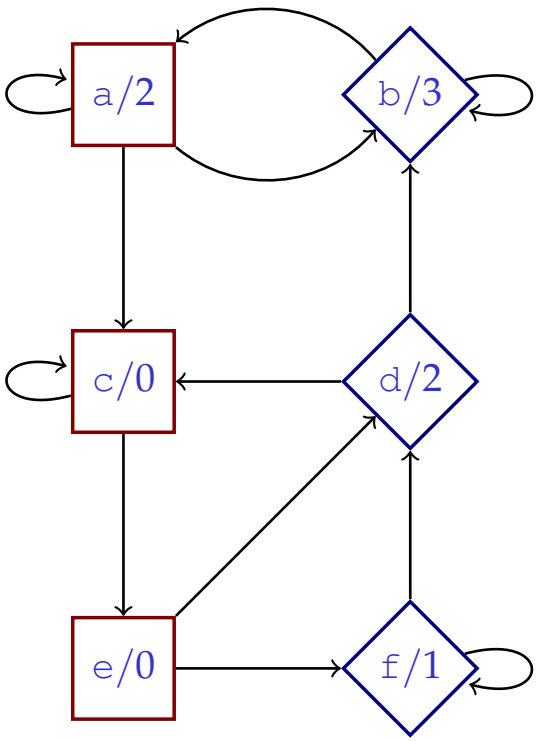
Solving Infinite Games on Graphs via Quasi-Dominions

Fabio Mogavero
(joint work with Massimo Benerecetti and Daniele Dell'Erba)

Università degli Studi di Napoli Federico II

Udine, January 22, 2020

Games on graphs



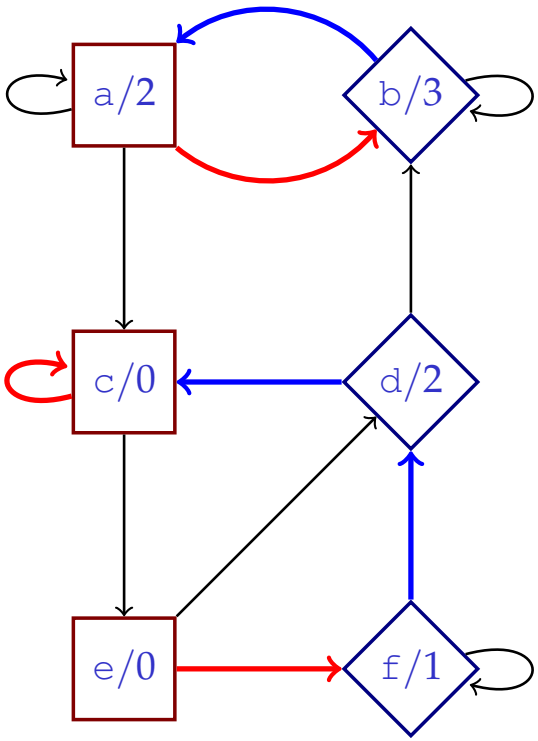
An Arena is a directed graph

- two players (player ♦ and player ■);
- each position (node) belongs to one of the two players (♦-positions and ■-positions) ;
- nodes are labeled with numerical values.

Turn-based infinite-duration game

- each player $\alpha \in \{♦, ■\}$ chooses an edge (move) at its positions;
- players choices induce plays: infinite paths in the arena;
- a play is winning for a player if it satisfies some winning condition;
- players have opposite winning conditions (zero-sum).

Parity Games



Parity Game

- player ♦ called Even, player ■ called Odd.
- positions are labeled with natural numbers called priorities

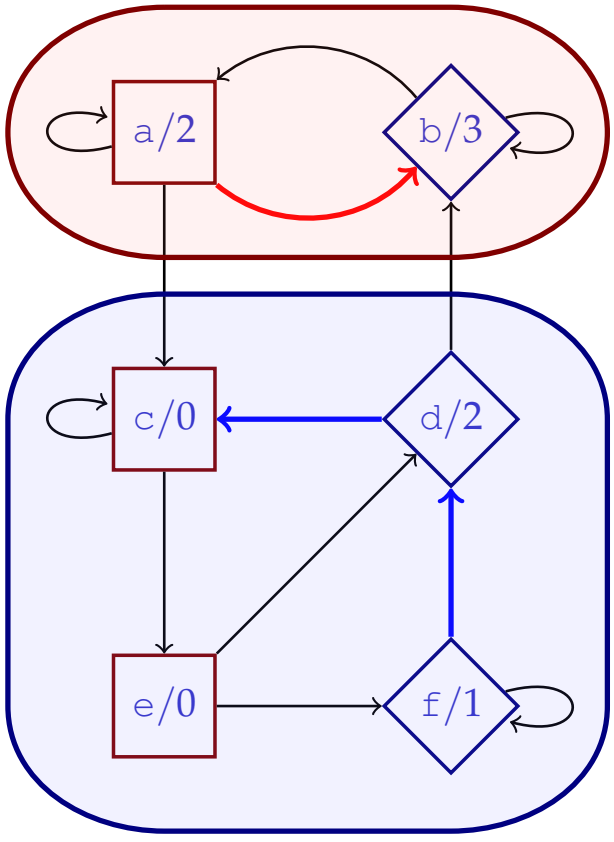
Winning Condition for $\alpha \in \{\blacklozenge, \blacksquare\}$

the highest priority occurring infinitely often along the play is of parity α.

Examples

- the maximal recurring priority in play $(\mathtt{ab})^\omega$ is 3;
- the maximal recurring priority in play $\mathtt{efd(c)}^\omega$ is 0.

Strategies and Determinacy



Memoryless strategy for player $\alpha \in \{\blacklozenge, \blacksquare\}$

function σ_α from α-positions to arbitrary positions.

Winning position for $\alpha \in \{\blacklozenge, \blacksquare\}$

v is winning for α if it has a strategy σ_α *s.t.*, regardless of the strategy of $\overline{\alpha}$, the induced plays from v are all winning for α.

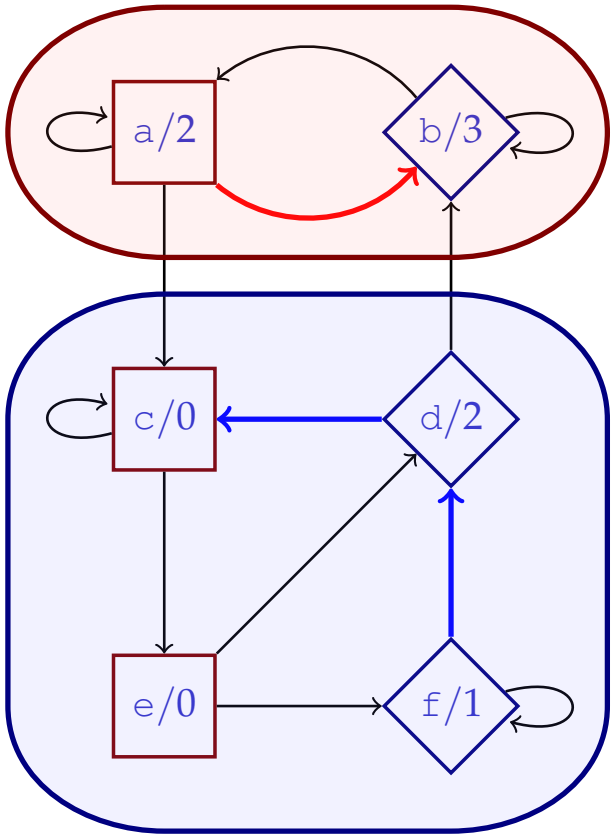
Winning Region for $\alpha \in \{\blacklozenge, \blacksquare\}$

maximal set Wn_α of winning positions for α.

The Decision Problem

Identify the winning regions of the players.

Strategies and Determinacy



Memoryless strategy for player $\alpha \in \{\blacklozenge, \blacksquare\}$

function σ_α from α-positions to arbitrary positions.

Winning position for $\alpha \in \{\blacklozenge, \blacksquare\}$

v is winning for α if it has a strategy σ_α *s.t.*, regardless of the strategy of $\overline{\alpha}$, the induced plays from v are all winning for α.

Winning Region for $\alpha \in \{\blacklozenge, \blacksquare\}$

maximal set Wn_α of winning positions for α.

The Decision Problem

Identify the winning regions of the players.

Determinacy

there is a *bipartition* $(\mathrm{Wn}_\blacklozenge, \mathrm{Wn}_\blacksquare)$ of the positions *s.t.*:

- player $\blacklozenge$ has a winning strategy on $\mathrm{Wn}_\blacklozenge$;
- player $\blacksquare$ has a winning strategy on $\mathrm{Wn}_\blacksquare$.

Relevance of Parity Games

Applications

- modal μCalculus model checking (linear-time interreducible)
- reactive synthesis from LTL specifications
- emptiness and membership problems for alternating tree automata;
- automata theory, *e.g.*, complementation and determinisation problems;
- ...

Complexity Status

Parity Games are *memoryless determined*, *i.e.*,
- there are *no draws*
- *no memory* is needed to win

Parity Games are in NPTime ∩ CoNPTime.

Open Problem

Is the *solution problem* in PTime? It is known to be PTime-hard!

Known Upper Bounds

- Quasi-PTime upper bound: $n^{O(\log k)}$ [Calude et al., 2017].

State of the Art

Some solution algorithms for Parity Games

Solution Algorithm	Time Complexity	Space Complexity
Recursive	$O(m \cdot n^k)$	$O(m \cdot n)$
Dominion Decomposition	$n^{O(\sqrt{n})}$	$O(m \cdot n)$
Big Step	$O(m \cdot n^{k/3})$	$O(n \cdot m + k \cdot \log n))$
Priority Promotion	$O\left(m \cdot \left(3\frac{n-2}{k-2}\right)^{k-1}\right)$	$O(n \cdot \log k)$
Small Progress Measure	$O\left(m \cdot k \cdot \left(\frac{n}{k}\right)^{k/2}\right)$	$O(k \cdot n \cdot \log n)$
Strategy Improvement	$O(m \cdot n \cdot 2^m)$	$O(n^2)$
Calude *et al.* QP	$O(n^{\log k+6})$	$O(n^{\log k+5})$
Jurdzinski & Lazic QP	$O(m \cdot n^{\log k-\log\log n+4.03})$	$O(n \cdot \log n \cdot \log k)$
Fearnley *et al.* QP	$O\left(m \cdot \frac{k}{\log n} \cdot n^{\log k-\log\log n+1.45}\right)$	$O(n \cdot \log n \cdot \log k)$

n number of positions m number of moves k number of priorities

Priority Promotion: Solving Parity Games with Quasi-Dominions

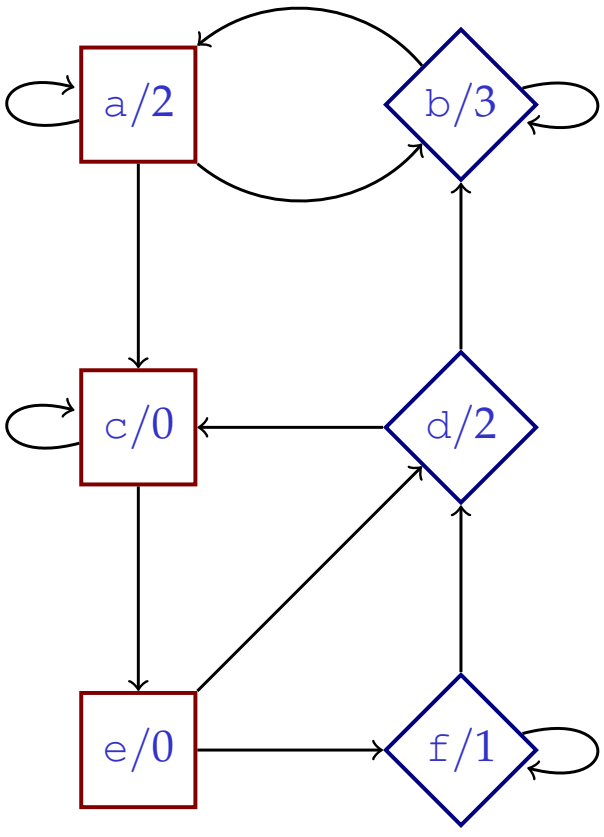
A novel approach based on the new notions of

- *quasi-dominion*
- *priority promotion*

- best proven asymptotic space complexity: $O(n \cdot \log k)$;
- outperforms the other solvers of at least *one order of magnitude*.

Priority Promotion Approach

Predecessor & Attractor

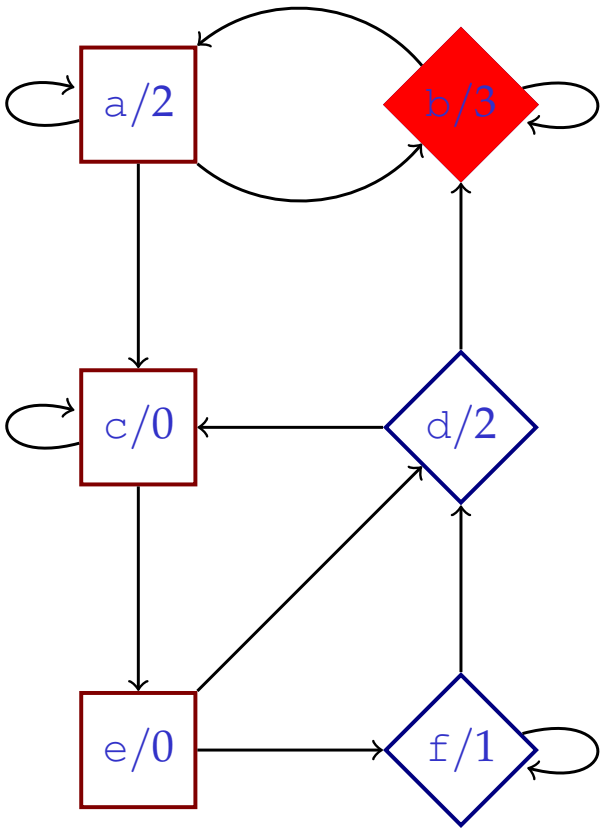


Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathsf{pre}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Predecessor & Attractor

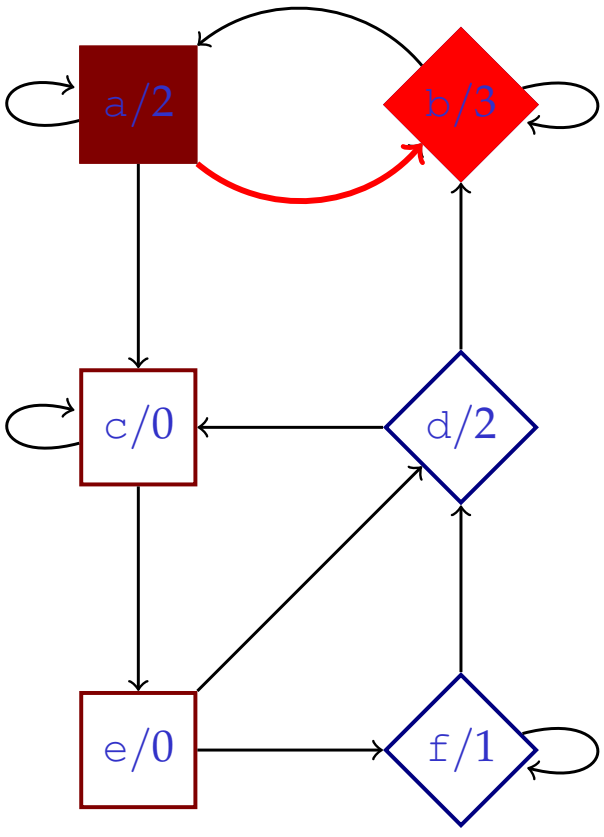


Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathsf{pre}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Predecessor & Attractor

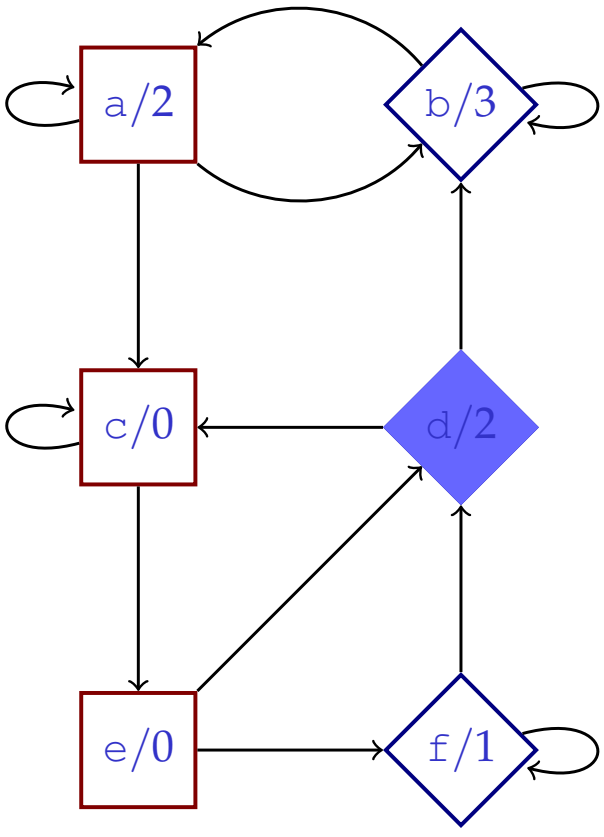


Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathsf{pre}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Predecessor & Attractor

