Parity Games, their algorithms and complexity

Tom van Dijk (University of Twente)

IPA Spring Days, 2023

ACKNOWLEDGEMENTS

`https://discord.gg/bR6sbjf6jT`

These slides were created in part based on lecture slides by Tim Willemse, Wolfgang Schreiner and Nathanael Fijalkow, Joost-Pieter Katoen and Javier Esparza

OUTLINE

1. Background / Applications of parity games
   - Ω-automata
   - Modal μ-calculus
   - LTL synthesis
2. Parity game solving algorithms
   - Introduction
   - Decomposition based algorithms
   - Local algorithms
3. Quasi-polynomial algorithms
   - Universal trees and progress measures
   - Ordered progress measures
   - Quasi-polynomial recursive algorithm
4. Final remarks

OUTLINE

1. Background / Applications of parity games
   - Ω-automata
   - Modal μ-calculus
   - LTL synthesis
2. Parity game solving algorithms
   - Introduction
   - Decomposition based algorithms
   - Local algorithms
3. Quasi-polynomial algorithms
   - Universal trees and progress measures
   - Ordered progress measures
   - Quasi-polynomial recursive algorithm
4. Final remarks

Recap: regular languages and finite automata

- Syntax of regular expressions: $r ::= \emptyset \mid \epsilon \mid a \mid r_1 + r_2 \mid r_1.r_2 \mid r^*$
- Finite automaton: $\langle S, S_0, \Sigma, R, F \rangle$
  - S is a finite set of states and S_0 is a set of initial states
  - Σ is an alphabet
  - $R \colon S \times \Sigma \to 2^S$ is a transition function
  - $F \subseteq S$ is a set of accepting (final) states
- Two types: nondeterministic NFA and deterministic DFA
- For every NFA $\mathcal{A}$, the accepted language of $\mathcal{A}$ is a regular language
- For every DFA $\mathcal{A}$, the accepted language of $\mathcal{A}$ is a regular language
- DFAs are as expressive as NFAs, but can be exponentially larger

Recap: regular languages and finite automata

- Syntax of regular expressions: $r ::= \emptyset \mid \epsilon \mid a \mid r_1 + r_2 \mid r_1.r_2 \mid r^*$
- Finite automaton: $\langle S, S_0, \Sigma, R, F \rangle$
- Constructions: RE $\leftrightarrow$ NFA, NFA $\leftrightarrow$ DFA
  - RE $\rightarrow$ NFA: based on the syntax tree
  - RE $\leftarrow$ NFA: state elimination
  - NFA $\rightarrow$ DFA: determinisation: power set construction (exponential)
  - NFA $\leftarrow$ DFA: trivial (every DFA is an NFA)

ω-AUTOMATA

- Finite automata accept/reject words of finite length
- ω-Automata accept/reject words of infinite length (ω-words)

ω-regular expressions

ω-regular expressions have syntax

$$s ::= r^\omega \mid r.s \mid s_1 + s_2$$

where r is an (ordinary) regular expression and $\epsilon \notin \mathcal{L}(r)$

Semantics of ω-regular expressions

The ω-language $\mathcal{L}_\omega(s)$ of an ω-regular expression s is inductively defined by

- $\mathcal{L}_\omega(r^\omega) = (\mathcal{L}(r))^\omega$
- $\mathcal{L}_\omega(rs) = \mathcal{L}(r).\mathcal{L}_\omega(s)$
- $\mathcal{L}_\omega(s_1 + s_2) = \mathcal{L}_\omega(s_1) \cup \mathcal{L}_\omega(s_2)$

ω-automata

ω-automata have the same syntax as finite automata, but different acceptance conditions

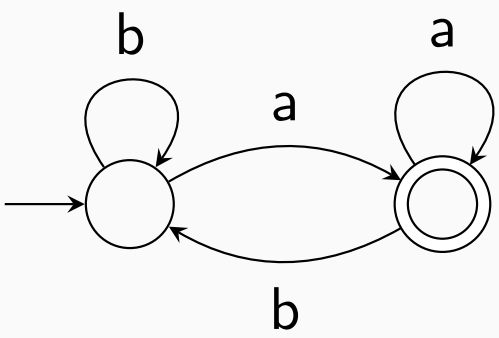
Büchi automaton

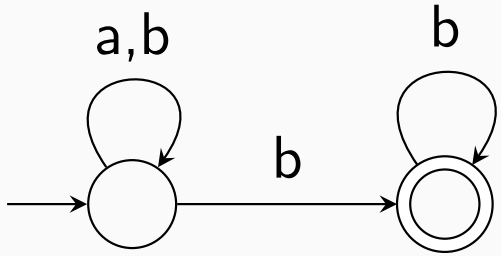
Büchi automaton: $\langle S, S_0, \Sigma, R, F \rangle$ (same as Finite automaton)

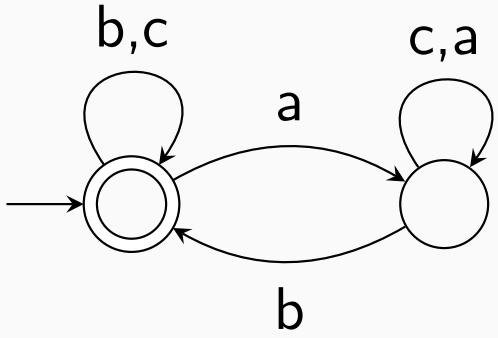
- S is a finite set of states and S_0 is a set of initial states
- Σ is an alphabet
- $R \colon S \times \Sigma \to 2^S$ is a transition function
- $F \subseteq S$ is a set of accepting states

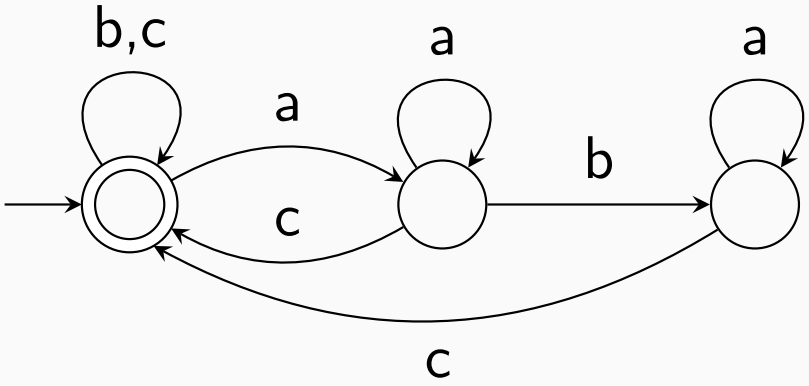
A run of a Büchi automaton on an ω-word is an infinite sequence of states and transitions. A run is accepting if it visits the set F infinitely often.

Examples









Büchi automaton

Büchi automaton: $\langle S, S_0, \Sigma, R, F \rangle$

A run is accepting if it visits the set F infinitely often.

NBAs and ω-regular languages

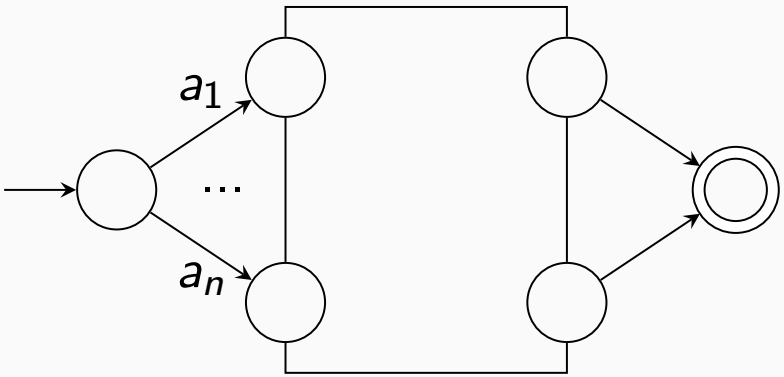
Theorem:

- For every ω-regular language L, there is an NBA $\mathcal{A}$ with $L = \mathcal{L}_\omega(\mathcal{A})$
- For every NBA $\mathcal{A}$, the language $\mathcal{L}_\omega(\mathcal{A})$ is ω-regular.

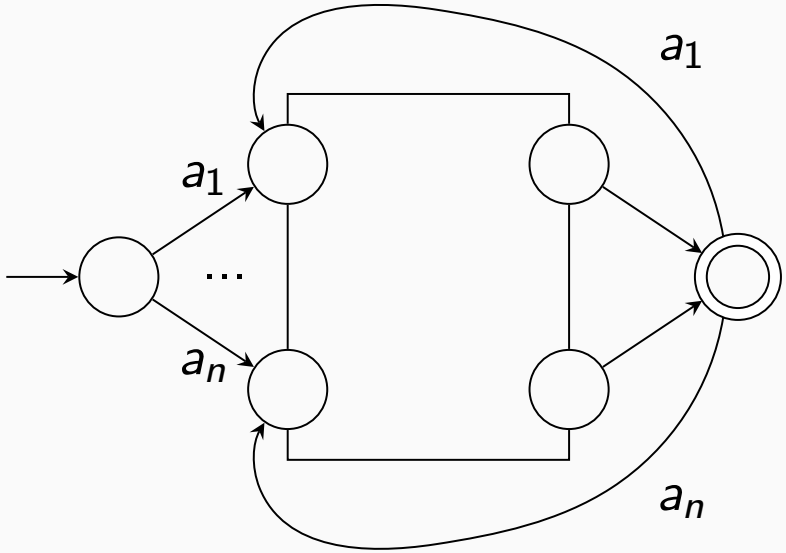
Proof: the constructions in the next slides:

