Course ITI8531: Software Synthesis and Verification

Lecture 14: Acacia+ LTL Synthesis part III

Spring 2019

Leonidas Tsiopoulos

leonidas.tsiopoulos@taltech.ee

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. (today)
  - **Compositional** approach for *large conjunctions* of LTL formulas. (today)
  - Performance is better or similar to other existing tools but its *main advantage* is the generation of *compact strategies.* (today)
- 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

$\downarrow 2^{O(n)}$

Universal coBüchi Word automata

$\downarrow O(1)$

Universal **KcoBüchi** Word automata

$\downarrow 2^{O(m^2)}$

Det. KcoBüchi Word automata

Realizability

~

Safety game

- Safety games are the simplest games to solve!

Acacia+ Safraless approach

LTL

$\downarrow 2^{O(n)}$

Universal coBüchi Word automata

$\downarrow O(1)$

Universal **KcoBüchi** Word automata

$\downarrow 2^{O(m^2)}$

Det. KcoBüchi Word automata

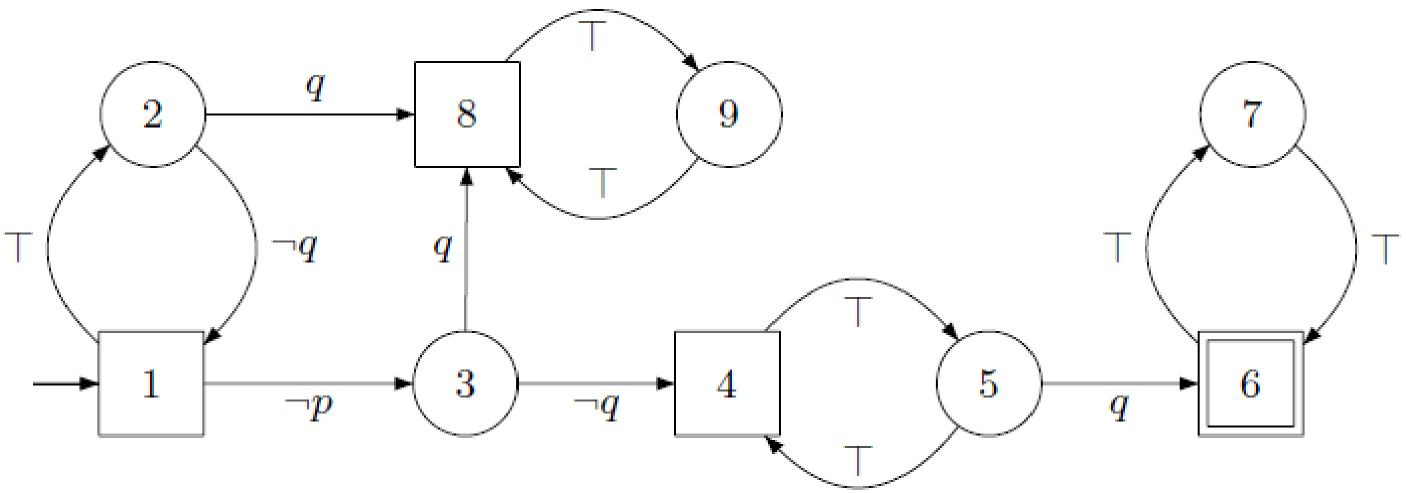
Doubly Exponential Size ...

... Game solved on-the-fly with **antichain** technics

- Safety games are the simplest games to solve!

Example of tbUCW

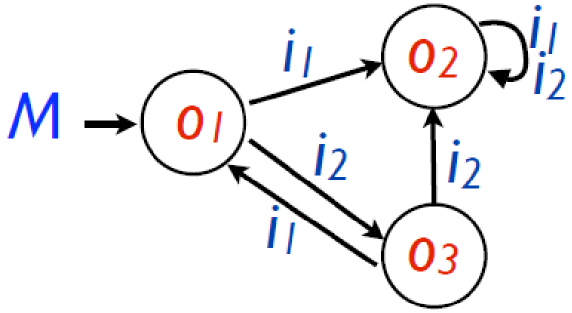
- tbUCW for $\mathbf{F}q \rightarrow (p\mathbf{U}q)$ where $I = \{q\}$ and $O = \{p\}$
- Output states $Q_O = \{1, 4, 6, 8\}$ are depicted by squares and input states $Q_I = \{2, 3, 5, 7, 9\}$ by circles
- $\top$ stands for the sets Σ_I or Σ_O, depending on the context, $\neg q$ (resp. $\neg p$) stands 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 $L(M) \subseteq L_{\mathrm{uc}}(A)$.
- The tbUCW is equivalent to an LTL formula given as input and is constructed by using tools *Wring* or *LTL2BA*.

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 tbUKCWs

- Let A be a tbUKCW (Σ_O, Σ_I, Q_O, Q_I, q_0, α, Δ_O, Δ_I) with $K \in \mathbb{N}$.
  - Let $Q = Q_O \cup Q_I$ and $\Delta = \Delta_O \cup \Delta_I$.
- Let det(A,K) = (Σ_O,Σ_I,$\mathbb{F}_O$, $\mathbb{F}_I$, F_0, α', δ_O, δ_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: $q \in Q_O \longrightarrow (q_0 \in \alpha)$ **if** $q = q_0$, -1 otherwise.
  - $\alpha' = \{F \in \mathbb{F}_O \cup \mathbb{F}_I \mid \exists q, F(q) > K\}$.
  - $\mathrm{succ}(F, \sigma) = q \longrightarrow \mathtt{max}\{\mathtt{min}(K + 1, F(p) + (q \in \alpha)) \mid q \in \Delta(p, \sigma), F(p) \neq -1\}$
    - There is a successor state if the run up to p visited finaal states.
  - $\delta_O = \mathrm{succ}|_{\mathbb{F}O \times \Sigma O}$, $\delta_I = \mathrm{succ}|_{\mathbb{F}I \times \Sigma I}$

Reduction to Safety Games

- The *game* $G(\mathcal{A},K)$ can be defined as follows:
  - it is $\det(\mathcal{A},K)$ 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 $\mathcal{T} = \{(F, F') \mid$