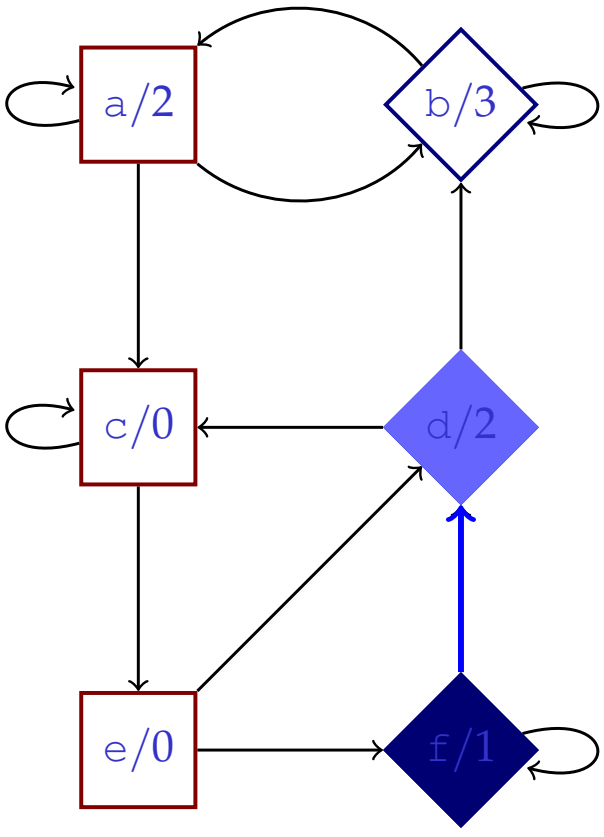


Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathsf{pre}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Predecessor & Attractor

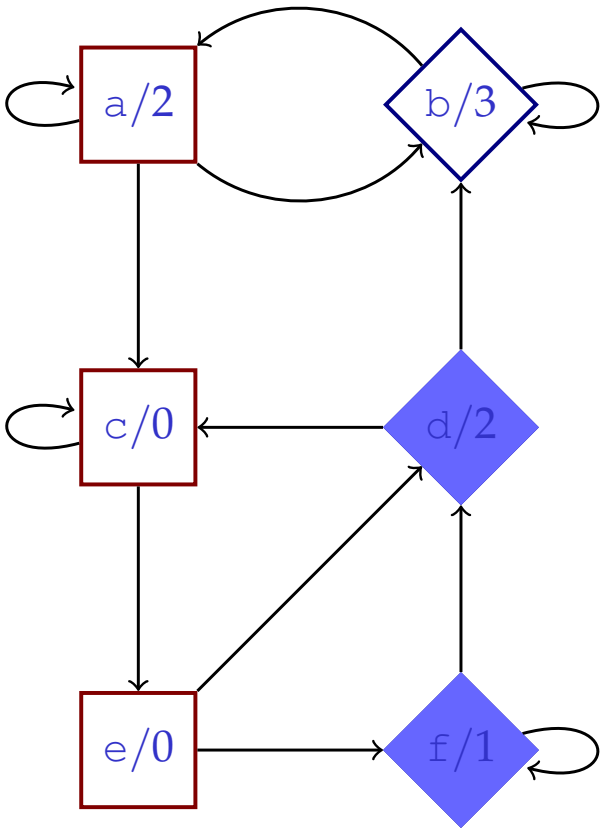


Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathrm{pre}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Predecessor & Attractor

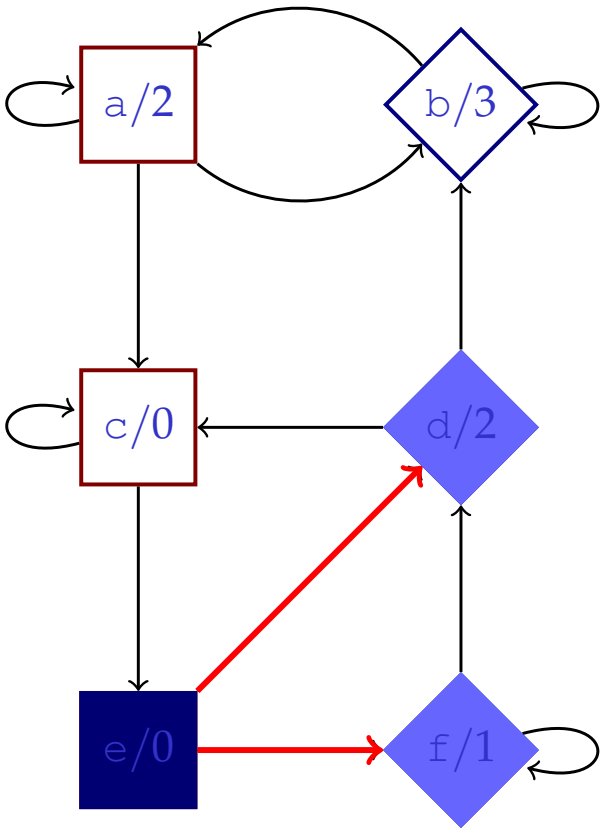


Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathsf{pre}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Predecessor & Attractor

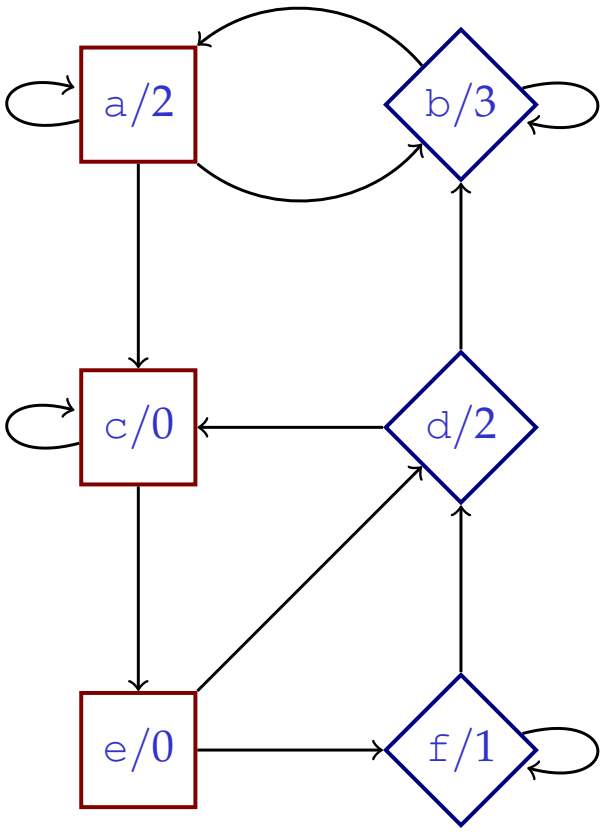


Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathsf{pre}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Predecessor & Attractor



Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathrm{pre}^{\alpha}(X)$ contains:

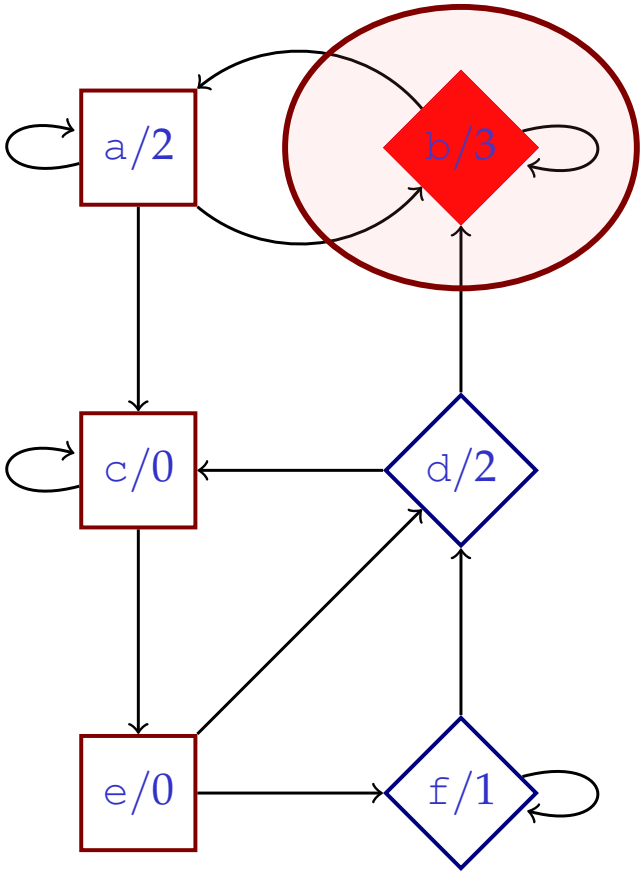
- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Attractor for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathrm{atr}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in an arbitrary number of moves;
- the $\overline{\alpha}$-positions that are forced to *reach* X in an arbitrary number of moves.

Predecessor & Attractor



Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathsf{pre}^{\alpha}(X)$ contains:

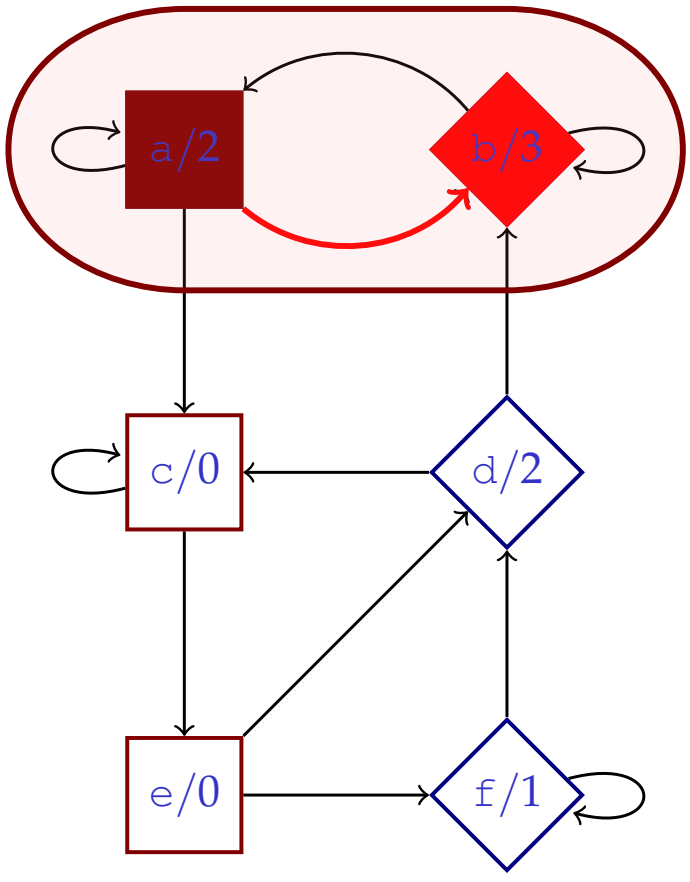
- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Attractor for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathsf{atr}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in an arbitrary number of moves;
- the $\overline{\alpha}$-positions that are forced to *reach* X in an arbitrary number of moves.

Predecessor & Attractor



Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathrm{pre}^{\alpha}(X)$ contains:

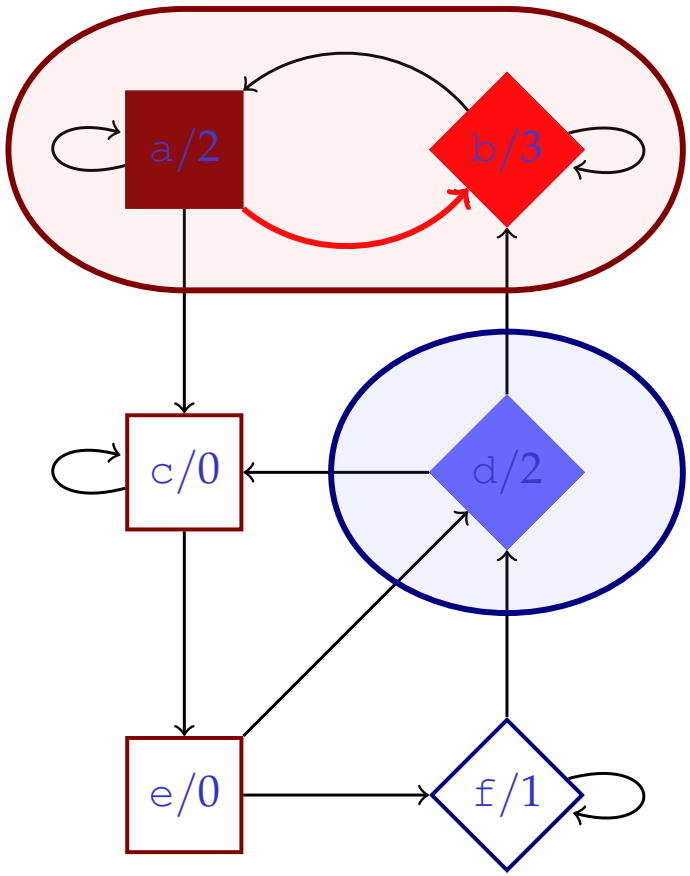
- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Attractor for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathrm{atr}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in an arbitrary number of moves;
- the $\overline{\alpha}$-positions that are forced to *reach* X in an arbitrary number of moves.

Predecessor & Attractor



Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathsf{pre}^{\alpha}(X)$ contains:

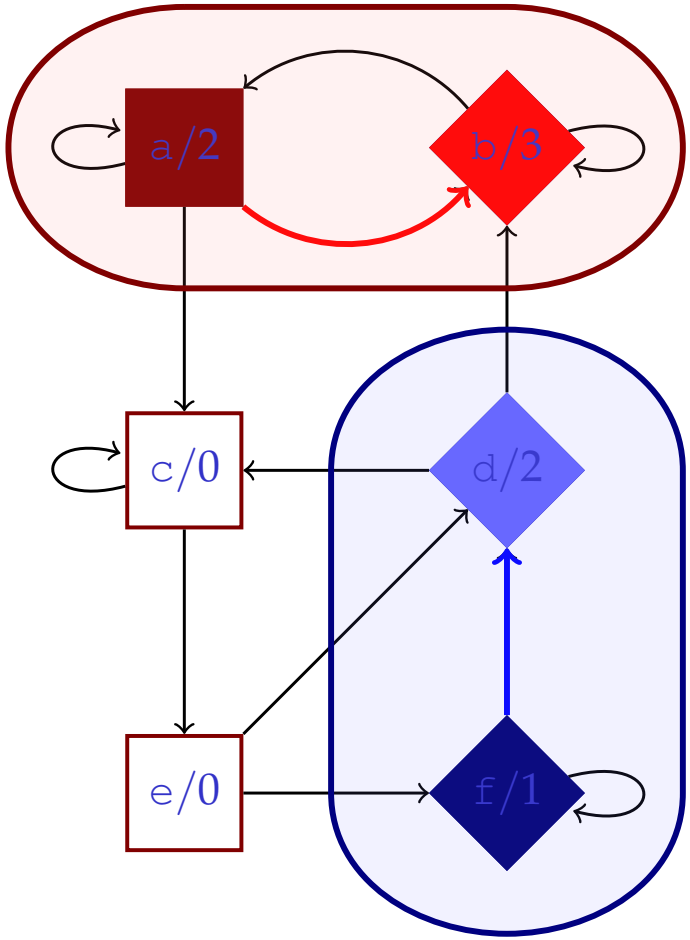
- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Attractor for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathsf{atr}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in an arbitrary number of moves;
- the $\overline{\alpha}$-positions that are forced to *reach* X in an arbitrary number of moves.