- From ω-regular expressions to NBAs
- From NBAs to ω-regular expressions

From ω-regular expressions to NBAs

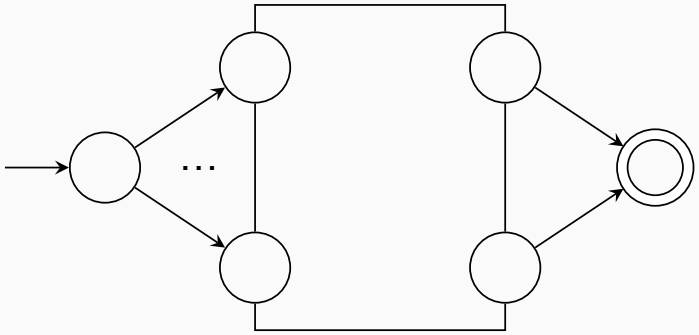


NFA for r

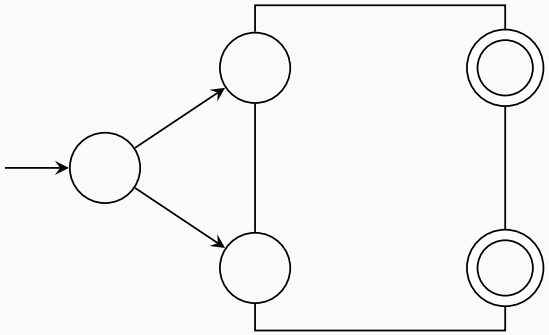


NBA for r^ω

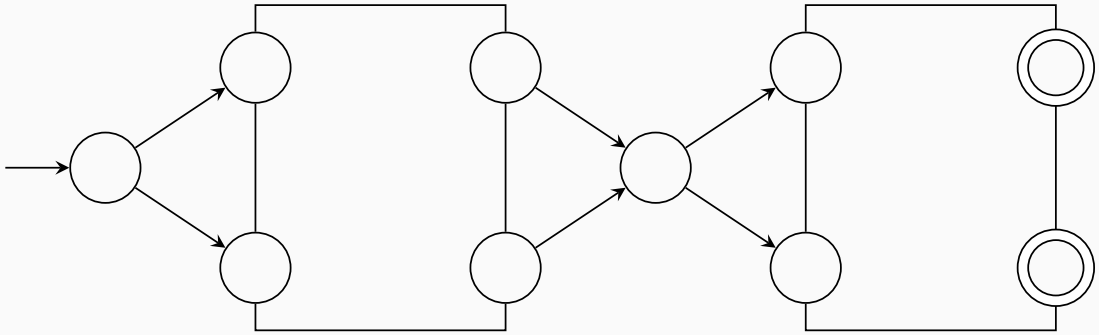
From ω-regular expressions to NBAs



NFA for r

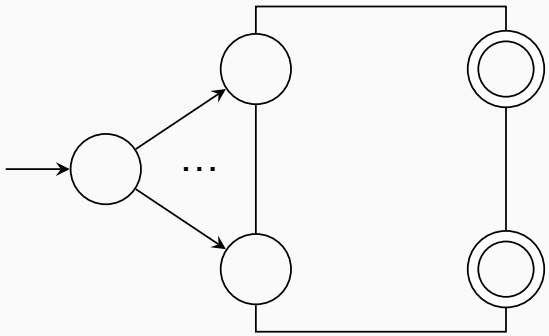


NBA for s

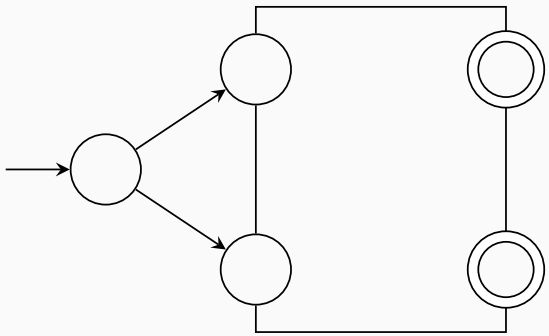


NBA for $r.s$

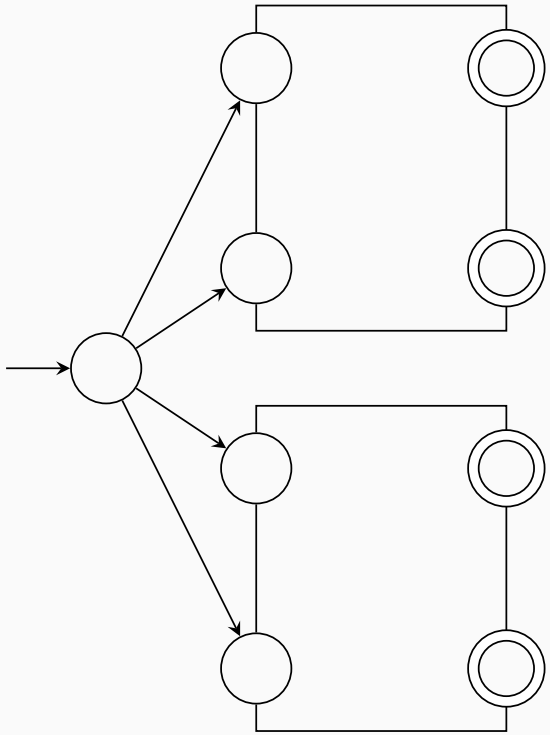
From ω-regular expressions to NBAs



NBA for s_1



NBA for s_2



NBA for $s_1 + s_2$

From NBAs to ω-regular expressions

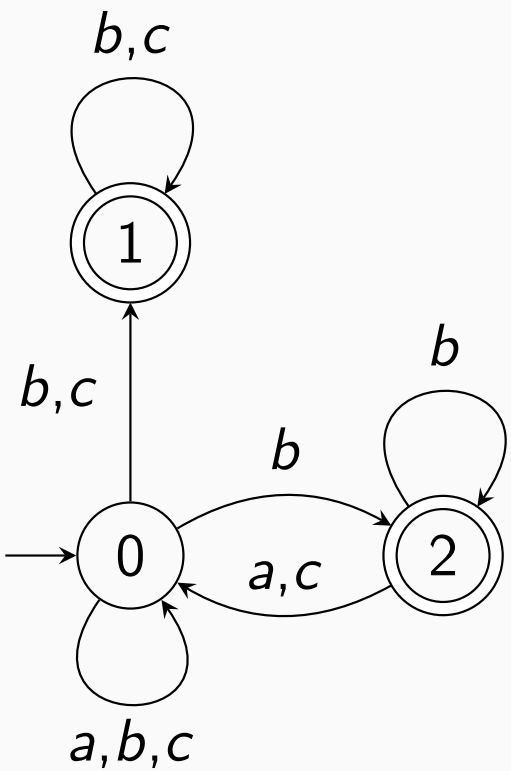
Intuition: build regular expressions for parts of the NBA: the expression from some initial state p to some accepting state q and the expression from some accepting state q to some accepting state q. The language $L_{p,q}$ of words with runs leading from p to q and visiting q exactly once is regular.

Let NBA $\mathcal{A} = \langle S, S_0, \Sigma, R, F \rangle$ and let $p, q \in S$ be states in $\mathcal{A}$. Define $\mathcal{A}_{p,q}$ as the NFA $\langle S, \{p\}, \Sigma, R, \{q\} \rangle$. Then:

$$\mathcal{L}_\omega(\mathcal{A}) = \bigcup_{p \in S_0} \bigcup_{q \in F} \mathcal{L}(\mathcal{A}_{p,q}).(\mathcal{L}(\mathcal{A}_{q,q}) \setminus \{\epsilon\})^\omega$$

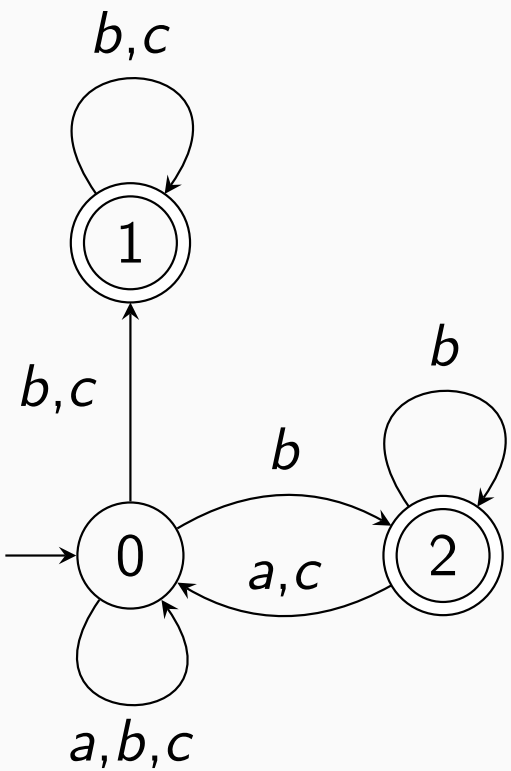
is ω-regular as $\mathcal{L}(\mathcal{A}_{p,q})$ and $\mathcal{L}(\mathcal{A}_{q,q})$ are regular.

Example



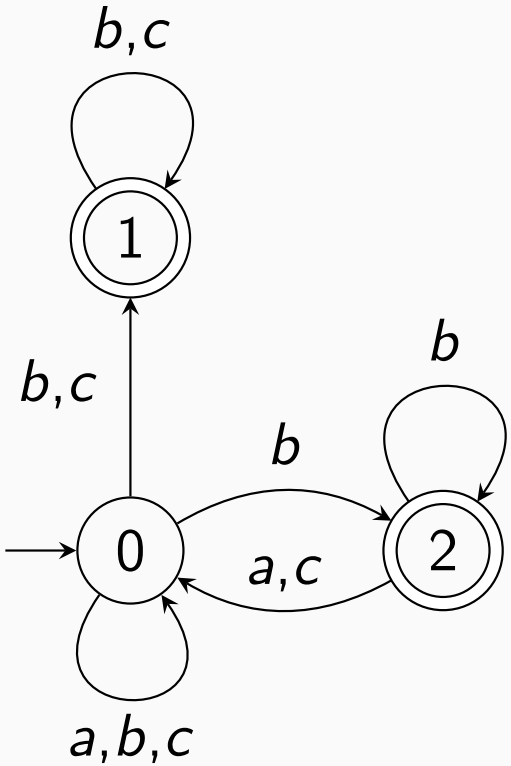
$$
\begin{aligned}
L_{0,1} &= \\
L_{1,1} &= \\
L_{0,2} &= \\
L_{2,2} &= \\
L_{\mathcal{A}} &=
\end{aligned}
$$

Example



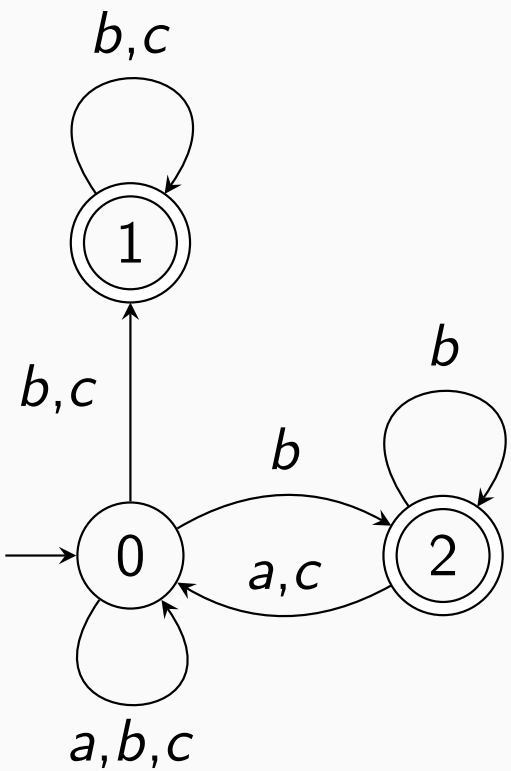
$$
\begin{aligned}
L_{0,1} &= (a+b+c)^*(b+c) \\
L_{1,1} &= \\
L_{0,2} &= \\
L_{2,2} &= \\
L_{\mathcal{A}} &=
\end{aligned}
$$

Example



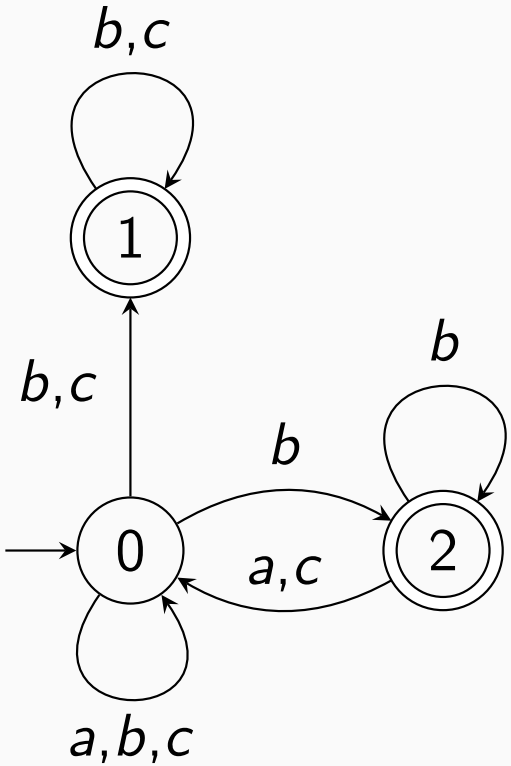
$$
\begin{aligned}
L_{0,1} &= (a+b+c)^*(b+c) \\
L_{1,1} &= b+c \\
L_{0,2} &= \\
L_{2,2} &= \\
L_{\mathcal{A}} &=
\end{aligned}
$$

Example



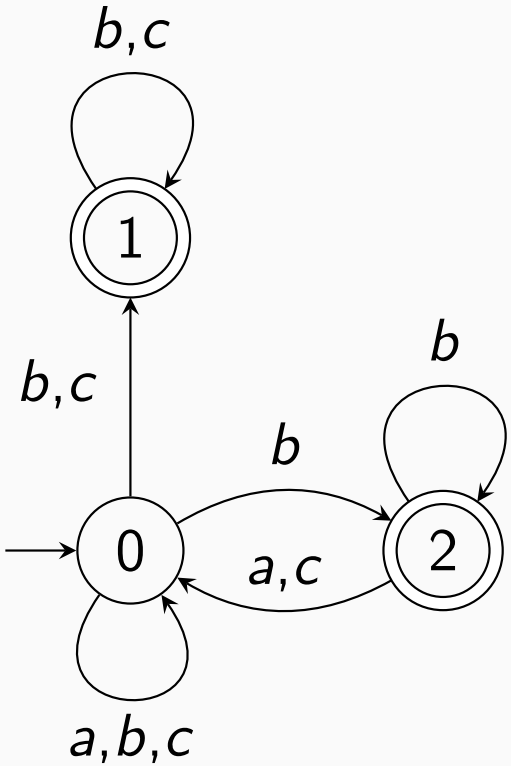
$$
\begin{aligned}
L_{0,1} &= (a+b+c)^*(b+c) \\
L_{1,1} &= b+c \\
L_{0,2} &= (a+b+c)^* b \\
L_{2,2} &= \\
L_{\mathcal{A}} &=
\end{aligned}
$$

Example



$$
\begin{aligned}
L_{0,1} &= (a+b+c)^*(b+c) \\
L_{1,1} &= b+c \\
L_{0,2} &= (a+b+c)^*b \\
L_{2,2} &= b+(a+c)(a+b+c)^*b \\
L_{\mathcal{A}} &=
\end{aligned}
$$

Example



$$
\begin{aligned}
L_{0,1} &= (a+b+c)^*(b+c) \\
L_{1,1} &= b+c \\
L_{0,2} &= (a+b+c)^* b \\
L_{2,2} &= b+(a+c)(a+b+c)^* b \\
L_{\mathcal{A}} &= L_{0,1}\left(L_{1,1}\right)^{\omega} + L_{0,2}\left(L_{2,2}\right)^{\omega}
\end{aligned}
$$

DBAs and ω-regular languages

- NFAs are as expressive as regular expressions
- DFAs are as expressive as regular expressions
- NBAs are as expressive as ω-regular expressions
- What about DBAs?