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

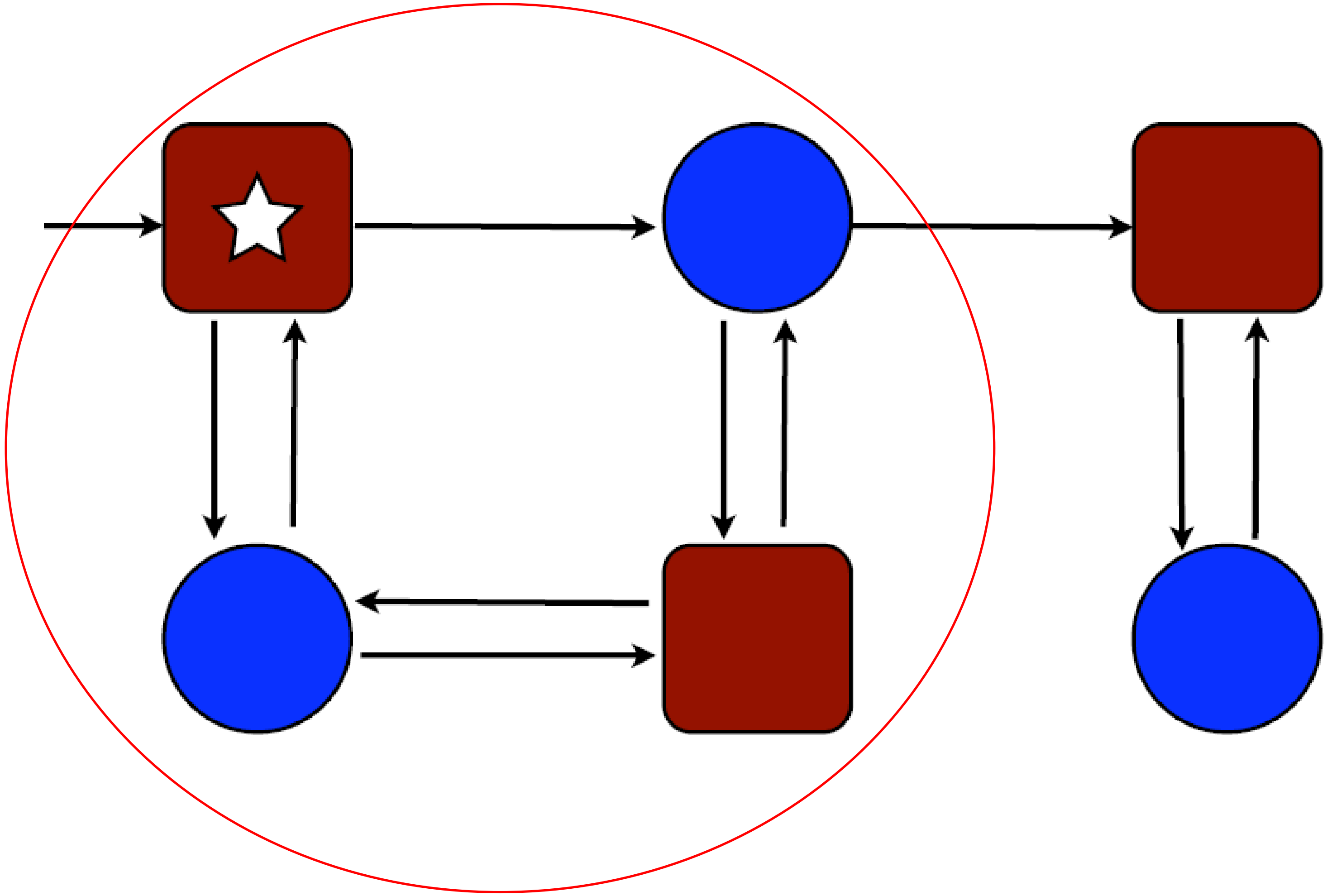
Theorem 2 (Reduction to a safety game). *Let $\mathcal{A}$ be a* tbU*K*CW *over inputs Σ_I and outputs Σ_O with n states* ($n > 0$)*, and let $K = 2n(n^{2n+2} + 1)$. The specification $\mathcal{A}$ is realizable iff Player O has a winning strategy in the game $G(\mathcal{A},K)$.*

Safety Game

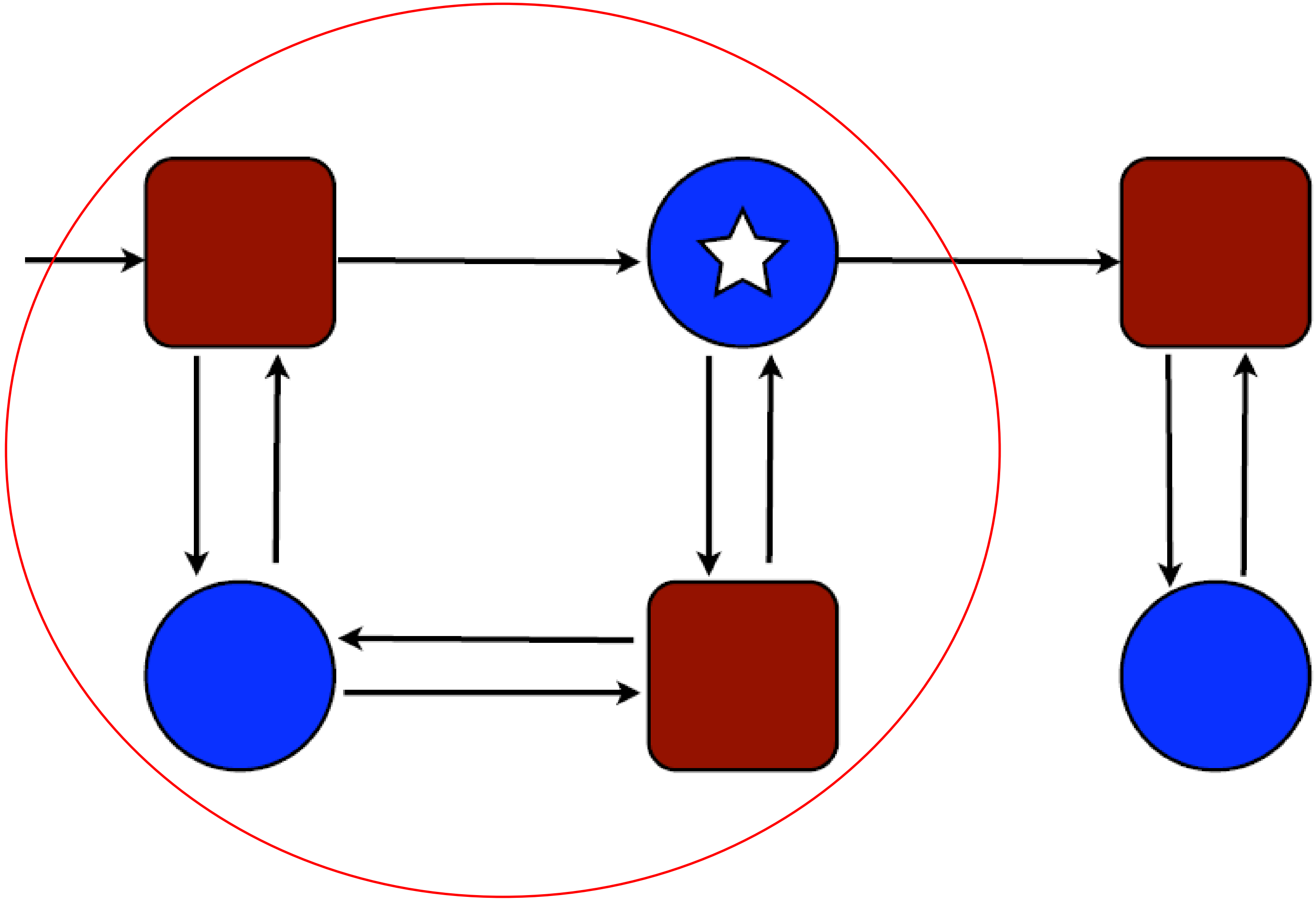
- *A game arena* is a tuple $G = (S_O, S_I, s_0, T, \text{safe})$ where S_I, S_O are disjoint sets of player states, $s_0 \in S_O$ is the *initial state*, $T \subseteq S_O \times S_I \cup S_I \times S_O$ is the *transition relation* and safe is the *safety consition*.
- A *finite play* on G of length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

