = \mathrm{Pre_t}(\mathrm{Pre}_c(X), \mathrm{Pre}_u(\overline{X}))$$

$$\mathrm{Pre_t}(X, Y) = \{s \in S | \exists d \geq 0, s' \in X \text{ s. t. } s \xrightarrow{d} s' \text{ and } \forall 0 \leq d' \leq d, s \xrightarrow{d'} \notin Y\}$$

Computing a winning strategy

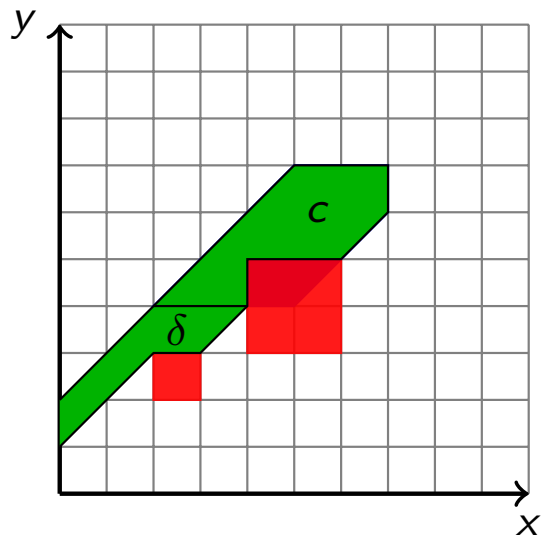


Computing a winning strategy



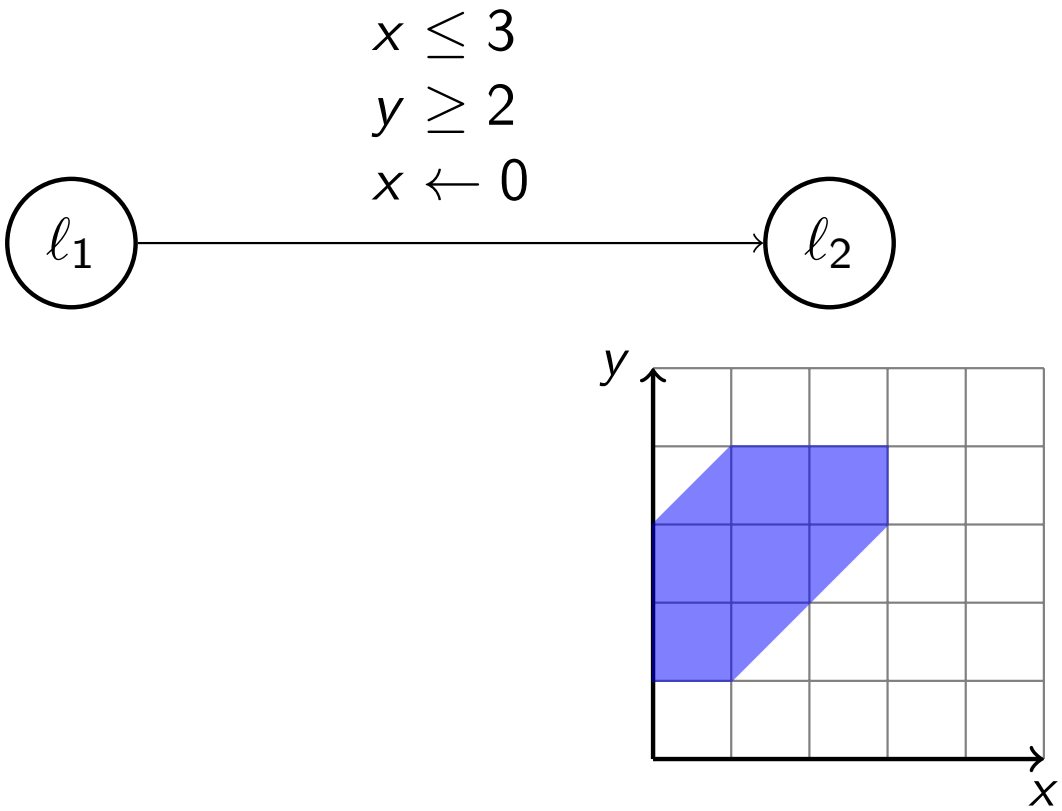
- For all states in $\mathrm{Pre}_c(W) \setminus \mathrm{Pre}_u(\overline{W})$, the strategy is the action leading to W.