DBAs and ω-regular languages

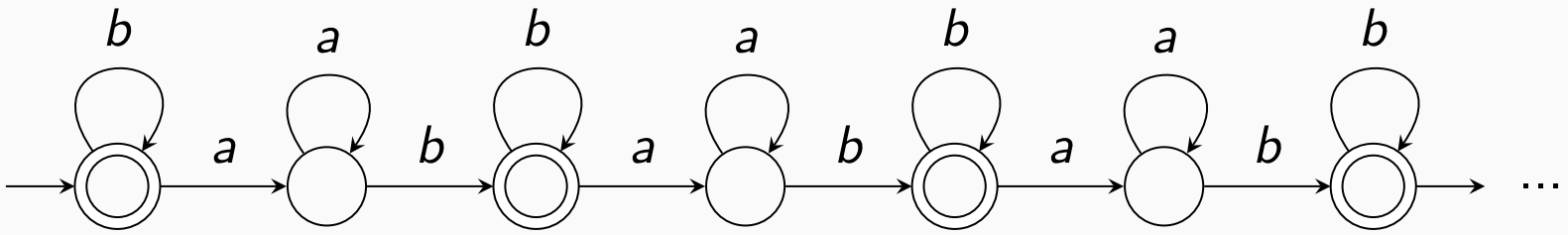
- NFAs are as expressive as regular expressions
- DFAs are as expressive as regular expressions
- NBAs are as expressive as ω-regular expressions
- What about DBAs?

Lemma: The language $(a + b)^* b^\omega$ is not recognized by any DBA.
Proof: By contradiction. Assume some DBA recognizes $(a + b)^* b^\omega$. Then it cannot be deterministic *and* have finitely many states.

DBAs and ω-regular languages

- NFAs are as expressive as regular expressions
- DFAs are as expressive as regular expressions
- NBAs are as expressive as ω-regular expressions
- What about DBAs?

Lemma: The language $(a + b)^* b^\omega$ is not recognized by any DBA.
Proof: By contradiction. Assume some DBA recognizes $(a + b)^* b^\omega$. Then it cannot be deterministic *and* have finitely many states.



Büchi automaton

Büchi automaton: $\langle S, S_0, \Sigma, R, F \rangle$

A run is accepting if it visits the set F infinitely often.

Generalized Büchi automaton

Generalized Büchi automaton: $\langle S, S_0, \Sigma, R, \mathcal{F} \rangle$

- $\mathcal{F}$ is a set of sets of accepting states $\{F_1, \ldots, F_n\}$
- A run is accepting if it visits each set of accepting states infinitely often
- Same expressive power as NBAs, but sometimes easier to use
  - From NBA to GNBA is trivial
  - From GNBA to NBA of size $|\mathcal{F}| \times |S|$

Other classes of ω-automata

DBA and NBA do not have the same expressive power.

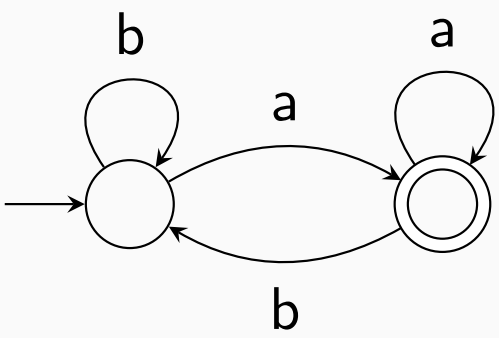
Question: Are there other classes of ω-automata with

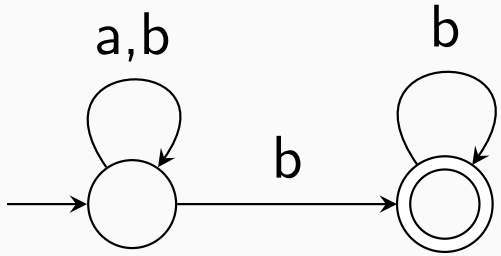
- the same expressive power as NBAs
- deterministic version has same expressive power

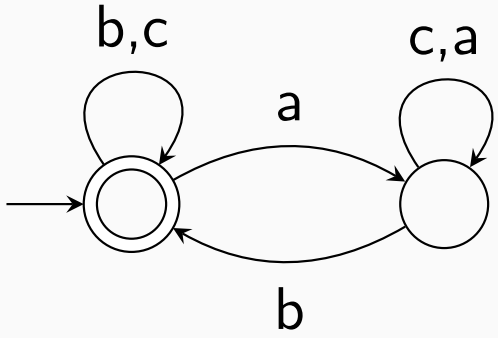
Co-Büchi automata

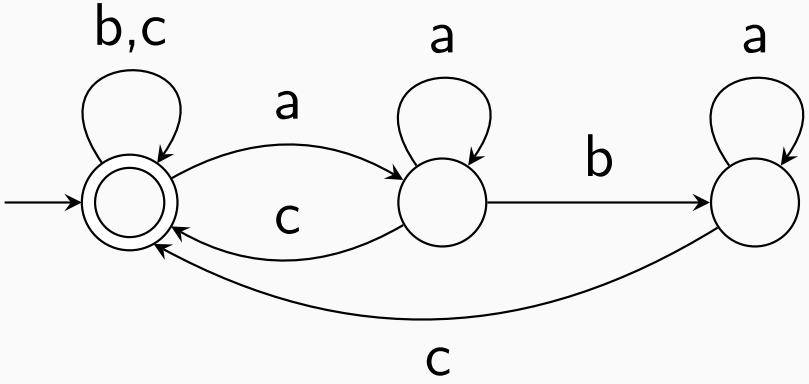
As Büchi automata, but a run is now accepting if and only if it visits accepting states finitely often.

Examples









Other classes of ω-automata

DCA and NCA do not have the same expressive power.

Question: Are there other classes of ω-automata with

- the same expressive power as NBAs
- deterministic version has same expressive power

Muller automata fulfill these properties, but have an exponential translation from NBAs

Question: Are there other classes of ω-automata with

- the same expressive power as NBAs
- deterministic version has same expressive power
- polynomial conversions to and from Büchi automata

Rabin automata

- The acceptance condition is a set of pairs

$$\{(F_1, G_1), \ldots, (F_n, G_n)\}$$

- A run ρ is accepting if there is a pair (F_i, G_i) such that ρ visits F_i infinitely often and G_i finitely often
- Conversion from NBA to NRA is trivial.
- A quadratic conversion from NRA to NBA exists.
- **Theorem** (Safra): Any NBA with n states can be transformed into a DRA with $n^{O(n)}$ states and $O(n)$ accepting pairs.

Rabin automata

- The acceptance condition is a set of pairs

$$\{(F_1, G_1), \ldots, (F_n, G_n)\}$$

- A run ρ is accepting if there is a pair (F_i, G_i) such that ρ visits F_i infinitely often and G_i finitely often
- Conversion from NBA to NRA is trivial.
- A quadratic conversion from NRA to NBA exists.
- **Theorem** (Safra): Any NBA with n states can be transformed into a DRA with $n^{O(n)}$ states and $O(n)$ accepting pairs.
- However: complementing a Rabin automaton increases its size exponentially

Other classes of ω-automata

Question: Are there other classes of ω-automata with

- the same expressive power as NBAs
- deterministic version has same expressive power
- polynomial conversions to and from Büchi automata
- no exponential blowup for complementation

Parity automata ("Chain" automata)

- The acceptance condition is a sequence $(F_1, \ldots, F_{2k})$ of sets of states such that $F_1 \subseteq F_2 \subseteq \cdots \subseteq F_{2k} = S$
- A run ρ is accepting if the minimal index i such that ρ visits the set F_i infinitely often is even
- NBA to NPA: $F \to (\emptyset, F, S, S)$
- NPA to NRA: $(F_1, \ldots, F_{2k}) \to \{(F_{2k}, F_{2k-1}), \ldots, (F_4, F_3), (F_2, F_1)\}$
- NPA to NBA: NPA to NRA to NBA
- **Theorem** (Safra, Piterman): Any NBA with n states can be transformed into a DPA with $n^{O(n)}$ states and $O(n)$ accepting sets.
- Complementation of DPAs: $(F_1, \ldots, F_{2k}) \to (\emptyset, F_1, \ldots, F_{2k}, S)$
- Also: memoryless strategies for both players

ω-Automata

- Finite automata accept/reject words of finite length
  - So an NFA accepts words that violate a regular safety property
- ω-Automata accept/reject words of infinite length (ω-words)
  - So an NBA accepts words that violate ω-regular properties
- Different ω-automata with different acceptance conditions
  - Büchi: a set of accepting states
  - Generalized Büchi: a set of sets of accepting states
  - Co-Büchi: a set of rejecting states
  - Rabin: a set of pairs of accepting/rejecting states
  - Parity: a sequence of sets / states labeled with priorities

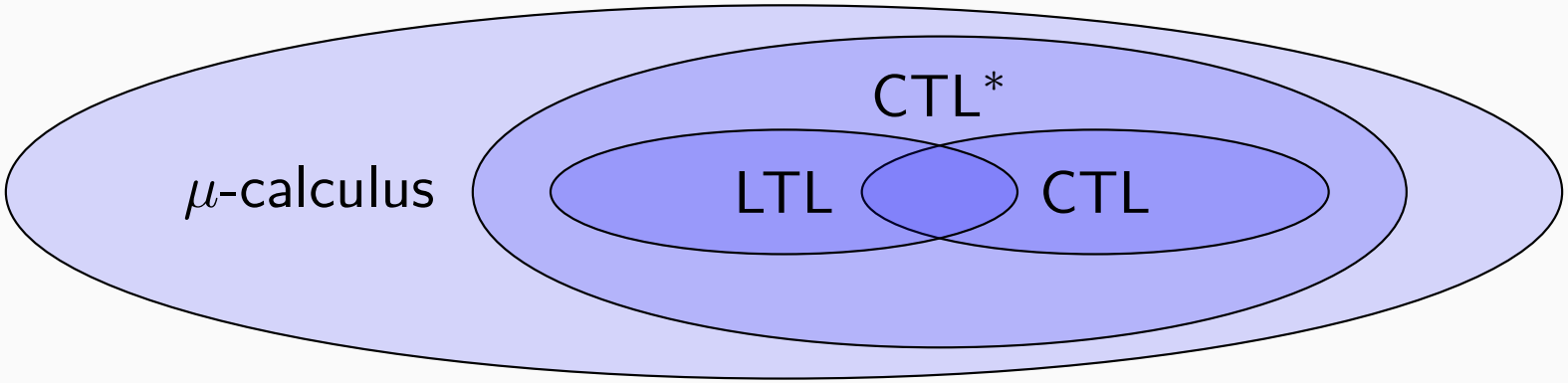
OUTLINE

1. Background / Applications of parity games
   - Ω-automata
   - Modal μ-calculus
   - LTL synthesis
2. Parity game solving algorithms
   - Introduction
   - Decomposition based algorithms
   - Local algorithms
3. Quasi-polynomial algorithms
   - Universal trees and progress measures
   - Ordered progress measures
   - Quasi-polynomial recursive algorithm
4. Final remarks

μ-CALCULUS: SYNTAX AND SEMANTICS

Idea of μ-calculus: add fixed point operators to basic modal logic.

