Course ITI8531: Software Synthesis and Verification

Lecture 13: Acacia+ LTL Synthesis - II

Spring 2019

Leonidas Tsiopoulos

leonidas.tsiopoulos@taltech.ee

Avoiding the Classical Approach to LTL Synthesis

- LTL synthesis is a challenging problem due to 2EXPTIME theoretical complexity and lack of scalable algorithms for determinization of automata and solving games.
- There are some LTL-based synthesis approaches offering „Safraless“ solutions to avoid the very complex determinisation step and also better algorithms working on „symbolic“ representation of the state space during the game.
  - Even translating LTL formulae to symbolic automaton in the first place.
    - More for this and other „Safraless“ approaches in the 4th lecture.
- Acacia+ and the techniques around it is one such „Safraless“ approach.

Acacia+: A tool for LTL synthesis

- Main contributions:
  - Efficient *symbolic* incremental algorithms based on *antichains* for game solving.
  - Synthesis of *small* winning strategies, when they exist.
  - *Compositional* approach for *large conjunctions* of LTL formulas.
  - Performance is better or similar to other existing tools but its *main advantage* is the generation of *compact strategies.*
- Application scenarios:
  - **Synthesis of control code from high-level LTL specifications**.
  - *Debugging* of unrealizable specifications by inspecting compact counter strategies.
  - *Generation of small deterministic automata* from LTL formulas, when they exist.

Acacia+ Safraless approach

LTL

↓ $2^{O(n)}$

Universal coBüchi Word automata

↓ $O(1)$

Universal **KcoBüchi** Word automata

↓ $2^{O(m^2)}$

Det. KcoBüchi Word automata

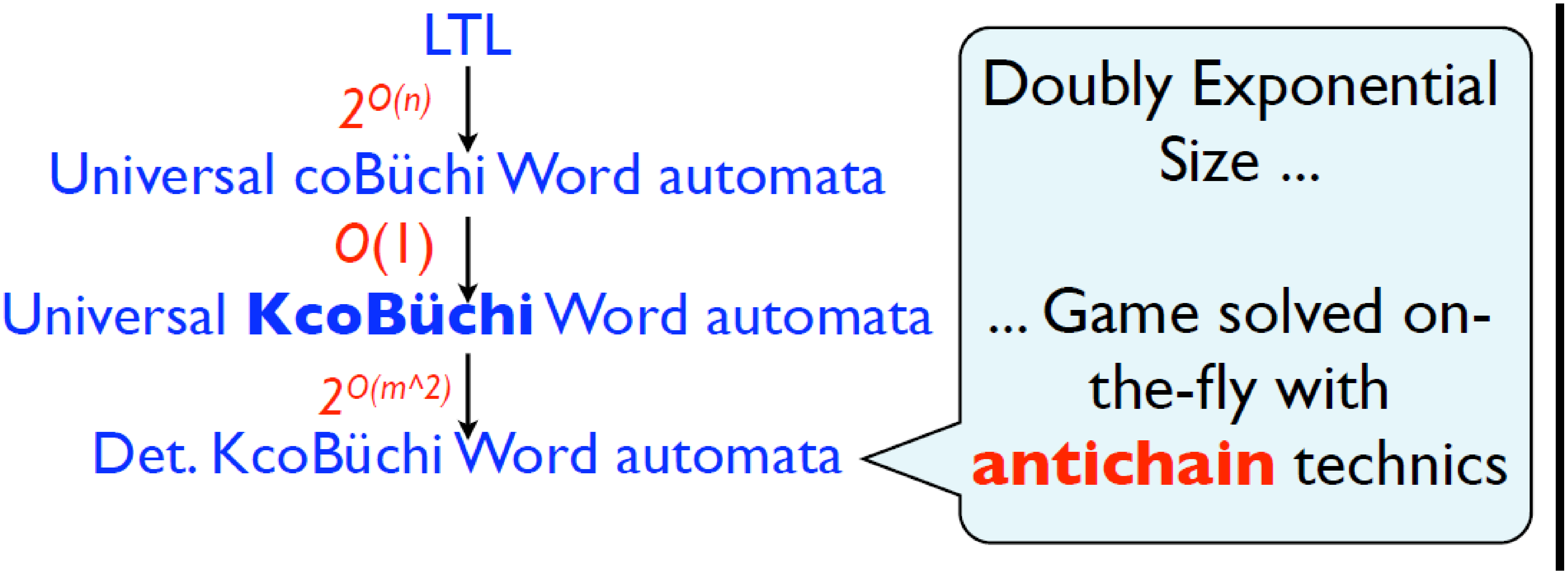
Realizability

~

Safety game

- Safety games are the simplest games to solve!
- Details and comparison to other games of other LTL-based synthesis approaches in Lectures III and IV

Acacia+ Safraless approach



- Safety games are the simplest games to solve!
- Details and comparison to other games of other LTL-based synthesis approaches in Lectures III and IV

Acacia+ and LTL Transformation to Automata (1)

- An *infinite word automaton* is a tuple $A = (\Sigma, Q, q_0, \alpha, \delta)$ where:
  - Σ is the *finite alphabet*,
  - Q is a *finite set of states*,
  - $q_0 \in Q$ is the *initial* state,
  - $\alpha \subseteq Q$ is a set of *final states* and
  - $\delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*.
    - For all $q \in Q$ and all $\sigma \in \Sigma$, $\delta(q, \sigma) = \{q' \mid (q, \sigma, q') \in \delta\}$.

- A is *deterministic* if $\forall q \in Q \cdot \forall \sigma \in \Sigma \cdot |\delta(q, \sigma)| \leq 1$.
- A is *complete* if $\forall q \in Q \cdot \forall \sigma \in \Sigma \cdot \delta(q, \sigma) = \emptyset$.

Acacia+ and LTL Transformation to Automata (2)

- A *run* of A on a *word* $w = \sigma_0\sigma_1 \cdot\cdot\cdot \in \Sigma^\omega$ is an infinite sequence of states $\rho = \rho_0\rho_1 \cdot\cdot\cdot \in Q^\omega$ such that $\rho_0 = q_0$ and $\forall i \geq 0 \cdot q_{i+1} \in \delta(q_i, \rho_i)$.
- The *set of runs* of A on w is denoted by $\text{Runs}_A(w)$.
- The number of times state q occurs along run ρ is denoted by $\text{Visit}(\rho, q)$.
- Three *acceptance conditions* (a.c.) are considered for infinite word automata. A word w is *accepted by* A if:
  - Non-deterministic Büchi : $\exists \rho \in \text{Runs}_A(w) \cdot \exists q \in \alpha \cdot \text{Visit}(\rho, q) = \infty$
    - Runs visit final states infinitely often.
  - Universal Co-Büchi : $\forall \rho \in \text{Runs}_A(w) \cdot \forall q \in \alpha \cdot \text{Visit}(\rho, q) < \infty$
    - Runs visit final states finitely often.
  - Universal K-Co-Büchi : $\forall \rho \in \text{Runs}_A(w) \cdot \forall q \in \alpha \cdot \text{Visit}(\rho, q) \leq K$
    - Runs visit at most K final states.

Acacia+ and LTL Transformation to Automata (3)

- The *set of words* accepted by A with the non-deterministic Büchi a.c. is denoted by $L_{\mathrm{b}}(A)$.
  - This implies that A is a non-deterministic Büchi word automaton (NBW).

- Similarly, the set of words accepted by A with the universal co-Büchi and universal *K*-co-Büchi a.c., are denoted respectively by $L_{\mathrm{uc}}(A)$ and $L_{\mathrm{uc},K}(A)$.
  - With those interpretations, A is a universal co-Büchi automaton (UCW) and that (A,K) is a universal *K*-co-Büchi automaton (U*K*CW) respectively.

- By duality, $L_{\mathrm{b}}(A) = \overline{L_{\mathrm{uc}}(A)}$ for any infinite word automaton A.

- Also, for any $0 \leq K_1 \leq K_2$, $L_{\mathrm{uc},K_1}(A) \subseteq L_{\mathrm{uc},K_2}(A) \subseteq L_{\mathrm{uc}}(A)$.

Infinite automata and LTL

- NBWs subsume LTL, i.e., for an LTL formula φ, there is a NBW A_φ (possibly exponentially larger) such that $L_\text{b}(A_\varphi) = \{w | w \vDash \varphi\}$.