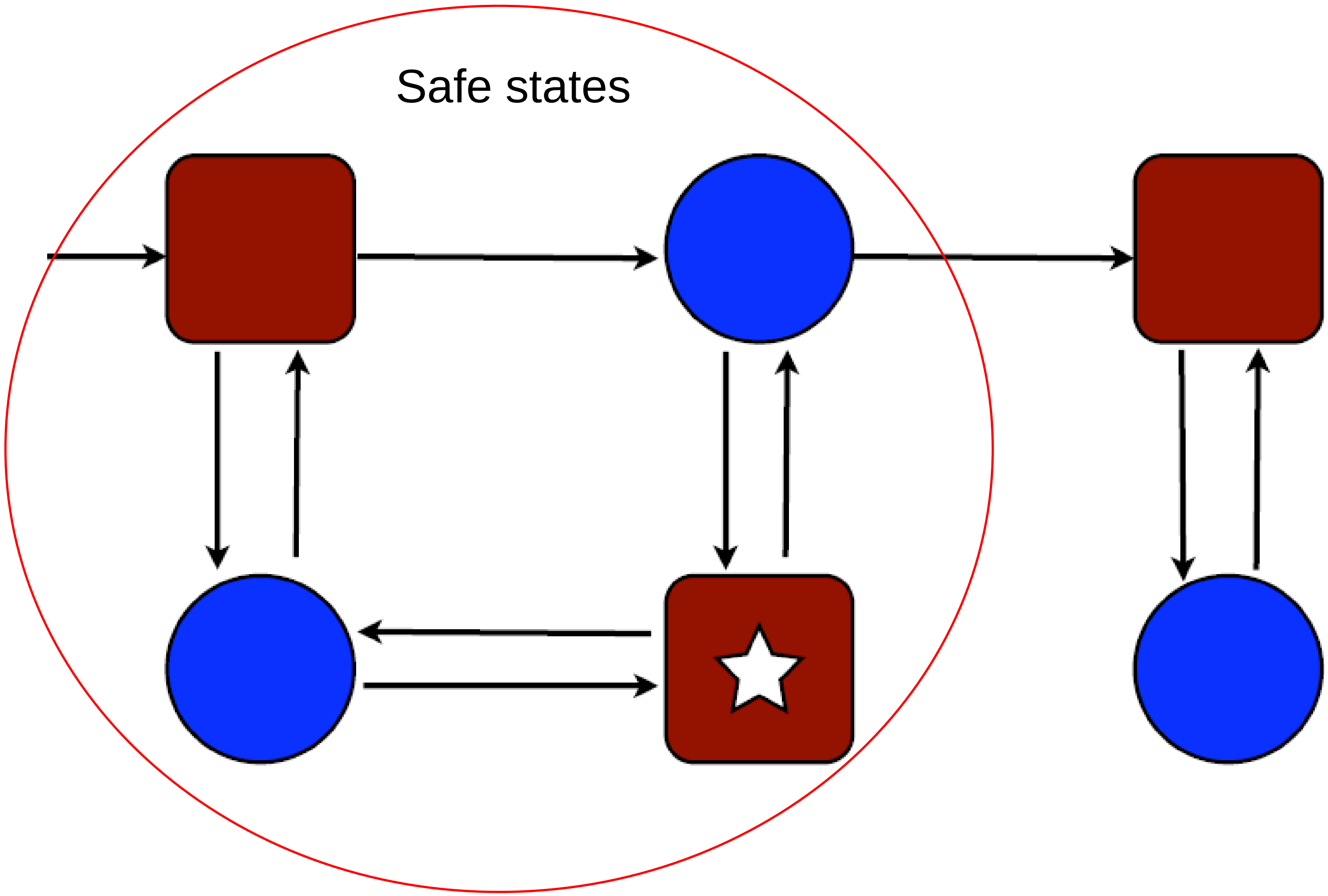
Safety Game



Safety Game



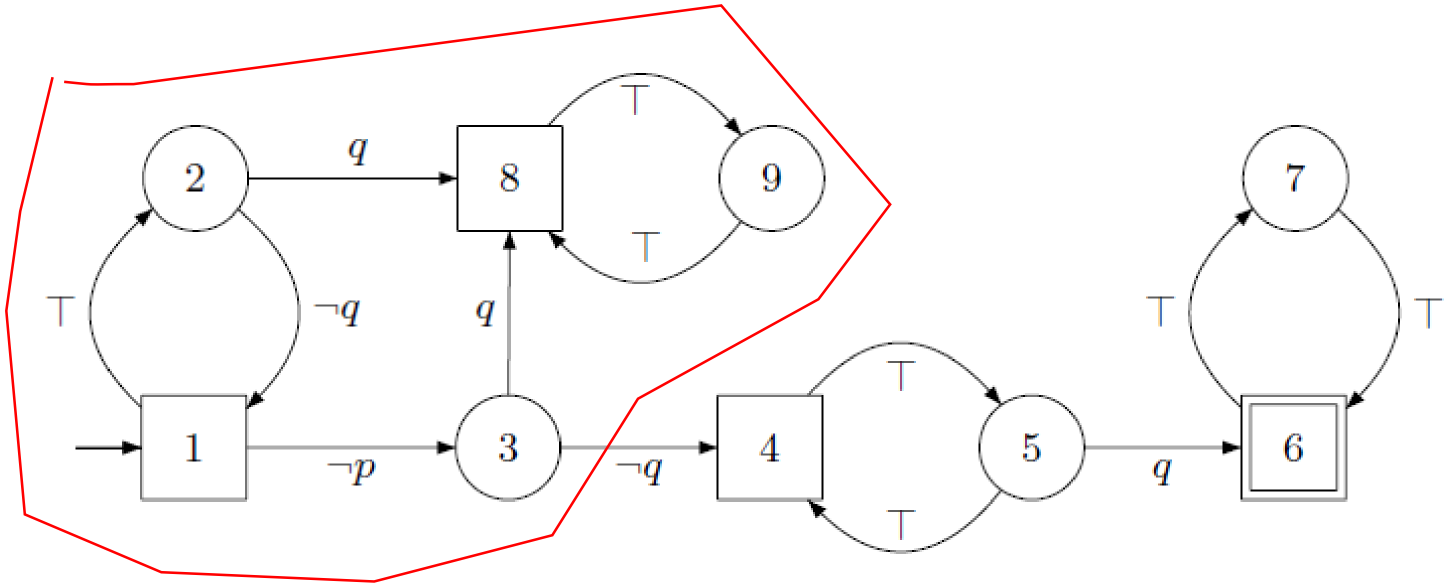
Safety Game



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

Example of tbUCW

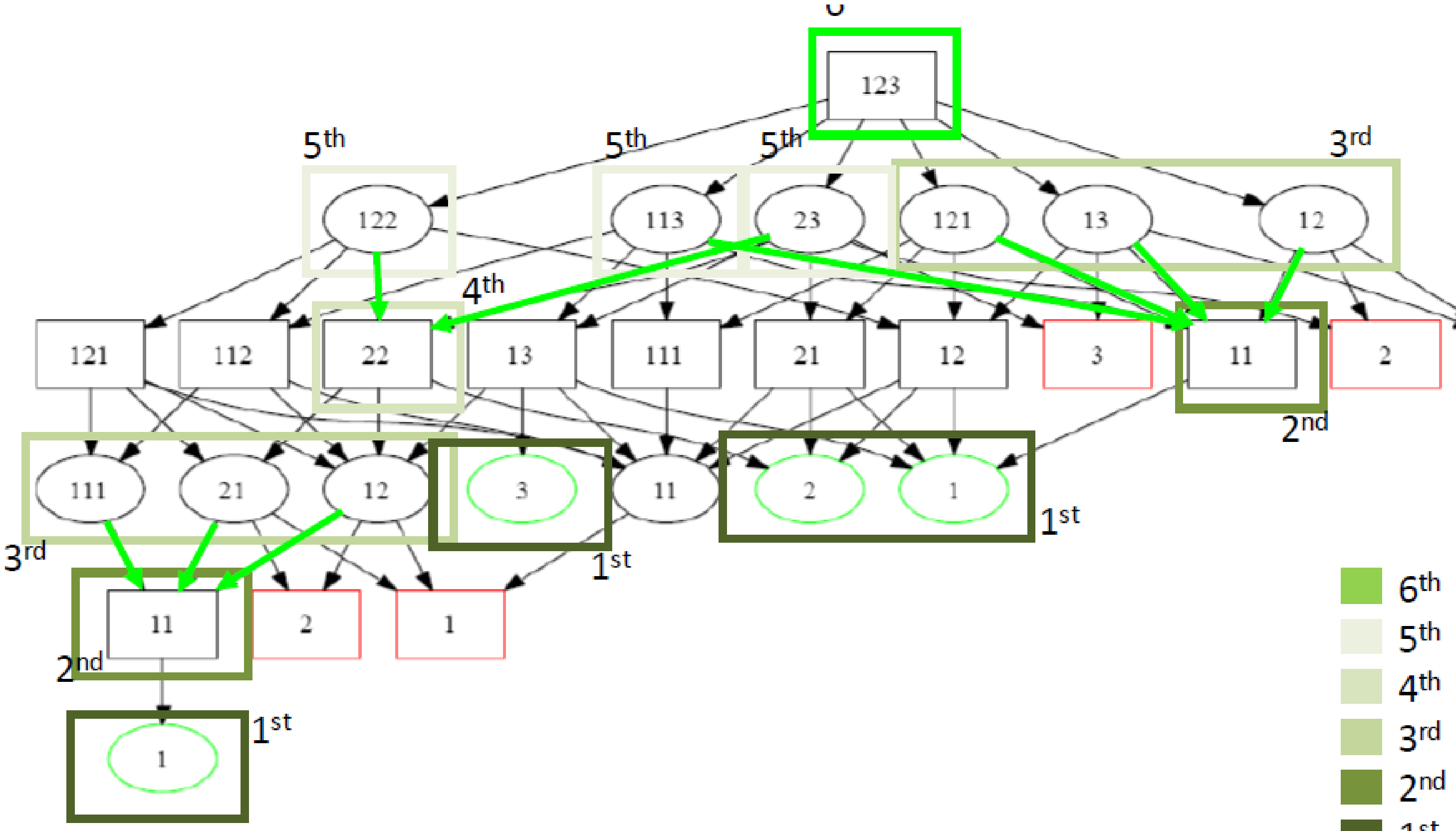
- tbUCW for **F**$q \rightarrow (p\mathbf{U}q)$ where $I = \{q\}$ and $O = \{p\}$
- Output states $Q_O = \{1, 4, 6, 8\}$ are depicted by squares and input states $Q_I = \{2, 3, 5, 7, 9\}$ by circles
- $\top$ stands for the sets Σ_I or Σ_O, depending on the context, $\neg q$ (resp. $\neg p$) stands for 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.

Solving safety games

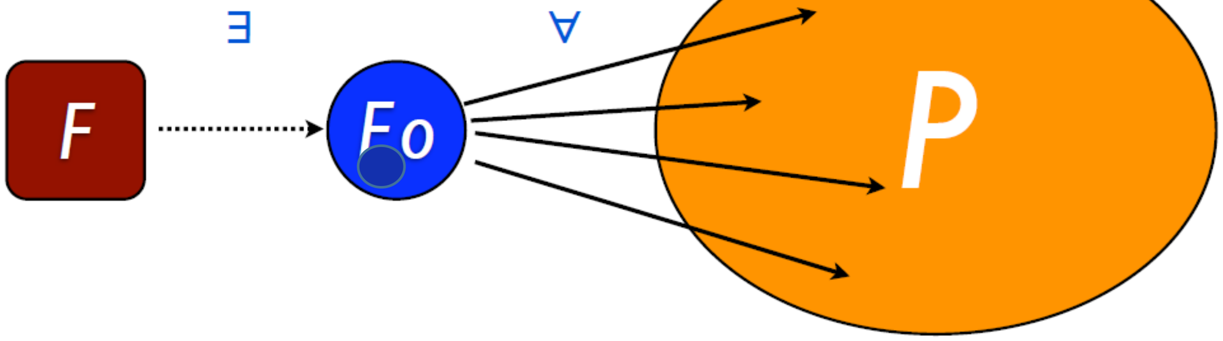
- Algorithms for solving safety games are constructed using the so-called *controllable predecessor operator*.



