A Game of Pawns

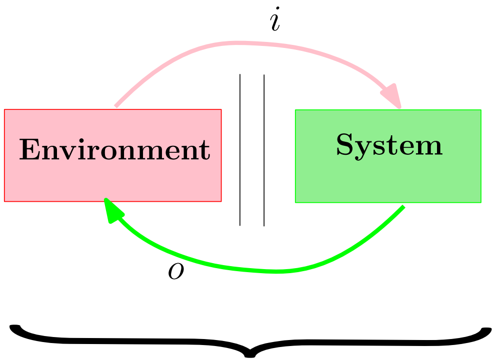
Shibashis Guha

Tata Institute of Fundamental Research

Joint work with Guy Avni and Pranav Ghorpade

November 1, 2022

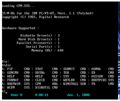
Reactive Systems

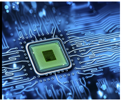


non-terminating interaction

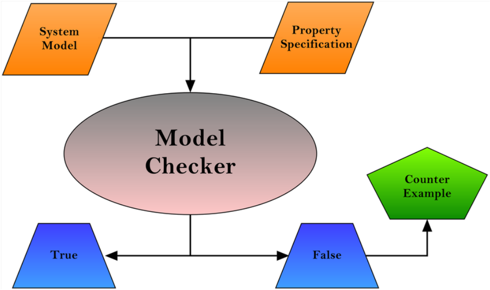
$\rho = (e_0, s_0) \xrightarrow{i_0, o_0} (e_1, s_1) \xrightarrow{i_1, o_1} (e_2, s_2) \ldots$ (infinite execution)

Reactive Systems





Verification: Model Checking

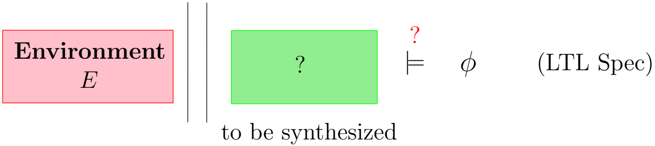


Verification

Environment E $\parallel$ **System** S $\overset{?}{\models}$ ϕ (LTL Spec)

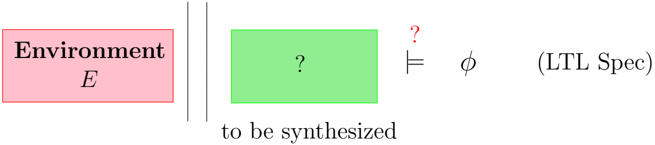
$L(E \times S) \subseteq L(A_\phi)$?

Synthesis



$L(E \times ?) \subseteq L(A_\phi)$?

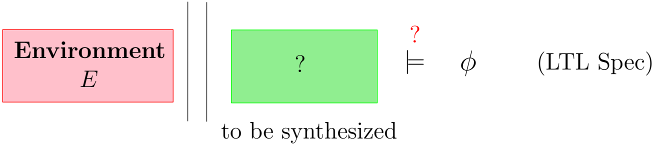
Synthesis



$L(E \times ?) \subseteq L(A_\phi)$?

- E and S are two players.
- Interaction: $\rho = (e_0, s_0) \xrightarrow{i_0/o_0} (e_1, s_1) \xrightarrow{i_1/o_1} (e_2, s_2) \ldots$
- Objective of S: ϕ, and Objective of E : $\neg\phi$ (adversarial environment)

Synthesis



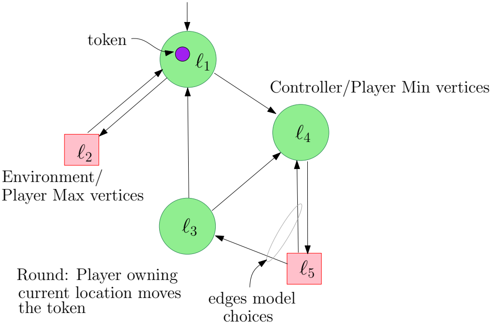
$L(E \times ?) \subseteq L(A_\phi)$?