- μ-calculus is very expressive (subsumes CTL, LTL, CTL*).
- μ-calculus is very pure ("assembly language" for modal logic, cf: λ-calculus for functional programming).
- drawback: lack of intuition.
- fragments of the μ-calculus are the basis for practical model checkers, such as μCRL, mCRL2, CADP, LTSmin



μ-CALCULUS: SYNTAX AND SEMANTICS

Elements of the language

- Constants tt and ff
- (Negated) propositions p and $\neg p$
- Operators conjunction $\wedge$ and disjunction $\vee$
- Modalities forall $\Box$ and exists $\Diamond$
- Variables $X \in \mathit{Var}$ representing sets of states
- Fixed point operators lfp $\mu X.\phi$ and gfp $\nu X.\phi$

Some notation and terminology

- The semantics of a μ-calculus formula ϕ is a set of states
- A valuation $\mathcal{V}\colon \mathit{Var} \to 2^S$ interprets variables as sets of states.
- $\mathcal{V}[X := Q]$ is a valuation like $\mathcal{V}$, but X is set to Q

μ-CALCULUS: SYNTAX AND SEMANTICS

Examples

- $\nu X.p \wedge \Box X$ is analogous to LTL **G** p
- $\mu X.p \vee (\Diamond tt \wedge \Box X)$ is analogous to LTL **F** p
- $\mu X.\nu Y.\Box X \vee (q \wedge \Box Y)$ is analogous to LTL **F G** q
- $\nu X.p \wedge \Diamond\Diamond X$ has no analogous LTL formula

μ-CALCULUS: SYNTAX AND SEMANTICS

Syntax

$$\phi, \psi ::= tt \mid ff \mid p \mid \neg p \mid \phi \wedge \psi \mid \phi \vee \psi \mid [a]\phi \mid \langle a \rangle \phi \mid X \mid \mu X.\phi \mid \nu X.\phi$$

Semantics for a transition system $\mathcal{M} := (S, \rightarrow, L)$

$$\begin{array}{lcl}
[\![tt]\!]^{\mathcal{M}} & = & S \\
[\![ff]\!]^{\mathcal{M}} & = & \emptyset \\
[\![p]\!]^{\mathcal{M}} & = & \{s \in S \mid p \in L(s)\} \\
[\![\neg p]\!]^{\mathcal{M}} & = & \{s \in S \mid p \notin L(s)\} \\
[\![\phi \vee \psi]\!]^{\mathcal{M}} & = & [\![\phi]\!]^{\mathcal{M}} \cup [\![\psi]\!]^{\mathcal{M}} \\
[\![\phi \wedge \psi]\!]^{\mathcal{M}} & = & [\![\phi]\!]^{\mathcal{M}} \cap [\![\psi]\!]^{\mathcal{M}}
\end{array}$$

(notice that there is no negation on formulae, only on the propositions)

μ-CALCULUS: SYNTAX AND SEMANTICS

Syntax

$$\phi, \psi ::= tt \mid ff \mid p \mid \neg p \mid \phi \wedge \psi \mid \phi \vee \psi \mid [a]\phi \mid \langle a \rangle \phi \mid X \mid \mu X.\phi \mid \nu X.\phi$$

Semantics for a transition system $\mathcal{M} := (S, \rightarrow, L)$

$$\begin{array}{lcl}
[\![tt]\!]^{\mathcal{M}} & = & S \\
[\![ff]\!]^{\mathcal{M}} & = & \emptyset \\
[\![p]\!]^{\mathcal{M}} & = & \{s \in S \mid p \in L(s)\} \\
[\![\neg p]\!]^{\mathcal{M}} & = & \{s \in S \mid p \notin L(s)\} \\
[\![\phi \vee \psi]\!]^{\mathcal{M}} & = & [\![\phi]\!]^{\mathcal{M}} \cup [\![\psi]\!]^{\mathcal{M}} \\
[\![\phi \wedge \psi]\!]^{\mathcal{M}} & = & [\![\phi]\!]^{\mathcal{M}} \cap [\![\psi]\!]^{\mathcal{M}} \\
[\![[a]\phi]\!]^{\mathcal{M}} & = & \{s \in S \mid \forall t.(s \xrightarrow{a} t) \rightarrow (t \in [\![\phi]\!]^{\mathcal{M}})\} \\
[\![\langle a \rangle \phi]\!]^{\mathcal{M}} & = & \{s \in S \mid \exists t.(s \xrightarrow{a} t) \wedge (t \in [\![\phi]\!]^{\mathcal{M}})\}
\end{array}$$

μ-CALCULUS: SYNTAX AND SEMANTICS

Syntax

$$\phi, \psi ::= tt \mid ff \mid p \mid \neg p \mid \phi \wedge \psi \mid \phi \vee \psi \mid [a]\phi \mid \langle a \rangle \phi \mid X \mid \mu X.\phi \mid \nu X.\phi$$

Semantics for a transition system $\mathcal{M} := (S, \rightarrow, L)$

$$\begin{array}{lcll}
[\![tt]\!]^{\mathcal{M}} & = & S & \\
[\![ff]\!]^{\mathcal{M}} & = & \emptyset & \\
[\![p]\!]^{\mathcal{M}} & = & \{s \in S \mid p \in L(s)\} & \\
[\![\neg p]\!]^{\mathcal{M}} & = & \{s \in S \mid p \notin L(s)\} & \\
[\![\phi \vee \psi]\!]^{\mathcal{M}}_{\mathcal{V}} & = & [\![\phi]\!]^{\mathcal{M}}_{\mathcal{V}} \cup [\![\psi]\!]^{\mathcal{M}}_{\mathcal{V}} & \\
[\![\phi \wedge \psi]\!]^{\mathcal{M}}_{\mathcal{V}} & = & [\![\phi]\!]^{\mathcal{M}}_{\mathcal{V}} \cap [\![\psi]\!]^{\mathcal{M}}_{\mathcal{V}} & \\
[\![[a]\phi]\!]^{\mathcal{M}}_{\mathcal{V}} & = & \{s \in S \mid \forall t.(s \xrightarrow{a} t) \rightarrow (t \in [\![\phi]\!]^{\mathcal{M}}_{\mathcal{V}})\} & \\
[\![\langle a \rangle \phi]\!]^{\mathcal{M}}_{\mathcal{V}} & = & \{s \in S \mid \exists t.(s \xrightarrow{a} t) \wedge (t \in [\![\phi]\!]^{\mathcal{M}}_{\mathcal{V}})\} & \\
[\![X]\!]^{\mathcal{M}}_{\mathcal{V}} & = & \mathcal{V}(X) & \\
[\![\mu X.\phi]\!]^{\mathcal{M}}_{\mathcal{V}} & = & \bigcap\{S' \subseteq S \mid [\![\phi]\!]^{\mathcal{M}}_{\mathcal{V}[S'/X]} \subseteq S'\} & \text{(lfp)} \\
[\![\nu X.\phi]\!]^{\mathcal{M}}_{\mathcal{V}} & = & \bigsqcup\{S' \subseteq S \mid S' \subseteq [\![\phi]\!]^{\mathcal{M}}_{\mathcal{V}[S'/X]}\} & \text{(gfp)}
\end{array}$$

where $\mathcal{V}\colon \mathrm{Var} \rightarrow 2^S$ assigns a set of states to the variables $X, Y, \ldots$

μ-CALCULUS: SYNTAX AND SEMANTICS

Syntax

$$\phi, \psi ::= tt \mid ff \mid p \mid \neg p \mid \phi \wedge \psi \mid \phi \vee \psi \mid [a]\phi \mid \langle a \rangle \phi \mid X \mid \mu X.\phi \mid \nu X.\phi$$

Semantics for a transition system $\mathcal{M} := (S, \rightarrow, L)$

$$\begin{array}{lcl}
[\![tt]\!]^{\mathcal{M}} & = & S \\
[\![ff]\!]^{\mathcal{M}} & = & \emptyset \\
[\![p]\!]^{\mathcal{M}} & = & \{s \in S \mid p \in L(s)\} \\
[\![\neg p]\!]^{\mathcal{M}} & = & \{s \in S \mid p \notin L(s)\} \\
[\![\phi \vee \psi]\!]^{\mathcal{M}}_{\mathcal{V}} & = & [\![\phi]\!]^{\mathcal{M}}_{\mathcal{V}} \cup [\![\psi]\!]^{\mathcal{M}}_{\mathcal{V}} \\
[\![\phi \wedge \psi]\!]^{\mathcal{M}}_{\mathcal{V}} & = & [\![\phi]\!]^{\mathcal{M}}_{\mathcal{V}} \cap [\![\psi]\!]^{\mathcal{M}}_{\mathcal{V}} \\
[\![[a]\phi]\!]^{\mathcal{M}}_{\mathcal{V}} & = & \{s \in S \mid \forall t.(s \xrightarrow{a} t) \rightarrow (t \in [\![\phi]\!]^{\mathcal{M}}_{\mathcal{V}})\} \\
[\![\langle a \rangle \phi]\!]^{\mathcal{M}}_{\mathcal{V}} & = & \{s \in S \mid \exists t.(s \xrightarrow{a} t) \wedge (t \in [\![\phi]\!]^{\mathcal{M}}_{\mathcal{V}})\} \\
[\![X]\!]^{\mathcal{M}}_{\mathcal{V}} & = & \mathcal{V}(X) \\
[\![\mu X.\phi]\!]^{\mathcal{M}}_{\mathcal{V}} & = & \mathrm{fp}(X := [\![\phi]\!]^{\mathcal{M}}_{\mathcal{V}[X \leftarrow x]}, \emptyset) \\
[\![\nu X.\phi]\!]^{\mathcal{M}}_{\mathcal{V}} & = & \mathrm{fp}(X := [\![\phi]\!]^{\mathcal{M}}_{\mathcal{V}[X \leftarrow x]}, S)
\end{array}$$

where $\mathcal{V}\colon \mathrm{Var} \rightarrow 2^S$ assigns a set of states to the variables $X, Y, \ldots$

NOTES ON μ-CALCULUS

- Increasing complexity with alternation of fixed point types
  - With one fix-point we talk about termination properties
  - With two fix-points we can write fairness formulas
  - CTL*/LTL operators require at most alternation depth 2
- See also Chapter 26 of the Handbook of Model Checking (2018)

PARITY GAMES AND THE MODAL μ-CALCULUS

Linear from μ-calculus to parity games

Structural decomposition of ϕ.

- Stirling (1995): Local Model Checking Games. In CONCUR.
- Emerson, Jutla & Sistla (2001): On model checking for the mu-calculus and its fragments. In TCS.

Linear from parity games to μ-calculus

Construction of a μ-calculus formula of the winning condition.

- Walukiewicz (1996): Monadic Second Order Logic on Tree-Like Structures. In STACS.

Hence: the modal μ-calculus is at least as expressive as ω-regular expressions.

OUTLINE

1. Background / Applications of parity games
   - Ω-automata
   - Modal μ-calculus
   - LTL synthesis
2. Parity game solving algorithms
   - Introduction
   - Decomposition based algorithms
   - Local algorithms
3. Quasi-polynomial algorithms
   - Universal trees and progress measures
   - Ordered progress measures
   - Quasi-polynomial recursive algorithm
4. Final remarks

LTL synthesis

Given an LTL formula ϕ over AP and a partition of AP into controllable and uncontrollable propositions AP_C and AP_U, synthesize a controller such that ϕ holds for all choices of the environment on AP_U.

LTL SYNTHESIS

LTL synthesis

Given an LTL formula ϕ over AP and a partition of AP into controllable and uncontrollable propositions AP_C and AP_U, synthesize a controller such that ϕ holds for all choices of the environment on AP_U.

Controller semantics

- Mealy machine: outputs based on input and state
  - environment moves first
  - controller reacts to ensure ϕ
- Moore machine: output based on state
  - environment moves last
  - controller ensures ϕ regardless of environment response

LTL SYNTHESIS

LTL synthesis

Given an LTL formula ϕ over AP and a partition of AP into controllable and uncontrollable propositions AP_C and AP_U, synthesize a controller such that ϕ holds for all choices of the environment on AP_U.

LTL to automata

1. Obtain an automaton that accepts the language of ϕ
   - Use a product automaton to synthesize a controller that is embedded in a larger system

LTL SYNTHESIS

LTL synthesis

Given an LTL formula ϕ over AP and a partition of AP into controllable and uncontrollable propositions AP_C and AP_U, synthesize a controller such that ϕ holds for all choices of the environment on AP_U.

LTL to automata

1. Obtain an automaton that accepts the language of ϕ
   - Use a product automaton to synthesize a controller that is embedded in a larger system
2. Split the transitions to obtain a game
   - Every transition $p \to q$ has a valuation of $AP_C \cup AP_U$
   - Mealy: environment chooses a transition on AP_U from p to an intermediate vertex v; controller chooses a transition on AP_C from v to q
   - Moore: controller chooses a transition on AP_C from p to an intermediate vertex v; environment chooses a transition on AP_U from v to q

LTL SYNTHESIS

LTL synthesis

Given an LTL formula ϕ over AP and a partition of AP into controllable and uncontrollable propositions AP_C and AP_U, synthesize a controller such that ϕ holds for all choices of the environment on AP_U.

LTL to automata

1. Obtain an automaton that accepts the language of ϕ
2. Split the transitions to obtain a game

The automaton must be deterministic! Why?

LTL SYNTHESIS

LTL synthesis

Given an LTL formula ϕ over AP and a partition of AP into controllable and uncontrollable propositions AP_C and AP_U, synthesize a controller such that ϕ holds for all choices of the environment on AP_U.

LTL to automata

1. Obtain an automaton that accepts the language of ϕ
2. Split the transitions to obtain a game

The automaton must be deterministic! Why?

- Otherwise conflicting strategies!
  (Sufficient if the environment player is deterministic)
- DBA not sufficient (Why?)
- Constructions exist from NBA to DPA, DRA
- DPA is ideal: solving a PG gives a positional strategy

LTL SYNTHESIS

LTL synthesis

Given an LTL formula ϕ over AP and a partition of AP into controllable and uncontrollable propositions AP_C and AP_U, synthesize a controller such that ϕ holds for all choices of the environment on AP_U.