Predecessor & Attractor



Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathrm{pre}^{\alpha}(X)$ contains:

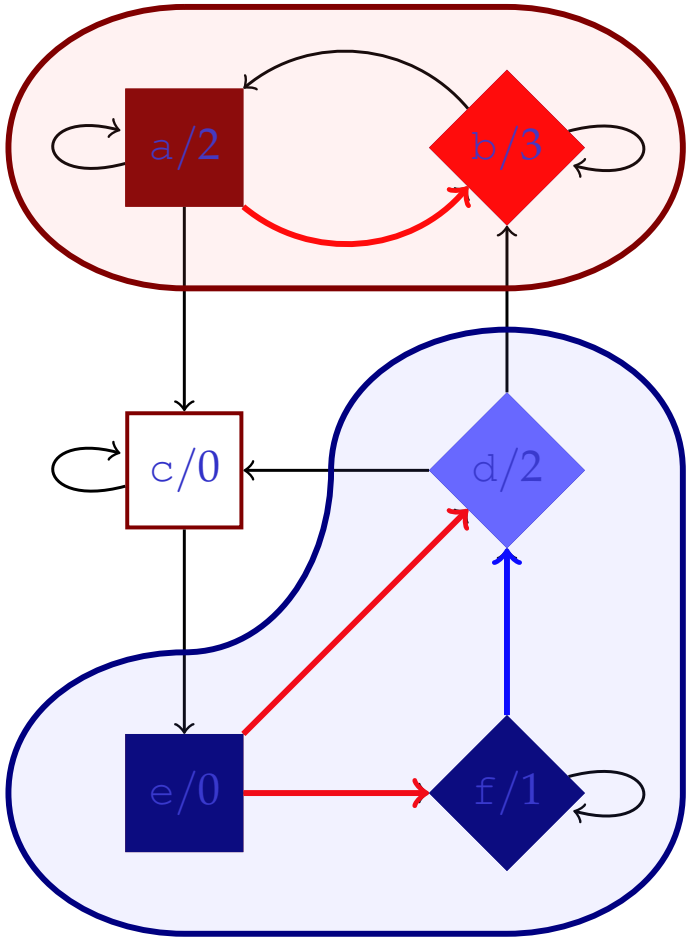
- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Attractor for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathrm{atr}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in an arbitrary number of moves;
- the $\overline{\alpha}$-positions that are forced to *reach* X in an arbitrary number of moves.

Predecessor & Attractor



Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathrm{pre}^{\alpha}(X)$ contains:

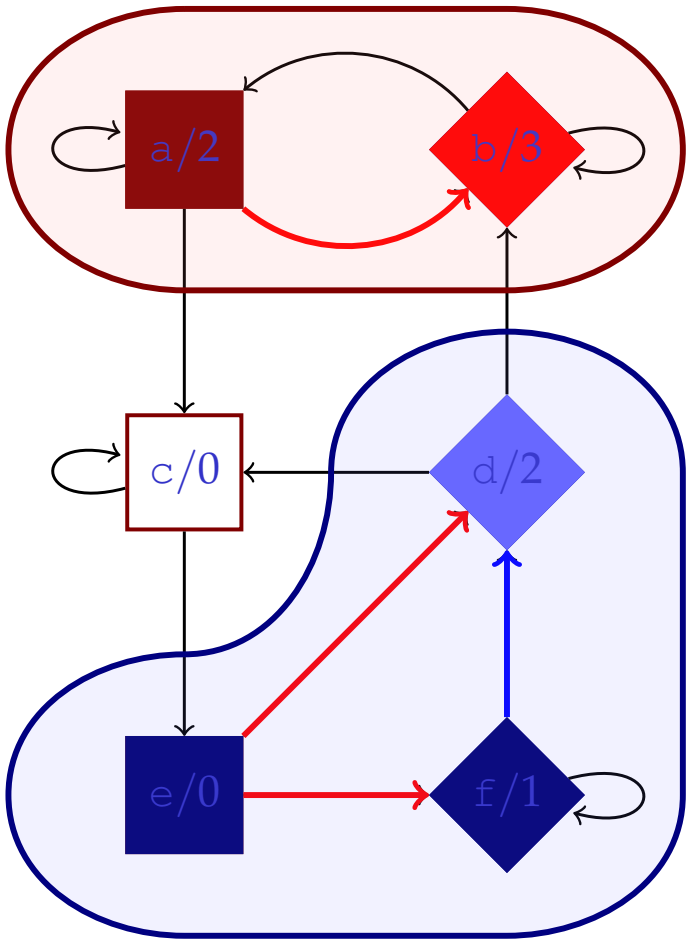
- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Attractor for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathrm{atr}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in an arbitrary number of moves;
- the $\overline{\alpha}$-positions that are forced to *reach* X in an arbitrary number of moves.

Predecessor & Attractor



Predecessors for $\alpha \in \{\blacklozenge, \blacksquare\}$

$\mathsf{pre}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in one move;
- the $\overline{\alpha}$-positions that are forced to *reach* X in one move.

Attractor for $\alpha \in \{\blacklozenge, \blacksquare\}$

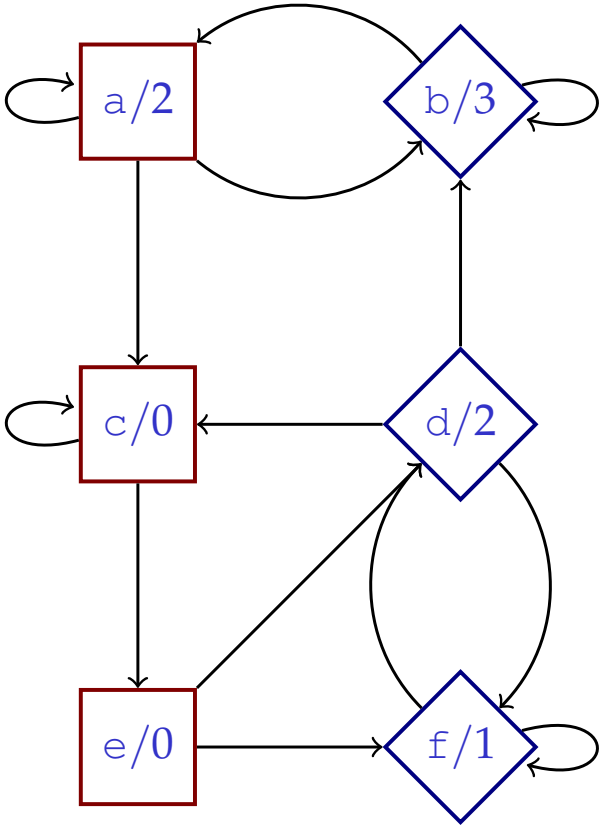
$\mathsf{atr}^{\alpha}(X)$ contains:

- the α-positions that can *reach* X in an arbitrary number of moves;
- the $\overline{\alpha}$-positions that are forced to *reach* X in an arbitrary number of moves.

Properties of $\mathsf{atr}^{\alpha}(X)$

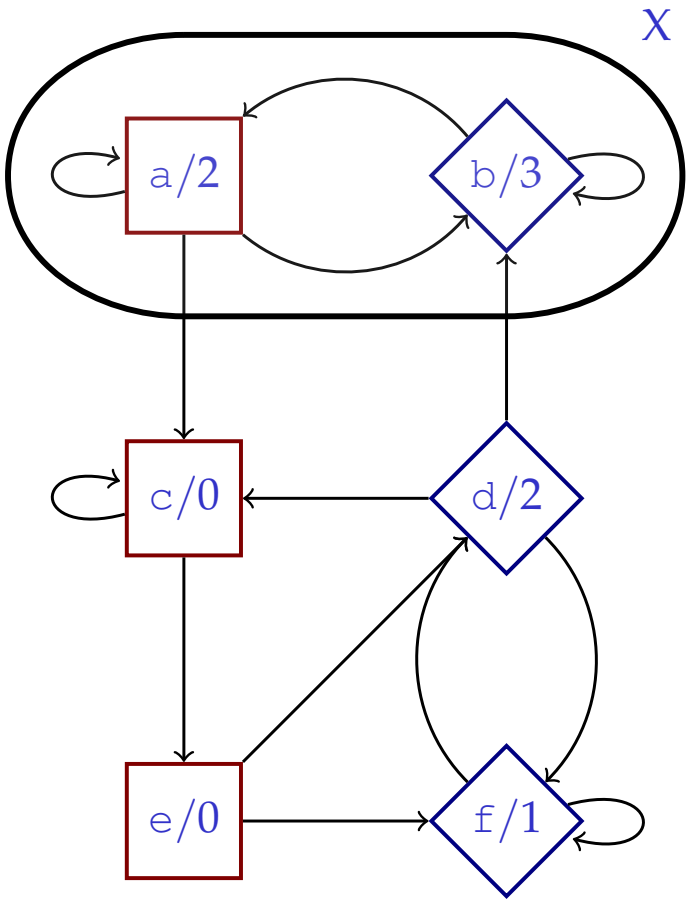
- An α-position not belonging to $\mathsf{atr}^{\alpha}(X)$ cannot have a move leading to it.
- Player α has a strategy to force any play that remains in $\mathsf{atr}^{\alpha}(X)$ forever to visit a position in X infinitely often.

Subgame



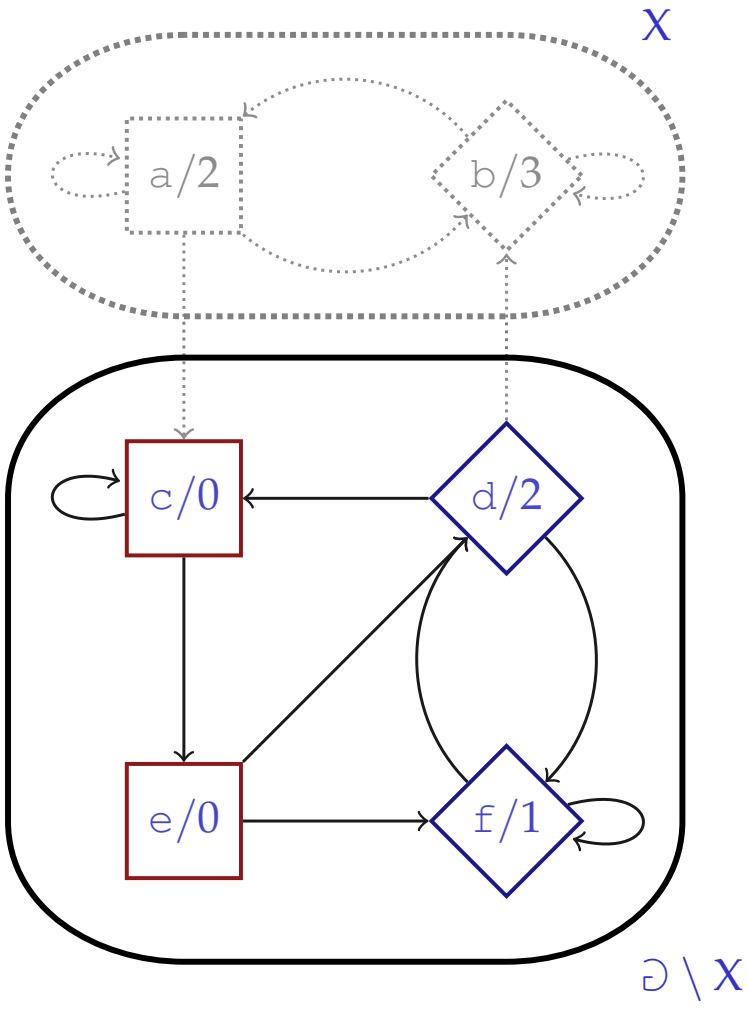
For a game $\mathcal{G}$ and a set of position X, $\mathcal{G} \setminus X$ denotes the subgame obtained by removing X and all the moves from and to X.

Subgame



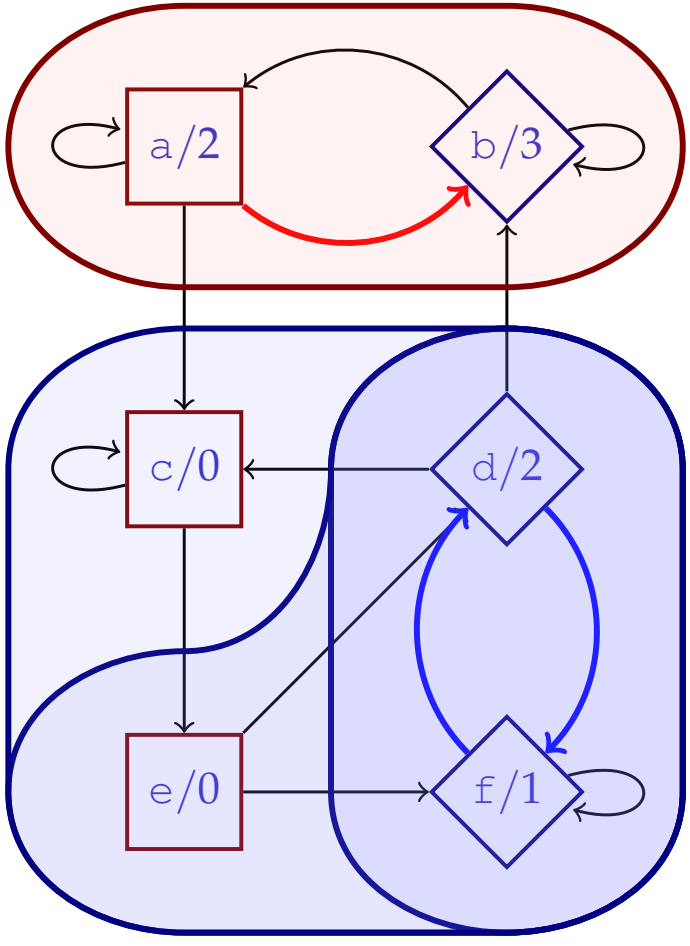
For a game $\mathfrak{G}$ and a set of position X, $\mathfrak{G} \setminus X$ denotes the subgame obtained by removing X and all the moves from and to X.

Subgame



For a game $\ni$ and a set of position X, $\ni \setminus X$ denotes the subgame obtained by removing X and all the moves from and to X.

Dominions

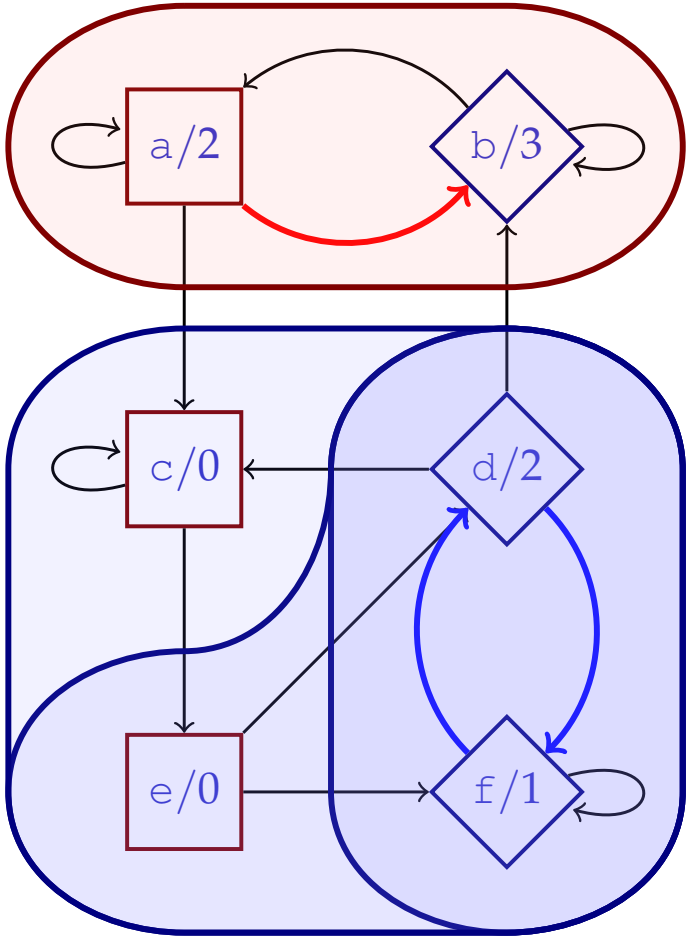


Dominion for $\alpha \in \{\blacklozenge, \blacksquare\}$

non-empty set of positions D *s.t.*

- α has a winning strategy on D
- $\overline{\alpha}$ cannot escape from D

Dominions



Dominion for $\alpha \in \{\blacklozenge, \blacksquare\}$

non-empty set of positions D *s.t.*

- α has a winning strategy on D
- $\overline{\alpha}$ cannot escape from D

Property

Let D an α-dominion and $\hat{D} \triangleq \mathsf{atr}^{\alpha}(D)$, then $\hat{D}$ is an α-dominion.