Computing a winning strategy

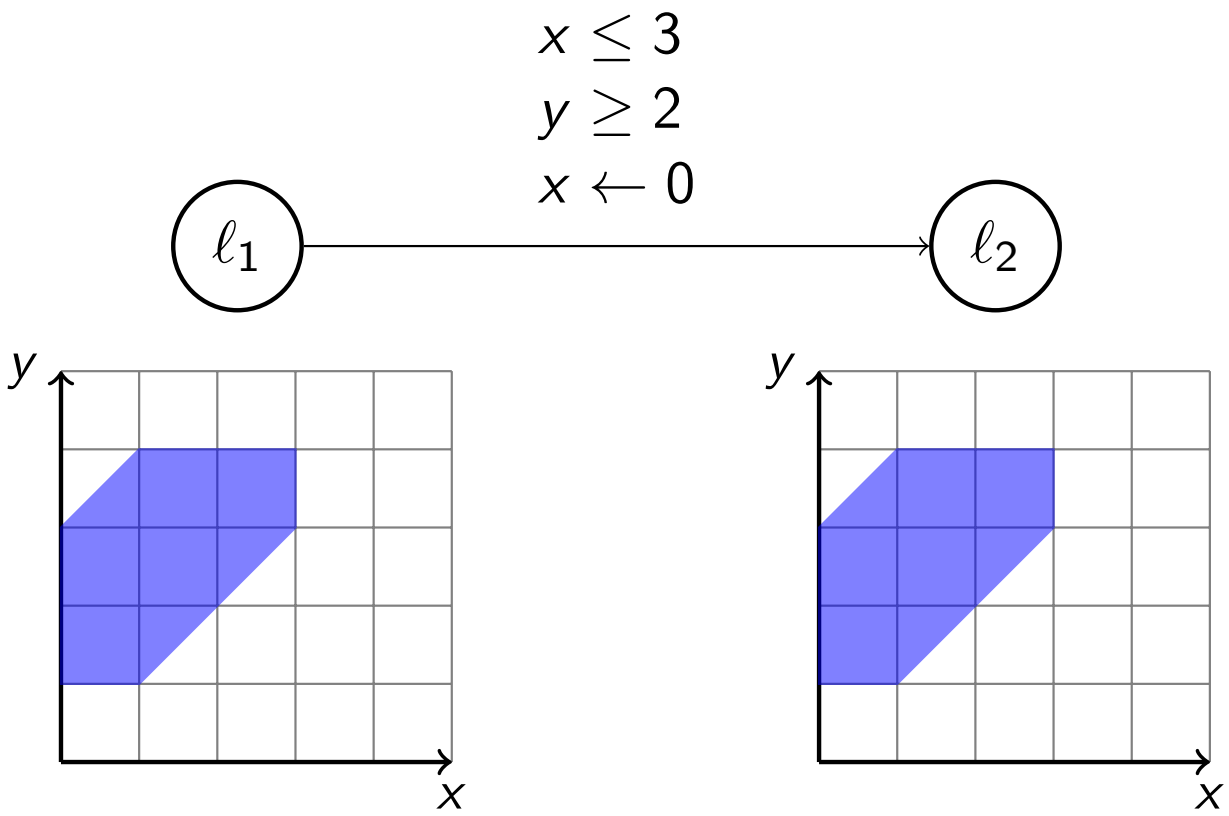


- For all states in $\mathrm{Pre}_c(W) \setminus \mathrm{Pre}_u(\overline{W})$, the strategy is the action leading to W.
- In all other winning states, the strategy is delay (δ).

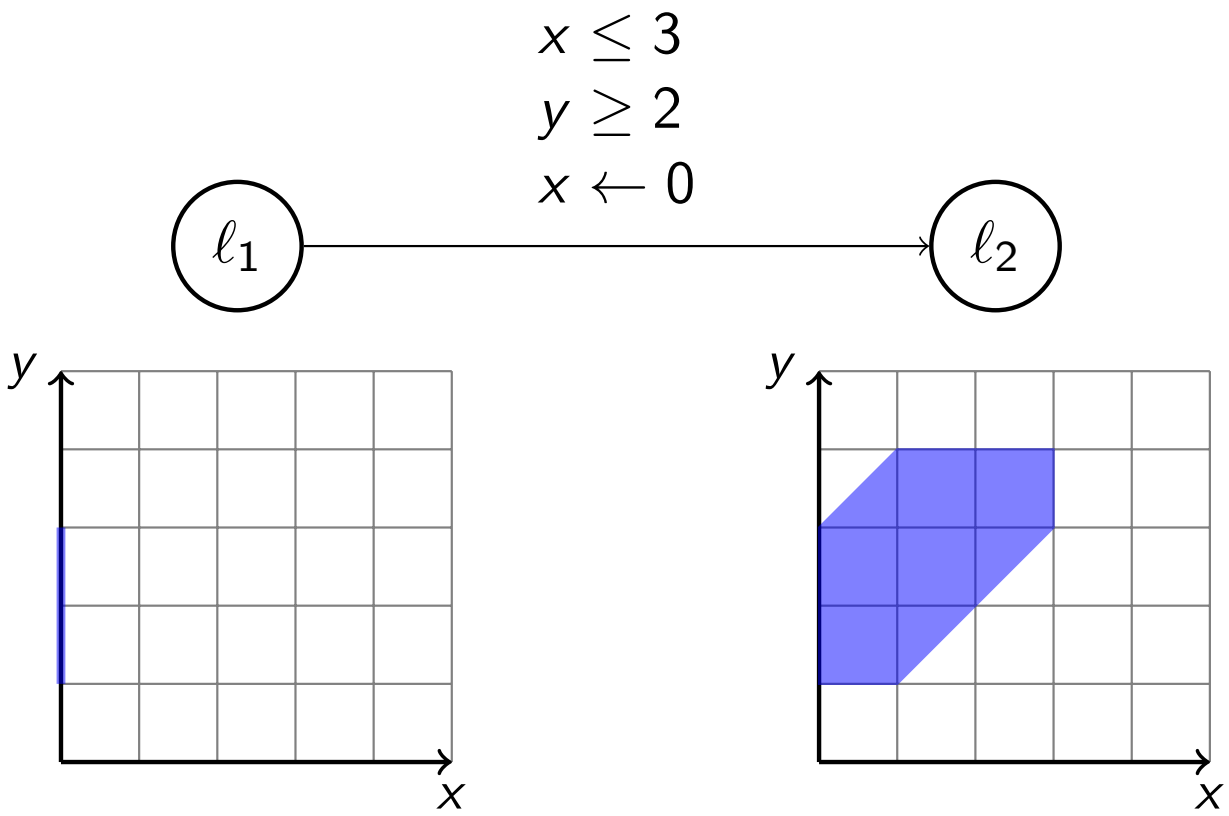
Computing action predecessors (with zones)



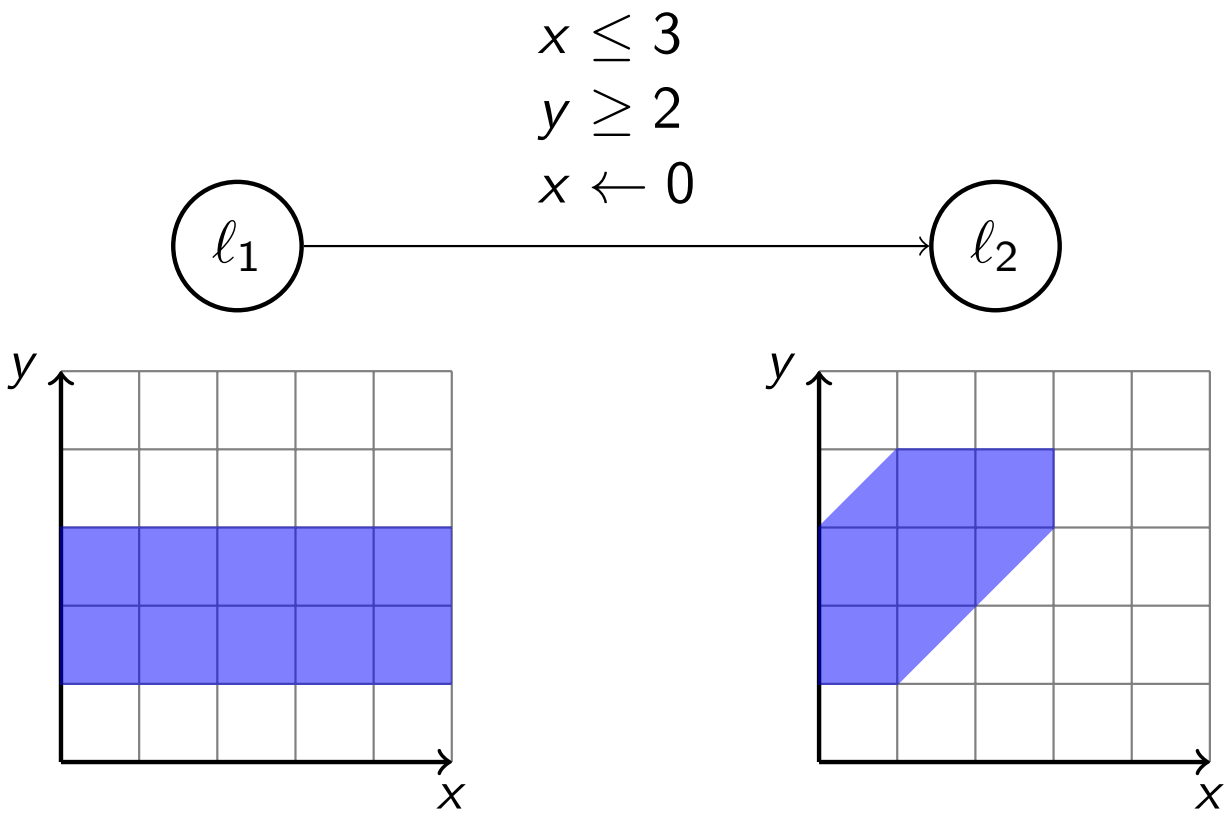
Computing action predecessors (with zones)