Solving safety games with Acacia+

- Let $G(\mathcal{A},K) = (\mathbb{F}_O, \mathbb{F}_I, F_0, \mathcal{T}, \text{safe})$ 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 $2^{\mathbb{F}}$:
  - $\text{Pre}_I : 2^{\mathbb{F}O} \to 2^{\mathbb{F}I}$, $\text{Pre}_O : 2^{\mathbb{F}I} \to 2^{\mathbb{F}O}$.
- Let $P \subseteq \mathbb{F}$ be a subset of system positions. The safe controllable predecessors of P are then:

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



Properties of the controllable predecessor - 1

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

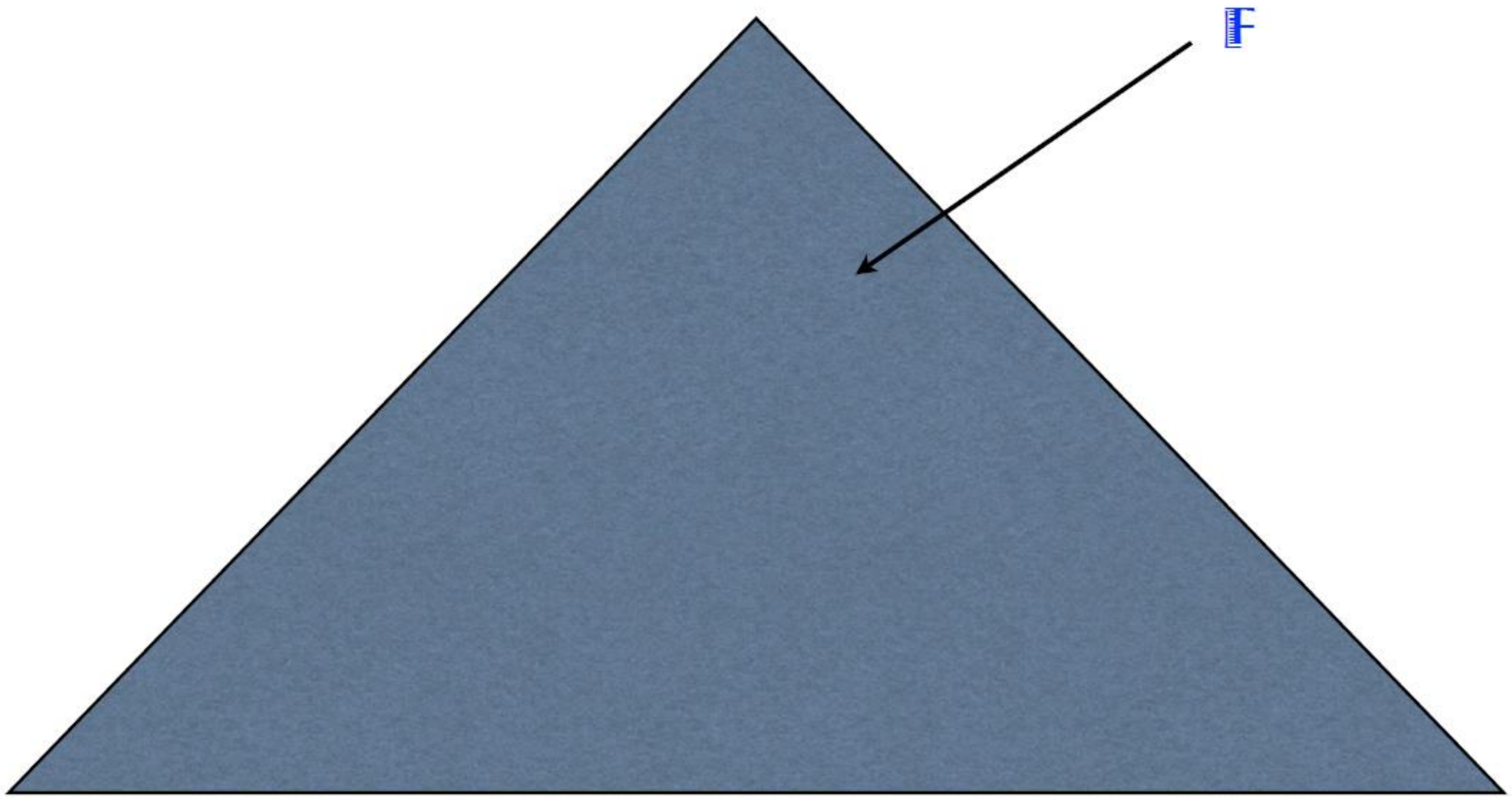
Theorem. *The set of states from which Player O* (the system) *has a winning strategy in* $G(\mathcal{A},K)$ *is equal to* CPre^*.