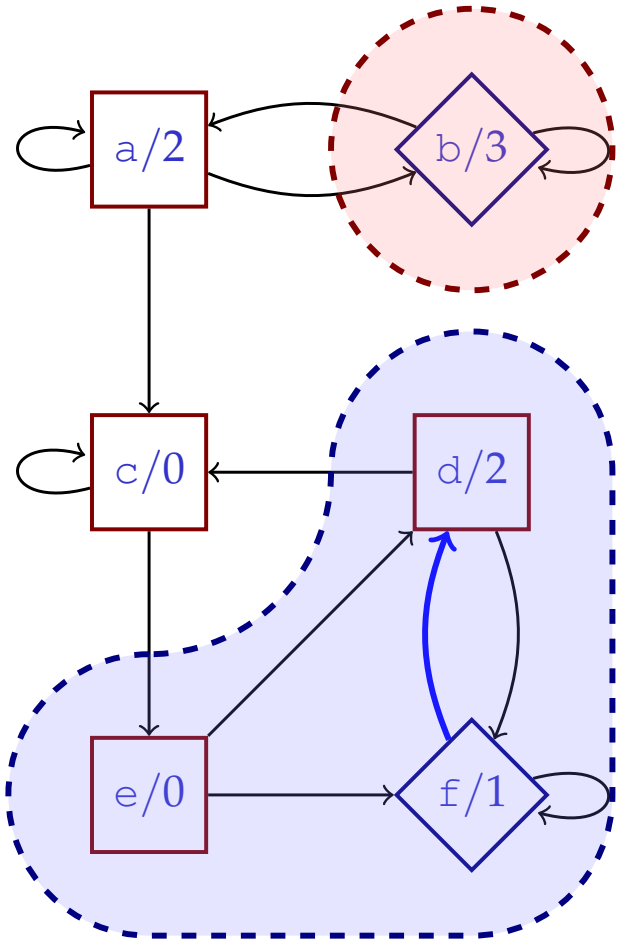
Solution via Dominions

Ⅰ $(\mathrm{Wn}_{\blacklozenge}, \mathrm{Wn}_{\blacksquare}) \leftarrow (\emptyset, \emptyset)$

Ⅱ WHILE ∂ is non-empty DO

1. find an α-dominion D for some $\alpha \in \{\blacklozenge, \blacksquare\}$
2. $\hat{D} \leftarrow \mathrm{atr}^{\alpha}(D)$ (compute the α-attractor of D)
3. $\mathrm{Wn}_{\alpha} \leftarrow \mathrm{Wn}_{\alpha} \cup \hat{D}$ (compose the results)
4. $\partial \leftarrow \partial \setminus \hat{D}$ (reduce the game)

Quasi Dominions



Quasi Dominion for $\alpha \in \{\blacklozenge, \blacksquare\}$

non-empty set of positions Q on which α has a strategy σ_α whose induced plays

- either remain in Q forever and are winning for α
- or are forced to exit from Q by the opponent $\overline{\alpha}$

Property

a quasi α-dominion Q from which $\overline{\alpha}$ cannot escape is an α-dominion

Dominion Problem

Problem

find an α-dominion for some player $\alpha \in \{\blacklozenge, \blacksquare\}$

Observations

dominions are hard to find

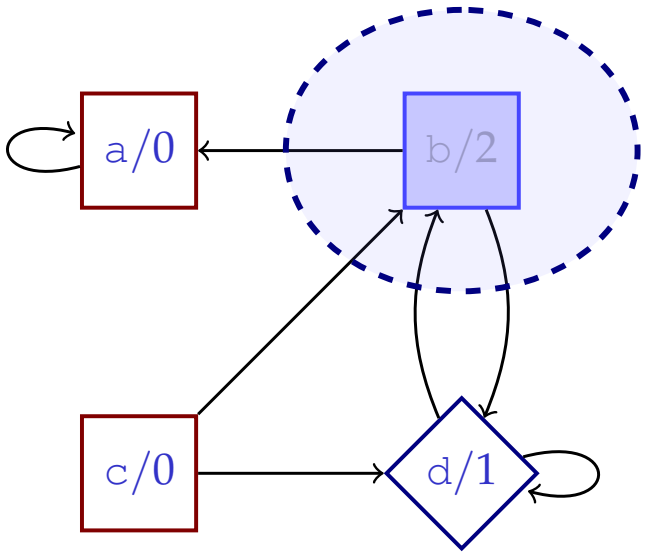
quasi dominions are easy to find

High-Level Idea

compute quasi dominions in the game and suitably *compose* quasi dominions of the same player until a dominion is eventually obtained:

- extract a quasi α-dominion Q for some player $\alpha \in \{\blacklozenge, \blacksquare\}$
- check if Q is an α-dominion in the entire game
- if not, try to merge Q with a previously computed quasi α-dominion and repeat

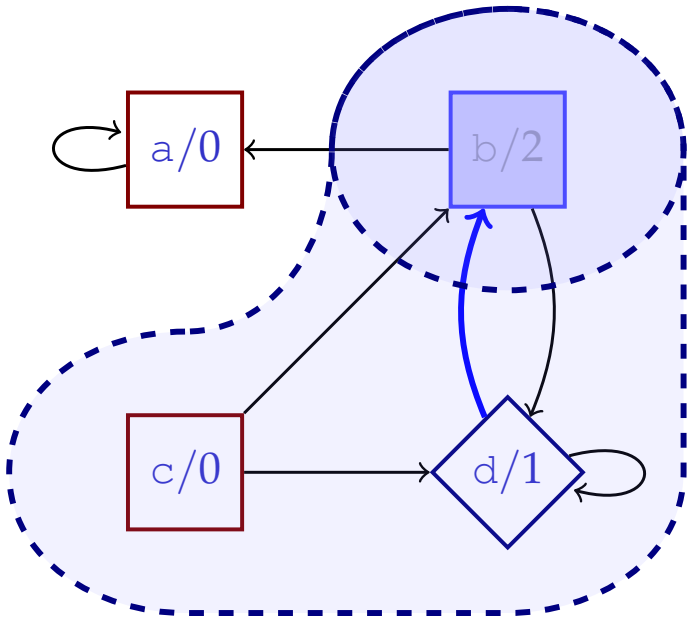
Quasi Dominion Construction



Quasi α-Dominion Computation

the set P of position with maximal priority of parity α is a quasi α-dominion

Quasi Dominion Construction



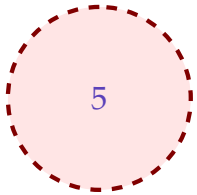
Quasi α-Dominion Computation

the set P of position with maximal priority of parity α is a quasi α-dominion

Quasi α-Dominion Maximization

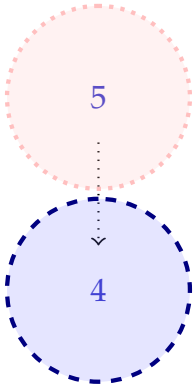
the α-attractor $R = \mathsf{atr}^{\alpha}(P)$ of P is a quasi α-dominion.

High-Level Simulation I



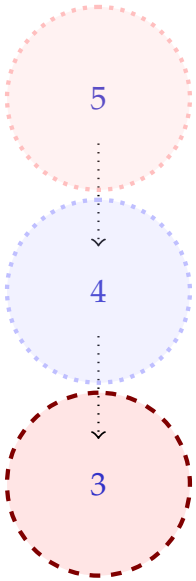
- Proceed in decreasing order of priorities
- Maximize the set of positions with maximal priority in the current subgame.
- Remove the quasi dominion obtained from the current subgame.

High-Level Simulation I



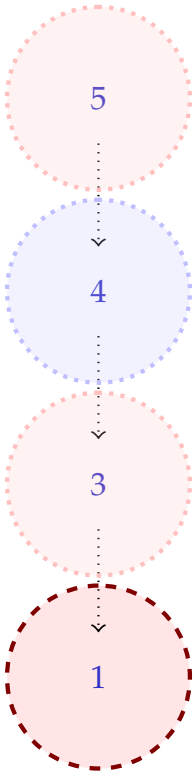
- Proceed in decreasing order of priorities
- Maximize the set of positions with maximal priority in the current subgame.
- Remove the quasi dominion obtained from the current subgame.

High-Level Simulation I



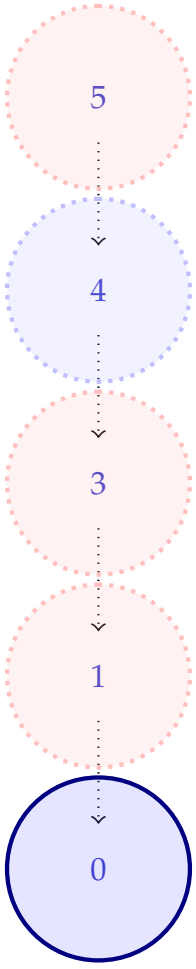
- Proceed in decreasing order of priorities
- Maximize the set of positions with maximal priority in the current subgame.
- Remove the quasi dominion obtained from the current subgame.

High-Level Simulation I



- Proceed in decreasing order of priorities
- Maximize the set of positions with maximal priority in the current subgame.
- Remove the quasi dominion obtained from the current subgame.

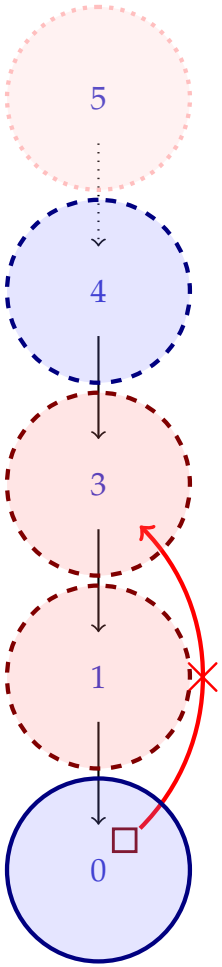
High-Level Simulation I



- Proceed in decreasing order of priorities
- Maximize the set of positions with maximal priority in the current subgame.
- Remove the quasi dominion obtained from the current subgame.

Clearly, a local α-dominion will be found in some subgame, eventually.

High-Level Simulation I

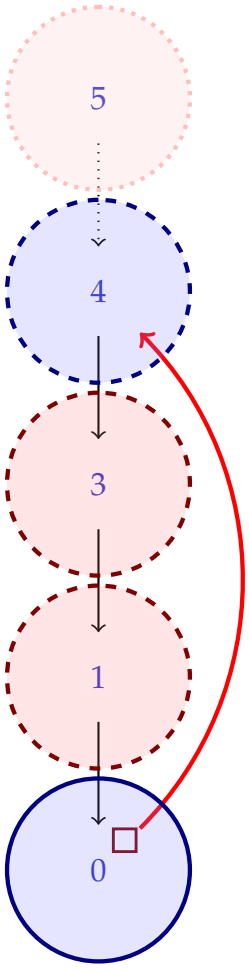


- Proceed in decreasing order of priorities
- Maximize the set of positions with maximal priority in the current subgame.
- Remove the quasi dominion obtained from the current subgame.

Clearly, a local α-dominion will be found in some subgame, eventually.

The opponent $\overline{\alpha}$ can only escape from this α-dominion by moving to a quasi α-dominion of higher priority.

High-Level Simulation I

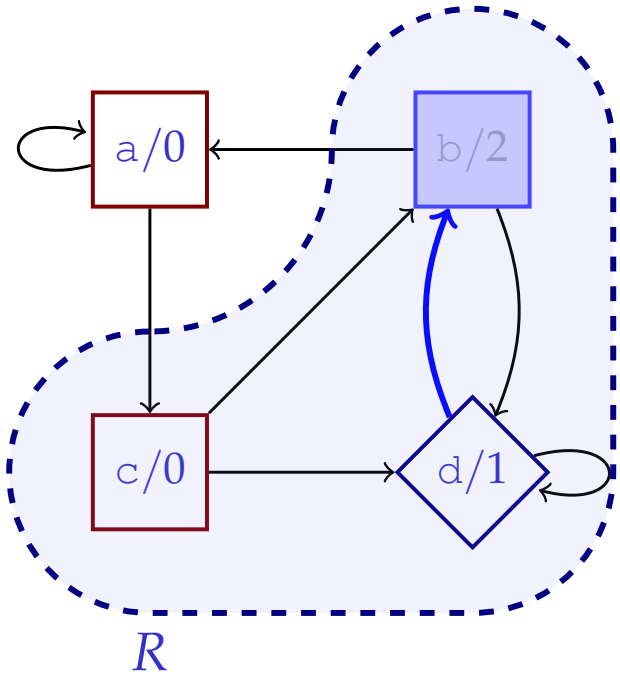


- Proceed in decreasing order of priorities
- Maximize the set of positions with maximal priority in the current subgame.
- Remove the quasi dominion obtained from the current subgame.

Clearly, a local α-dominion will be found in some subgame, eventually.

The opponent $\overline{\alpha}$ can only escape from this α-dominion by moving to a quasi α-dominion of higher priority.

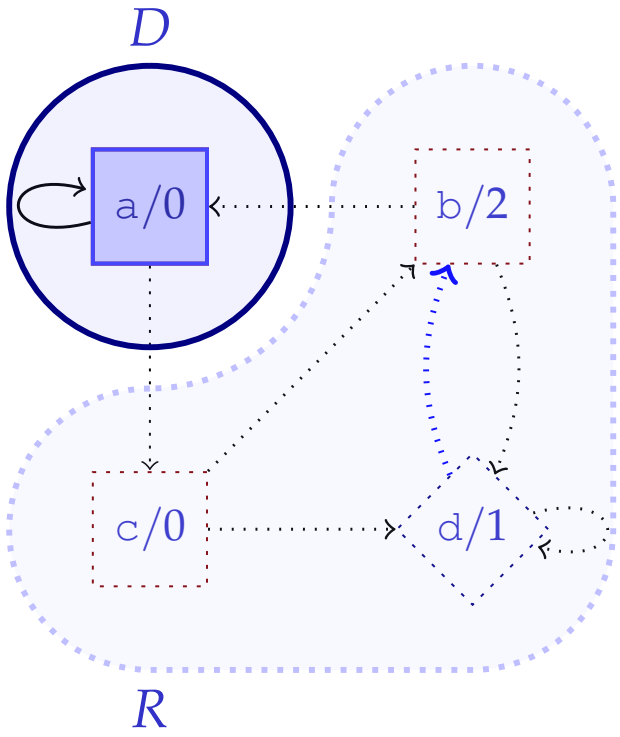
Quasi Dominion Merging



Property 2: Merge

if R is a quasi α-dominion in the game $\mathfrak{Z}$ and D an α-dominion in the subgame $\mathfrak{Z} \setminus R$, then $R \cup D$ is a quasi α-dominion in $\mathfrak{Z}$.

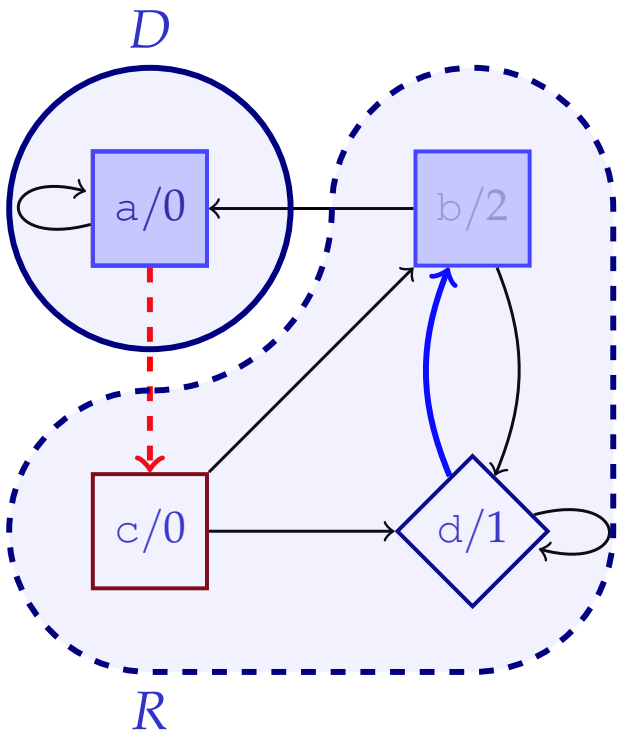
Quasi Dominion Merging



Property 2: Merge

if R is a quasi α-dominion in the game $\mathfrak{D}$ and D an α-dominion in the subgame $\mathfrak{D} \setminus R$, then $R \cup D$ is a quasi α-dominion in $\mathfrak{D}$.

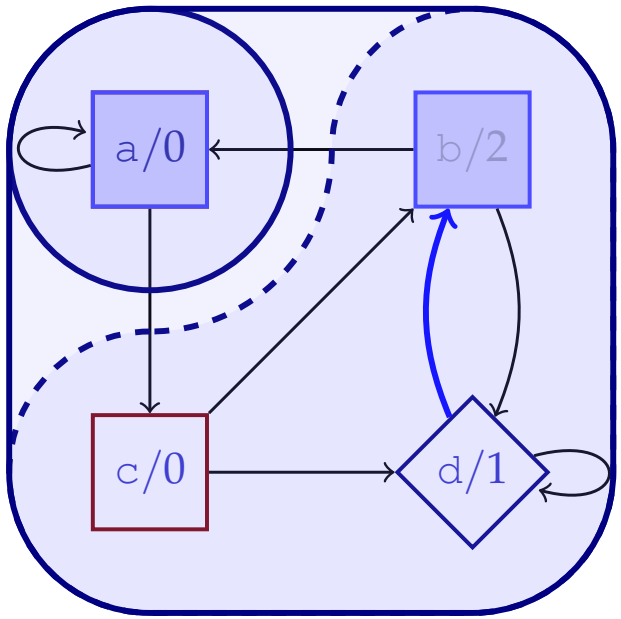
Quasi Dominion Merging



Property 2: Merge

if R is a quasi α-dominion in the game $\mathfrak{D}$ and D an α-dominion in the subgame $\mathfrak{D} \setminus R$, then $R \cup D$ is a quasi α-dominion in $\mathfrak{D}$.