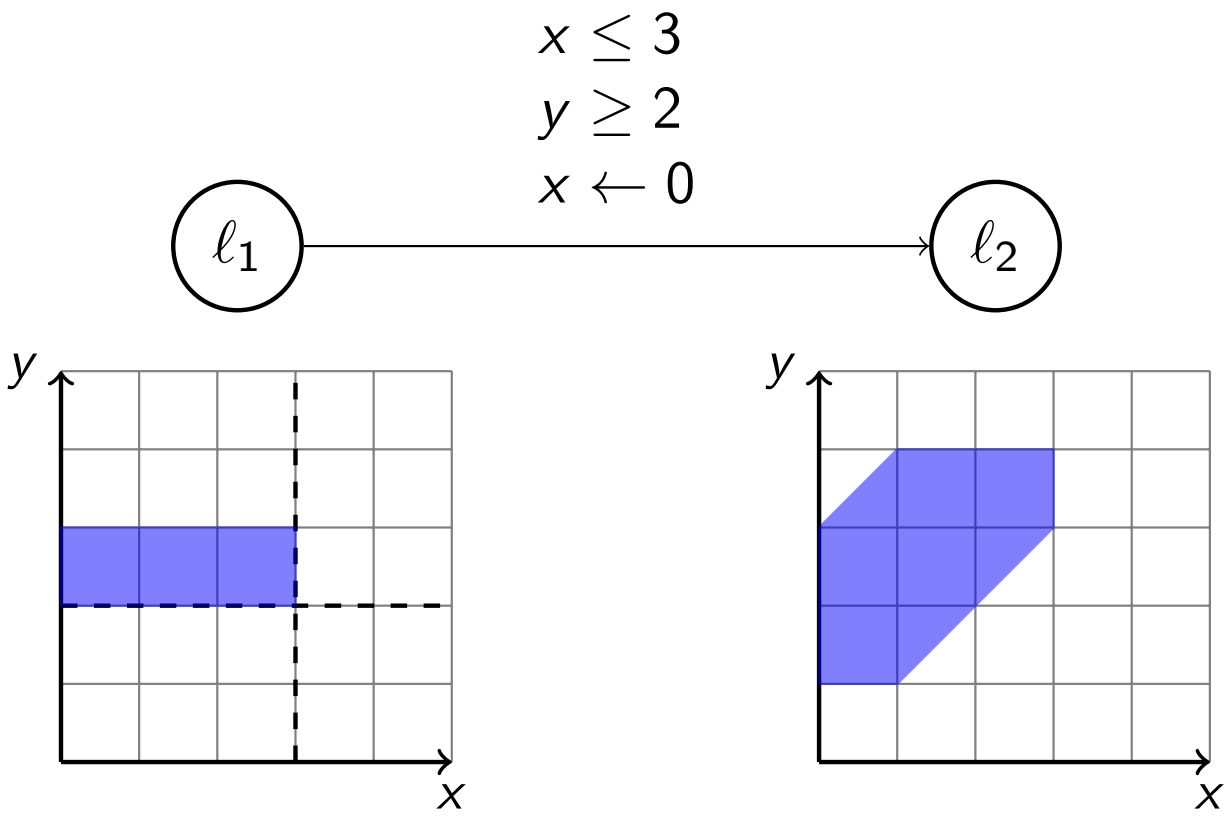
Computing action predecessors (with zones)