- E and S are two players.
- Interaction: $\rho = (e_0, s_0) \xrightarrow{i_0/o_0} (e_1, s_1) \xrightarrow{i_1/o_1} (e_2, s_2) \ldots$
- Objective of S: ϕ, and Objective of E : $\neg\phi$ (adversarial environment)
- S wins if $\rho \in L(\phi)$, otherwise E wins.
- We want σ_S that wins against any σ_E

  Winning strategy = Correct system

Two-player reachability games

Mathematical model for controller synthesis.

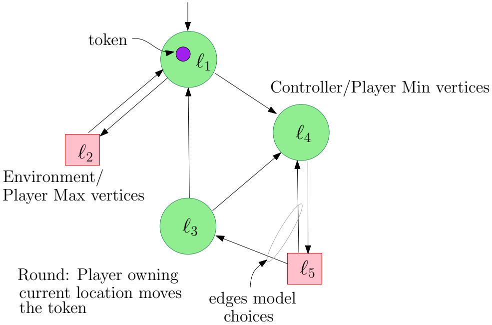


System / Player Min / **Player** 1

Environment / Player Max / **Player** 2

Two-player reachability games

Mathematical model for controller synthesis.



System / Player Min / **Player** 1

Environment / Player Max / **Player** 2

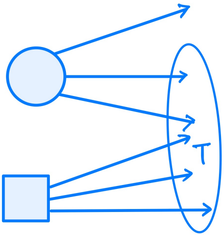
Reachability objective: Does Player 1 have a strategy to reach ℓ_3?

A strategy for a player from a vertex v that he owns is an edge/action chosen from v given a finite run ending in v.

Solving reachability games: Attractor computation

Controlled predecessor operator:

$\mathrm{CPre}(T) = \{v \in V_1 \mid v' \in T \text{ for some successor } v' \text{ of } v\} \cup \{v \in V_2 \mid v' \in T \text{ for all successors } v' \text{ of } v\}$.



Player 1 attractor: $\mathrm{Attr}_1(T)$ of T is defined inductively by applying the controlled predecessor operator as:

- $\mathrm{Attr}_1^0(T) = T$,
- $\mathrm{Attr}_1^{n+1}(T) = \mathrm{Attr}_1^n(T) \cup \mathrm{CPre}(\mathrm{Attr}_1^n)$, and
- $\mathrm{Attr}_1(T) = \bigcup_{n \in \mathbb{N}} \mathrm{Attr}_1^n(T)$.

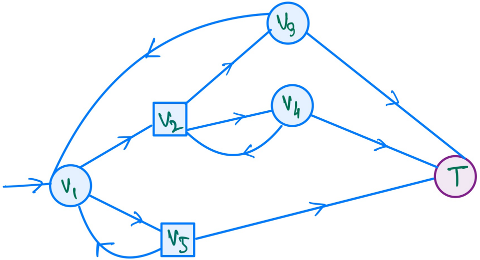
Pawn games

- Two players
- Finite set P of pawns; each *pawn owns some vertices*.
- Pawns are partitioned among players.
- Pawns *dynamically change hands* modelling dynamic change of resources.
- Pawns are entities controlling resources without having their own objectives.

Always-grabbing pawn games

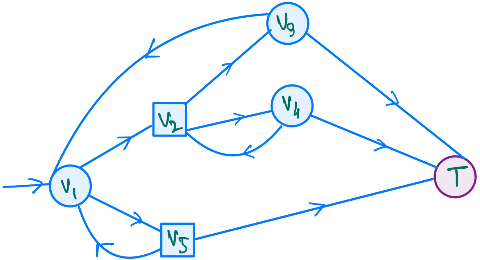
- Two players: Player 1 and Player 2
- Starts from an initial configuration: (v, W), a vertex v and a set W of pawns being controlled by Player 1 to start with.
- Rules of the game: How pawns change hands.

  If Player i makes a move, then the other player grabs a pawn.

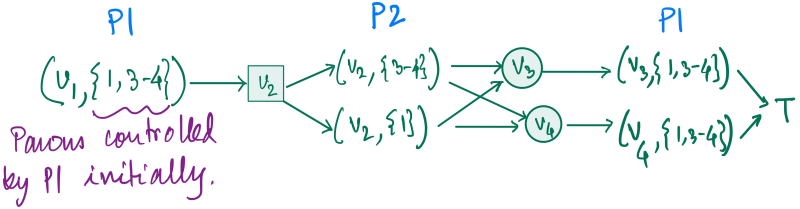


v_3, v_4 belong to the same pawn.