Quasi Dominion Merging

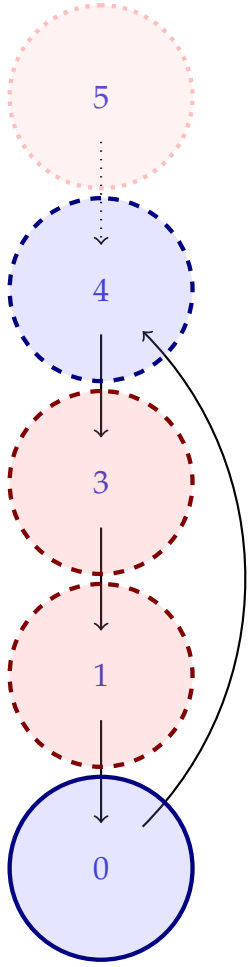


$R \cup D$

Property 2: Merge

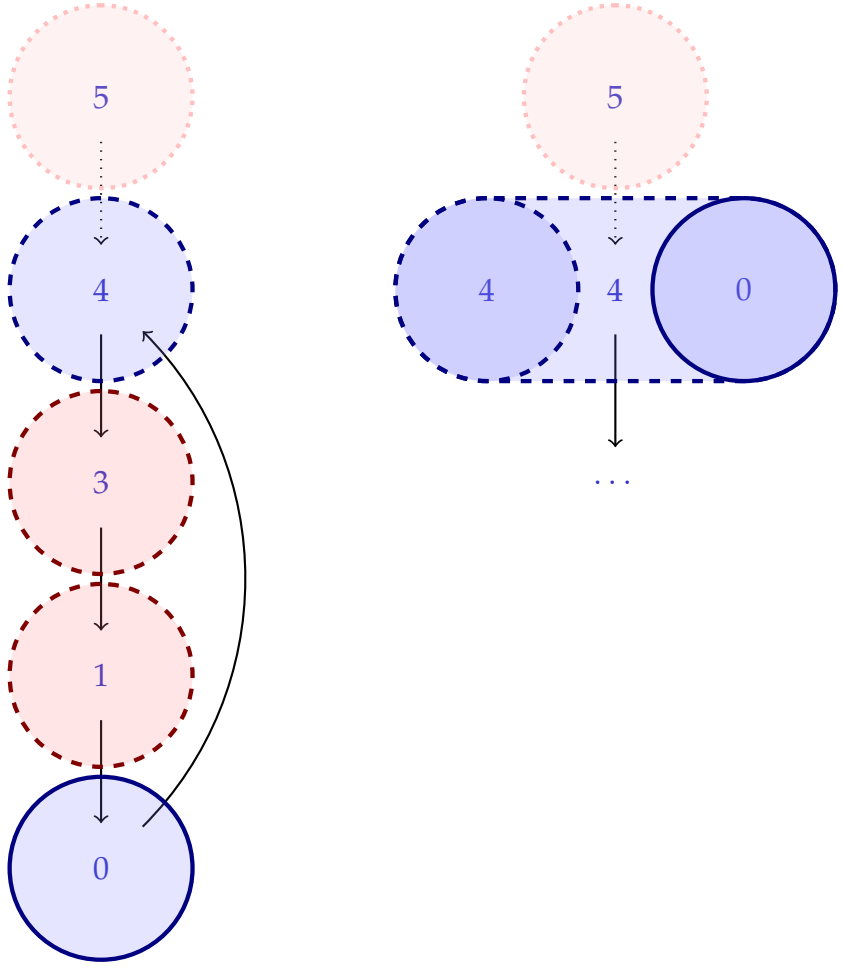
if R is a quasi α-dominion in the game $\mathfrak{Z}$ and D an α-dominion in the subgame $\mathfrak{Z} \setminus R$, then $R \cup D$ is a quasi α-dominion in $\mathfrak{Z}$.

High-Level Simulation II



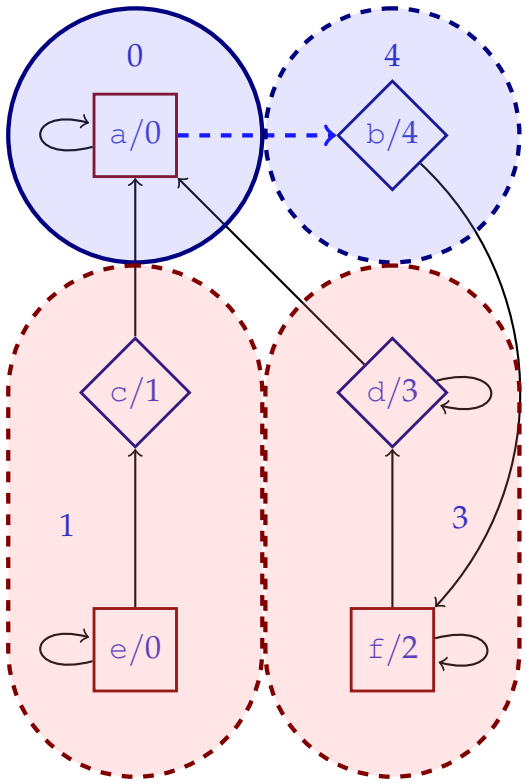
When a α-dominion in a subgame is eventually found, we can apply Property 2, merge it with another α-quasi dominion and maximize the result.

High-Level Simulation II

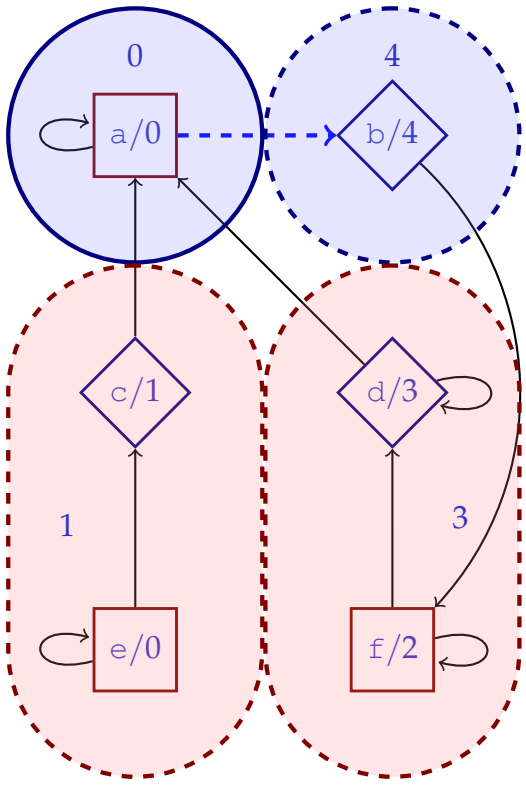


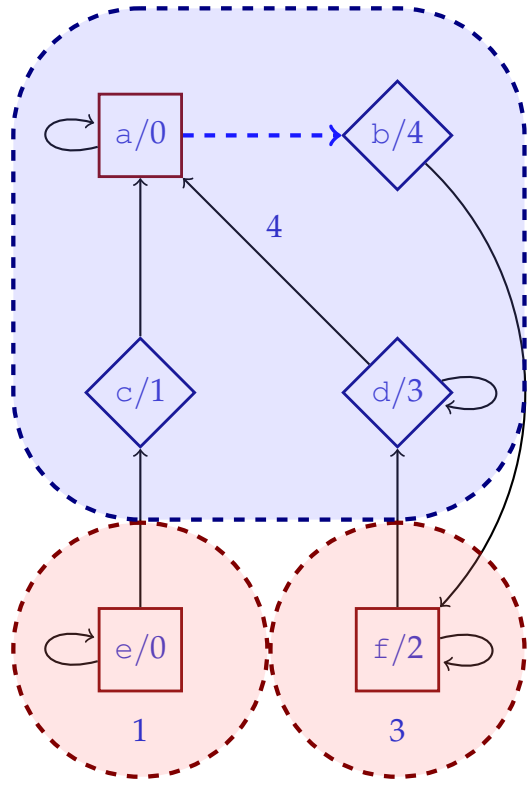
When a α-dominion in a subgame is eventually found, we can apply Property 2, merge it with another α-quasi dominion and maximize the result.