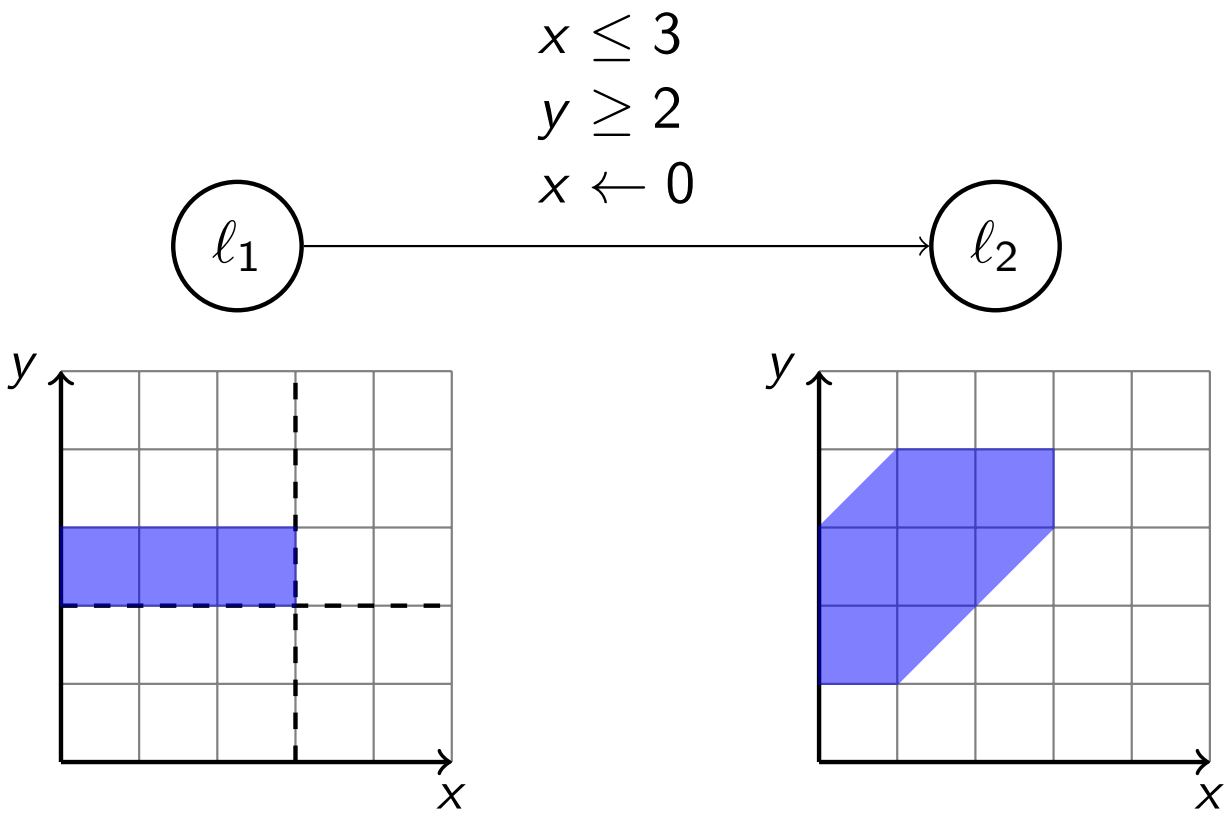
Computing action predecessors (with zones)



Computing action predecessors (with zones)

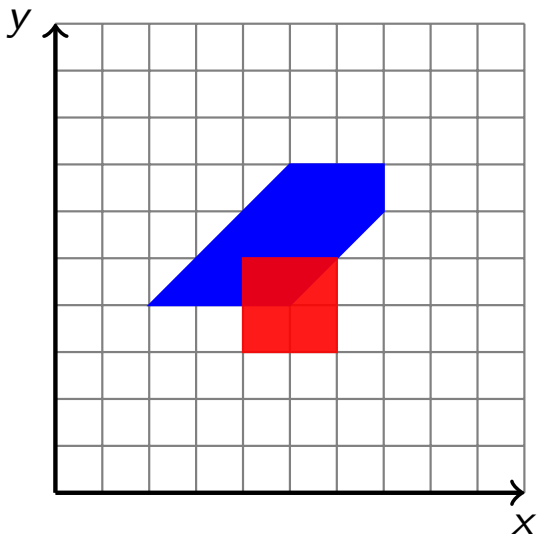


Computing action predecessors (with zones)



$$Z_1 = \text{unreset}(Z_2 \wedge (x == 0)) \wedge (x \leq 3 \wedge y \geq 2)$$

Computing time predecessors (with zones)



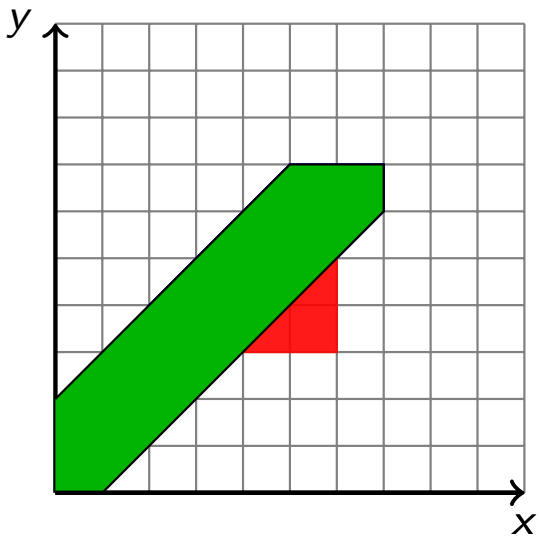
For G convex :

$$\mathrm{Pre_t}(G, B) = (G^{\swarrow} \setminus B^{\swarrow}) \cup (G \cap B^{\swarrow} \cap \overline{B})^{\swarrow}$$

And :

$$\mathrm{Pre_t}(\bigcup_i G_i, \bigcup_j B_j) = \bigcup_i \bigcap_j \mathrm{Pre_t}(G_i, B_j)$$

Computing time predecessors (with zones)



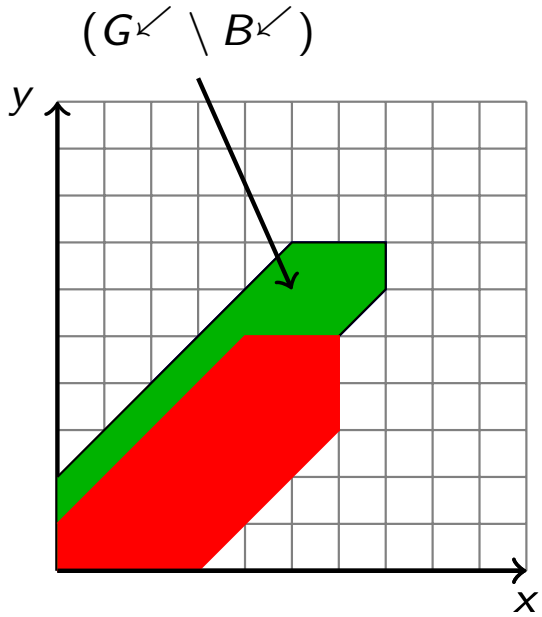
For G convex :

$$\mathrm{Pre_t}(G, B) = (G^{\swarrow} \setminus B^{\swarrow}) \cup (G \cap B^{\swarrow} \cap \overline{B})^{\swarrow}$$

And :

$$\mathrm{Pre_t}(\bigcup_i G_i, \bigcup_j B_j) = \bigcup_i \bigcap_j \mathrm{Pre_t}(G_i, B_j)$$

Computing time predecessors (with zones)