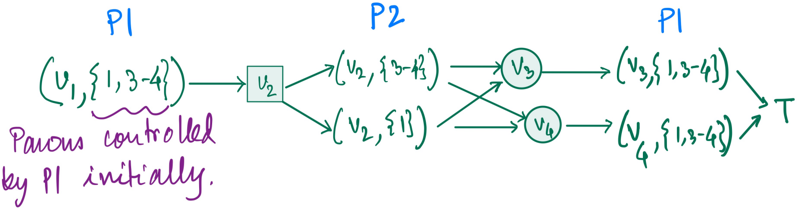
Always-grabbing pawn games



v_3, v_4 belong to the same pawn.

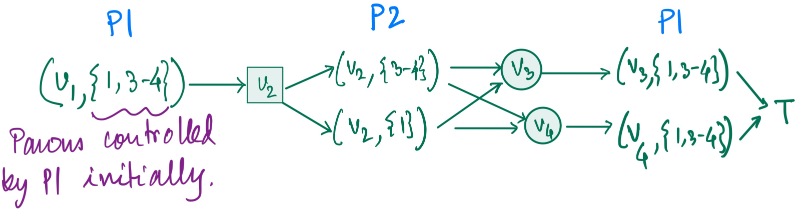


Pawn game: Configuration graph



- A configuration is of the form $\langle v, W \rangle$ for $v \in V$ and $W \subseteq P$.
- Size of configuration graph is exponential in the size of the input.

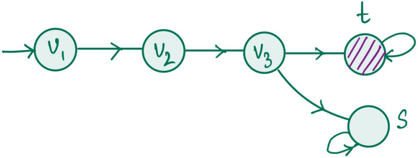
Pawn game: Configuration graph



- Size of configuration graph is exponential in the size of the input.
- Traditional reachability games can be solved in PTIME.

Pawn games with reachability objective can be solved in EXPTIME.

Non-monotonicity of pawn games



Pawn i owns vertex v_i for $i \in \{1,2,3\}$.

P1	P2	P1	
$\langle v_1, \{1\} \rangle$	$\langle v_2, \{\} \rangle$	$\langle v_3, \{3\} \rangle$	t
P1	P1	P2	
$\langle v_1, \{1,2\} \rangle$	$\langle v_2, \{2\} \rangle$	$\langle v_3, \{\} \rangle$	s

Pawn i owns vertex v_i for $i \in \{1, 2, 3\}$.

- Player i wins with with an initial set of pawns $\{1\}$ while loses with the initial set $\{1, 2\}$.

Grabbing mechanisms in pawn games

Player i's opponent is Player $3 - i$ (Player $-i$).

- **always grabbing**: Following a move of Player i, Player $-i$ always has to grab one of Player i's pawns.
- **always grabbing-or-giving**: Following a move of Player i, Player $-i$ always either has to grab one of Player i's pawns or give Player i one of his pawns.
- **optional grabbing**: Following a move of Player i, Player $-i$ has the option of grabbing one of Player i's pawns.
- **k-grabbing**: Following a move by one of the players, Player 1 has the option of grabbing a pawn from Player 2, and he can grab at most k pawns.

Ownership of vertices

- **one vertex per pawn**: Each V_i is a singleton; a pawn owns exactly one vertex.
- **multiple vertices per pawn**: $V_1, \ldots, V_d$ is a partition of V.
- **overlapping multiple vertices per pawn**: Each pawn may own multiple vertices, and each vertex may be owned by multiple pawns.

Problem Definition

Let $\alpha \in \{OVPP, MVPP, OMVPP\}$ and $\beta \in \{$*always*-grabbing, always grabbing-or-giving, optional-grabbing, *k*-grabbing $\}$.

> Given an input an $\alpha\beta$ pawn game $\mathcal{G}$ with target set T and an initial configuration c, decide if Player 1 has a strategy to reach T from c.