- By duality, one can associate an equivalent UCW with any LTL formula φ:
  - Take $A_{\neg\varphi}$ with the universal co-Büchi a.c., so
    - $L_\text{uc}(A_{\neg\varphi}) = \overline{L_\text{b}(A_{\neg\varphi})} = L_\text{b}(A_\varphi) = \{w | w \vDash \varphi\}$.

Turn-based Automata for Realizability of Games (1)

- To reflect the game point of view of the realizability problem the notion of *turn-based automata* is used to define the specification.
- *A turn-based automaton* A over the *input alphabet* Σ_I and the *output alphabet* Σ_O is a tuple $A = (\Sigma_I, \Sigma_O, Q_I, Q_O, q_0, \alpha, \delta_I, \delta_O)$ where:
  - Q_I, Q_O are *finite sets of input and output states* respectively,
  - $q_0 \in Q_O$ the *initial state*,
  - $\alpha \subseteq Q_I \cup Q_O$ is the set of *final states*,
  - $\delta_I \subseteq Q_I \times \Sigma_I \times Q_O$ and $\delta_O \subseteq Q_O \times \Sigma_O \times Q_I$ are the *input* and *output transition relations.*
- A is *complete* if for all $q_I \in Q_I$, and all $\sigma_I \in \Sigma_I$, $\delta_I(q_I, \sigma_I) \neq \emptyset$, **and** for all $q_O \in Q_O$ and all $\sigma_O \in \Sigma_O$, $\delta_O(q_O, \sigma_O) \neq \emptyset$.

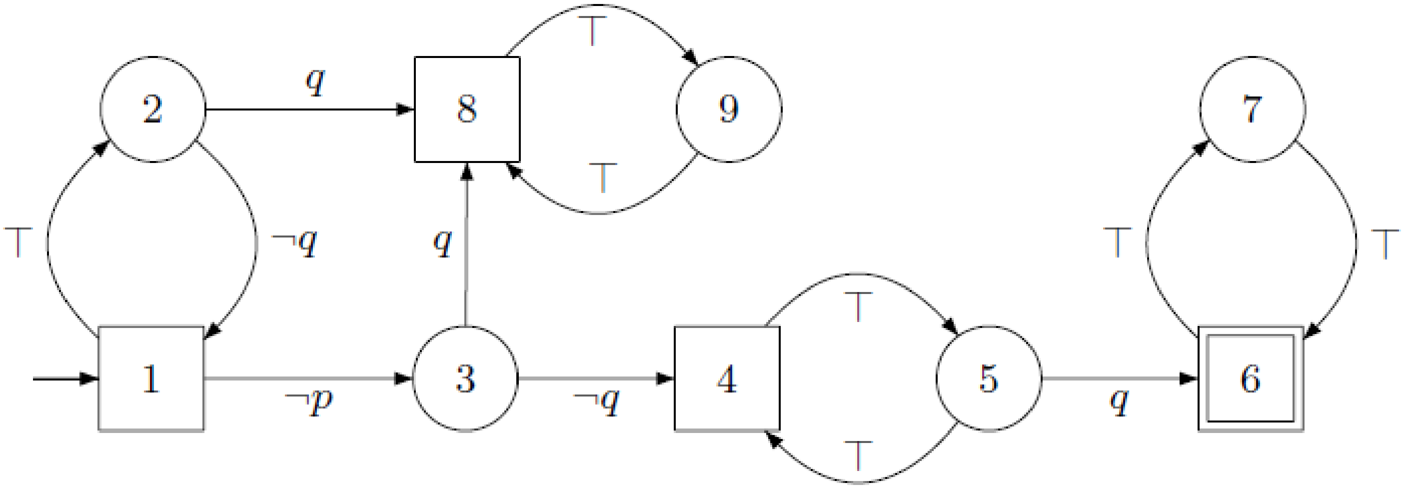
Turn-based Automata for Realizability of Games (2)

- Turn-based automata A run on words from Σ^ω.
- A *run* on a word $w = (o_0 \cup i_0)(o_1 \cup i_1) \cdot \cdot \cdot \in \Sigma^\omega$ is an infinite sequence of states $\rho = \rho_0\rho_1 \cdot \cdot \cdot \in (Q_O Q_I)^\omega$ such that $\rho_0 = q_0$ and for all $j \geq 0$,

  $(\rho_{2j}, o_j, \rho_{2j+1}) \in \delta_O$ and $(\rho_{2j+1}, i_j, \rho_{2j+2}) \in \delta_I$.
- All acceptance conditions we saw carry over to turn-based automata.
- Every UCW (resp. NBW) with state set Q and transition set Δ is equivalent to a turn-based UCW (tbUCW) (resp. tbNBW) with $|Q| + |\Delta|$ states:
  - the new set of states is $Q \cup \Delta$,
  - final states remain the same,
  - and each transition $r = q \xrightarrow{\sigma_i \cup \sigma_o} q' \in \Delta$ where $\sigma_o \in \Sigma_O$ and $\sigma_i \in \Sigma_I$ is split into a transition $q \xrightarrow{\sigma_o} r$ and a transition $r \xrightarrow{\sigma_i} q'$.

Example of tbUCW

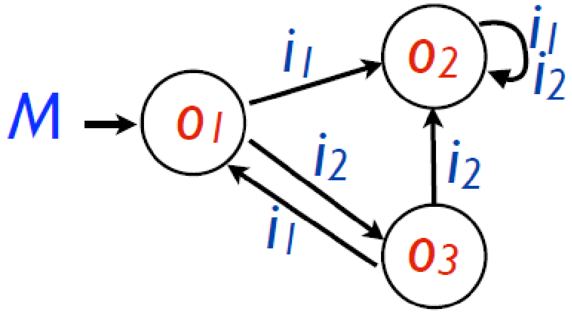
- tbUCW for **F**$q \rightarrow (p$**U**$q)$ where $I = \{q\}$ and $O = \{p\}$
- Output states $Q_O = \{1, 4, 6, 8\}$ are depicted by squares and input states $Q_I = \{2, 3, 5, 7, 9\}$ by circles
- $\top$ stands for the sets Σ_I or Σ_O, depending on the context, $\neg q$ (resp. $\neg p$) stands for the sets that do not contain q (resp. p), i.e. the empty set.
- At state 1, if controller does not assert p and next the environment does not assert q, then the run is in state 4. From this state, whatever the controller does, if the environment asserts q, then the controller loses, as state 6 will be visited infinitely often.



- A strategy for the controller is to assert p all the time, therefore the runs will loop in states 1 and 2 until the environment asserts q. Afterwards the runs will loop in states 8 and 9, which are non-final.

Finite state strategies

- We know that if an LTL formula is realizable, there exists a finite-state strategy that realizes it [PR89].
- Finite-state strategies are represented as complete Moore machines in Acacia+.



$L(M)$ = *traces of infinite paths*

E.g. $(o_1 \cup i_1)(o_2 \cup i_2)^\omega$

- The LTL realizability problem reduces to decide, given a tbUCW A over inputs Σ_I and outputs Σ_O, whether there is a non-empty Moore machine M such that $L(M) \subseteq L_{uc}(A)$.
- The tbUCW is equivalent to an LTL formula given as input and is constructed by using tools *Wring* or *LTL2BA*.

Bounding the number of *visited* final states

Lemma 1. Given a Moore machine *M* with *m* states, and a tbUCW A with *n* states, if $L(M) \subseteq L_{\text{uc}}(A)$, then all runs on words of $L(M)$ visit at most $m \times n$ final states.

Proof. The infinite paths of *M* starting from the initial state define words that are accepted by A. Therefore in the product of *M* and A, there is no cycle visiting an accepting state of A, which allows one to bound the number of visited final states by the number of states in the product.

Corollary. $L(M) \subseteq L_{\text{uc}}(A)$ iff $L(M) \subseteq L_{\text{uc},\, m\text{x}n}(A)$

Reduction to a bounded universal *K*-co-Büchi automaton

Lemma 2. Given a realizable tbUCW $\mathcal{A}$ over *inputs* Σ_I and *outputs* Σ_O with *n* states, there exists a non-empty Moore machine with at most $n^{2n+2} + 1$ states that realizes it.

Proof*. In the paper. Re-using an older result by Safra.*

Theorem. Let $\mathcal{A}$ be a tbUCW over Σ_I, Σ_O with *n* states and $K = 2n(n^{2n+2} + 1)$ (from above proof). Then $\mathcal{A}$ is realizable iff ($\mathcal{A}$,*K*) is realizable.