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

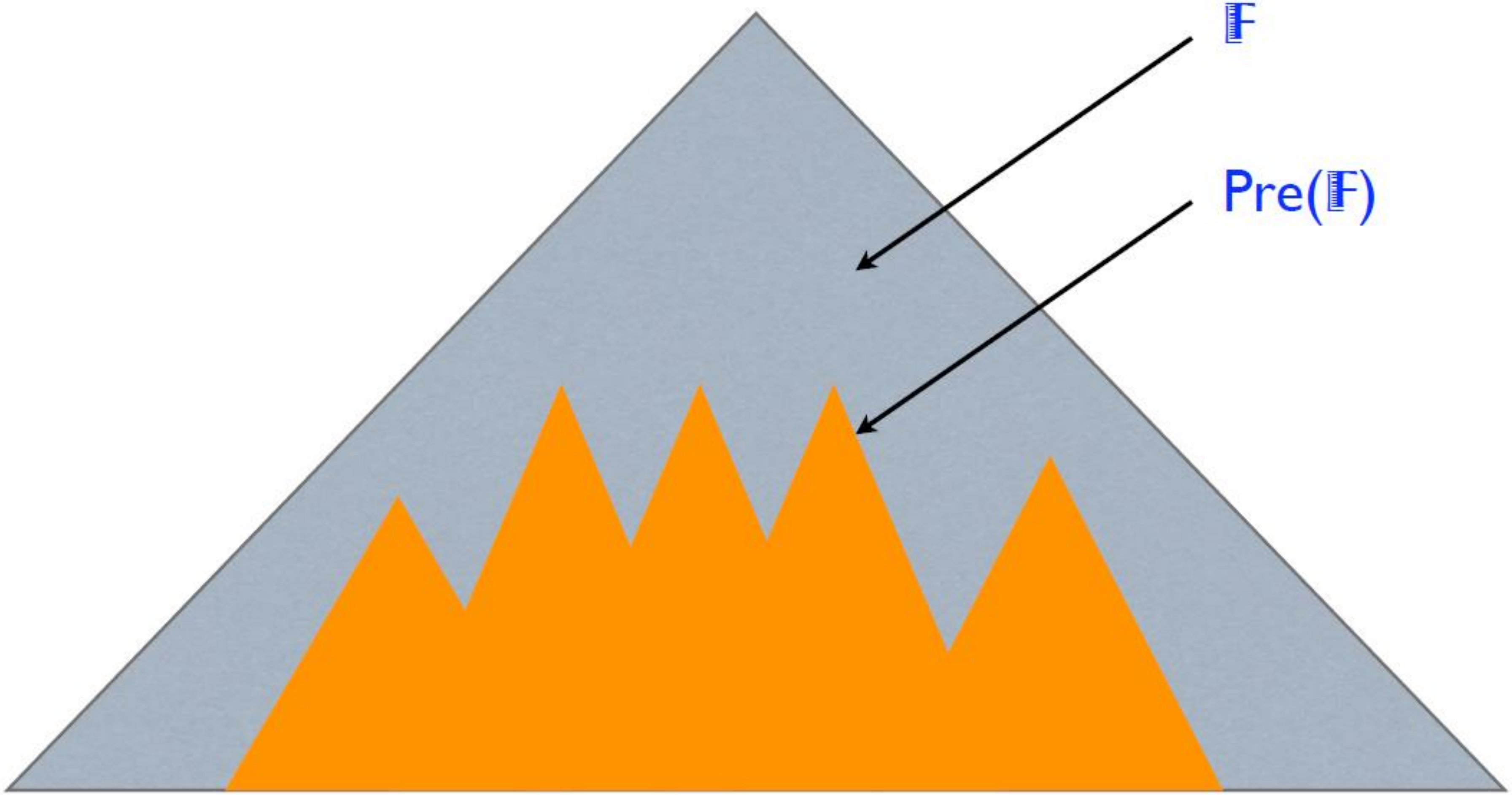
Properties of the controllable predecessor - 2

- $\mathbb{F}$ 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**, $\boldsymbol{\mathcal{S}} = \{F \in S \mid \nexists F' \in S \cdot F' \neq F \wedge F \preccurlyeq F\}$.
- 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.