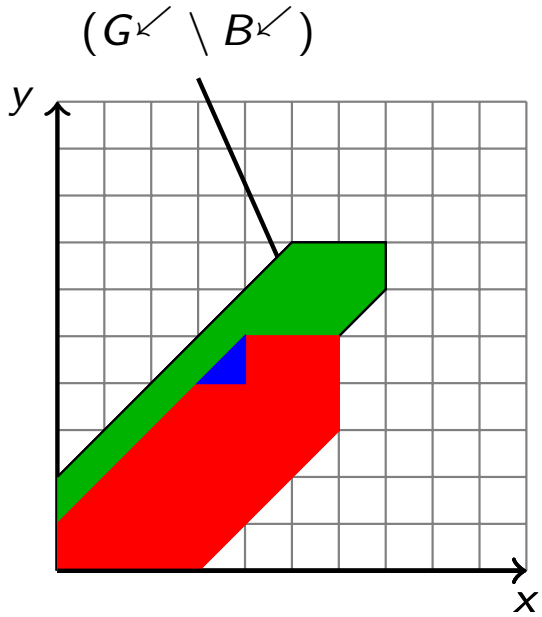
For G convex :

$$\mathrm{Pre_t}(G, B) = (G^{\swarrow} \setminus B^{\swarrow}) \cup (G \cap B^{\swarrow} \cap \overline{B})^{\swarrow}$$

And :

$$\mathrm{Pre_t}(\bigcup_i G_i, \bigcup_j B_j) = \bigcup_i \bigcap_j \mathrm{Pre_t}(G_i, B_j)$$

Computing time predecessors (with zones)



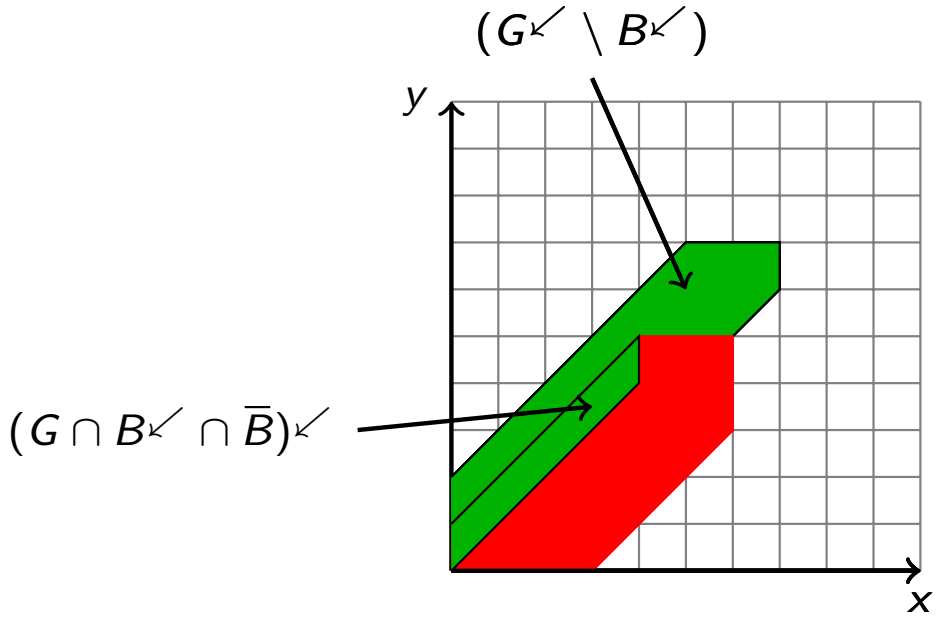
For G convex :

$$\mathrm{Pre_t}(G, B) = (G^{\swarrow} \setminus B^{\swarrow}) \cup (G \cap B^{\swarrow} \cap \overline{B})^{\swarrow}$$

And :

$$\mathrm{Pre_t}(\bigcup_i G_i, \bigcup_j B_j) = \bigcup_i \bigcap_j \mathrm{Pre_t}(G_i, B_j)$$

Computing time predecessors (with zones)



For G convex :

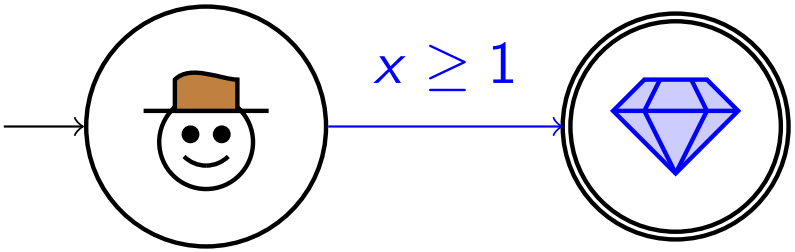
$$\mathrm{Pre_t}(G, B) = (G^{\swarrow} \setminus B^{\swarrow}) \cup (G \cap B^{\swarrow} \cap \overline{B})^{\swarrow}$$

And :

$$\mathrm{Pre_t}(\bigcup_i G_i, \bigcup_j B_j) = \bigcup_i \bigcap_j \mathrm{Pre_t}(G_i, B_j)$$

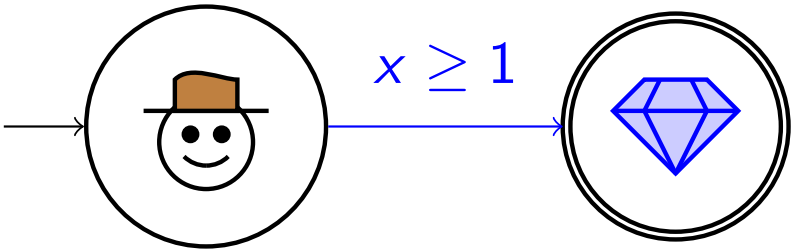
Urgency and invariants

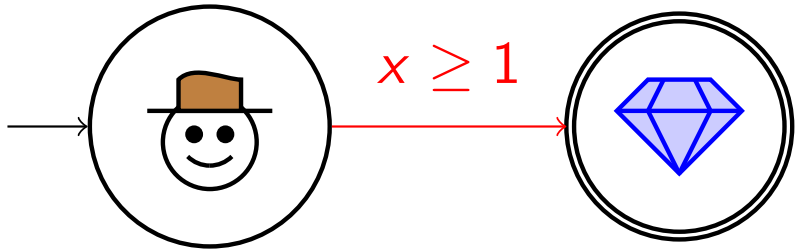
Who wins ?



Urgency and invariants

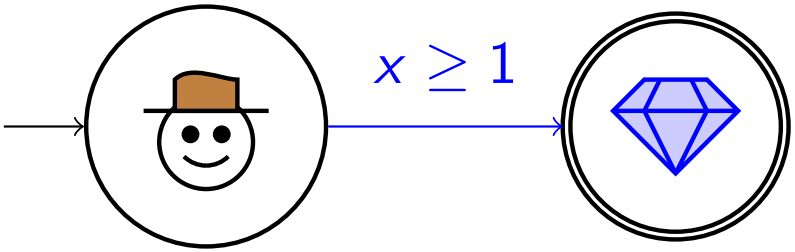
Who wins ?