Determinization of U*K*CWs

- In the previous lecture we saw how an LTL formula can be transformed to a tbU*K*CW in a stepwise manner.

- What remains before solving the safety game and realize with a Moore machine the winning strategy for the system (if it exists) against the environment is to determinize the tbU*K*CW.

- The deterministic tbU*K*CWs can be viewed as safety games.

Determinization of U*K*CWs

- **Lemma**: U*K*CWs are determinizable.
- **Sketch of Proof**: Let $A = (\Sigma, Q, q_0, \alpha, \Delta, K)$ be a U*K*CW.
- *For each state q*, ***count*** *the maximal number of final states visited by runs ending up in q.*
  - Extending the usual subset construction with counters.
- Set of states $\mathbb{F}$: ***counting functions*** *F* from *Q* to [-1,0,...,*K*+1].
  - The counter of a state *q* is set to −1 when no run up to *q* visited final states.
- ***Initial*** *counting function* F_0: $q \rightarrow (q_0 \in \alpha)$ **if** $q = q_0$, -1 otherwise.
- ***Final*** *states* are *functions F* such that $\exists q$: $F(q) > K$.
  - The final states are the sets in which a state has its counter **greater than** *K.*

Determinization of tbU*K*CWs

- Let A be a tbU*K*CW ($\Sigma_O, \Sigma_I, Q_O, Q_I, q_0, \alpha, \Delta_O, \Delta_I$) with $K \in \mathbb{N}$.
  - Let $Q = Q_O \cup Q_I$ and $\Delta = \Delta_O \cup \Delta_I$.
- Let det(A,K) = ($\Sigma_O, \Sigma_I, \mathbb{F}_O, \mathbb{F}_I, F_0, \alpha', \delta_O, \delta_I$) where:
  - Set of states $\mathbb{F}_O$: ***counting functions*** F_O from Q_O to [-1,0,...,K+1].
  - Set of states $\mathbb{F}_I$: ***counting functions*** F_I from Q_I to [-1,0,...,K+1].
  - ***Initial*** *counting function* F_0: $q \in Q_O \rightarrow (q_0 \in \alpha)$ **if** $q = q_0$, -1 otherwise.
  - $\alpha' = \{F \in \mathbb{F}_O \cup \mathbb{F}_I \mid \exists q, F(q) > K\}$.
  - $\text{succ}(F, \sigma) = q \rightarrow \max\{\min(K + 1, F(p) + (q \in \alpha)) \mid q \in \Delta(p, \sigma), F(p) \neq -1\}$
    - There is a successor state if the run up to *p* visited finaal states.
  - $\delta_O = \text{succ}|_{\mathbb{F}O \times \Sigma O}$, $\delta_I = \text{succ}|_{\mathbb{F}I \times \Sigma I}$

Reduction to Safety Games

- The *game* $G(\mathcal{A},K)$ can be defined as follows:
  - it is det(A,K) where *input states* are viewed as Player *I*'s states (env.) and output states as Player *O*'s states (system).
- $G(\mathcal{A},K) = (\mathbb{F}_O, \mathbb{F}_I, F_0, \mathcal{T}, \text{safe})$ where safe $= \mathbb{F} \setminus \alpha'$ and $\mathcal{T} = \{(F, F') \mid$

  $\exists \sigma \in \Sigma_O \cup \Sigma_I, F' = \text{succ}(F, \sigma)\}$.

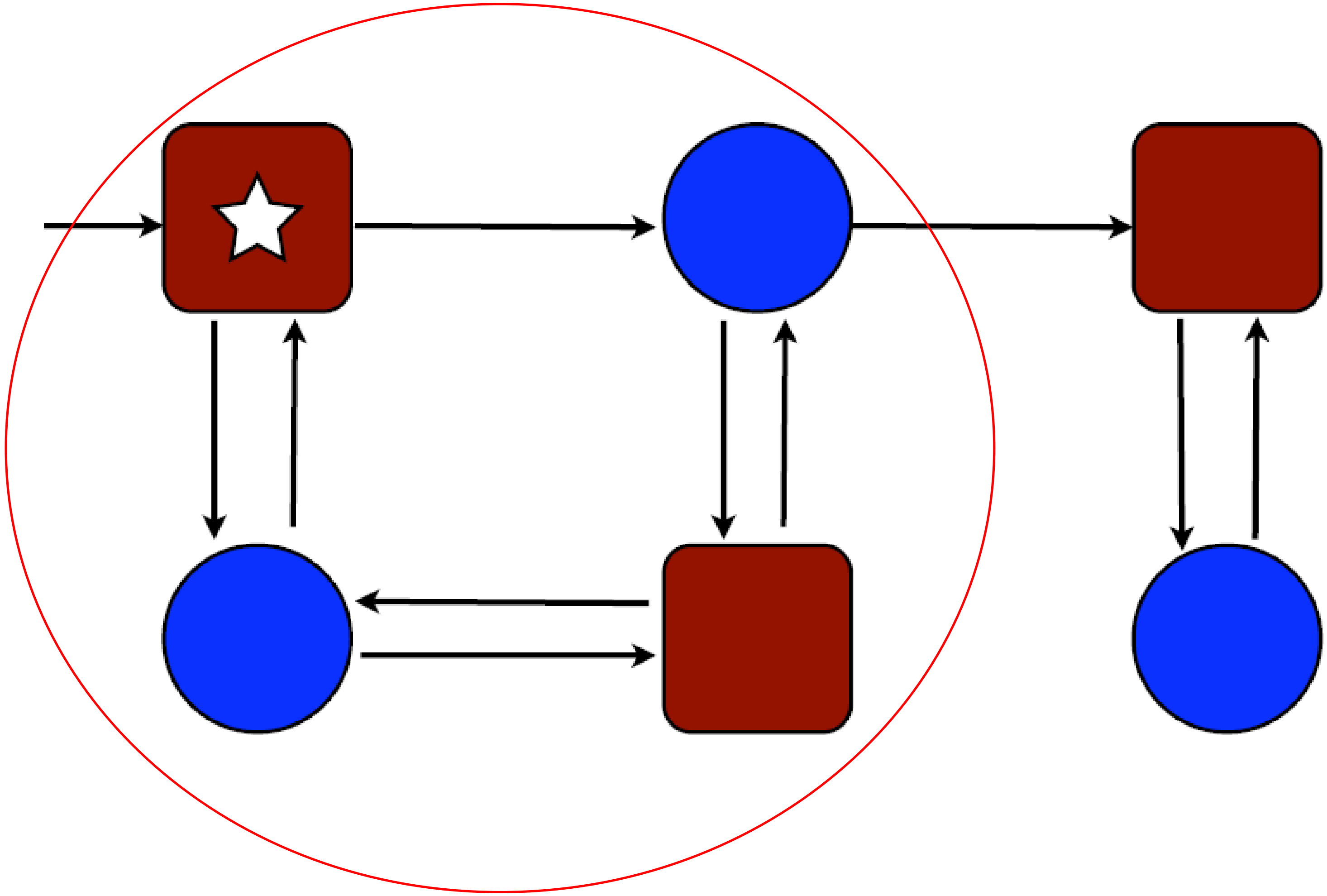
Theorem 2 (Reduction to a safety game). *Let* $\mathcal{A}$ *be a* tbU*K*CW *over inputs* Σ_I *and outputs* Σ_O *with n states* ($n > 0$)*, and let* $K = 2n(n^{2n+2} + 1)$. *The specification* $\mathcal{A}$ *is realizable iff Player O has a winning strategy in the game* $G(\mathcal{A},K)$.