Quasi-Dominion Reset

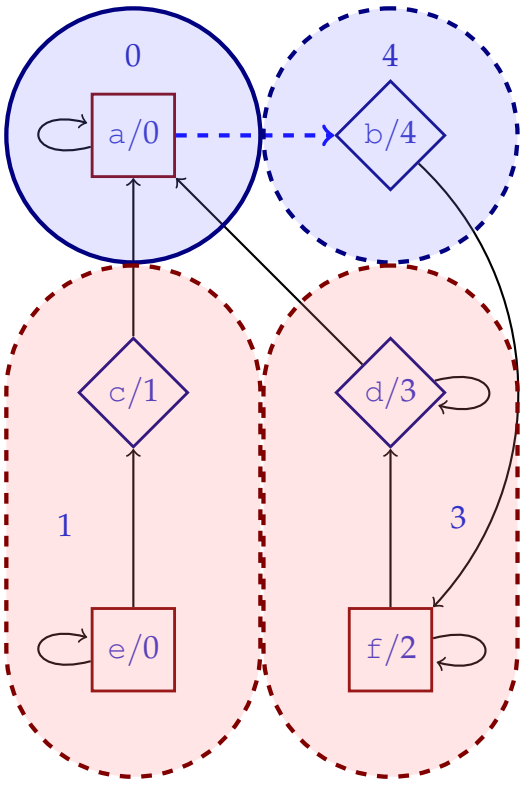


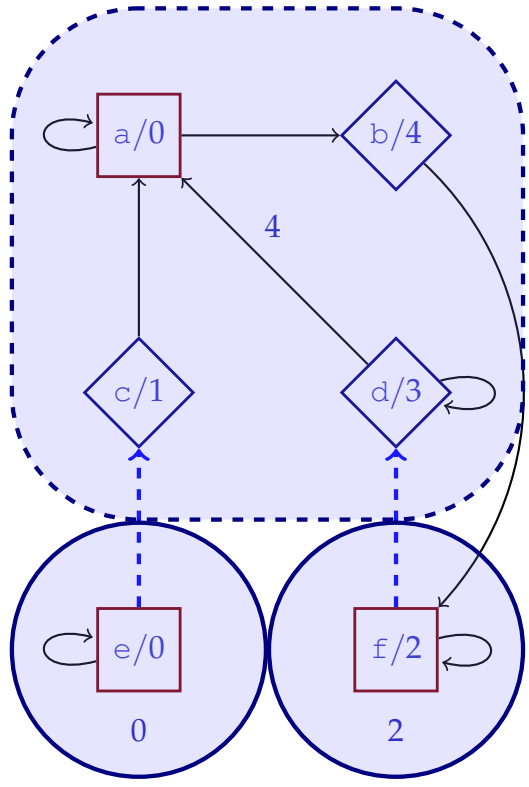
Quasi-Dominion Reset



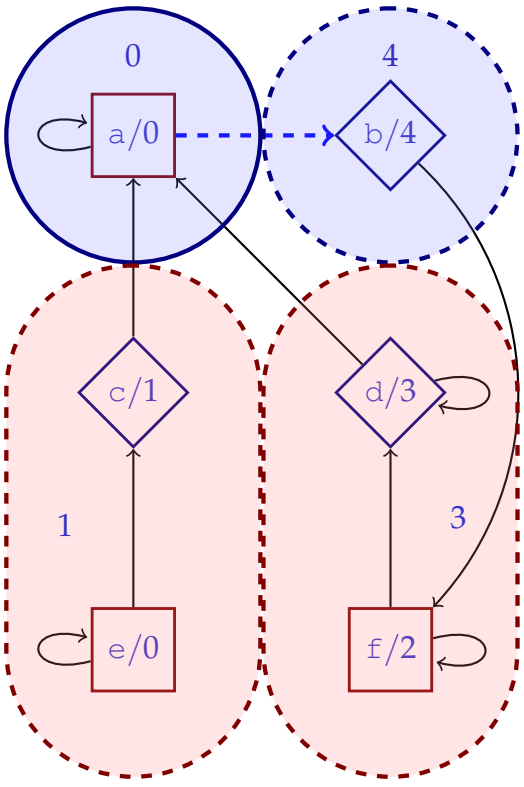


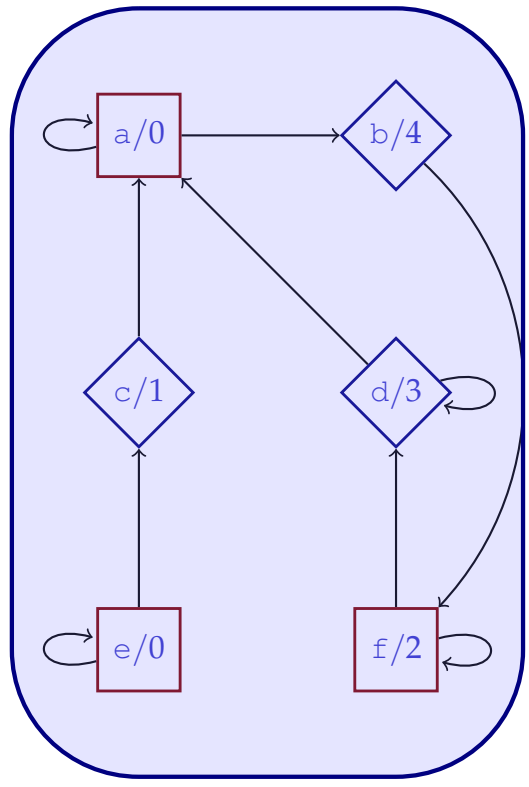
Quasi-Dominion Reset





Quasi-Dominion Reset



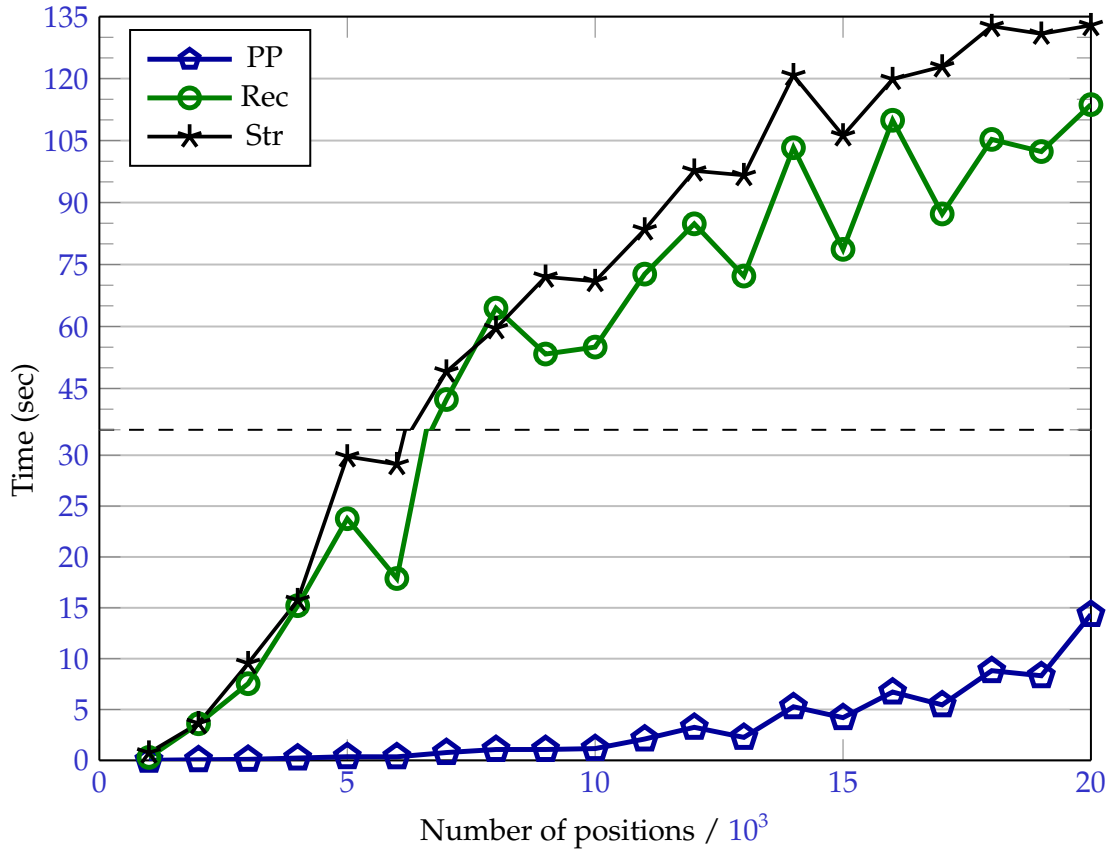


Results

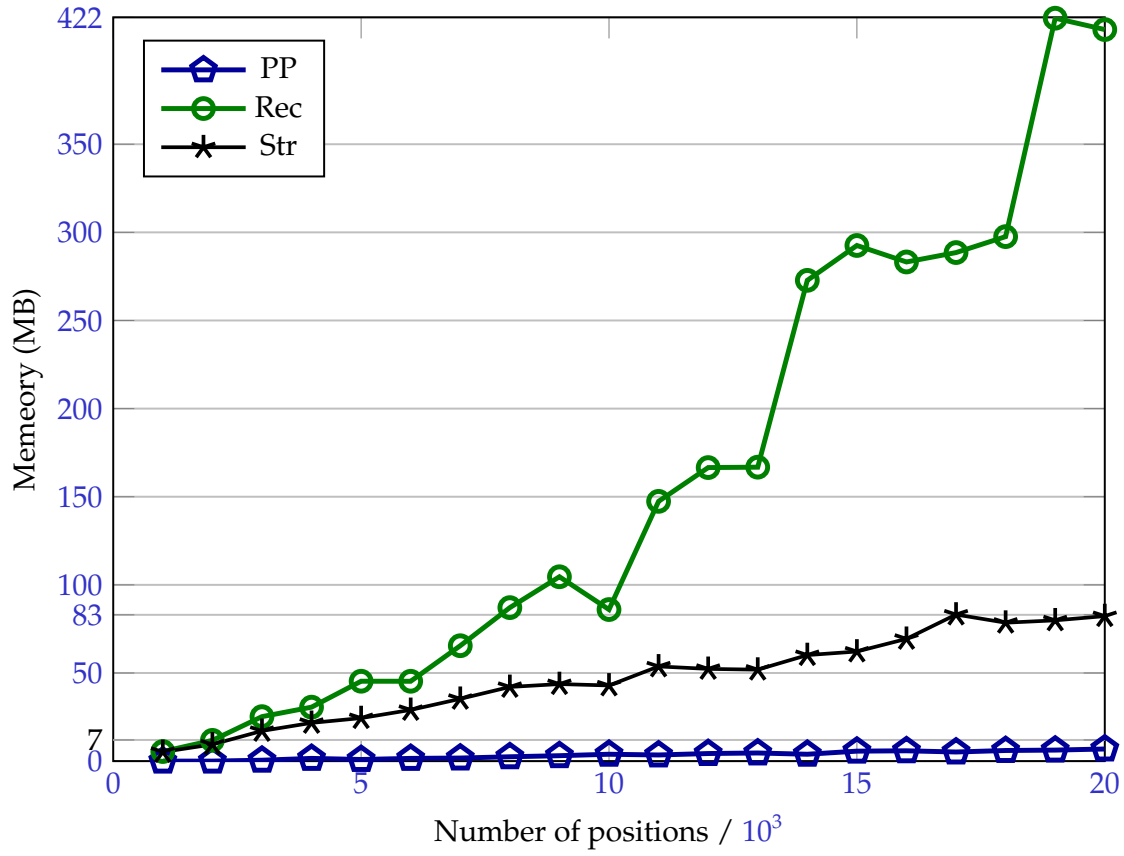
Experimental Evaluation on Concrete Benchmarks

Benchmark	Positions	**FJSSW-QP** (C++)	**JL-QP** (OCaml)	**PP** (OCaml)	**Rec** (OCaml)	**StrImp** (OCaml)
Hanoi	$1.9 \cdot 10^4$	7.7	†	**0.0**	**0.0**	†
Hanoi	$5.9 \cdot 10^4$	72.5	†	**0.0**	0.1	†
Elevator	$2.7 \cdot 10^3$	13.0	†	**0.0**	**0.0**	208.5
Elevator	$1.5 \cdot 10^4$	342.3	†	**0.0**	0.1	†
Elevator	$1 \cdot 10^5$	†	†	**0.2**	0.9	†
Lang. Incl.	$4.4 \cdot 10^4$	25.7	†	**0.0**	0.2	†
Lang. Incl.	$7.8 \cdot 10^4$	71.8	†	**0.0**	0.3	†
Lang. Incl.	$1.2 \cdot 10^5$	149.3	†	**0.1**	0.4	†
Ladder	$1 \cdot 10^4$	121.1	†	**0.0**	**0.0**	30.0
Ladder	$2 \cdot 10^4$	501.5	†	**0.0**	**0.0**	131.6
Str. Imp.	$1.2 \cdot 10^4$	3.7	†	**0.0**	**0.0**	†
Str. Imp.	$4.6 \cdot 10^4$	42.6	†	**0.0**	0.2	†
Str. Imp.	$1 \cdot 10^5$	227.9	†	**0.1**	0.4	†
Clique	$2 \cdot 10^2$	0.8	†	**0.0**	**0.0**	0.1
Clique	$1 \cdot 10^3$	102.1	†	**0.1**	4.6	8.4
Clique	$1.5 \cdot 10^3$	518.6	†	**0.4**	15.2	31.5
MC. Lad.	$3 \cdot 10^4$	9.2	†	**0.0**	**0.0**	0.9
MC. Lad.	$1.5 \cdot 10^5$	254.5	†	**0.0**	0.1	2.7
Rec. Lad.	$5 \cdot 10^3$	13.1	†	**0.5**	†	570.2
Rec. Lad.	$2.5 \cdot 10^4$	288.2	†	**13.8**	†	†
Jurdziński	$7.5 \cdot 10^3$	54.6	†	**3.4**	316.7	41.1
Jurdziński	$1.2 \cdot 10^4$	89.6	†	**9.6**	†	67.2
Jurdziński	$1.9 \cdot 10^4$	320.4	†	**25.41**	†	312.7

Experimental Evaluation on Random Games: Time



Experimental Evaluation on Random Games: Memory



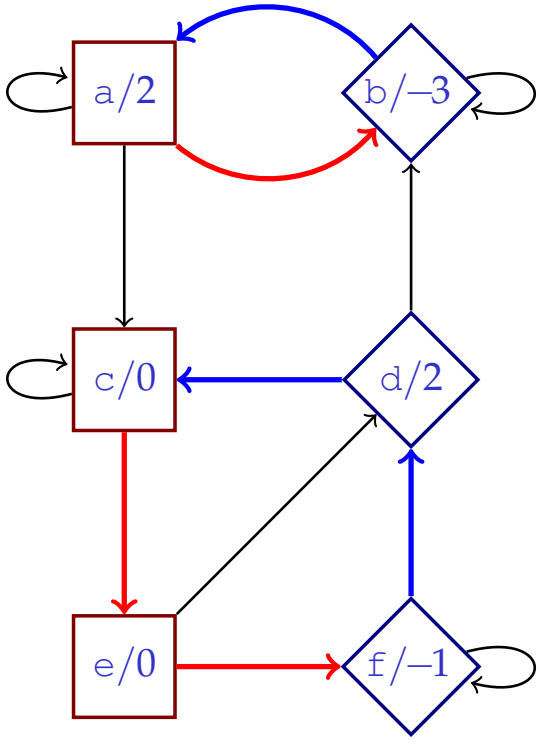
Experimental Evaluation: on Random Games

Positions	Dom	Big	Str	Rec	SmPr	PP
$1 \cdot 10^4$	60.00	15.22	55.24	0.65	60.00	**0.14**
$3 \cdot 10^4$	60.00	35.31	59.37	2.94	60.00	**0.40**
$5 \cdot 10^4$	60.00	45.04	60.00	5.79	60.00	**1.39**
$7 \cdot 10^4$	60.00	50.67	60.00	5.12	59.70	**0.79**
$1 \cdot 10^5$	60.00	58.43	60.00	8.89	60.00	**1.83**
Timeout	100%	49%	95.4%	1%	99.67%	**0%**

Table: Logarithmic number of priorities, average time over 120 games per row.

Mean-Payoff Games

Mean-Payoff Games



Mean-Payoff Game

- positions are labeled with integer numbers called weights;
- the payoff of a play is the sum of the weights of its positions.

Winning Condition for $\alpha \in \{\blacklozenge, \blacksquare\}$

the mean payoff of the play π

$$\liminf_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n} w(\pi_i)$$

is positive (resp., non-positive).

Examples

- the mean payoff of play $(\mathtt{ab})^\omega$ is $-\frac{1}{2}$;
- the mean payoff of play $(\mathtt{efdc})^\omega$ is $\frac{1}{4}$.

A play is winning for player $\blacklozenge$ $\iff$ its payoff diverges to $+\infty$.

Relevance of Mean-Payoff Games

Applications

- *specification*, *verification* of systems against quantitative properties;
- *synthesis* of resource bounded systems: memory, power, bandwidth, etc.
- consistency of Temporal Network (used in temporal planning, scheduling).
- ...

Complexity Status

Mean-Payoff Games are *memoryless determined*, *i.e.*,

- there are *no draws*
- *no memory* is needed to win

mean payoff $\rightsquigarrow$ discounted payoff $\rightsquigarrow$ simple stochastic

$\rightsquigarrow$

NPTime $\cap$ CoNPTime

Known Upper Bounds

- Pseudo-PTime upper bound: $O(m \cdot n \cdot W \cdot \log(n \cdot W))$ [Brim *et al.*, 2011];
- with n the number of positions, m the number of moves and W the maximal positive weight in the game.

Solving MPG with Quasi-Dominions

Best algorithm for Mean-Payoff Games

- Based on the notion of ■*-progress measure*: a local condition among adjacent positions witnessing the existence of a winning ■-strategy
- A progress measure is computed by iteratively updating a *Payoff measure function* μ that associates a natural number with each position
- The measure $\mu(v)$ of position v estimates the payoff that player ♦ can enforce along finite plays starting in v

The Goal

- Speed up the convergence of the most efficient algorithm for mean payoff games: *Small Energy Progress Measure* (Brim *et al.*)

The Progress Measure Approach: Intuitions

Measure Invariant

The measure of any position v under-approximates the payoff that player $\blacklozenge$ can enforce along some finite play starting from v.

The Progress Measure Approach: Intuitions

Measure Invariant

The measure of any position v under-approximates the payoff that player $\blacklozenge$ can enforce along some finite play starting from v.

Observation

The payoff of a simple play cannot exceed the sum of all the positive weights in the game ($S = \sum\{w(v) : w(v) > 0\}$).

The Progress Measure Approach: Intuitions

Measure Invariant

The measure of any position v under-approximates the payoff that player ♦ can enforce along some finite play starting from v.

Observation

The payoff of a simple play cannot exceed the sum of all the positive weights in the game ($S = \sum\{w(v) : w(v) > 0\}$).

Observation + Invariant $\Rightarrow \mu(v) > S$ signals a win for player ♦ and is replaced by ∞.

The Progress Measure Approach: Intuitions

Measure Invariant

The measure of any position v under-approximates the payoff that player ♦ can enforce along some finite play starting from v.

Observation

The payoff of a simple play cannot exceed the sum of all the positive weights in the game ($S = \sum\{w(v) : w(v) > 0\}$).

Observation + Invariant $\Rightarrow \mu(v) > S$ signals a win for player ♦ and is replaced by ∞.

Computing a Progress Measure

Players are allowed to change the measure of their positions only by choosing moves:

- choosing move (v, u) will grants to position v the measure $\mu(v) = w(v) + \mu(u)$
- Player ♦ will choose moves that make the measures of his positions increase as much as possible.
- Player ■ will choose move that make the measures of his positions as low as possible.

Mean-payoff progress measure

Progress Measure

A measure function $\mu : V \to \mathbb{N} \cup \{\infty\}$ is a progress measure if

1. for all ♦-positions v, $\mu(u) + w(v) \leq \mu(v)$, for all moves (v, u).
2. for all ■-positions v, $\mu(u) + w(v) \leq \mu(v)$, for some move (v, u).

Mean-payoff progress measure

Progress Measure

A measure function $\mu : V \to \mathbb{N} \cup \{\infty\}$ is a progress measure if

1. for all ♦-positions v, $\mu(u) + w(v) \leq \mu(v)$, for all moves (v, u).
2. for all ■-positions v, $\mu(u) + w(v) \leq \mu(v)$, for some move (v, u).

The ■-strategy that chooses only moves satisfying Condition 2 ensures that every induced play eventually gets trapped into a non-positive cycle.

Mean-payoff progress measure

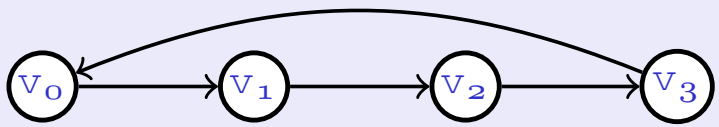
Progress Measure

A measure function $\mu : V \to \mathbb{N} \cup \{\infty\}$ is a progress measure if

1. for all ♦-positions v, $\mu(u) + w(v) \leq \mu(v)$, for all moves (v, u).
2. for all ■-positions v, $\mu(u) + w(v) \leq \mu(v)$, for some move (v, u).

The ■-strategy that chooses only moves satisfying Condition 2 ensures that every induced play eventually gets trapped into a non-positive cycle.

Example



$$\begin{aligned}
\textstyle\sum_{i=0}^{3}(\mu(v_{i+1}) + w(v_i)) &\leq \textstyle\sum_{i=0}^{3} \mu(v_i) \\
\textstyle\sum_{i=0}^{3} w(v_i) &\leq \textstyle\sum_{i=0}^{3}(\mu(v_i) - \mu(v_{i+1})) = \mu(v_0) - \mu(v_4) \\
\textstyle\sum_{i=0}^{3} w(v_i) &\leq \mu(v_0) - \mu(v_0) = 0
\end{aligned}$$

Mean-payoff progress measure

Progress Measure

A measure function $\mu : V \to \mathbb{N} \cup \{\infty\}$ is a progress measure if

1. for all ♦-positions v, $\mu(u) + w(v) \leq \mu(v)$, for all moves (v, u).
2. for all ■-positions v, $\mu(u) + w(v) \leq \mu(v)$, for some move (v, u).

The Solution

The winning regions of the two player:

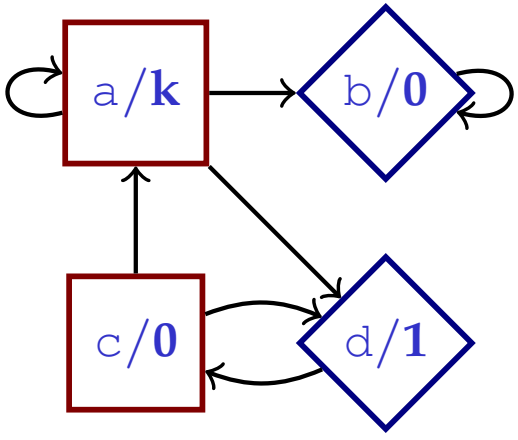
- $\mathrm{Wn}_{\blacklozenge}$ contains the positions v *s.t.* $\mu(v) = \infty$;
- $\mathrm{Wn}_{\blacksquare}$ contains the positions v *s.t.* $\mu(v) \neq \infty$

Computing a Progress Measure

The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + w(v) : (v,u) \in \supset\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + w(v) : (v,u) \in \supset\}, & \text{if } v \in \blacksquare. \end{cases}$$

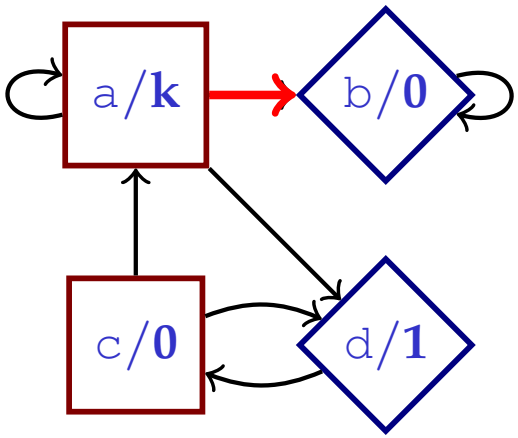
	a	b	c	d
μ_0	0	0	0	0



Computing a Progress Measure

The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + w(v) : (v,u) \in \Im\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + w(v) : (v,u) \in \Im\}, & \text{if } v \in \blacksquare. \end{cases}$$

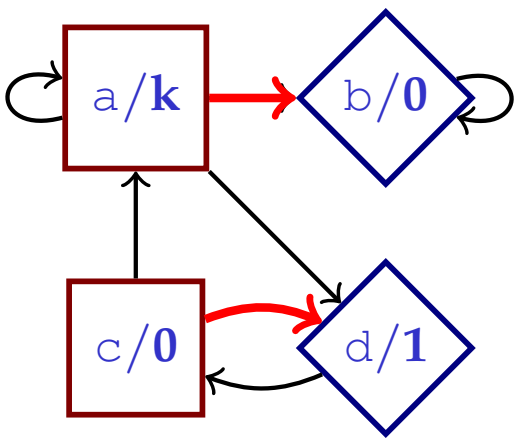


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	0	1

Computing a Progress Measure

The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + w(v) : (v,u) \in \ni\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + w(v) : (v,u) \in \ni\}, & \text{if } v \in \blacksquare. \end{cases}$$