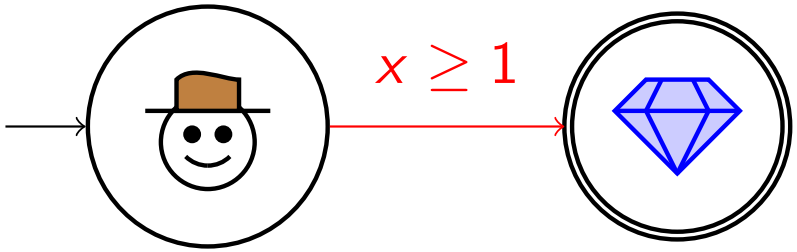


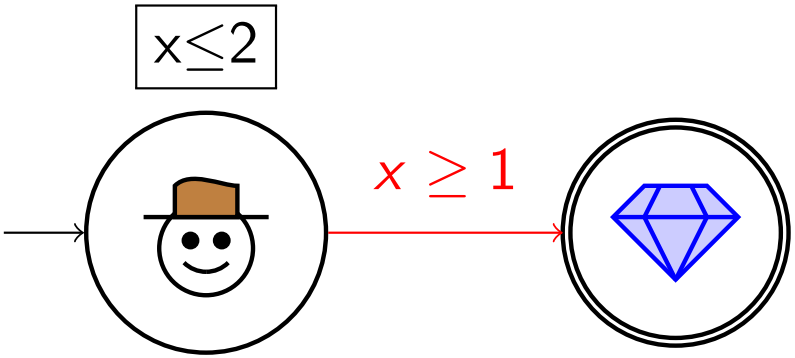


Urgency and invariants

Who wins ?

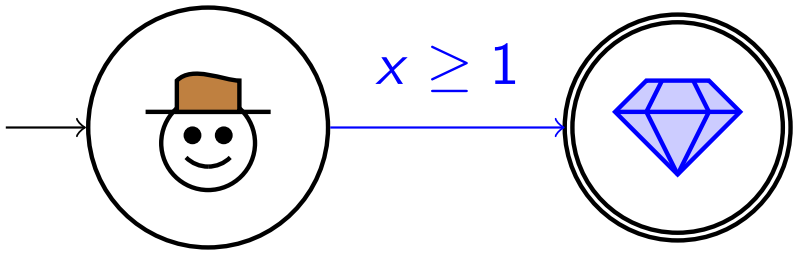


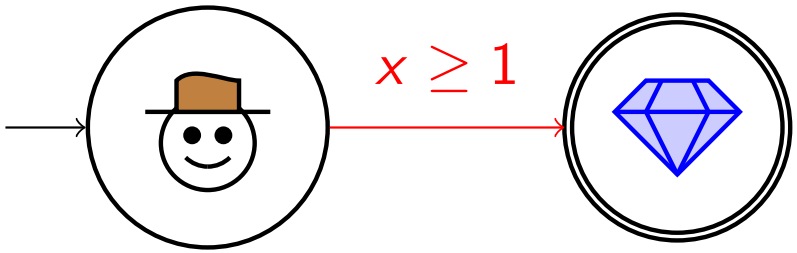


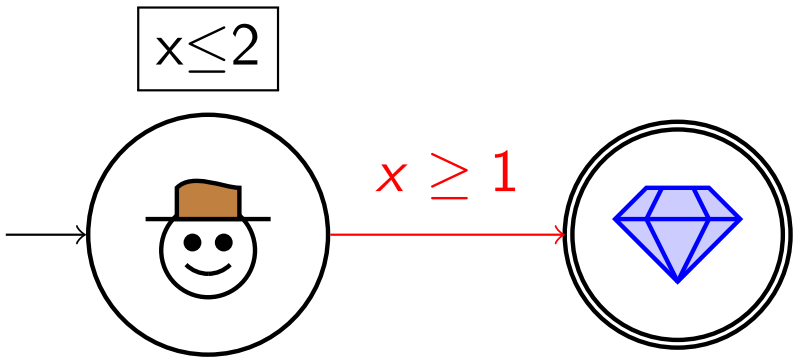


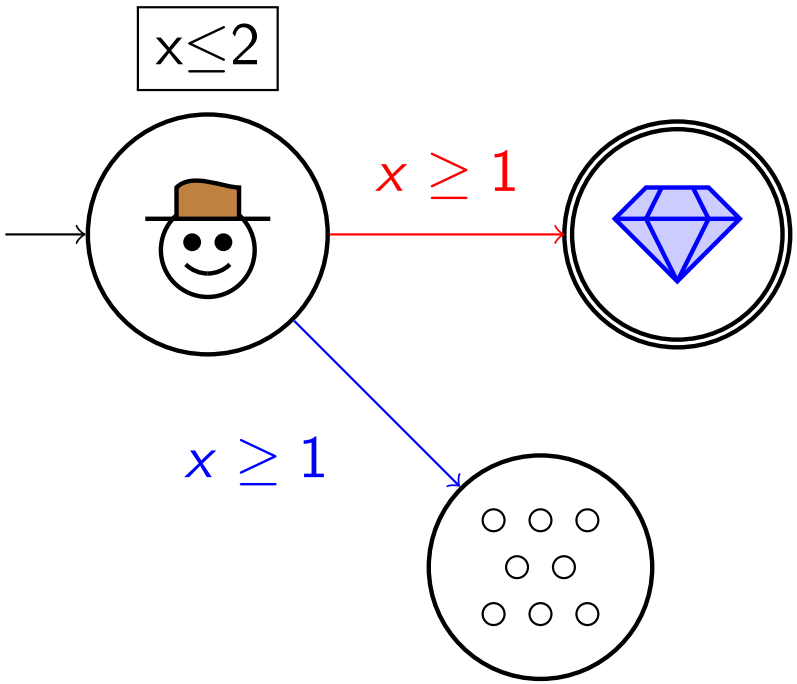
Urgency and invariants

Who wins ?