Safety Game

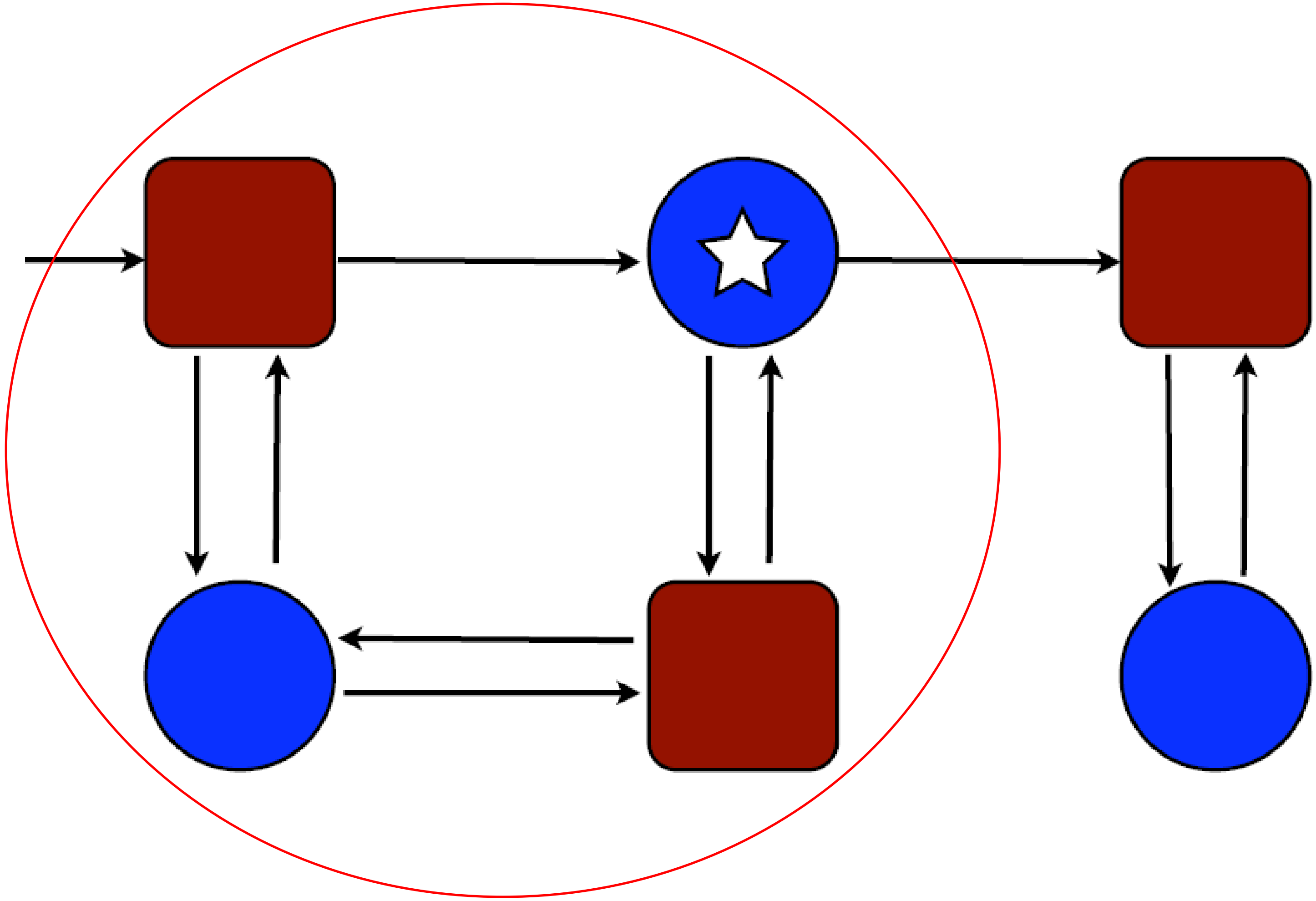
- *A game arena* is a tuple $G = (S_O, S_I, s_0, T, \text{safe})$ where S_I, S_O are disjoint sets of player states, $s_0 \in S_O$ is the *initial state*, $T \subseteq S_O \times S_I \cup S_I \times S_O$ is the *transition relation* and safe is the *safety consition*.
- A *finite play* on G of length n is a finite word $\pi = \pi_0\pi_1 \ldots \pi_n \in (S_O \cup S_I)^*$

  s. t. $\pi_0 = s_0$ and for all $i = 0, \ldots, n - 1$, $(\pi_i, \pi_{i+1}) \in T$.
- A *winning condition W* is a subset of $(S_O S_I)^*$.
- A *play* π is won by Player *O* if $\pi \in W$, otherwise it is won by Player *I*.
- A *strategy* λ_i for Player i ($i \in \{I,O\}$) is a *mapping* that maps any finite play whose last state s is in S_i to a state s' s. t. $(s, s') \in T$.
- The *outcome* of a strategy λ_i of Player i is the set $\text{Outcome}_G(\lambda_i)$ of infinite plays $\pi = \pi_0\pi_1\pi_2 \ldots$ s.t. for all $j \geq 0$, if $\pi_j \in S_i$, then $\pi_{j+1} = \lambda_i(\pi_0, \ldots, \pi_j)$.
- A strategy λ_O for Player *O* is *winning* if $\text{Outcome}_G(\lambda_O) \subseteq \text{safe}^\omega$.
  - Must void the *bad* states!