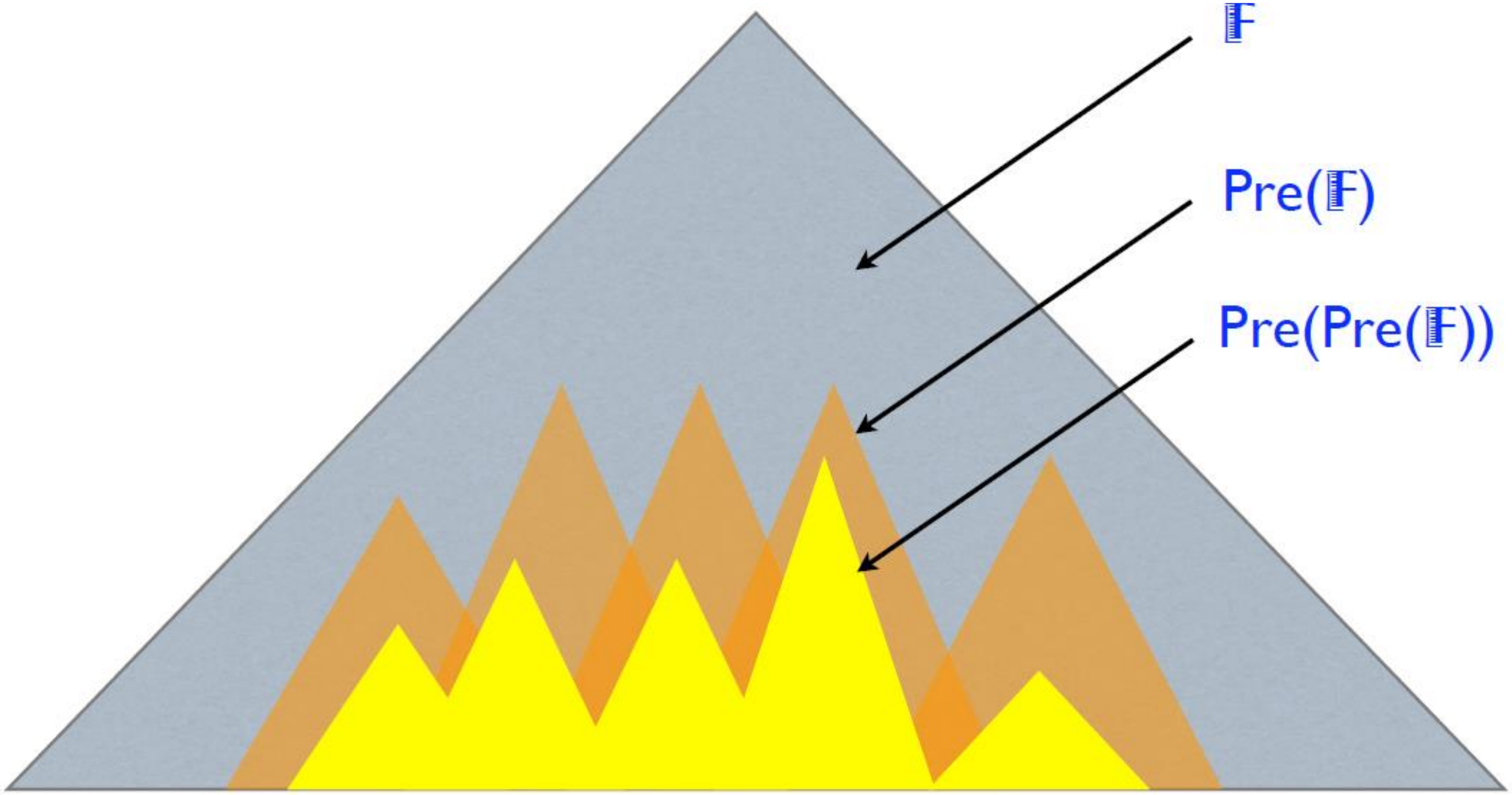
Symbolic Fixpoint Computation



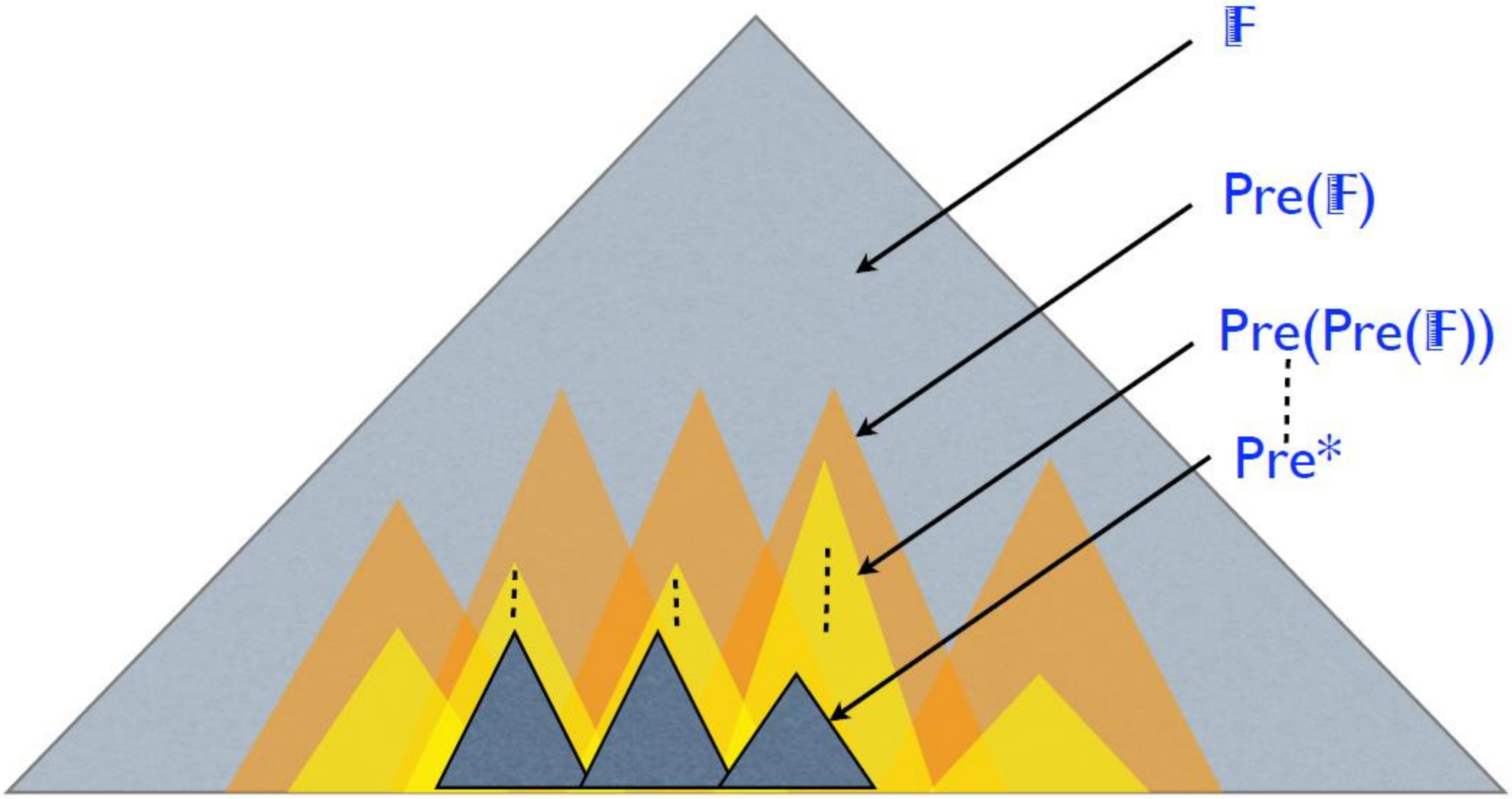
Symbolic Fixpoint Computation



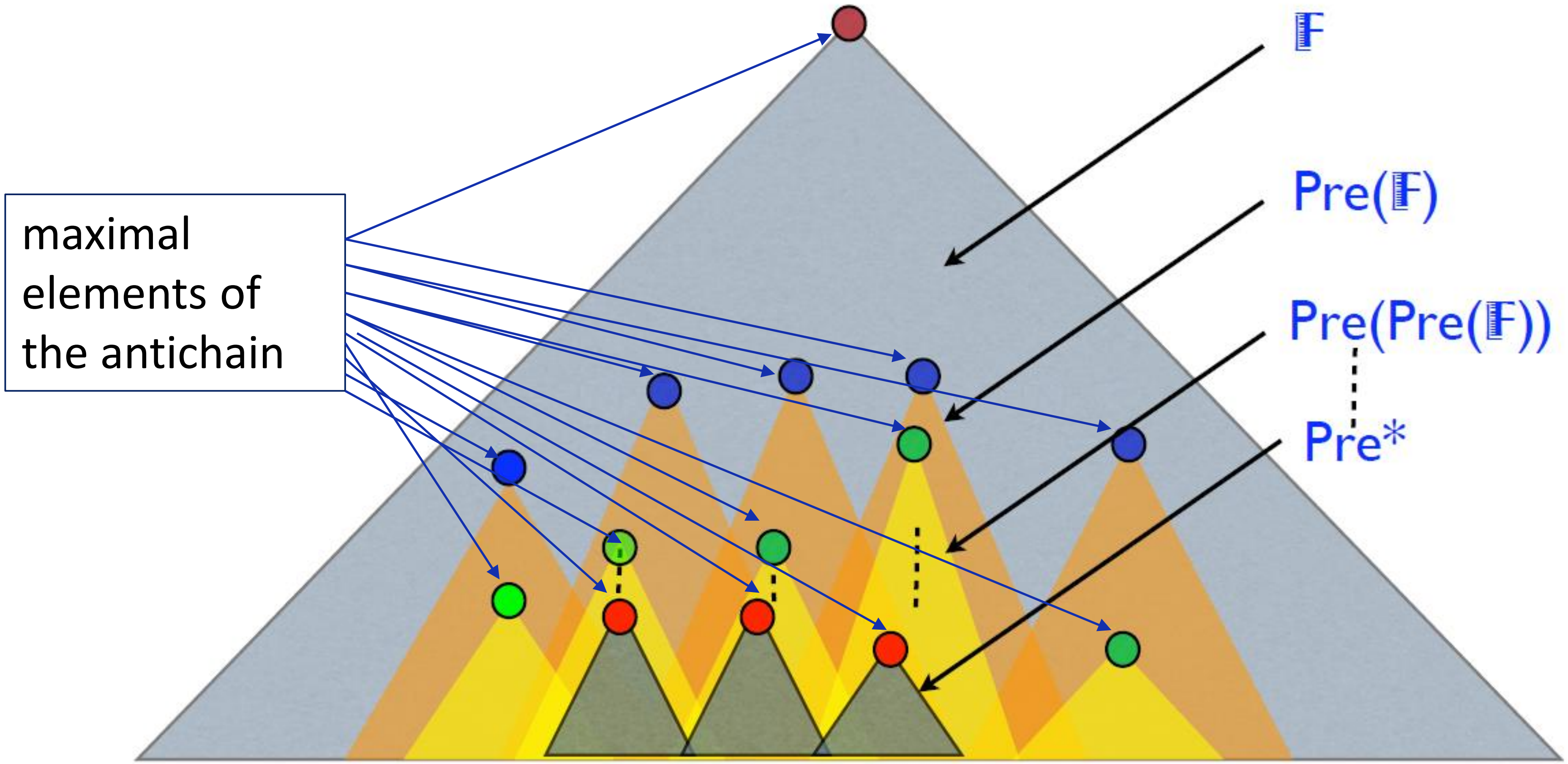
Symbolic Fixpoint Computation



Symbolic Fixpoint Computation



Symbolic Fixpoint Computation



Synthesis of winning strategies

- If a formula φ is realizable, *extract* from the *greatest fixpoint computation* a Moore machine that realizes it.
- Let $\Pi_I \subseteq \mathbb{F}_I \cap$ safe and $\Pi_O \subseteq \mathbb{F}_O \cap$ safe be the two sets obtained by the *greatest fixpoint computation*.
- $\mathrm{Pre}_O(\Pi_I) = \Pi_O$, $\mathrm{Pre}_I(\Pi_O) = \Pi_I$ --- Π_I and Π_O are *downward-closed.*
- By definition of Pre_O for all $F \in \lceil \Pi_O \rceil$, $\exists\ \sigma_F \in \Sigma$ such that

  $\mathrm{succ}(F, \sigma_F) \in \Pi_I$, and this σ_F can be computed.
- A Moore machine can be extracted:
  - set of states is $\lceil \Pi_O \rceil$,
  - the *output function* maps any state $F \in \lceil \Pi_O \rceil$ to σ_F ,
  - the *transition function* maps F to a partially-ordered state F' according to the succ operator,
  - and the *initial state* F_0 is a state partially-ordered with F.

Example of Moore machine synthesis

- tbUCW for $\mathbf{F}q \rightarrow (p\mathbf{U}q)$
- Start with the safe state in the game for the system, denoted by

$F1 = (1 \rightarrow 1, 4 \rightarrow 1, 6 \rightarrow 1, 8 \rightarrow 1)$.

- Then, for the system predecessor (Env.),

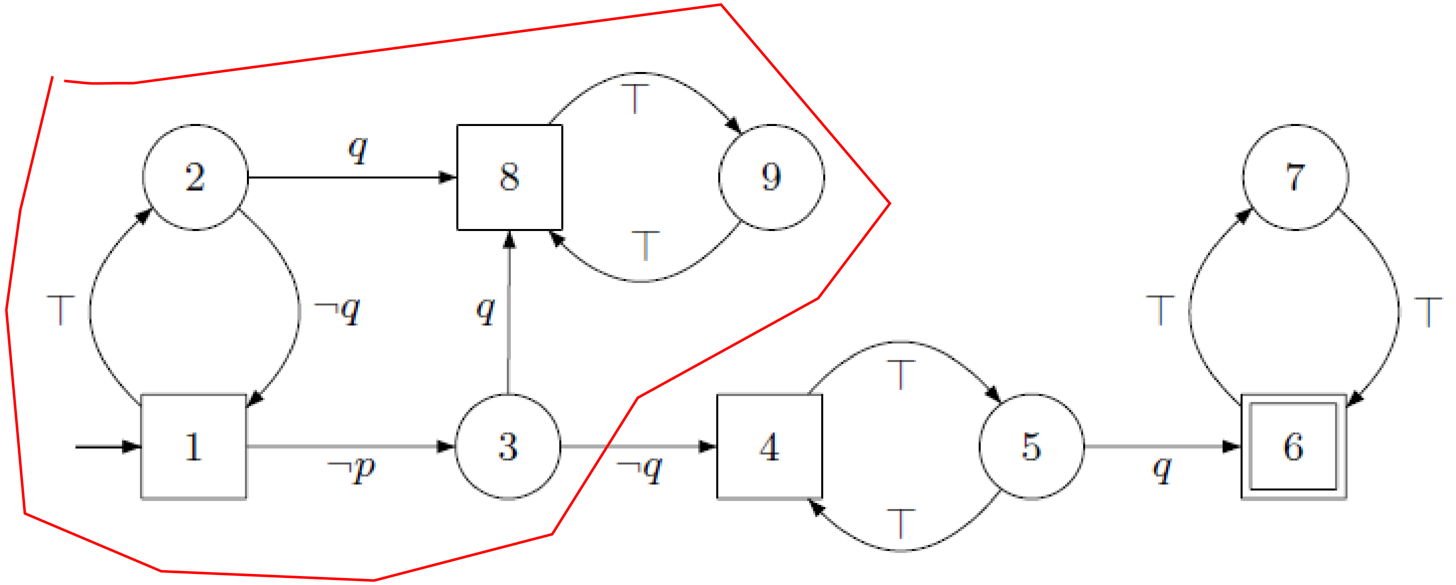
$F2 := (2 \rightarrow 1, 3 \rightarrow 1, 5 \rightarrow 0, 7 \rightarrow 0, 9 \rightarrow 1)$