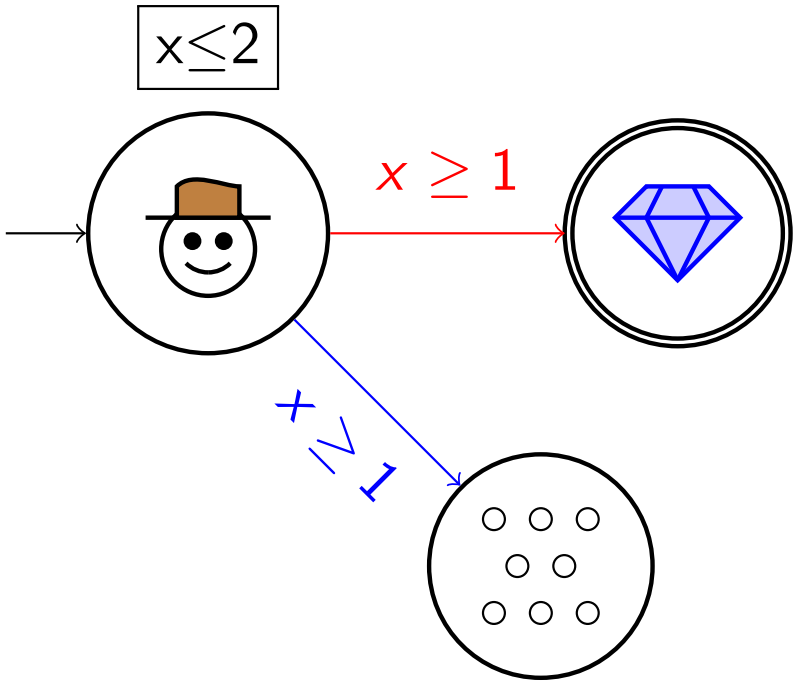




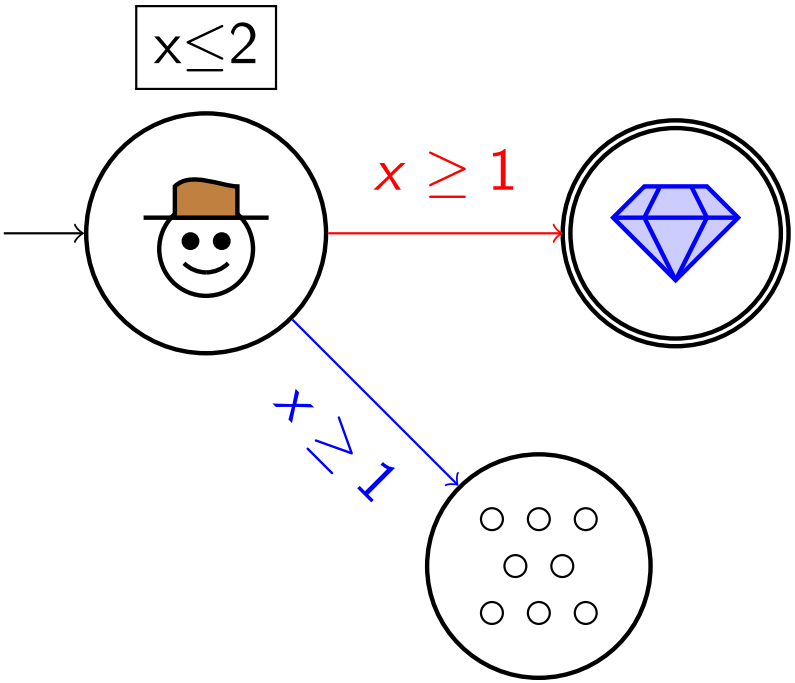


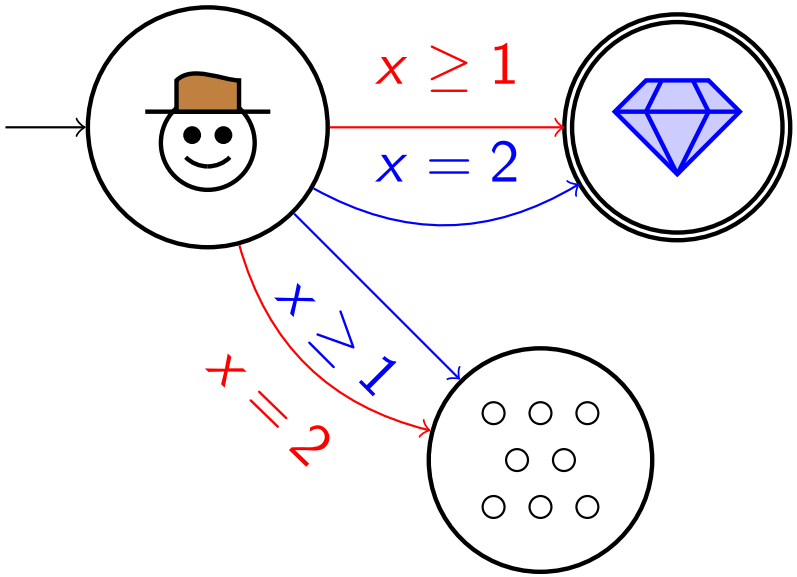


Urgency and invariants

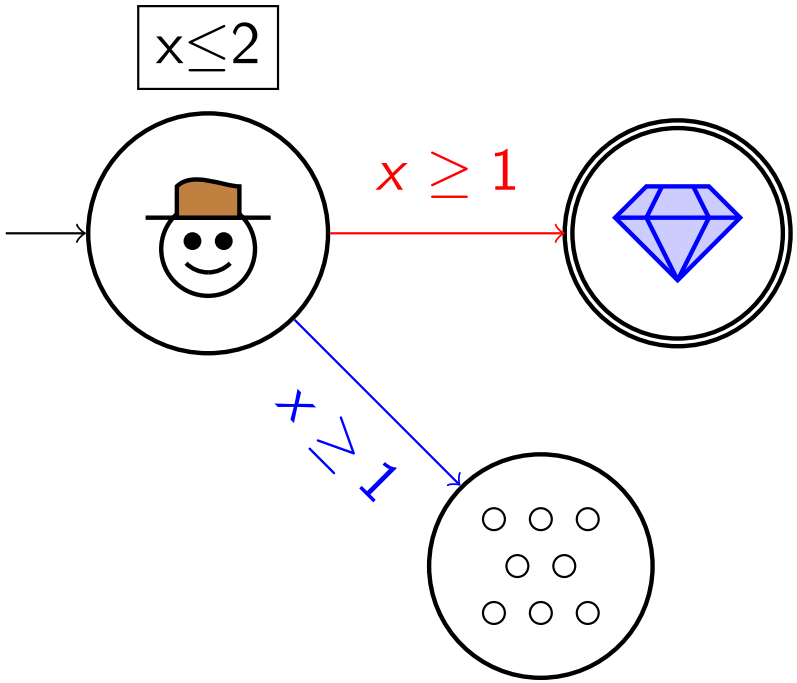


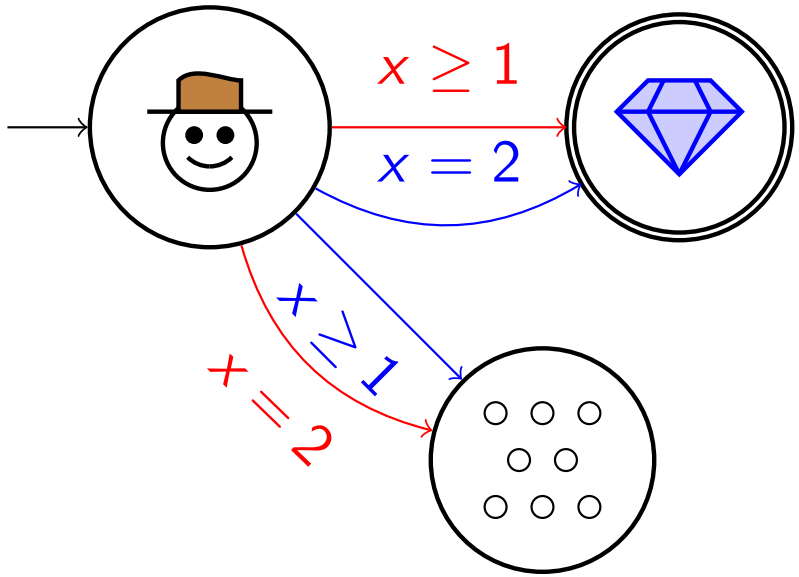
Urgency and invariants





Urgency and invariants





Actually, we compute :

$$\mathrm{Pre_t}\Big(\mathrm{Pre}_c(X) \cup \mathrm{ForcedPre}_u(X), \mathrm{Pre}_u(\overline{X}) \cup (\mathrm{ForcedPre}_c(\overline{X}) \setminus \mathrm{Pre}_c(X))\Big)$$

Safety timed game

- As before, we can solve safety timed games in two different ways
  There are actually subtle differences between the two wrt. when waiting is winning

Safety timed game

- As before, we can solve safety timed games in two different ways
  There are actually subtle differences between the two wrt. when waiting is winning
- Either, we can compute a greatest fixpoint as before :

$$X_0 = G$$
$$X_{n+1} = X_n \cap \pi(X_n)$$

With :

$$\pi(X) = \mathrm{Pre}_t(\mathrm{Pre}_c(X), \mathrm{Pre}_u(\overline{X}))$$

Safety timed game

- Or, we can compute the losing states :

$$Y_0 = \text{Bad}$$
$$Y_{n+1} = Y_n \cup \pi_u(Y_n)$$

With :

$$\pi_u(Y) = \text{Pre}_t^u(\text{Pre}_{u(Y)}, \text{Pre}_c(\overline{Y}))$$

and

$$\text{Pre}_t^u(X, Y) = \{s \in S | \exists d \geq 0, s' \in X \text{ s. t. } s \xrightarrow{d} s' \text{ and } \forall 0 \leq d' < d, s \xrightarrow{d'} \notin Y\}$$

Safety timed game

- Or, we can compute the losing states :

$$Y_0 = \text{Bad}$$
$$Y_{n+1} = Y_n \cup \pi_u(Y_n)$$

With :

$$\pi_u(Y) = \text{Pre}^{\text{u}}_{\text{t}}(\text{Pre}_{u(Y)}, \text{Pre}_c(\overline{Y}))$$

and

$$\text{Pre}^{\text{u}}_{\text{t}}(X, Y) = \{s \in S | \exists d \geq 0, s' \in X \text{ s. t. } s \xrightarrow{d} s' \text{ and } \forall 0 \leq d' < d, s \xrightarrow{d'} \notin Y\}$$

- Strategy : compute the winning states $W = \overline{Y}$ and,
  - in $\text{Pre}_c(W)$ do the corresponding action ;
  - in the other winning states, wait.

Conclusion

A word about complexity

- Existence of a winning strategy in reachability/safety games :

	one-player	two-player
discrete	NL-complete	P-complete
timed	PSPACE-complete	EXPTIME-complete

Conclusion

- Summary
  - A very expressive framework ;
  - “Easy” to extend, e. g., with time ;