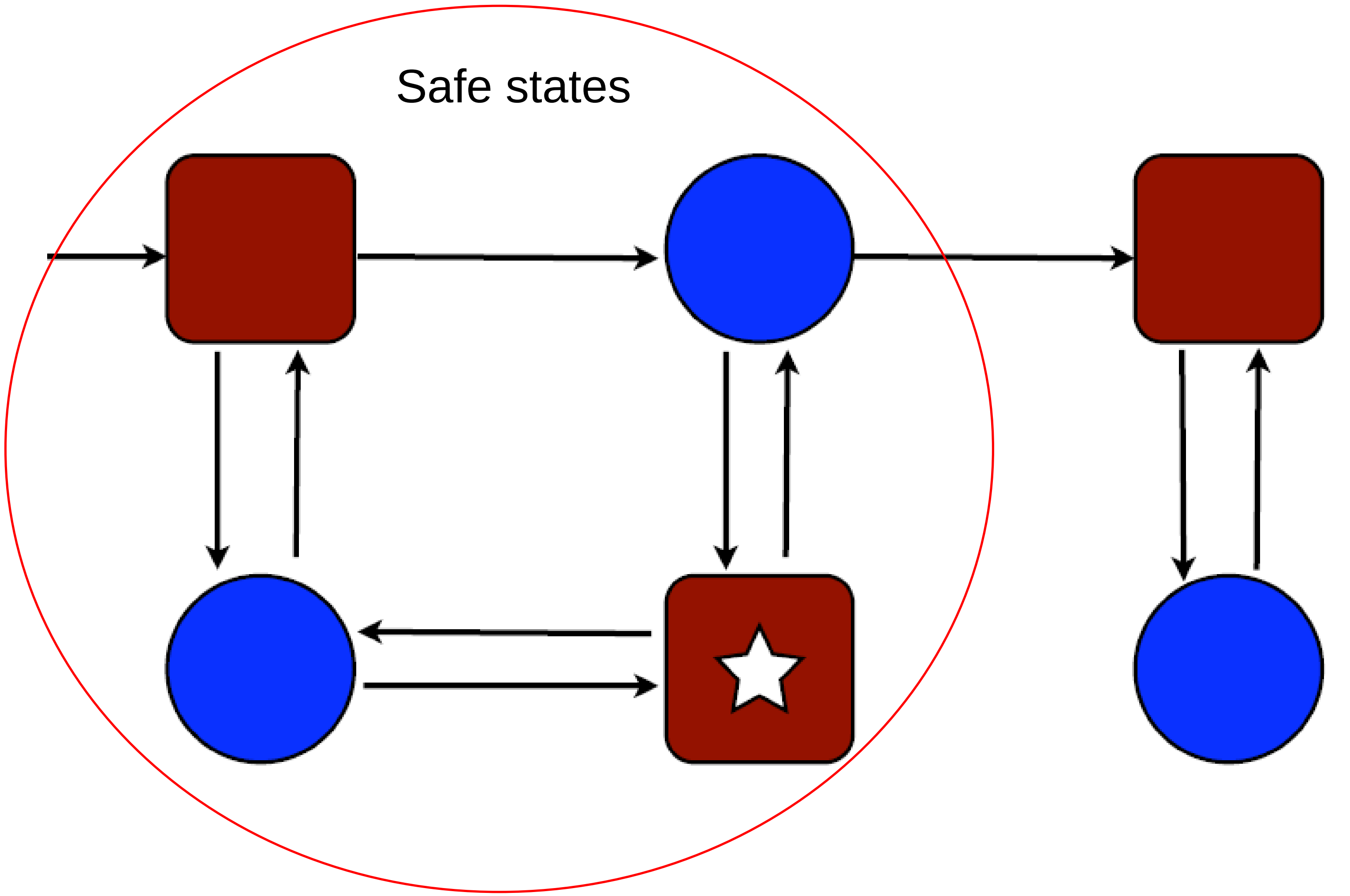
Safety Game



Safety Game



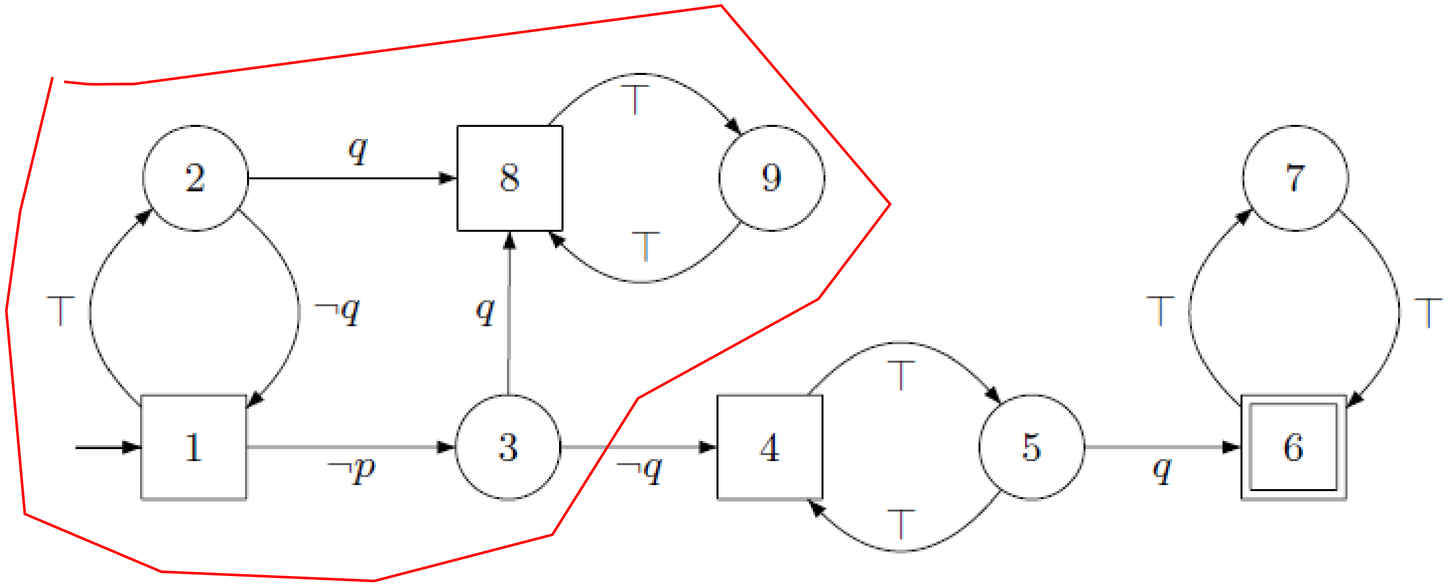
Safety Game



System controller wins if it has a strategy to keep the system in safe states.

Example of tbUCW

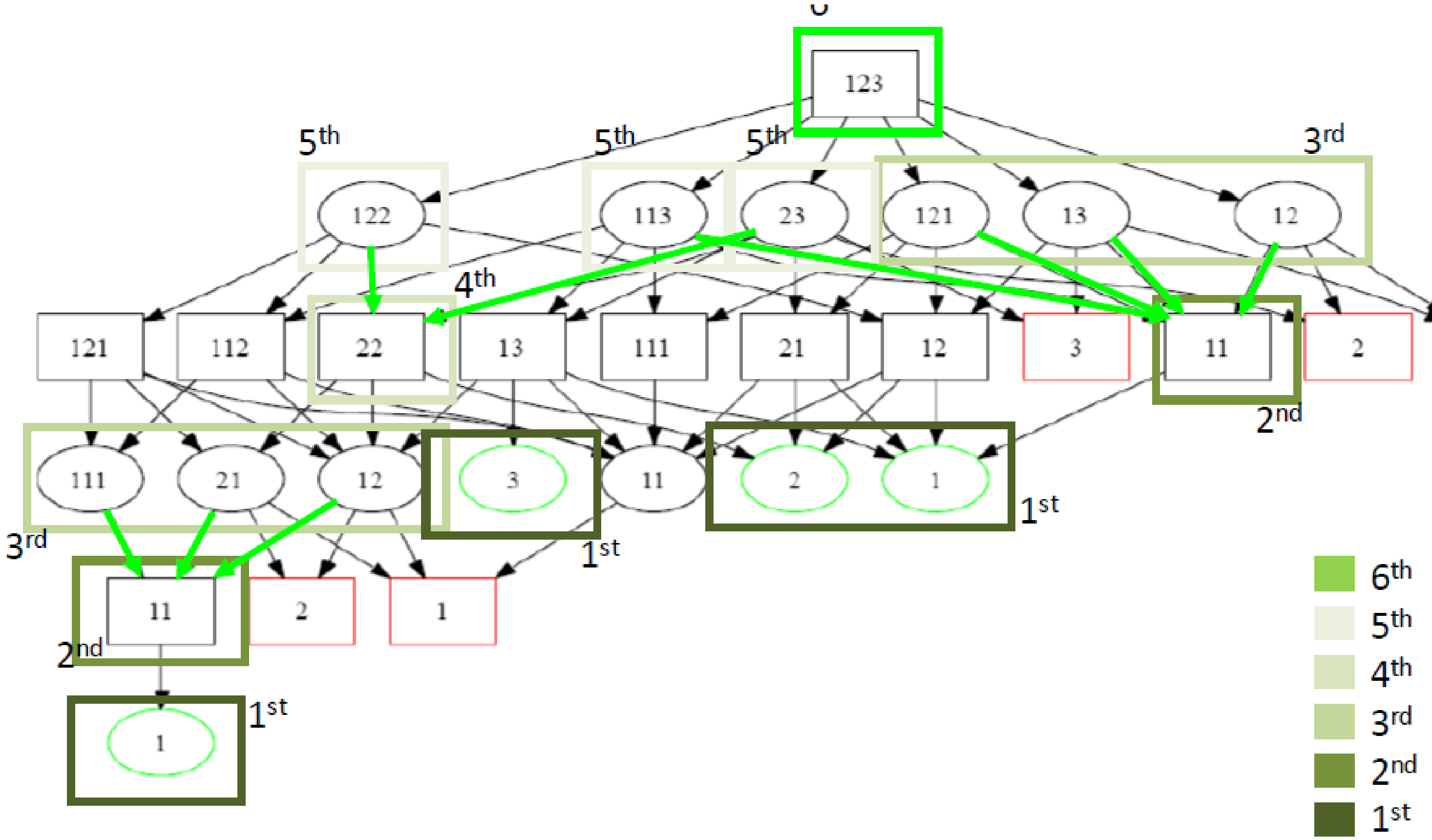
- tbUCW for **F**q → (p**U**q) where I = {q} and O = {p}
- Output states Q_O = {1, 4, 6, 8} are depicted by squares and input states Q_I = {2, 3, 5, 7, 9} by circles
- ⊤ stands for the sets Σ_I or Σ_O, depending on the context, ¬q (resp. ¬p) stands for the sets that do not contain q (resp. p), i.e. the empty set.
- At state 1, if controller does not assert p and next the environment does not assert q, then the run is in state 4. From this state, whatever the controller does, if the environment asserts q, then the controller loses, as state 6 will be visited infinitely often.



- A strategy for the controller is to assert p all the time, therefore the runs will loop in states 1 and 2 until the environment asserts q. Afterwards the runs will loop in states 8 and 9, which are non-final.