Typical procedure

1. Construct a NBA for ϕ (using the traditional approach, or using the construction to limit-deterministic NBA)
2. Compute a product automaton (also an NBA)
3. Construct a DPA from the NBA
4. Split the parity automaton into a parity game
5. Solve the parity game (realizability)
6. Use the strategy to construct the controller (synthesis)

OUTLINE

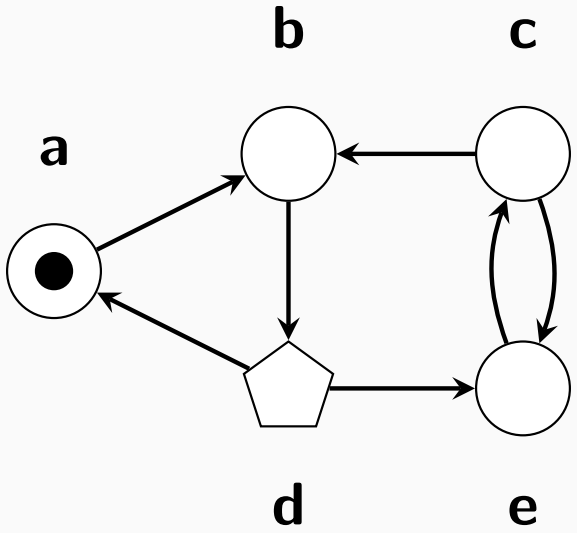
1. Background / Applications of parity games
   - Ω-automata
   - Modal μ-calculus
   - LTL synthesis
2. Parity game solving algorithms
   - Introduction
   - Decomposition based algorithms
   - Local algorithms
3. Quasi-polynomial algorithms
   - Universal trees and progress measures
   - Ordered progress measures
   - Quasi-polynomial recursive algorithm
4. Final remarks

OUTLINE

1. Background / Applications of parity games
   - Ω-automata
   - Modal μ-calculus
   - LTL synthesis
2. Parity game solving algorithms
   - Introduction
   - Decomposition based algorithms
   - Local algorithms
3. Quasi-polynomial algorithms
   - Universal trees and progress measures
   - Ordered progress measures
   - Quasi-polynomial recursive algorithm
4. Final remarks

PARITY GAMES

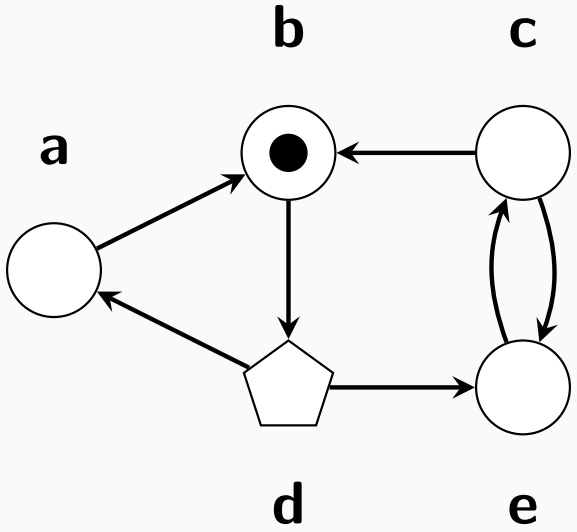
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a**

PARITY GAMES

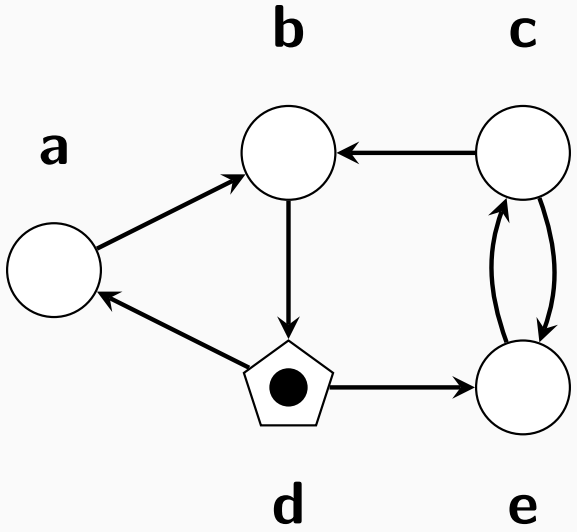
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b**

PARITY GAMES

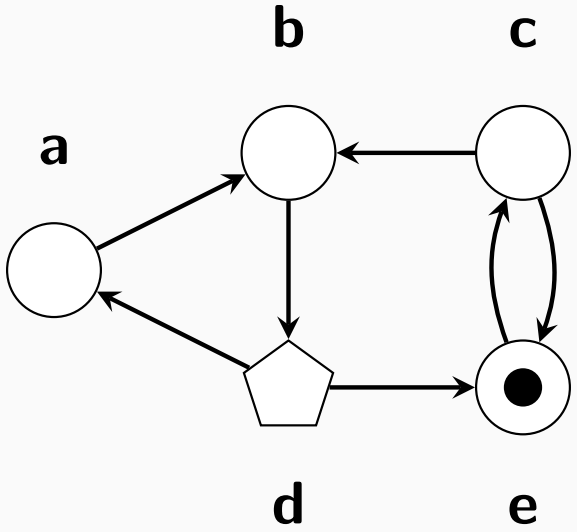
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d**

PARITY GAMES

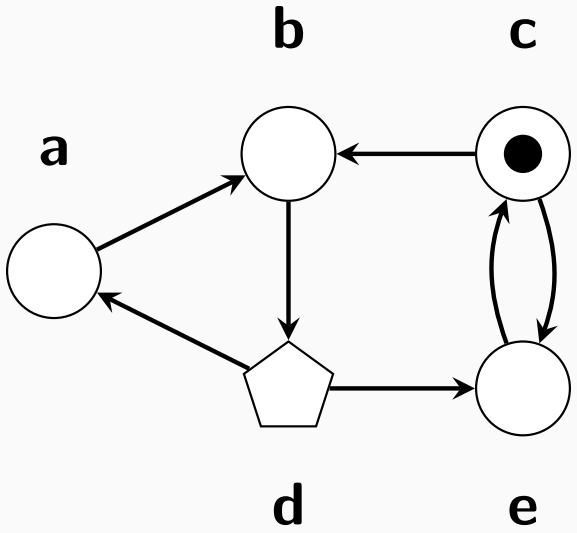
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e**

PARITY GAMES

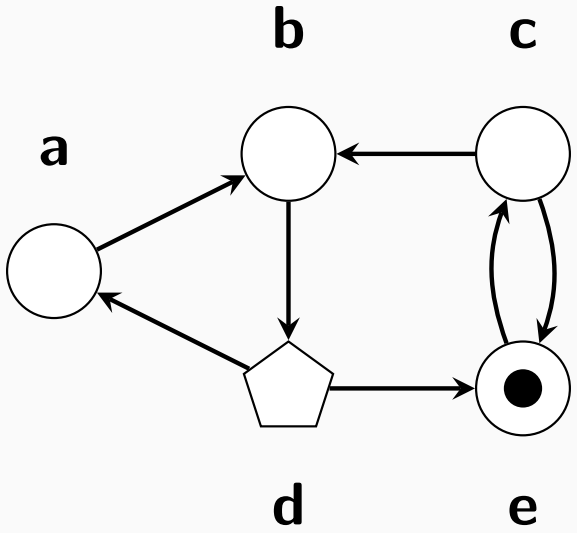
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c**

PARITY GAMES

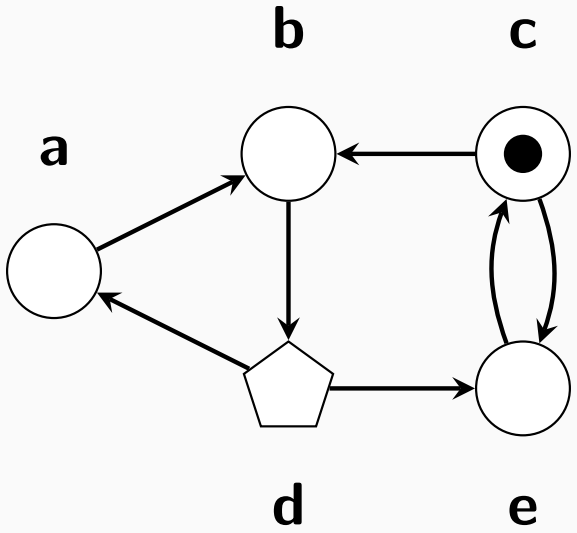
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e**

PARITY GAMES

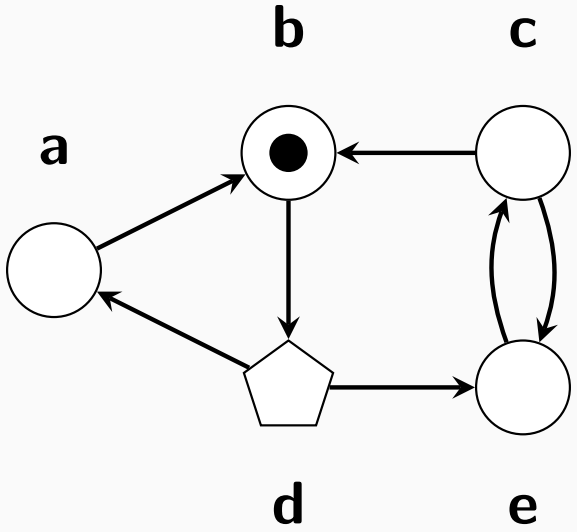
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c**

PARITY GAMES

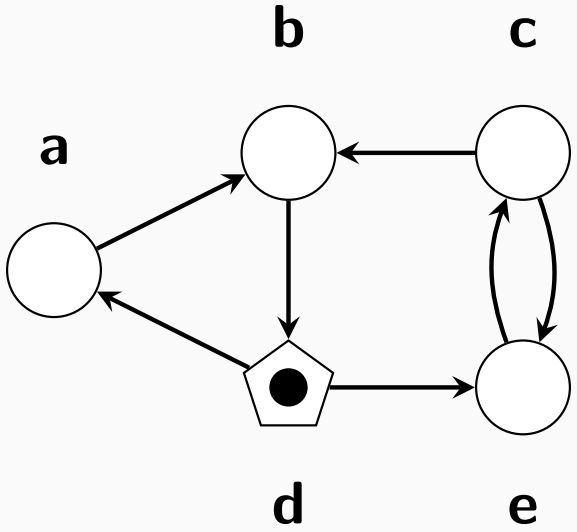
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b**

PARITY GAMES

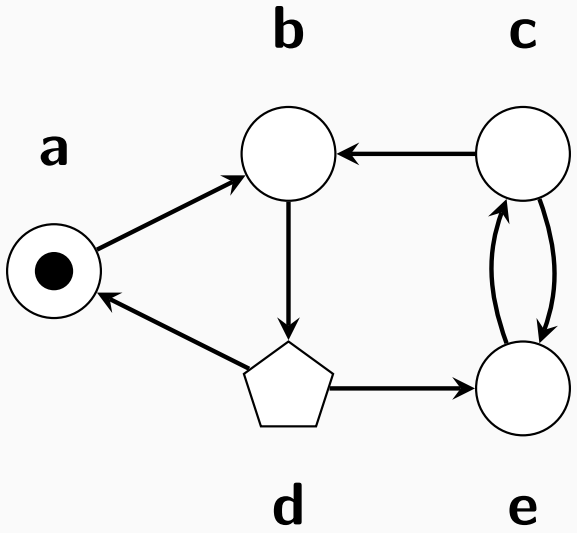
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b d**

PARITY GAMES

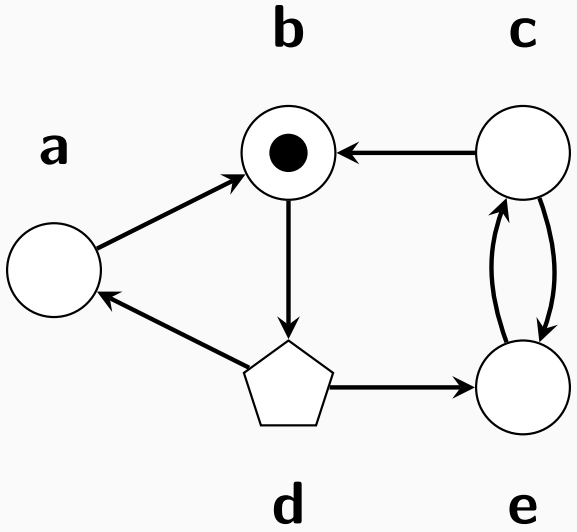
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b d a**