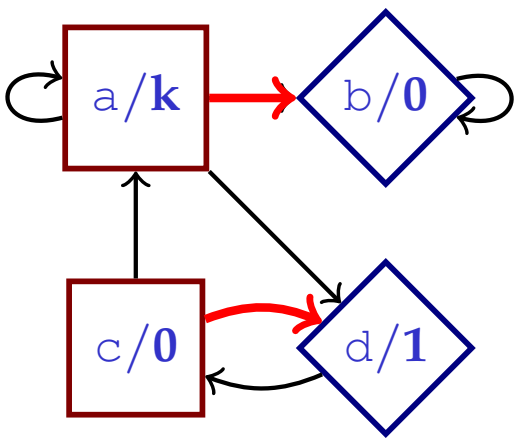


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	0	1
μ_2	k	0	1	1

Computing a Progress Measure

The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + w(v) : (v,u) \in \ni\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + w(v) : (v,u) \in \ni\}, & \text{if } v \in \blacksquare. \end{cases}$$

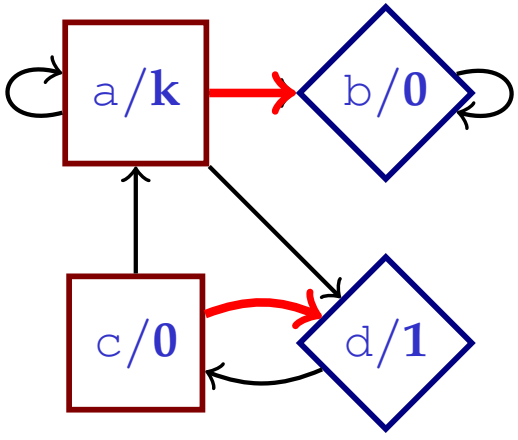


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	0	1
μ_2	k	0	1	1
μ_3	k	0	1	2

Computing a Progress Measure

The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + w(v) : (v,u) \in \ni\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + w(v) : (v,u) \in \ni\}, & \text{if } v \in \blacksquare. \end{cases}$$

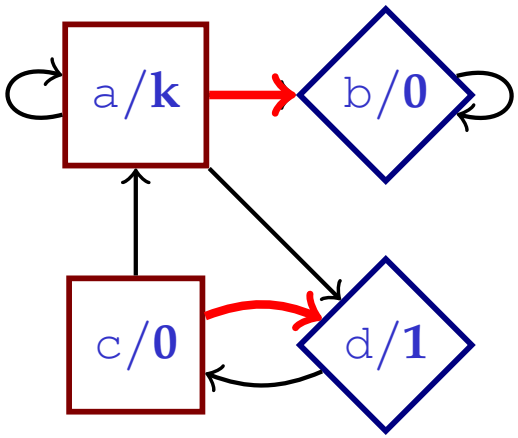


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	0	1
μ_2	k	0	1	1
μ_3	k	0	1	2
μ_4	k	0	2	2
μ_5	k	0	2	3
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
μ_{2k}	k	0	k	k

Computing a Progress Measure

The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + w(v) : (v,u) \in \ni\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + w(v) : (v,u) \in \ni\}, & \text{if } v \in \blacksquare. \end{cases}$$

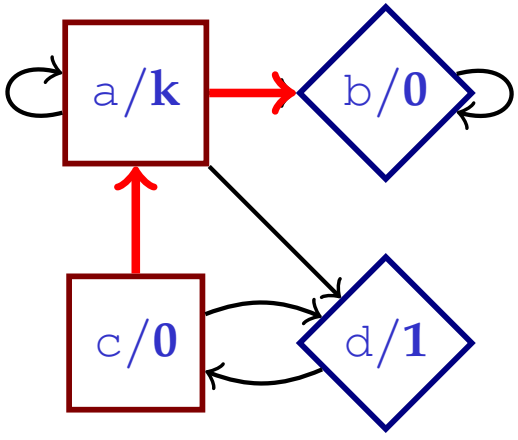


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	0	1
μ_2	k	0	1	1
μ_3	k	0	1	2
μ_4	k	0	2	2
μ_5	k	0	2	3
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
μ_{2k}	k	0	k	k
μ_{2k+1}	k	0	k	$k+1$

Computing a Progress Measure

The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + w(v) : (v,u) \in \ni\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + w(v) : (v,u) \in \ni\}, & \text{if } v \in \blacksquare. \end{cases}$$

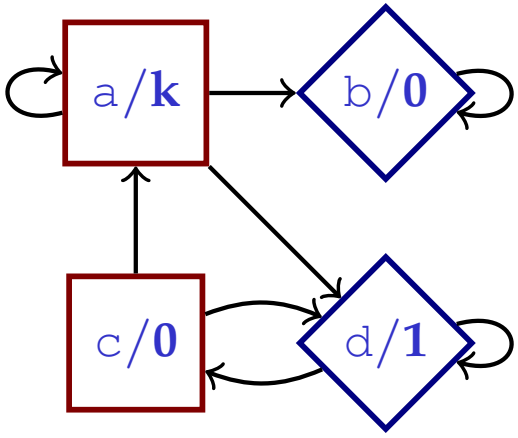


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	0	1
μ_2	k	0	1	1
μ_3	k	0	1	2
μ_4	k	0	2	2
μ_5	k	0	2	3
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
μ_{2k}	k	0	k	k
μ_{2k+1}	k	0	k	$k+1$
μ_{2k+2}	k	0	k	$k+1$

Computing the minimal progress measure

The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacksquare. \end{cases}$$

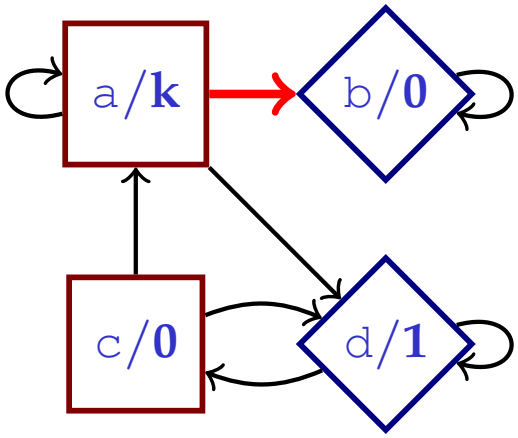


	a	b	c	d
μ_0	0	0	0	0

Computing the minimal progress measure

The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacksquare. \end{cases}$$

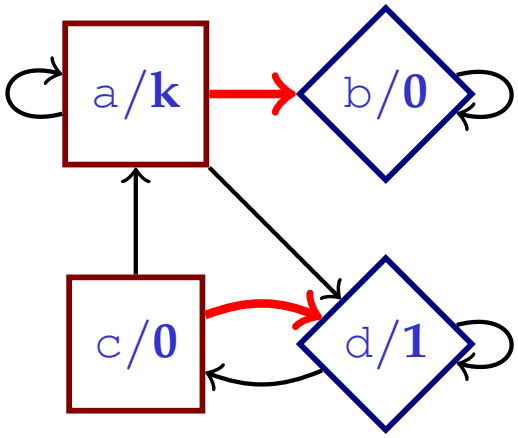


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	0	1

Computing the minimal progress measure

The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacksquare. \end{cases}$$

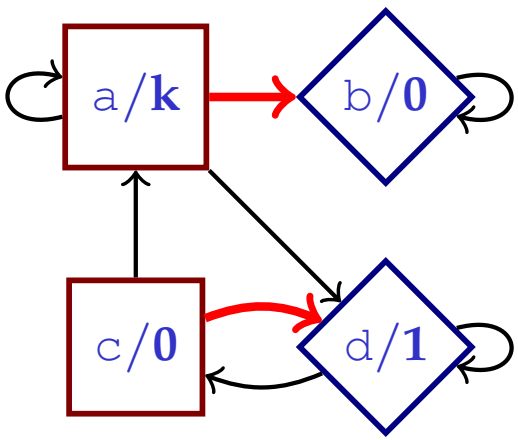


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	0	1
μ_2	k	0	1	2

Computing the minimal progress measure

The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacksquare. \end{cases}$$

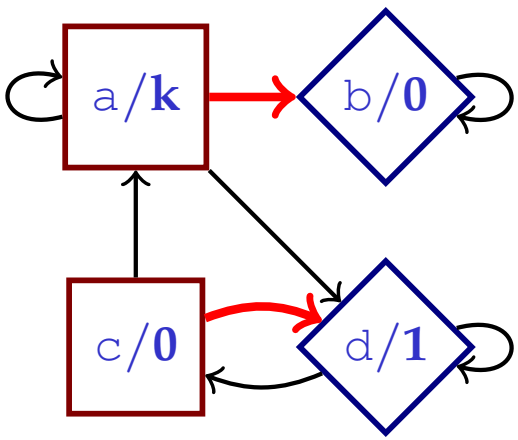


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	0	1
μ_2	k	0	1	2
μ_3	k	0	2	3

Computing the minimal progress measure

The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacksquare. \end{cases}$$

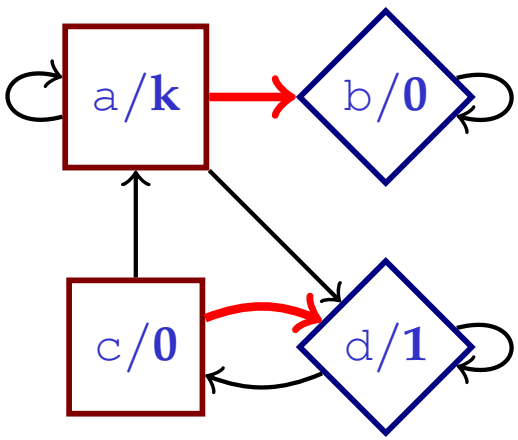


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	0	1
μ_2	k	0	1	2
μ_3	k	0	2	3
μ_4	k	0	3	4

Computing the minimal progress measure

The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacksquare. \end{cases}$$

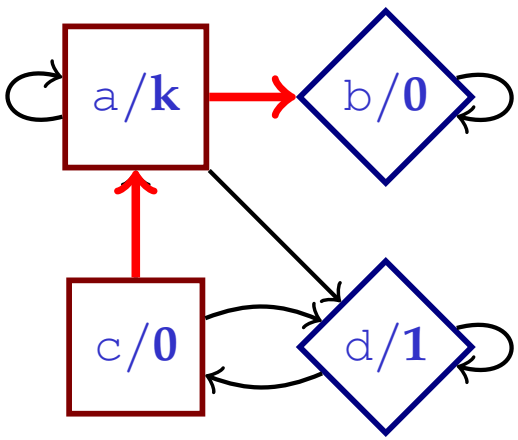


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	0	1
μ_2	k	0	1	2
μ_3	k	0	2	3
μ_4	k	0	3	4
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
μ_{k+1}	k	0	k	$k+1$

Computing the minimal progress measure

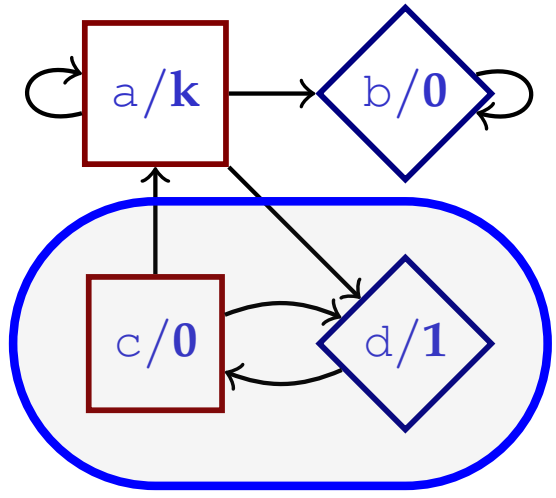
The least fixpoint of the following lift operator

$$lift(\mu)(v) \triangleq \begin{cases} \max\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacklozenge; \\ \min\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \blacksquare. \end{cases}$$



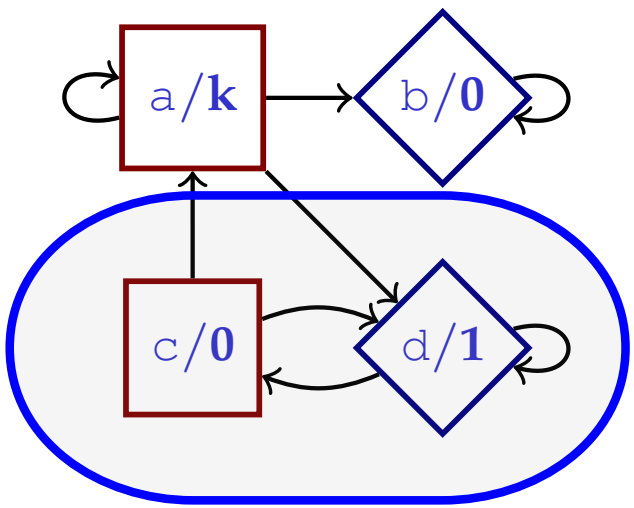
	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	0	1
μ_2	k	0	1	2
μ_3	k	0	2	3
μ_4	k	0	3	4
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
μ_{k+1}	k	0	k	$k+1$
μ_{k+2}	k	0	k	∞

The Problem



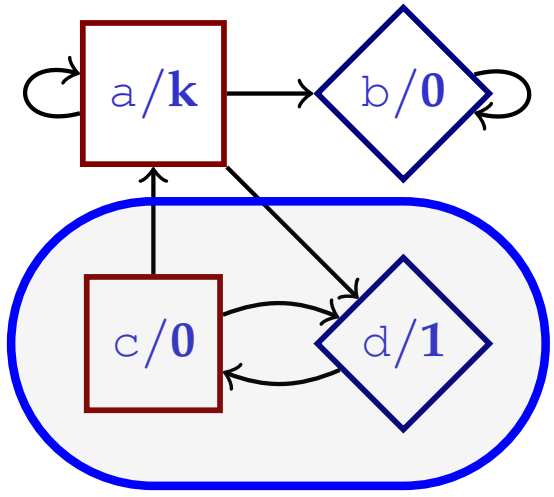
The number of iteration to reach a progress measure is linear in the weight k for both games.

This is due to the minimal measure increase policy employed by player ■ within the *lift* operator.



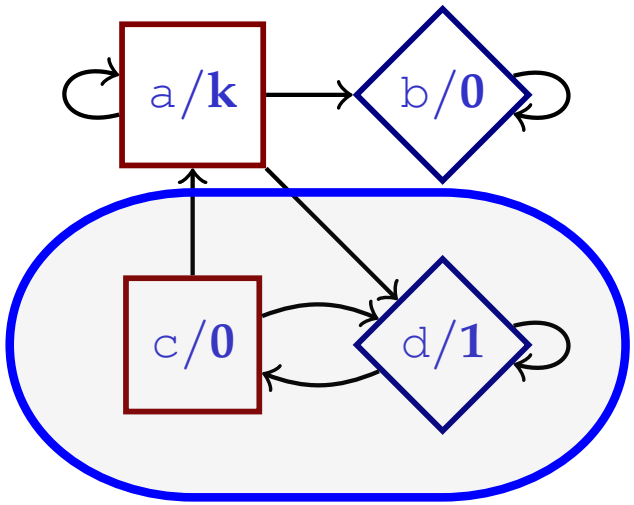
The move from $(\mathtt{c}, \mathtt{d})$ is clearly a loosing strategy for player ■.

The Problem



The number of iteration to reach a progress measure is linear in the weight k for both games.

This is due to the minimal measure increase policy employed by player ■ within the *lift* operator.



The move from $(\mathtt{c}, \mathtt{d})$ is clearly a loosing strategy for player ■.

Crucial observation

The set $\{\mathtt{c}, \mathtt{d}\}$ forms a quasi ♦-dominion in both games.

A Quasi-dominion progress measure

A new approach based on the merge of notions of

- *quasi-dominion*
- *progress measure*

Basic Concepts

Quasi ♦-dominion: set of positions where player ♦ has a strategy to win all the play that remain in the set forever (*i.e.*, those where the opponent ■ does not escape).

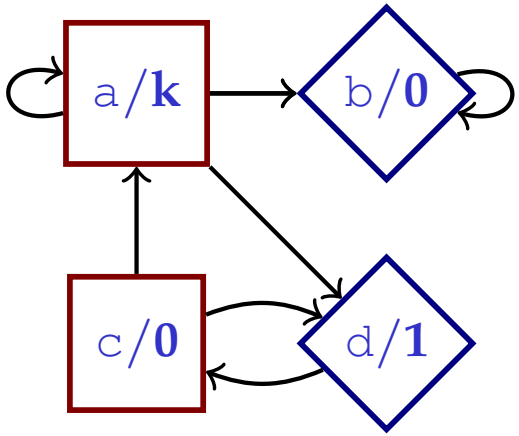
Crucial Property

The set of positions with a *positive* measure always form a quasi ♦-dominion.