Solving safety games

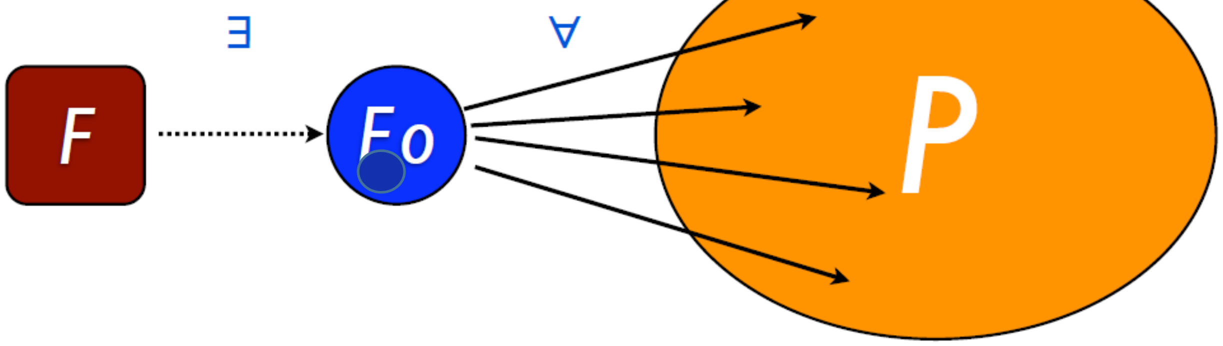
- Algorithms for solving safety games are constructed using the so-called *controllable predecessor operator*.



Solving safety games with Acacia+

- Let $G(A,K) = (\mathbb{F}_O, \mathbb{F}_I, F_0, T, \text{safe})$ and set of all *counting functions* $\mathbb{F} = \mathbb{F}_O \cup \mathbb{F}_I$.
- The controllable predecessor operator is based on the two following monotonic functions over the superset of the counting functions $2^{\mathbb{F}}$:
  - $\text{Pre}_I : 2^{\mathbb{F}O} \rightarrow 2^{\mathbb{F}I}$, $\text{Pre}_O : 2^{\mathbb{F}I} \rightarrow 2^{\mathbb{F}O}$.
- Let $P \subseteq \mathbb{F}$ be a subset of system positions. The safe controllable predecessors of P are then:

$$\text{CPre}(P) = \{F \mid \exists o \subseteq O, \forall F', ((Fo),F') \in T \Rightarrow F' \in P\} \cap \text{safe}$$



Properties of the controllable predecessor - 1

- Let CPre $=$ $\mathrm{Pre}_O \circ \mathrm{Pre}_I$. Function CPre is monotonic over the *complete lattice* $(2^{\mathbb{F}O}, \subseteq)$, and so it has a ***greatest fixed point*** denoted by CPre^*.

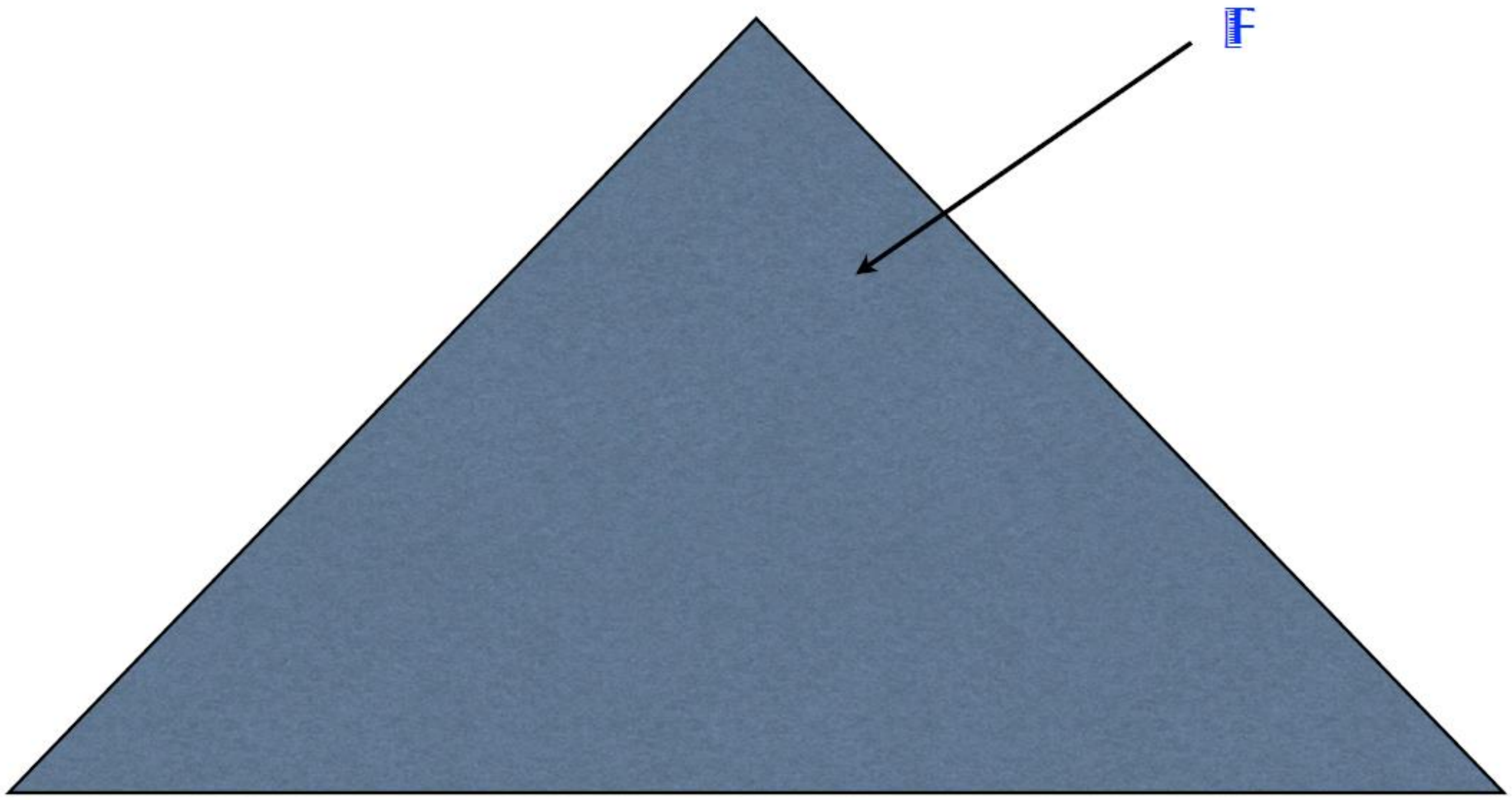
Theorem. *The set of states from which Player O* (the system) *has a winning strategy in G*($\mathcal{A}$,*K*) *is equal to* CPre^**.*

- By Theorem for the Reduction to a Safety Game, system has a winning strategy in *G*($\mathcal{A}$,*K*) iff the initial state $F_0 \in \mathrm{CPre}^*$.

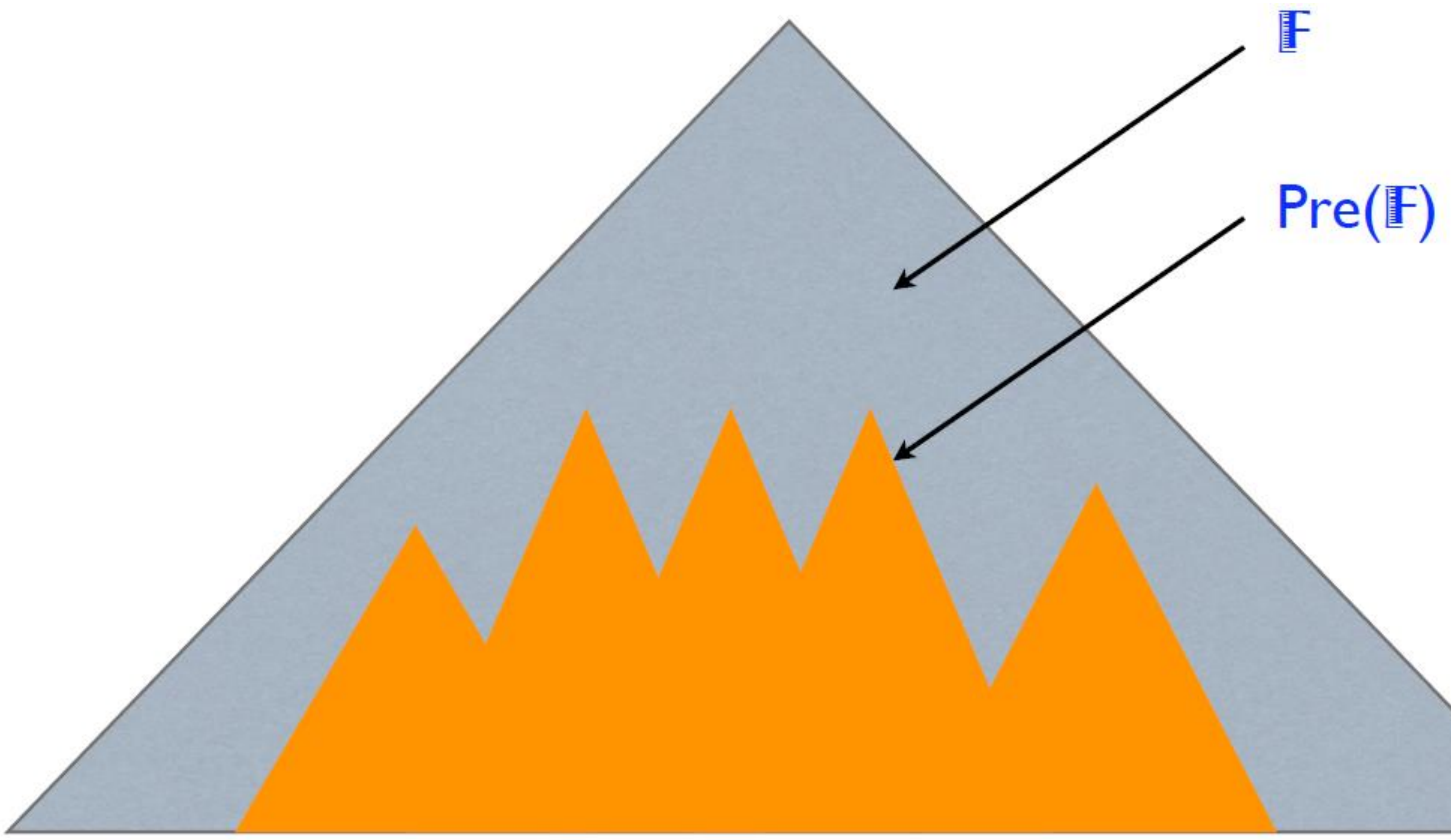
Properties of the controllable predecessor - 2