Pawn games: Applications

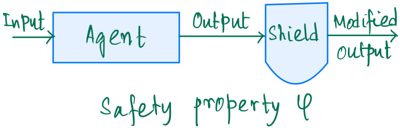
- For modelling *dynamic* resource contention in general.

Pawn games: Applications

- For modelling *dynamic* resource contention in general.

Shield synthesis

Shield synthesis (Könighofer et al. 2017): Consider an agent which is trained to do a specific task in an optimal manner but it violates the safety objective φ.



A shield *modifies the output so that safety is not violated*.

Consider the Kripke structure modelling the agent, at most k-pawns may be grabbed to model that the shield can change the action of the agent at most k times.

Pawn games: Applications

Sabotage games

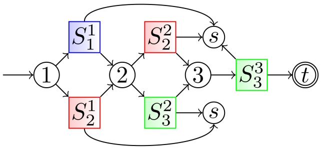
Sabotage games (Löding, Rohde. 2003). Two-player game on a graph in which a *saboteur crashes an edge* in the graph with the goal of preventing Player 1 reaching its target.

Think of Player 2 to grab and crash an edge.

k-grabbing mechanism

NP-hardness of MVPP

Checking existence of winning strategy in MVPP k-grabbing game is NP-hard.



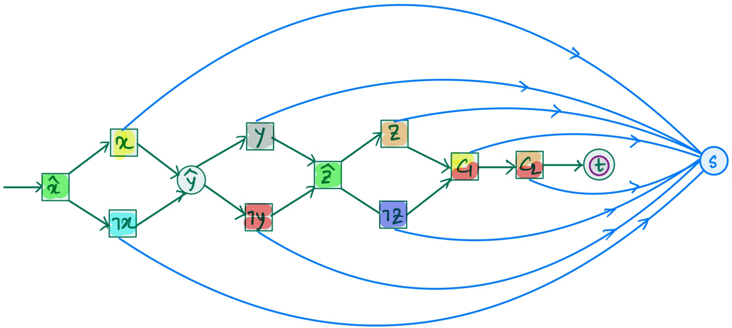
$U = \{1, 2, 3\}$, and $\mathcal{S} = \{\{1\}, \{1, 2\}, \{2, 3\}\}$

Each neighbour of $i \in U$ corresponds to a set $S \in \mathcal{S}$ such that $i \in S$.

k-grabbing mechanism

PSPACE-hardness of OMVPP

Reduction from TQBF: $\forall x \exists y \forall z (x \vee \neg y) \wedge (\neg y \vee z)$.



For every variable x, there is a vertex $\widehat{x}$ with neighbours x_i and $\neg x_i$.

Initial configuration: Player 1 controls only pawn p_1.

Player 2 has a winning strategy with n grabs iff the TQBF formula is satisfiable.

k-grabbing mechanism

PSPACE-membership of OMVPP

If Player 1 has a winning strategy, then he has one which ends in $n \cdot (k+1)$ rounds, where n is the number of vertices in the pawn game.

Every n rounds, Player 1 must grab a pawn; otherwise, there exists a *cycle in the configuration graph* that Player 2 can enforce and T is not reached.

Consider a *game tree* obtained by *unwinding the configuration graph* $n \cdot (k+1)$ times.

PSPACE-membership as we only need to *store a branch* of the tree; branch can be of length at most $n \cdot (k+1)$.

k-grabbing mechanism

- MVPP is NP-hard.
- OMVPP is PSPACE-complete.
- OVPP is in PTIME.

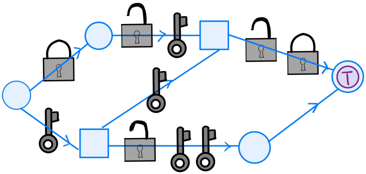
Optional-grabbing mechanism

Following a move of Player i, Player $-i$ has the option of grabbing one of Player i's pawns.

Lock & key game

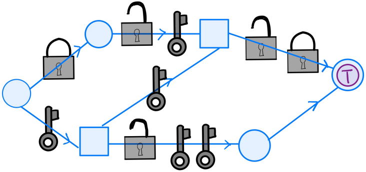
- Each edge has some locks and some keys.
- $L = \ell_1, \ldots, \ell_n$, and $K = k_1, \ldots, k_n$.
- Complexity of optional-grabbing using Lock & key game.