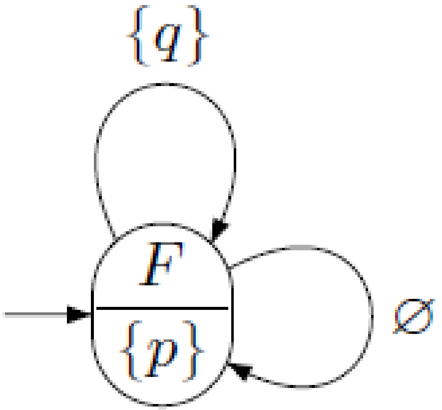
- Then for the controlled (System) predecessor

CPre = $(1 \rightarrow 1, 4 \rightarrow 0, 6 \rightarrow 0, 8 \rightarrow 1)$

- At end of computation, the fixpoint is:

$F := (1 \rightarrow 1, 4 \rightarrow -1, 6 \rightarrow -1, 8 \rightarrow 1)$





Forward algorithm for solving games

- In Acacia+ also a *forward* algorithm can be applied to solve games.
- Compared to the backward algorithm, the *forward* algorithm has the advantage that it computes ***only*** the winning positions *F* (for the System) which are reachable from the initial position.
    - But it can compute only one winning strategy.
- The algorithm explores the positions of the game and *once a position is known to be losing*, this information *is back propagated to the predecessors*.
- A position of Player System is *losing* iff it has *no* successors or *all* its successors are losing.
- A position of Player Env. is losing iff *one of* its successors is losing.

Forward algorithm – Sketch (1)

- At each step, maintain an *under-approximation* **Losing** of the set of losing positions.
- A waiting-list **Waiting** for reachable position exploration and re-evaluation of positions is used.
- An edge is put in the *waiting*-list if it is the first time it has been reached, or the status of its target position has changed.
- If a position is known that is losing, this is back-propagated to all its predecessors.
- A set **Passed** records the visited positions.
- a set **Depend** stores the edges (*s, s´*) which need to be re-evaluated when the value of *s´* changes.