  - Can express different control objectives (e.g, Safety and reachability)
  - Tools for timed systems : Uppaal ($\geq$ 5.0) and Roméo.
- To go further :
  - More complex objectives : parity, Büchi, co-Büchi, etc.
  - Partial observation ;
  - More than two players ;
  - Costs and rewards, cost bounds, optimality, Nash equilibria.

Références bibliographiques

Références bibliographiques I

Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K. G., and Lime, D. (2007).
Uppaal-tiga : Time for playing games !
In Damm, W. and Hermanns, H., editors, *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125. Springer.

Brihaye, T., Goeminne, A., Main, J. C. A., and Randour, M. (2023).
Reachability games and friends : A journey through the lens of memory and complexity (invited talk).
In Bouyer, P. and Srinivasan, S., editors, *43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2023, December 18-20, 2023, IIIT Hyderabad, Telangana, India*, volume 284 of *LIPIcs*, pages 1:1–1:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Références bibliographiques II

Cassez, F., David, A., Fleury, E., Larsen, K. G., and Lime, D. (2005).
Efficient on-the-fly algorithms for the analysis of timed games.
In Abadi, M. and de Alfaro, L., editors, *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer.

Jurdzinski, M. and Trivedi, A. (2007).
Reachability-time games on timed automata.
In Arge, L., Cachin, C., Jurdzinski, T., and Tarlecki, A., editors, *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 838–849. Springer.

Références bibliographiques III

Lime, D., Roux, O. H., Seidner, C., and Traonouez, L. (2009).
Romeo : A parametric model-checker for petri nets with stopwatches.
In Kowalewski, S. and Philippou, A., editors, *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57. Springer.

Maler, O., Pnueli, A., and Sifakis, J. (1995).
On the synthesis of discrete controllers for timed systems (an extended abstract).
In Mayr, E. W. and Puech, C., editors, *STACS 95, 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, March 2-4, 1995, Proceedings*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer.

Remerciements et crédits

Équipe pédagogique

Auteur.rice.s : Didier Lime.

Intervenant.e.s : Didier Lime.

Crédits