PARITY GAMES

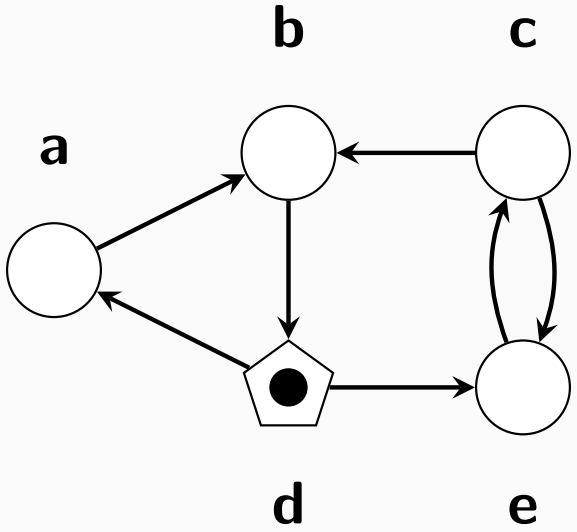
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b d a b**

PARITY GAMES

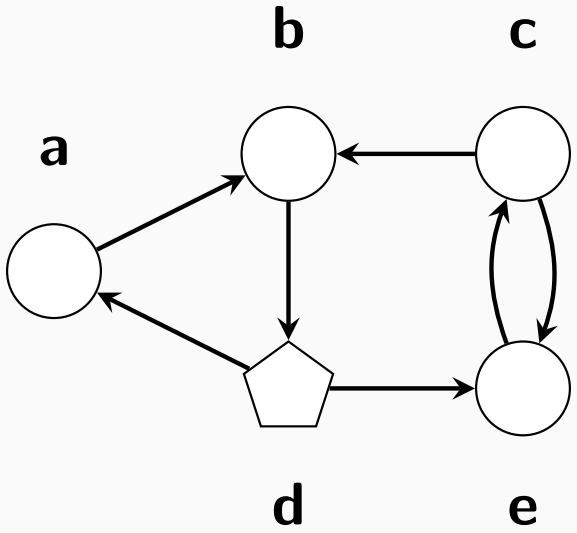
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b** $(\mathbf{d\ a\ b})^{\omega}$

PARITY GAMES

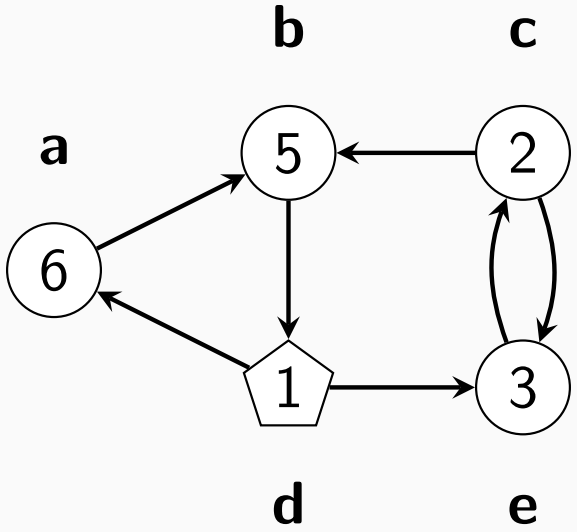
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



Who wins a play?

PARITY GAMES

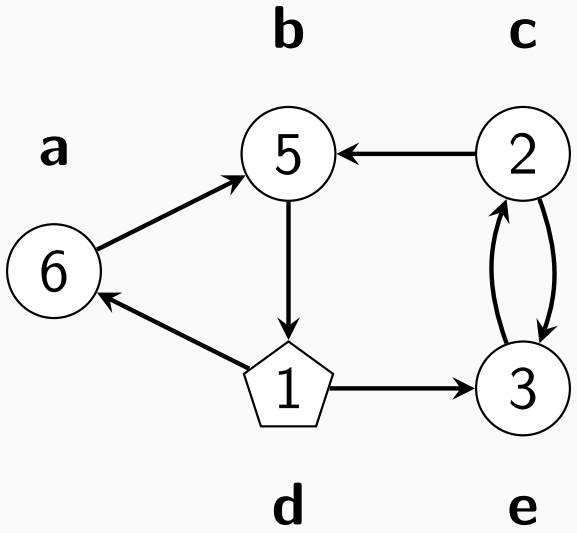
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



- Each vertex has a priority $\{0, 1, 2, \ldots, d\}$
- Player Even wins if the highest priority seen infinitely often is even

PARITY GAMES

- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor

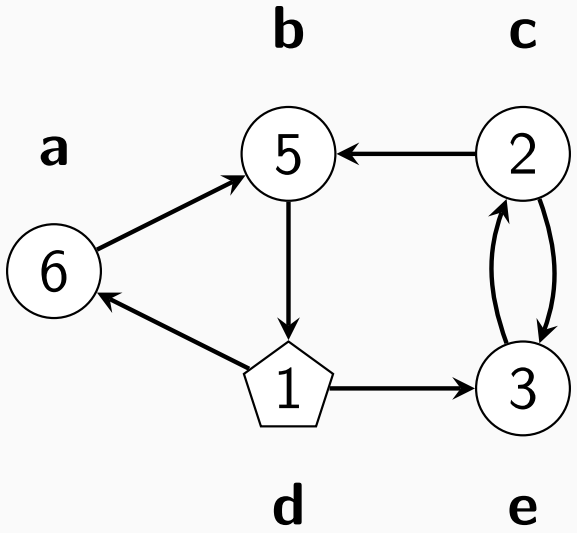


The play π: 6 5 1 3 2 3 2 5 $(1\ 6\ 5)^{\omega}$

Who wins this play?

PARITY GAMES

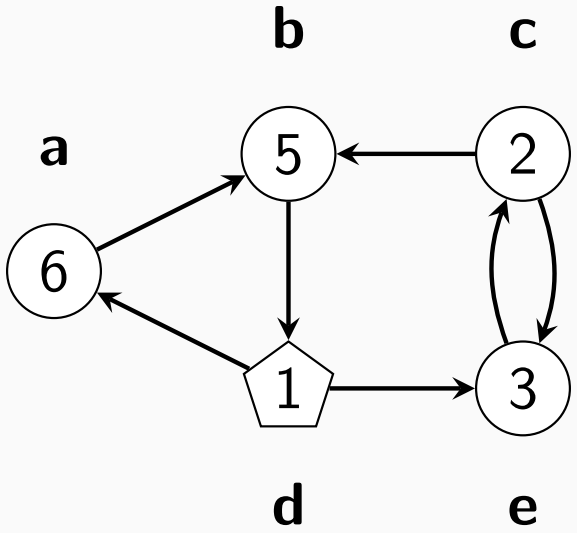
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



How do we determine who wins a vertex?

PARITY GAMES

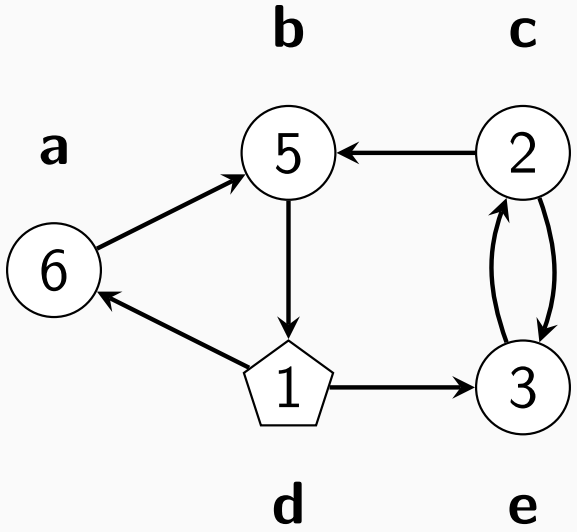
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



A player wins a vertex v if they have a positional strategy σ such that every play from v consistent with σ is won by that player.

PARITY GAMES

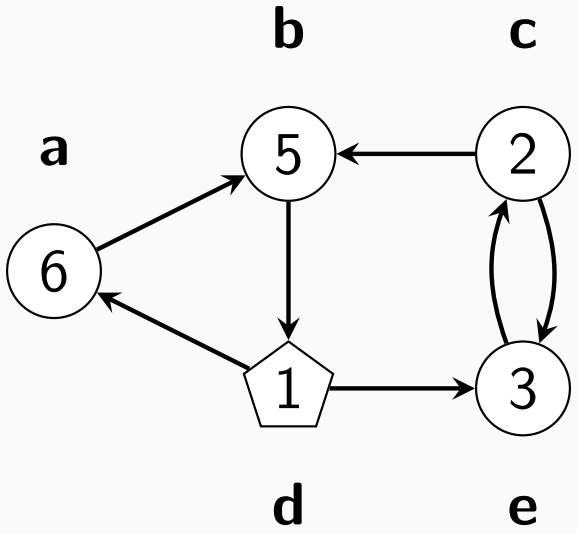
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



A player wins a (subgame) $\mathcal{G}$ if they have a positional strategy σ such that all cycles in the induced graph $\mathcal{G}[\sigma]$ are won by that player

PARITY GAMES

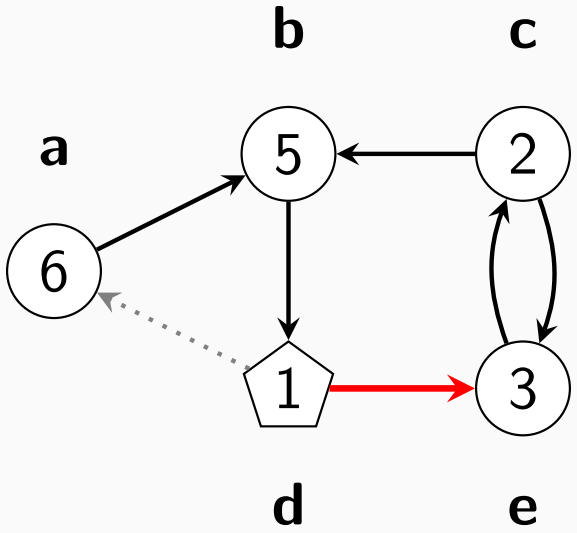
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



Which vertices are won by which player?

PARITY GAMES

- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor

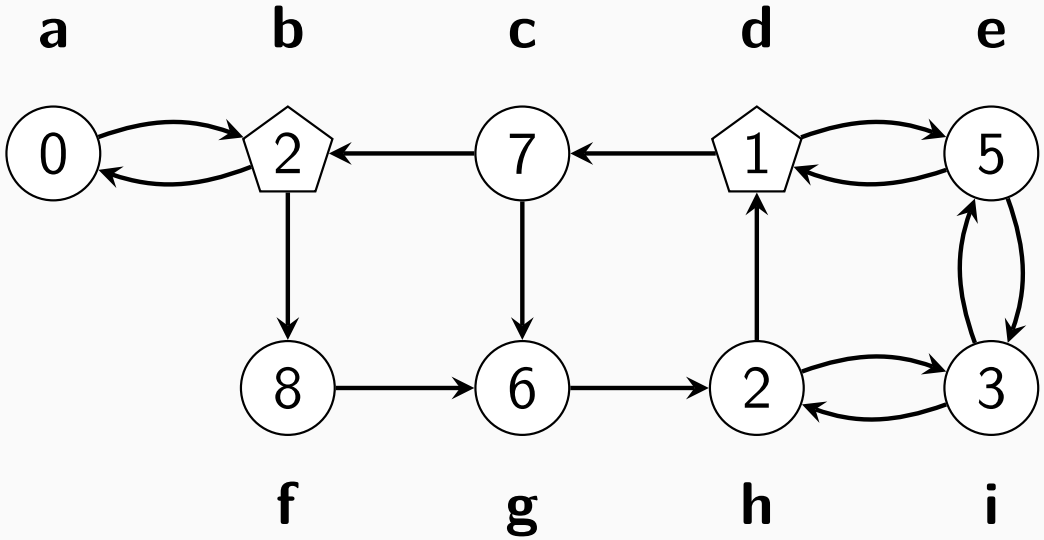


Player Odd wins with strategy $\{\, \mathbf{d} \to \mathbf{e} \,\}$
Only two (Odd) cycles remain

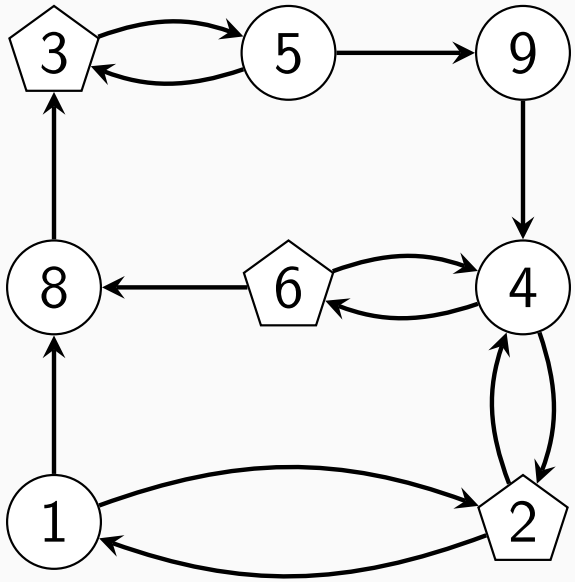
Solving a parity game

- Compute the winning regions $W_{\bigcirc}$ and $W_{\pentagon}$
- Compute the winning positional strategies $\sigma_{\bigcirc}$ and $\sigma_{\pentagon}$