- $\mathbb{F}$ can be *partially ordered* by $F \preccurlyeq F'$ iff $\forall q, F(q) \leq F'(q)$.
  - If system wins from *F´*, it can also win from *F.*
- CPre() preserves *downward*-closed sets.
  - A set $S \subseteq \mathbb{F}$ is *closed for* $\preccurlyeq$, if $\forall F \in S \cdot \forall F' \preccurlyeq F \cdot F' \in S$.
  - For all *closed* sets $S \subseteq \mathbb{F}$, the closure of *S* denoted by $\downarrow S$, is equal to *S*.
- A set $S \subseteq \mathbb{F}$ is an *antichain* if all elements of *S* are incomparable for $\preccurlyeq$.
- The set of *maximal elements* of *S* is an **antichain***,* $\boldsymbol{S} = \{F \in S \mid \nexists F' \in S \cdot F' \neq F \land F \preccurlyeq F\}$.
- For Acacia+ antichains are a compact and efficient representation to manipulate closed sets in $\mathbb{F}$.
- Each (downward) set of the fixpoint computation is represented by its maximal elements.

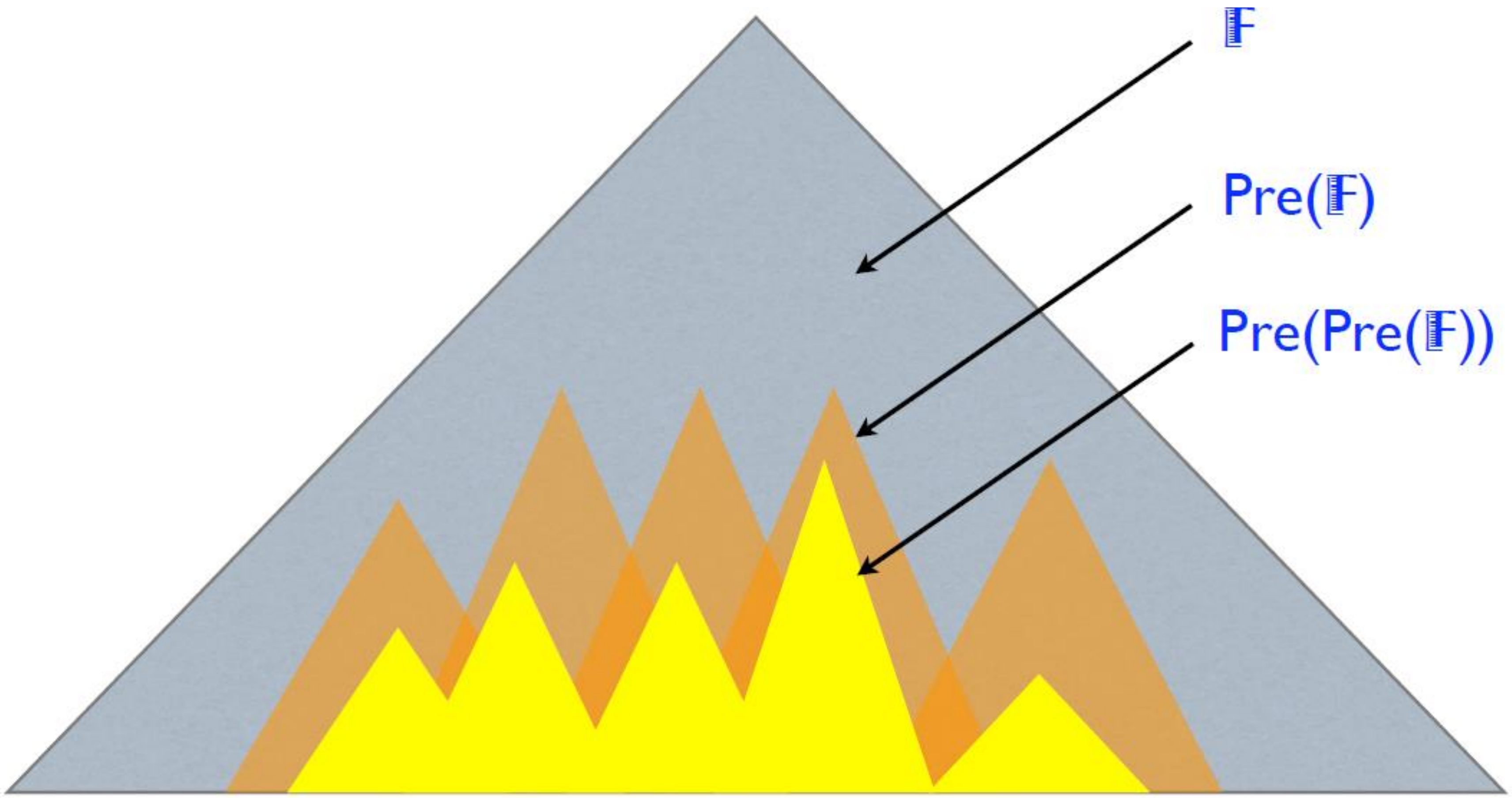
Symbolic Fixpoint Computation



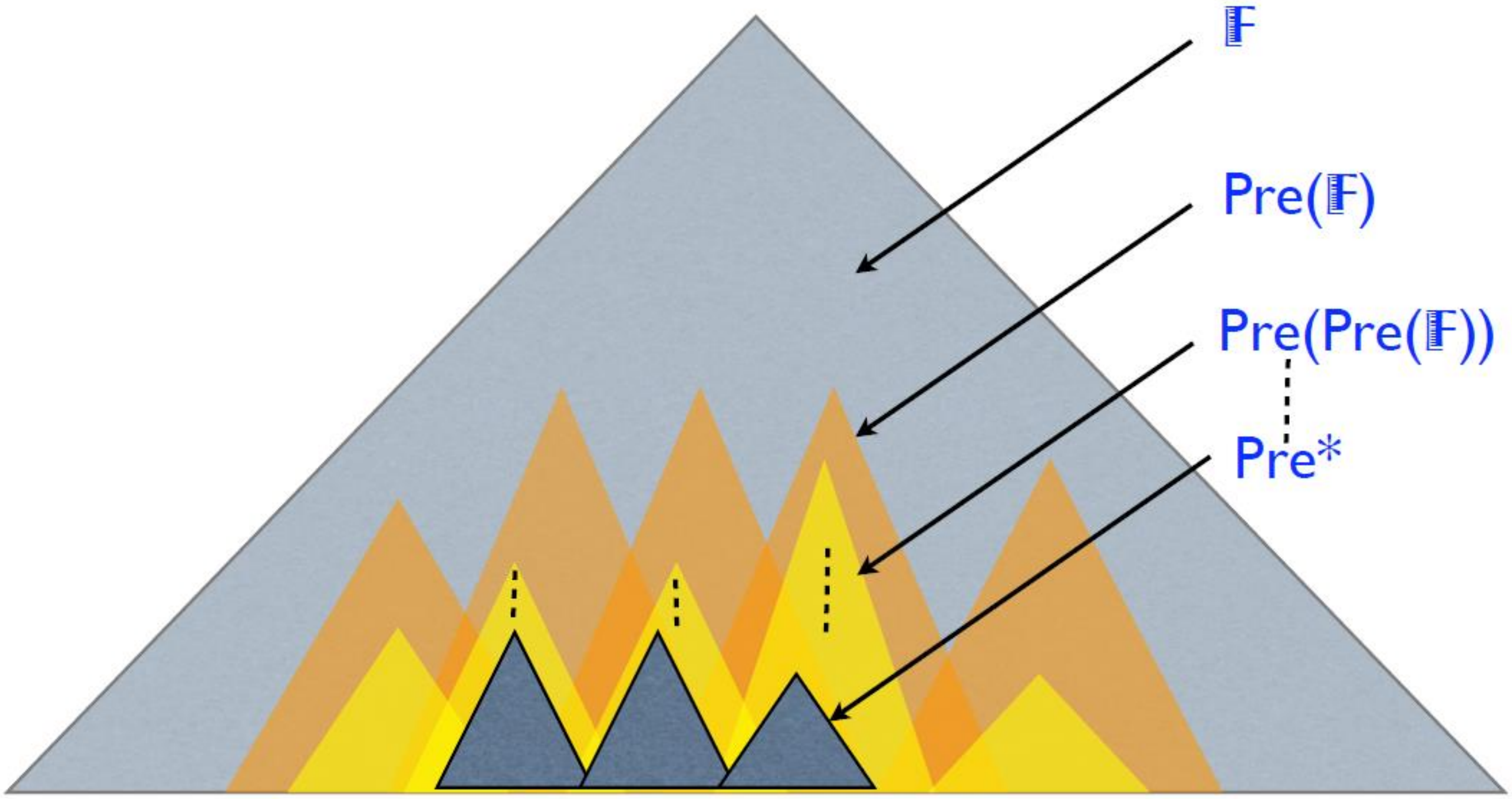
Symbolic Fixpoint Computation



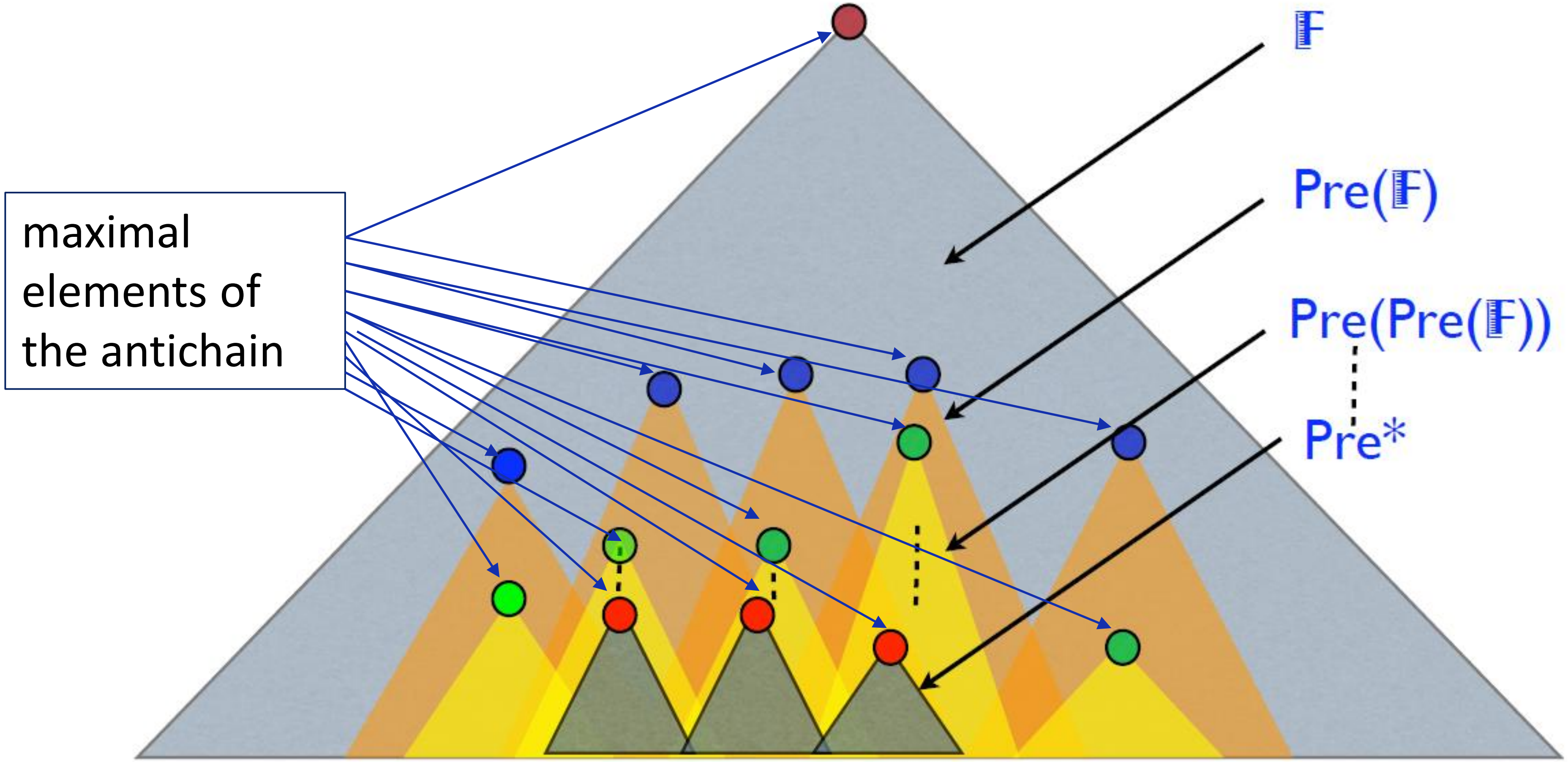
Symbolic Fixpoint Computation



Symbolic Fixpoint Computation



Symbolic Fixpoint Computation



Incremental realizability checking

- For checking the existence of a winning strategy for Player *O* in the safety game, the following property of U*K*CWs:

$$\text{for all } K_1, K_2 \;\cdot\; 0 \leq K_1 \leq K_2 \;\cdot\; L_{uc,K1}(A) \subseteq L_{uc,K2}(A) \subseteq L_{uc}(A).$$

```