The algorithm: Intuitions

We treat positions inside a quasi dominion differently from the ones outside

- Positions outside the quasi ♦-dominion (with non-positive measure) are lifted as in the standard way;
- Positions of player ■ within the quasi ♦-dominion try to escape following the move that exits the quasi dominion and grants the minimal measure increase;
- Positions of player ♦ within the quasi ♦-dominion try to remain in the quasi dominion if they can;

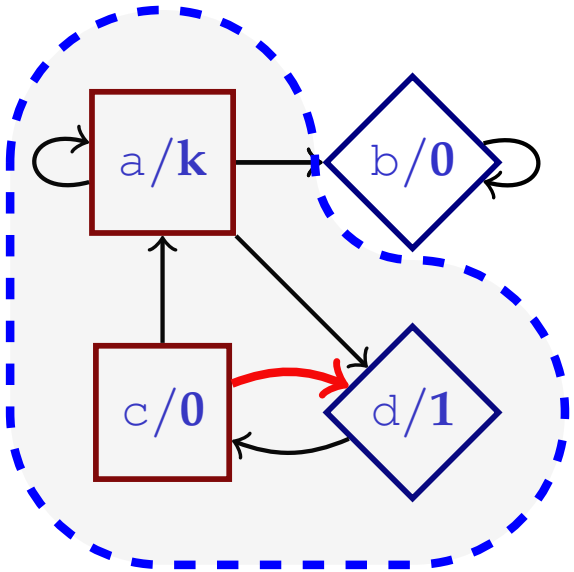


	a	b	c	d
μ_0	0	0	0	0

The algorithm: Intuitions

We treat positions inside a quasi dominion differently from the ones outside

- Positions outside the quasi ♦-dominion (with non-positive measure) are lifted as in the standard way;
- Positions of player ■ within the quasi ♦-dominion try to escape following the move that exits the quasi dominion and grants the minimal measure increase;
- Positions of player ♦ within the quasi ♦-dominion try to remain in the quasi dominion if they can;

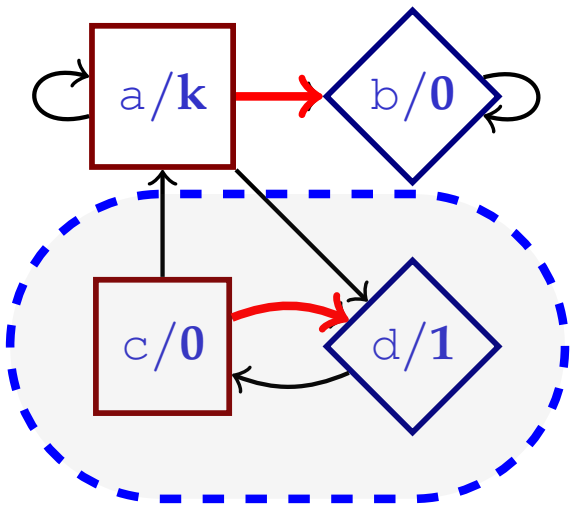


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	1	1

The algorithm: Intuitions

We treat positions inside a quasi dominion differently from the ones outside

- Positions outside the quasi ♦-dominion (with non-positive measure) are lifted as in the standard way;
- Positions of player ■ within the quasi ♦-dominion try to escape following the move that exits the quasi dominion and grants the minimal measure increase;
- Positions of player ♦ within the quasi ♦-dominion try to remain in the quasi dominion if they can;

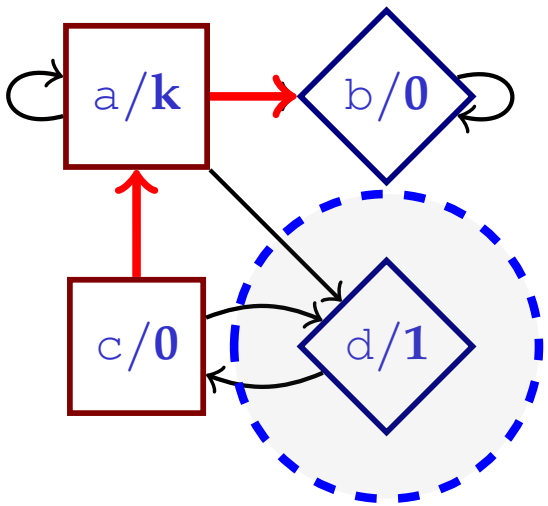


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	1	1
μ_2	k	0		

The algorithm: Intuitions

We treat positions inside a quasi dominion differently from the ones outside

- Positions outside the quasi ♦-dominion (with non-positive measure) are lifted as in the standard way;
- Positions of player ■ within the quasi ♦-dominion try to escape following the move that exits the quasi dominion and grants the minimal measure increase;
- Positions of player ♦ within the quasi ♦-dominion try to remain in the quasi dominion if they can;

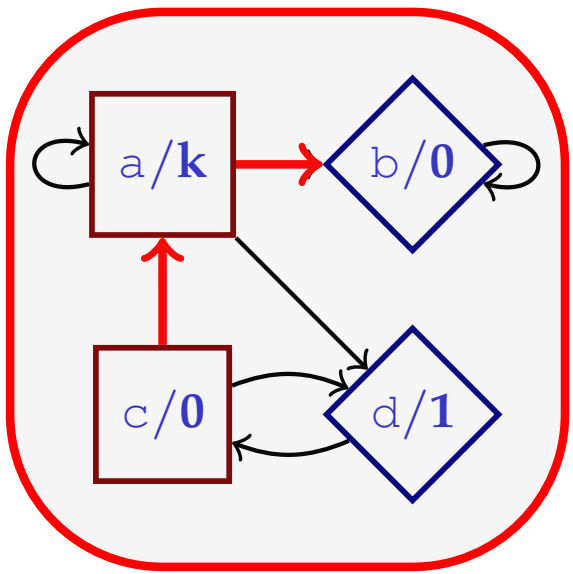


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	1	1
μ_2	k	0	k	

The algorithm: Intuitions

We treat positions inside a quasi dominion differently from the ones outside

- Positions outside the quasi ♦-dominion (with non-positive measure) are lifted as in the standard way;
- Positions of player ■ within the quasi ♦-dominion try to escape following the move that exits the quasi dominion and grants the minimal measure increase;
- Positions of player ♦ within the quasi ♦-dominion try to remain in the quasi dominion if they can;

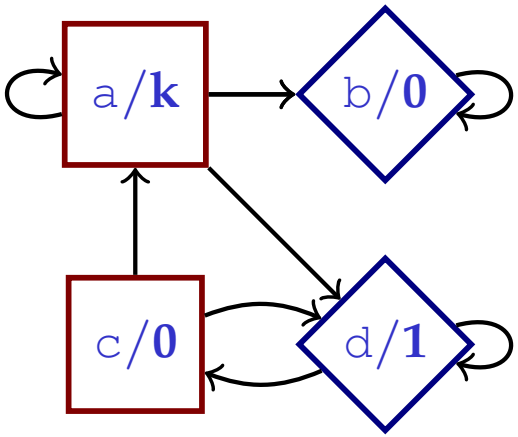


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	1	1
μ_2	k	0	k	$k+1$

The algorithm: Intuitions

We treat positions inside a quasi dominion differently from the ones outside

- Positions outside the quasi ♦-dominion (with non-positive measure) are lifted as in the standard way;
- Positions of player ■ within the quasi ♦-dominion try to escape following the move that exits the quasi dominion and grants the minimal measure increase;
- Positions of player ♦ within the quasi ♦-dominion try to remain in the quasi dominion if they can;

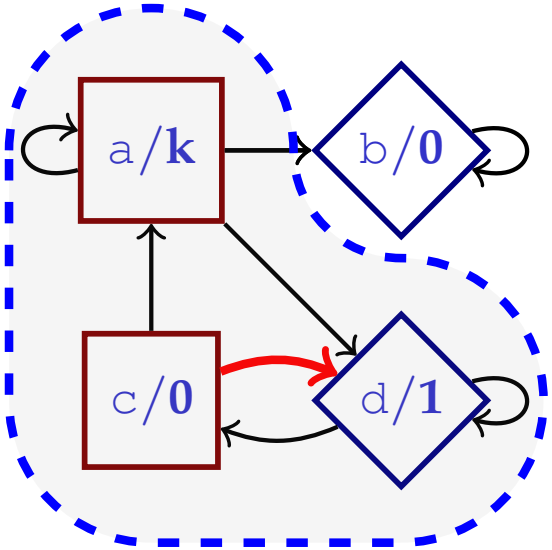


	a	b	c	d
μ_0	0	0	0	0

The algorithm: Intuitions

We treat positions inside a quasi dominion differently from the ones outside

- Positions outside the quasi ♦-dominion (with non-positive measure) are lifted as in the standard way;
- Positions of player ■ within the quasi ♦-dominion try to escape following the move that exits the quasi dominion and grants the minimal measure increase;
- Positions of player ♦ within the quasi ♦-dominion try to remain in the quasi dominion if they can;

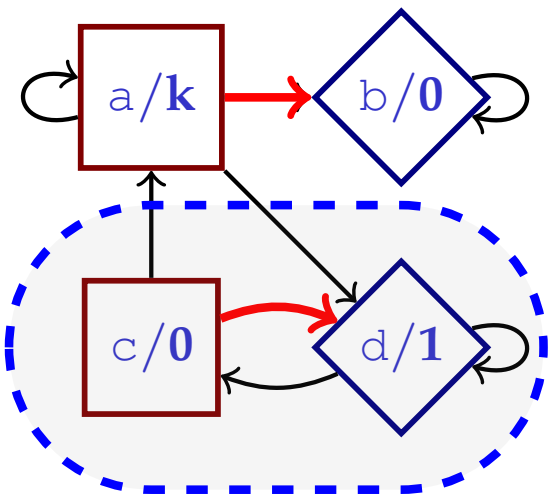


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	1	1

The algorithm: Intuitions

We treat positions inside a quasi dominion differently from the ones outside

- Positions outside the quasi ♦-dominion (with non-positive measure) are lifted as in the standard way;
- Positions of player ■ within the quasi ♦-dominion try to escape following the move that exits the quasi dominion and grants the minimal measure increase;
- Positions of player ♦ within the quasi ♦-dominion try to remain in the quasi dominion if they can;

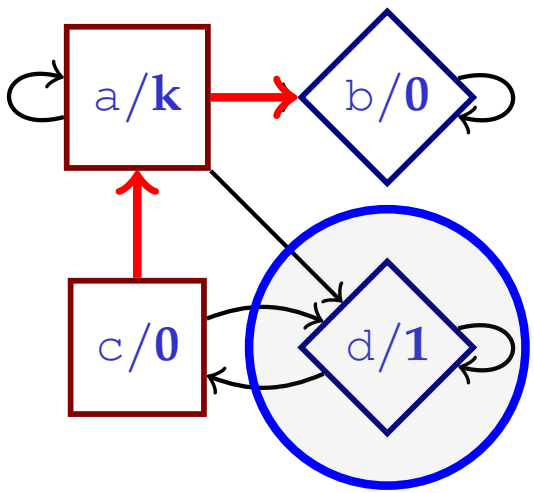


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	1	1
μ_2	k	0		

The algorithm: Intuitions

We treat positions inside a quasi dominion differently from the ones outside

- Positions outside the quasi ♦-dominion (with non-positive measure) are lifted as in the standard way;
- Positions of player ■ within the quasi ♦-dominion try to escape following the move that exits the quasi dominion and grants the minimal measure increase;
- Positions of player ♦ within the quasi ♦-dominion try to remain in the quasi dominion if they can;

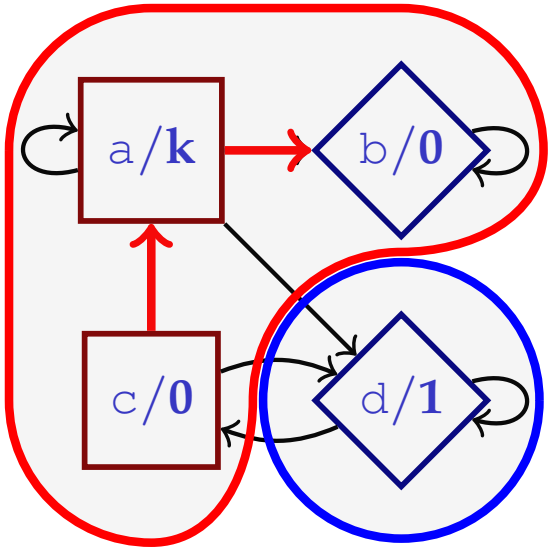


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	1	1
μ_2	k	0	k	

The algorithm: Intuitions

We treat positions inside a quasi dominion differently from the ones outside

- Positions outside the quasi ♦-dominion (with non-positive measure) are lifted as in the standard way;
- Positions of player ■ within the quasi ♦-dominion try to escape following the move that exits the quasi dominion and grants the minimal measure increase;
- Positions of player ♦ within the quasi ♦-dominion try to remain in the quasi dominion if they can;

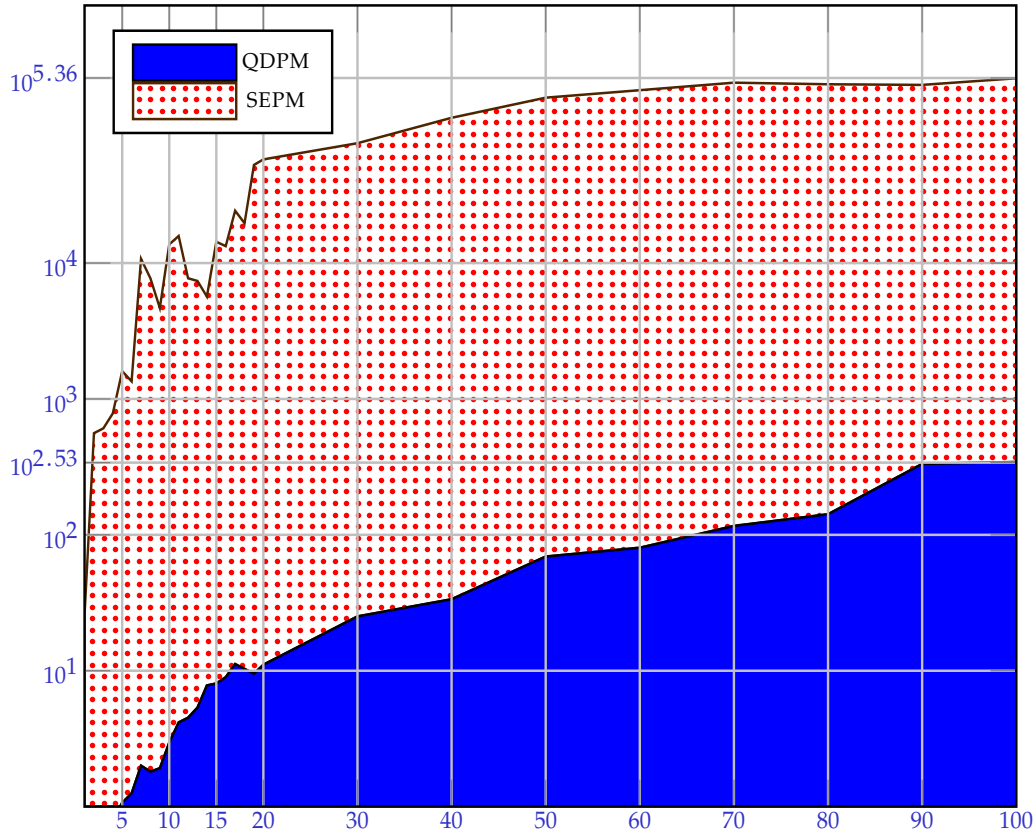


	a	b	c	d
μ_0	0	0	0	0
μ_1	k	0	1	1
μ_2	k	0	k	∞

Experimental Evaluation

Benchmark	(Pos,Moves)	SEPM	QDPM
Elevator 1	(144, 234)	0.04	**0.0010**
Elevator 2	(564, 950)	8.80	**0.0042**
Elevator 3	(2688, 4544)	4675.71	**0.0064**
Lang. Incl. 1	(170, 1094)	3.18	**0.0021**
Lang. Incl. 2	(304, 1222)	16.76	**0.0019**
Lang. Incl. 3	(428, 878)	20.25	**0.0033**
Lang. Incl. 4	(628, 1538)	135.51	**0.0029**
Lang. Incl. 5	(509, 2126)	148.37	**0.0034**
Lang. Incl. 6	(835, 2914)	834.90	**0.0051**
Lang. Incl. 7	(1658, 4544)	2277.87	**0.0100**

concrete verification problems



maximal out-degree 40

Number of positions: $n \times 10^3$

SEPM: Small Energy Progress Measure

QDPM: Quasi Domionion Progress Measure

Concluding Remarks

A novel notion of Quasi Dominion that seems to

- be general enough to be applicable to different types of games
- be simple enough to be easily computed
- convey enough information to accelerate the convergence to the solution of the games considered so far.

It has led to a *New approach* for solving parity games

- exponentially more efficient in space complexity (*w.r.t.* # priorities)
- very effective in practice

Integration of *Mean-Payoff* progress measures and *Quasi Dominions*

- matches the state-of-art asymptotic time complexity: $\mathrm{O}(n \cdot m \cdot W \cdot \log(n \cdot W))$;
- very effective in practice;
- existence of a families of games on which it performs arbitrarily better than best solution algorithm.