EXAMPLE PARITY GAME



EXAMPLE PARITY GAME



COMPLEXITY OF SOLVING PARITY GAMES

- Deciding the winning regions $W_{\bigcirc}$ and $W_{\pentagon}$ is in NP $\cap$ co-NP
- Deciding the winning regions $W_{\bigcirc}$ and $W_{\pentagon}$ is in UP $\cap$ co-UP
  - Reduction to simple stochastic games (which are in UP $\cap$ co-UP)
  - Also contains the integer factorization problem
  - P is strictly contained in UP is strictly contained in NP
  - Unambiguous NP: NP with a *unique* solution
- Open problem: is solving parity games in P?
  - Is model checking of the modal μ-calculus in P?
  - Open for 30/40 years (Emerson, Jutla, Sistla 1993, Kozen 1983)

OINK

- Modern (parallel) implementation of parity game algorithms
  - Strategy Improvement (parallel)
  - Distraction fixed point iteration (parallel)
  - Recursive algorithm (parallel attractor)
  - Priority promotion
  - Tangle learning
  - Small progress measures (with optimizations)
  - Succinct progress measures
  - Ordered progress measures
- The usual preprocessing algorithms
  - Inflation and compression
  - Removing self-loops
  - Detecting winner-controlled winning cycles
  - SCC decomposition
- `https://www.github.com/trolando/oink`
- Simple to use/extend library in C++

OINK

Easy to use

```
#include "oink.hpp"

pg::Game parity_game;
parity_game.parse_pgsolver(cin);

pg::Oink solver(parity_game);
solver.setSolver("tl");
solver.run();

parity_game.write_sol(cout);
```

Easy to extend

- Implement Solver interface
- Add one line to `solvers.cpp`

Notation for parity games

- A parity game $\mathcal{G}$ is a tuple $(V_{\bigcirc}, V_{\pentagon}, E, \mathrm{pr})$
- Vertices $V = V_{\bigcirc} \cup V_{\pentagon}$ controlled by players Even and Odd
- Transitions E: $V \times V$ such that E is left-total.
  - We write $u \to v$ for $(u, v) \in E$
  - $E(u)$ denotes the successors of u: $\{v \mid u \to v\}$
  - $E(U)$ denotes all successors of vertices in $\{v \mid u \to v \mid u \in U\}$
  - Each vertex has at least one successor.
- Priority function $\mathrm{pr}\colon V \to \{0, 1, 2, \ldots, d\}$
- A play is an infinite sequence $v_0 v_1 v_2 \ldots$ consistent with E
- A play π is won by player Even iff $\max(\mathrm{pr}(\inf \pi))$ is even

Notation for strategies

- A strategy for player α is a function $\sigma\colon V_\alpha \to V$ that assigns one successor to each vertex of player α.
- A play is consistent with σ if each v_i for which $\sigma(v_i)$ is defined, $v_{i+1} = \sigma(v_i)$.
- $\mathrm{Plays}(v, \sigma)$ is the set of plays consistent with σ starting in v.
- σ is a winning strategy from v for player α if all plays in $\mathrm{Plays}(v, \sigma)$ are winning for α

Notation for closed sets and dominions

- A set W is closed w.r.t. a strategy σ if for all $v \in W$:
  - if v is owned by α, then $\sigma(v) \in W$ (strategy in W)
  - if v is owned by $\overline{\alpha}$, then $E(v) \subseteq W$ (all successors in W)
- A set D is a dominion of player α if α has a strategy σ that is winning for all $v \in D$ and D is closed w.r.t. σ.
- The winning regions of either player are dominions.

ATTRACTOR COMPUTATION

Playing from **A** to **B**

- How to show that a player $\alpha \in \{\bullet, \pentagon\}$ wins?

  or...

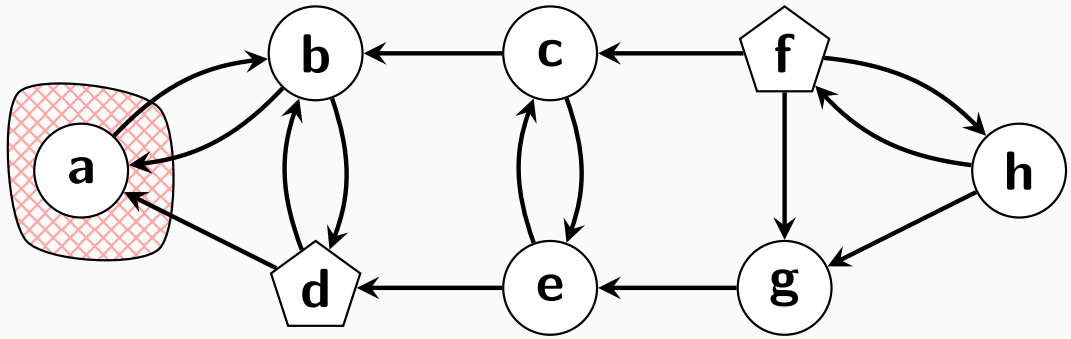
- How to show that the opponent $\overline{\alpha}$ must play to certain "bad" vertices?

Attractor computation

- "Backward reachability but in a game"
- Given target set Z and a player α,
  compute all vertices from which player α can ensure arrival in Z
- Add to Z (exhaustively):
  - all vertices of the player α that *can* play to Z
  - all vertices of opponent $\overline{\alpha}$ that *must* play to Z

Example of attractor computation

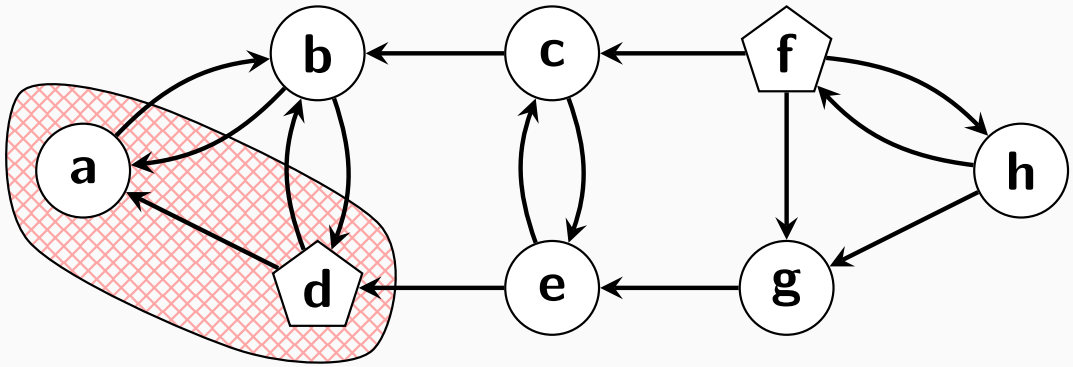
Computing the ⬟-attractor to **a**



Attractor set: $\{\mathbf{a}\}$
Can attract: **d** but not **b**

Example of attractor computation

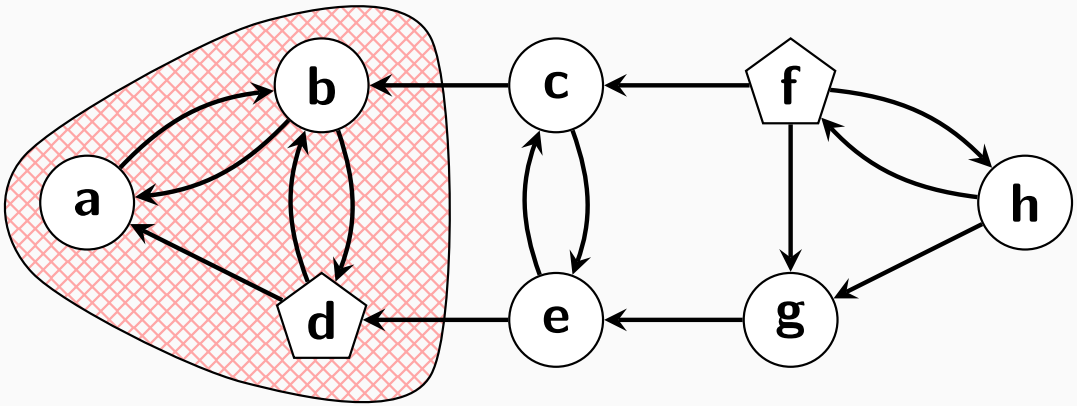
Computing the ⬠-attractor to **a**



Attractor set: $\{\mathbf{a}\}$, $\{\mathbf{a}, \mathbf{d}\}$
Can attract: **b** but not **e**

Example of attractor computation

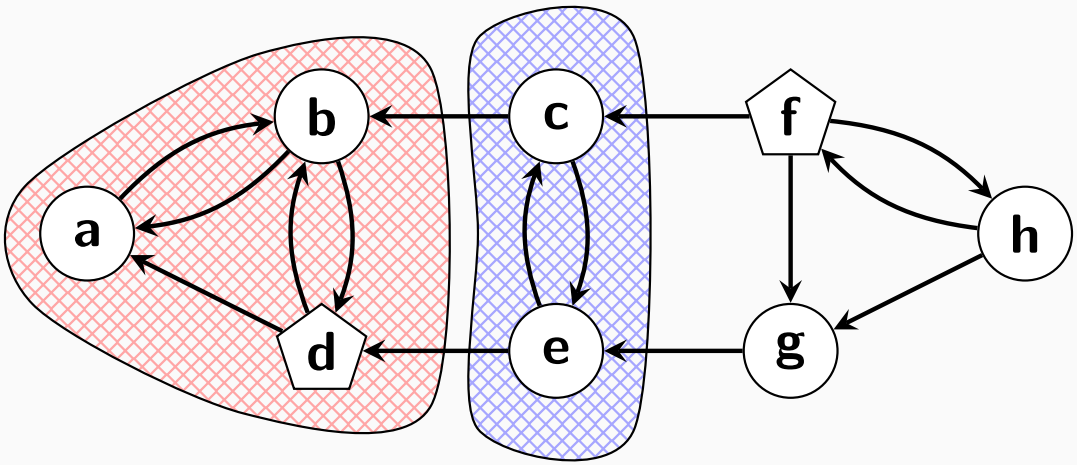
Computing the ⬟-attractor to **a**



Attractor set: $\{\mathbf{a}\}$, $\{\mathbf{a}, \mathbf{d}\}$, $\{\mathbf{a}, \mathbf{b}, \mathbf{d}\}$
Can attract: neither **c** nor **e**

Example of attractor computation

Computing the ⬟-attractor to **a**

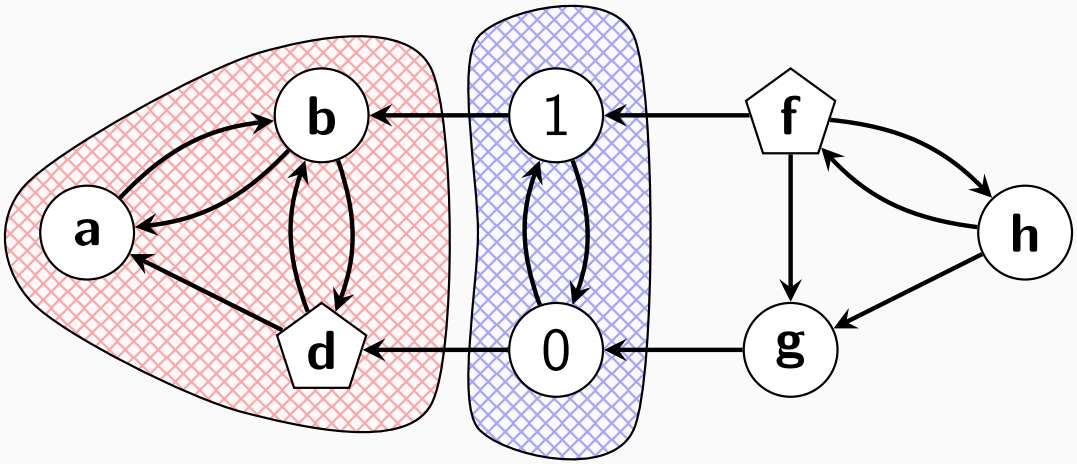


Attractor set: $\{\mathbf{a}\}$, $\{\mathbf{a}, \mathbf{d}\}$, $\{\mathbf{a}, \mathbf{b}, \mathbf{d}\}$

What if player ● cannot stay in $\{\mathbf{c}, \mathbf{e}\}$??

Example of attractor computation

Computing the ⬠-attractor to **a**



Attractor set: $\{\mathbf{a}\}$, $\{\mathbf{a}, \mathbf{d}\}$, $\{\mathbf{a}, \mathbf{b}, \mathbf{d}\}$

Every play inside the blue area is won by ⬠!