Forward algorithm – Sketch (2)

- At each step, pick an edge *e* = (*s, s´*) in the *waiting*-list.
- If its target *s´* has never been visited, check if this target is losing
  - When it has no successors.
- If losing, add *e* in the *waiting*-list for re-evaluation.
  - Back propagate the information on *s´*.
- Otherwise add all the successors of *s´* in the *waiting*-list for re-evaluation.
- If *s´* has already been visited, then compute the value of *s*.
- If *s* is losing, this information is back propagated to the positions whose *safeness* depends on *s*.

Compositional safety games and LTL synthesis

- Acacia+ implements a compositional approach for synthesis of large conjunctions of LTL formulas.

- Realistic systems cannot be specified by just a couple of simple LTL formulae.

- A scalable approach is very beneficial!

Overview of compositional algorithms

- Two *compositional* algorithms for LTL formulas of the form

  $\varphi = \varphi_1 \wedge \cdot \cdot \cdot \wedge \varphi_n$ are implemented in Acacia+.

- *Backward* algorithm: At each stage of the *parenthesizing*, the antichains W_i of the subformulae φ_i are computed backward and the antichain of the formula φ itself is also computed backward from the W_i's.
  - **All** winning strategies for φ are computed and compactly represented by the final antichain.
- *Forward* algorithm: At each stage of the *parenthesizing*, the antichains W_i of the subformulae φ_i are computed backward, **except** at the last stage where a forward algorithm seeks for **one** winning strategy by exploring the game arena *on the fly* in a forward fashion.

Compositional safety games

- Compositional reasoning on safety games is supported by the existence of a most permissive strategy, a *master plan*.
- The *master plan* of System can be interpreted as a *compact representation* of all the winning strategies of System against the Environment.
  - It contains **all** the moves that System can play in a state *s* in order to win the safety game.
- The master plan associated with a game can be computed in a *backward* fashion by using variants of the controllable operator CPre and sequence of positions *W.*

Composition of safety games

- Let G^i , $i \in \{1, . . . , n\}$, be n safety games $G^i = (S^i{}_1, S^i2, \Gamma^i{}_1, \Delta^i{}_1, \Delta^i{}_2)$ defined on the same sets of moves, Moves = $\text{Moves}_1 \uplus \text{Moves}_2$.

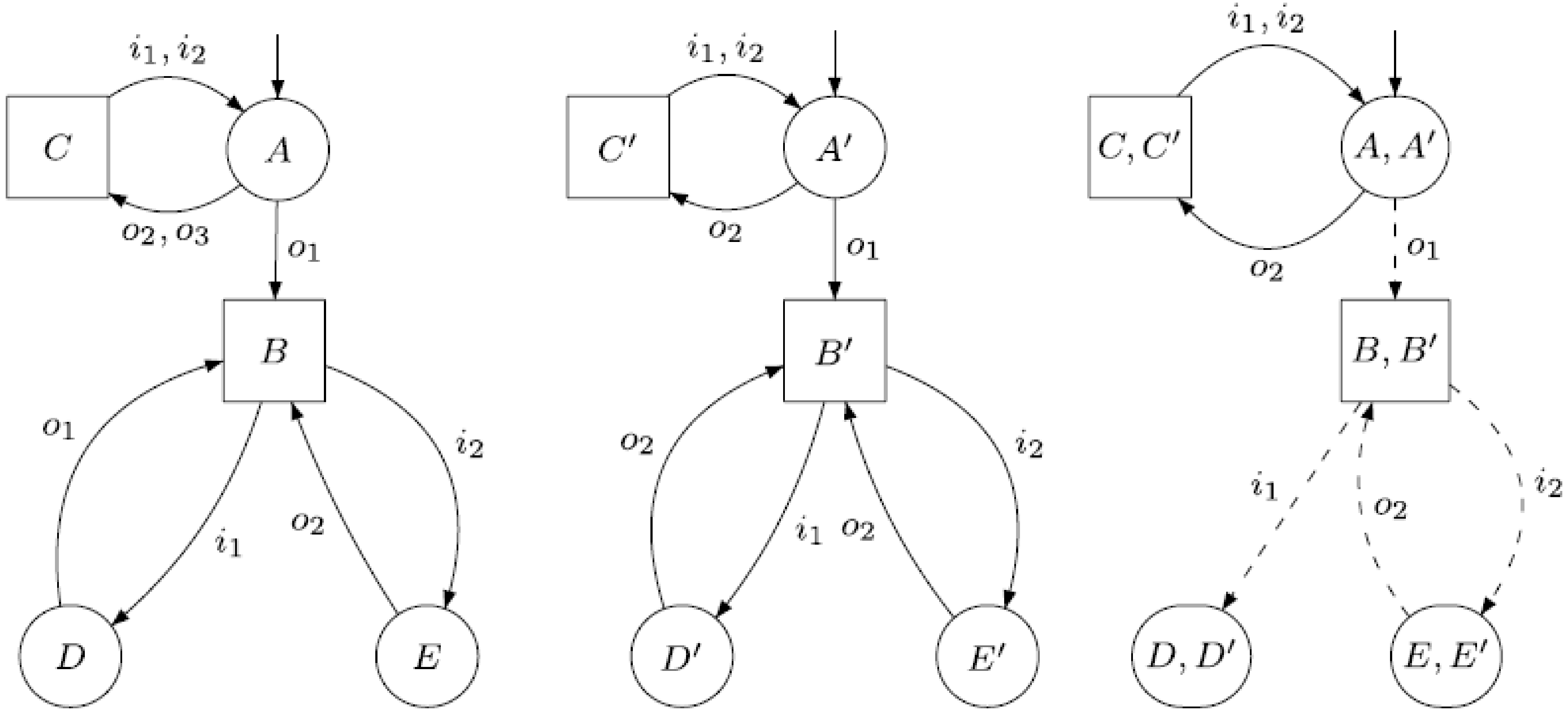
- Their product is the safety game $G^\otimes = (S^\otimes{}_1 , S^\otimes{}_2 , \Gamma^\otimes{}_1 , \Delta^\otimes{}_1 , \Delta^\otimes{}_2)$ over the *product of the state spaces* of the players, the *intersection (common) winning strategies* of the System and the *transitions* conforming to the winning strategy of System or to the moves of the Environment.

Backward compositional solving of G $^{\otimes}$

- First, compute locally the master plans of the components.
- Then compose the local master plans and apply one time the CPre operator to this composition to compute a function that contains information about the one-step inconsistencies between local master plans.
- Project back on the local components the information gained by the function , and iterate.

Forward compositional solving of G $^{\otimes}$

- Interested in computing a master plan only for the winning and *reachable* positions, **common for all** sub-games. **Example**:



Game G_1 Game G_2 $(G_1 \otimes G_2)[\mathsf{MP}_{\mathsf{Reach}}(G_1 \otimes G_2)]$

Compositional LTL synthesis

- When a formula is given as a *conjunction* of subformulas i.e.,

  $\psi = \varphi_1 \wedge \varphi_2 \wedge \cdot\cdot\cdot \wedge \varphi_n$ the safety game associated with this formula can be defined *compositionally*.

- For each subformula φ_i the corresponding tbUKCW $A_{\varphi i}$ on the alphabet of ψ is constructed and also their associated safety games $G(\varphi_i, K)$.
  - The notion of product is used at the level of turn-based automata.
  - Executing the $A_1 \otimes A_2$ *o*n a word *w* is equivalent to execute both A_1 and A_2 on this word.
- The game *G(ψ,K)* for the conjunction *ψ* is *isomorphic* to the game composition.
- The game is then solved compositionally by first computing the local master plans to finally produce a compact (global) Moore machine, if it exists.

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
- Filiot, E., Jin, N. & Raskin, JF. Antichains and compositional algorithms for LTL synthesis, Form Methods Syst Des (2011) 39: 261. https://doi.org/10.1007/s10703-011-0115-3