Input: an LTL formula Φ, a partition I,O
A ← UCW with n states equivalent to Φ
K ← n^(2n+2)
for k=0…K do
    if System wins then G(A,k) return realizable
endfor
return unrealizable
```

Not reasonable to test for unrealizable specifications. Need to reach the upper bound for *K*.

Unrealizability Checking

- As a consequence of the determinacy theorem for Borel games:
- φ is unrealizable for the System iff $\neg\varphi$ is realizable for the Environment.
- The previous algorithm is adapted to test unrealizability.
- Realizability by Player O of φ is checked, and *in parallel* realizability by Player I of $\neg\varphi$, incrementing the value of K.
- When one of the two processes stops, it is known if φ is realizable or not.
- In practice, realizability or unrealizability are obtained for small values of K.

References

- An Antichain Algorithm for LTL Realizability . http://lit2.ulb.ac.be/acaciaplus/slides/cav09.pdf

  Slides of presentation of the following paper at CAV 2009 conference.
- Filiot E., Jin N., Raskin JF. (2009) An Antichain Algorithm for LTL Realizability. In: Bouajjani A., Maler O. (eds) Computer Aided Verification. CAV 2009. Lecture Notes in Computer Science, vol 5643. Springer, Berlin, Heidelberg.
- http://lit2.ulb.ac.be/acaciaplus/ - link to the Acacia+ tool
- A. Pnueli and R. Rosner. On the synthesis of a reactive module. In Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '89) ACM, NY, USA, 179-190. DOI=http://dx.doi.org/10.1145/75277.75293, 1989