Lock & Key game



- Each edge has some locks and some keys.
- $L = \ell_1, \ldots, \ell_n$, and $K = k_1, \ldots, k_n$.
- An edge with *only open locks* can be crossed.
- While crossing an edge labelled with a *key changes the state of the corresponding lock*.
- A *configuration* is of the form $\langle v, A \rangle$, where $A \subseteq 2^L$ leading to EXPTIME-membership.
- EXPTIME-hardness follows from a reduction from APSPACE-TM.

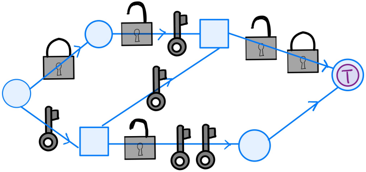
Lock & Key game to MVPP optional-grabbing game



- For each lock and each key, we have a gadget in the MVPP optional-grab game.

MVPP optional-grabbing game is EXPTIME-complete.

Lock & Key game to MVPP optional-grabbing game



- For each lock and each key, we have a gadget in the MVPP optional-grab game.

MVPP optional-grabbing game is $\mathrm{EXPTIME}$-complete.

OVPP optional-grabbing game is in PTIME.

Always-grabbing game

Following a move of Player i, Player $-i$ always has to grab one of Player i's pawns.

MVPP always-grabbing game is $\mathrm{EXPTIME}$-complete.

- Reduce an *instance of optional-grabbing game* obtained from $\mathrm{APSPACE}$-TM to an instance of always-grabbing game.
- We add some *isolated vertices* and one-vertex pawns owning them.
- The action of not grabbing in the optional-grabbing game can be replaced with *grabbing a pawn owning an isolated vertex* in the always-grabbing game.
- Consider Player 1 has a winning strategy in the optional-grabbing game. The challenge is to *ensure that there will be enough isolated pawns with Player* 2 *for Player* 1 *to grab*.

Always-grabbing-or-giving game

Every time Player i moves the token from vertex some v to some vertex u, it entirely depends on Player $-i$ to decide whether he wants to control u or not.

- If Player $-i$ does not have the pawn p_u that owns u and he wants to control u, he can grab p_u from Player i.
- If he does not want to control u and if he has p_u, he can give it to Player i.

Configuration graph has *two copies of each vertex*: One controlled by Player 1, and the other controlled by Player 2.

MVPP always-grab-or-give game is in PTIME.

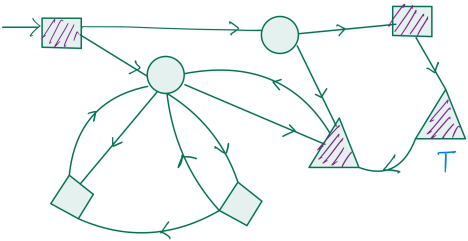
Discussions: Alternating-time temporal logic. (Alur, Henzinger, Kupferman. 2002)

- **Extends computational tree logic to multiple players**:

  for example, it allows specifications of the form $\langle\langle\{a, b\}\rangle\rangle(Fp \wedge Gq)$.

- Players are grouped as protagonist and antagonist, and becomes a two player game.

- For synchronous turn-based game, vertices are partitioned among the players.



Conclusion: Pawn Games

- Class of two-player turn-based zero-sum games in which *control of vertices changes dynamically*.
- Constitute succinctly represented turn-based games.
- Complexity results from PTIME to $\mathrm{EXPTIME}$-complete.
- We considered reachability games: Other ω-regular objectives, mean-payoff etc.
- Concurrent, stochastic games ...

Conclusion: Pawn Games

- Class of two-player turn-based zero-sum games in which *control of vertices changes dynamically*.
- Constitute succinctly represented turn-based games.
- Complexity results from PTIME to $\mathrm{EXPTIME}$-complete.
- We considered reachability games: Other ω-regular objectives, mean-payoff etc.
- Concurrent, stochastic games ...

Thank you for your attention!