ATTRACTOR COMPUTATION

In game $\mathcal{G}$, attract to Z as player α from the vertices R.
Update the strategy in σ for:

- new vertices in Z of player α
- vertices in Z of player α that can now play inside Z

```
1  def attract(𝒢, Z, R, α, σ):
2      Q ← Z                                  // initialize with all target vertices
3      while Q ≠ ∅ :
4          v ← take(Q)                                  // take a v ∈ Q
5          foreach u : v ∈ E(u) do                  // check incoming edges
6              if u ∈ R ∧ u ∉ Z ∧ (u ∈ V_α ∨ E(u) ⊆ Z) :        // attract?
7                  Z ← Z ∪ {u}
8                  Q ← Q ∪ {u}
9              if u ∈ Z ∩ V_α ∧ u ∉ σ : σ[u] ← v          // set strategy?
10     return Z, σ
```

PARITY GAME ALGORITHMS

Decomposition/Attractor-based

Properties of sets of vertices computed with attractors.

- recursive algorithm (+ quasi-polynomial variation)
- priority promotion
- tangle learning

Local value improvement

Improve the *value of individual vertices* based on direct successors.

- small progress measures
- succinct progress measures (quasi-polynomial)
- ordered progress measures (quasi-polynomial)
- strategy improvement
- nested fixed point iteration

Decomposition-based algorithms

- Use the attractor computation to decompose the game into regions
- Refine/improve decomposition until dominion found

- First attract for some player α to the "best vertex", then continue decomposition of the remaining game

- Region invariant
  - Player α wins all plays that stay in the region
  - Player $\overline{\alpha}$ either stays in the region (and loses), or visits the target vertex, or escapes to a higher region of player α

PARITY GAME ALGORITHMS

Zielonka's recursive algorithm (1998)

Bottom-up attract to lower opponent regions in a strict order.
After each change, recompute lower regions of the loser.

Priority promotion (2016)

Merge locally closed regions upwards.
Attract to the merged region, then recompute lower regions.

Tangle learning (2018)

The attractor now also attracts using tangles.
Every iteration finds new tangles.
Use new tangles to that refine the decomposition.

Local value improvement

- Every vertex has some value (e.g. "how good for Even")
- Locally improve each vertex based on the value of the successors
- Value "propagates" through the game
- Intuition: playing a game in reverse
- Challenge: reasoning about cycles and nested cyclic structures

PARITY GAME ALGORITHMS

Progress measures (1998)

Players play their "best" game backwards, each vertex having a monotonically increasing value representing the current play value.

Strategy improvement (2000)

Players take turns improving their strategy. That is:

- Players start with random strategies
- Opponent maximally improves their strategy (monotonically decreasing)
- Main player improves their strategy once, then repeat
- Best response always monotonically increases value
- Strategies are improved based on best successor (local property)

OUTLINE

1. Background / Applications of parity games
   - Ω-automata
   - Modal μ-calculus
   - LTL synthesis
2. Parity game solving algorithms
   - Introduction
   - Decomposition based algorithms
   - Local algorithms
3. Quasi-polynomial algorithms
   - Universal trees and progress measures
   - Ordered progress measures
   - Quasi-polynomial recursive algorithm
4. Final remarks

ZIELONKA'S RECURSIVE ALGORITHM

Outline

- Recursive algorithm by McNaughton 1993, Zielonka 1998
- Decomposition of the game into regions
- Initially assume highest vertex to be "good" for associated player
- Recompute recursively if this was not the case

- Recent variations run in quasi-polynomial time
  Parys 2019, Lehtinen 2019, Jurdzinski 2020

RECURSIVE STRUCTURE

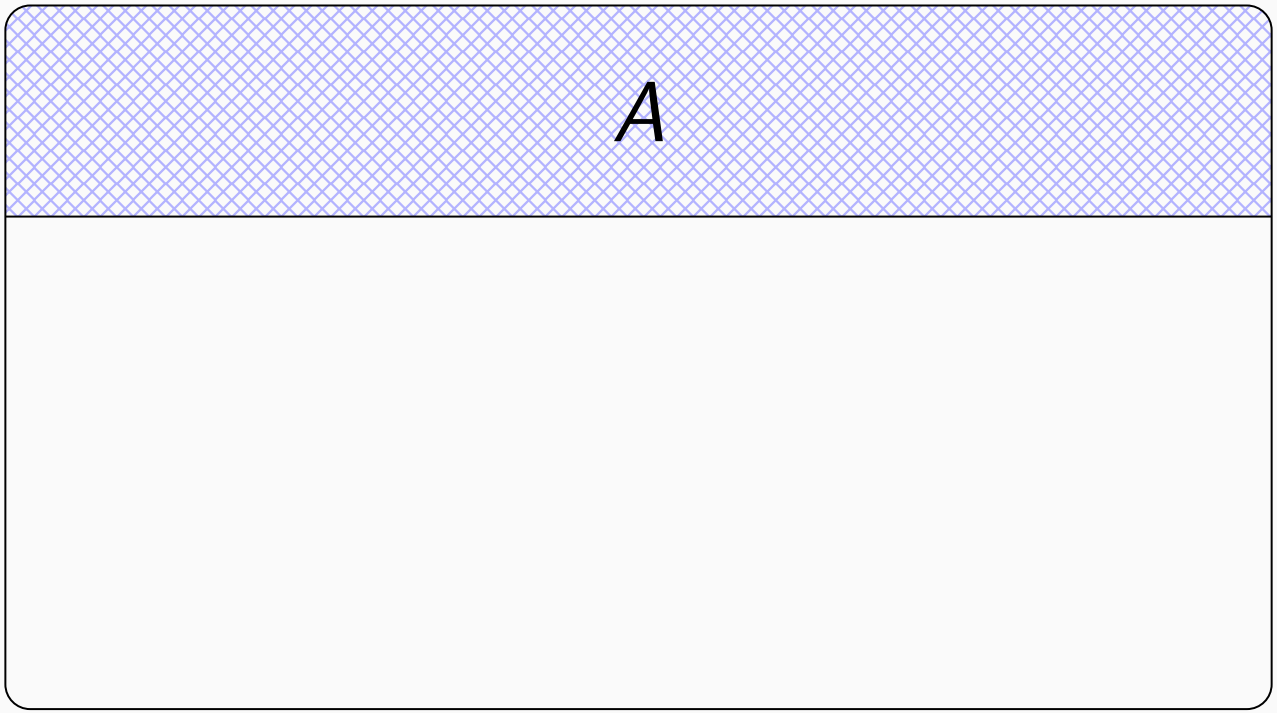
Task: solve the game recursively

Set $A :=$ attractor to highest-priority vertex

RECURSIVE STRUCTURE

Task: solve the game recursively

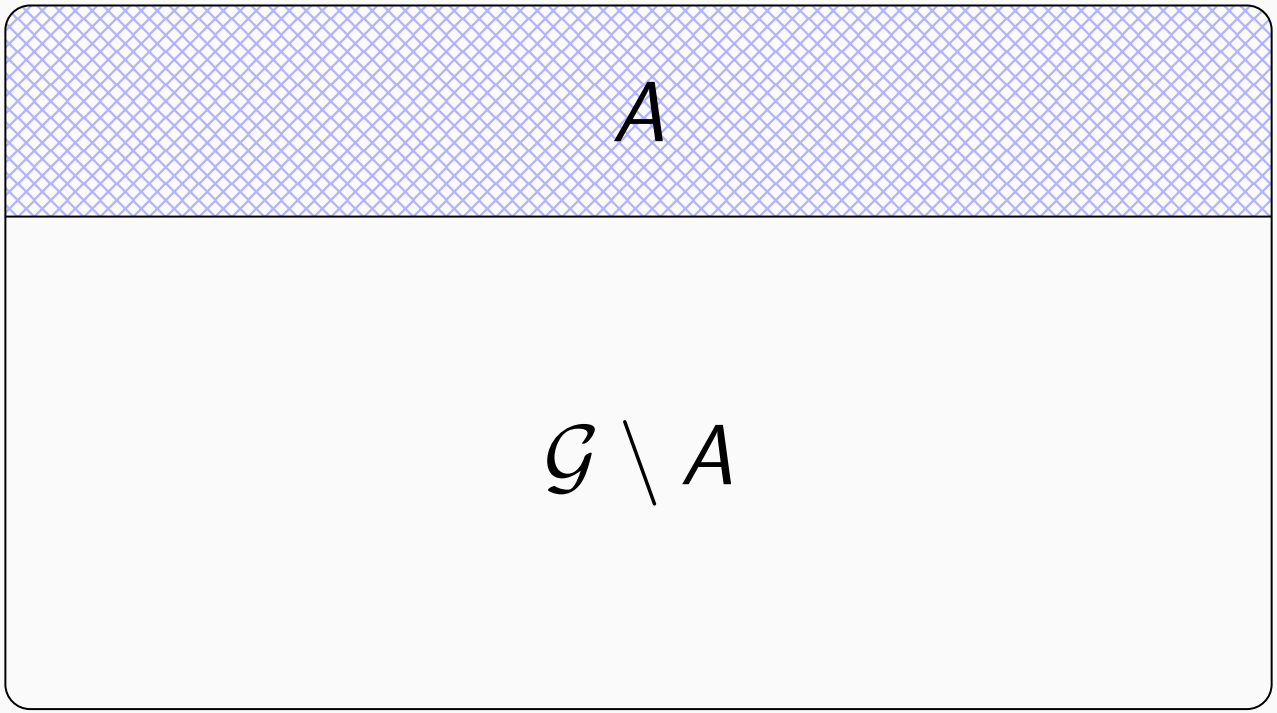
Set $A :=$ attractor to highest-priority vertex



RECURSIVE STRUCTURE

Task: solve the game recursively

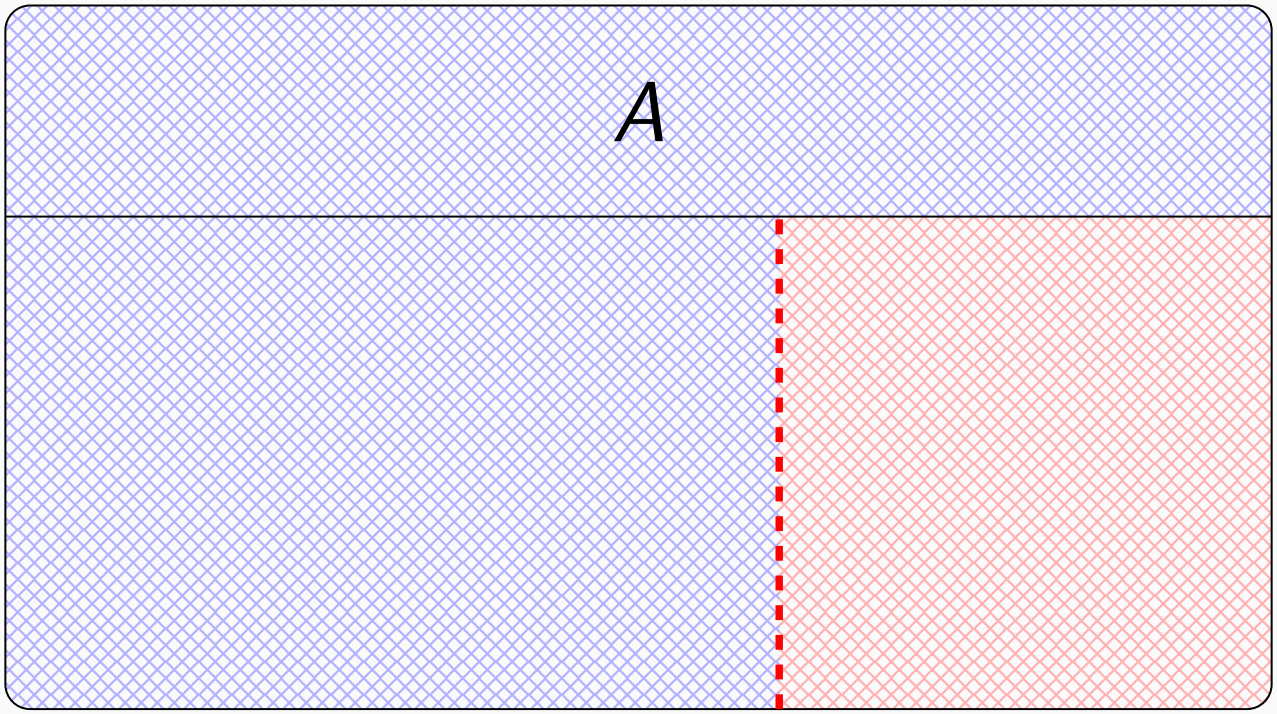
Recursively solve $\mathcal{G} \setminus A$



RECURSIVE STRUCTURE

Task: solve the game recursively

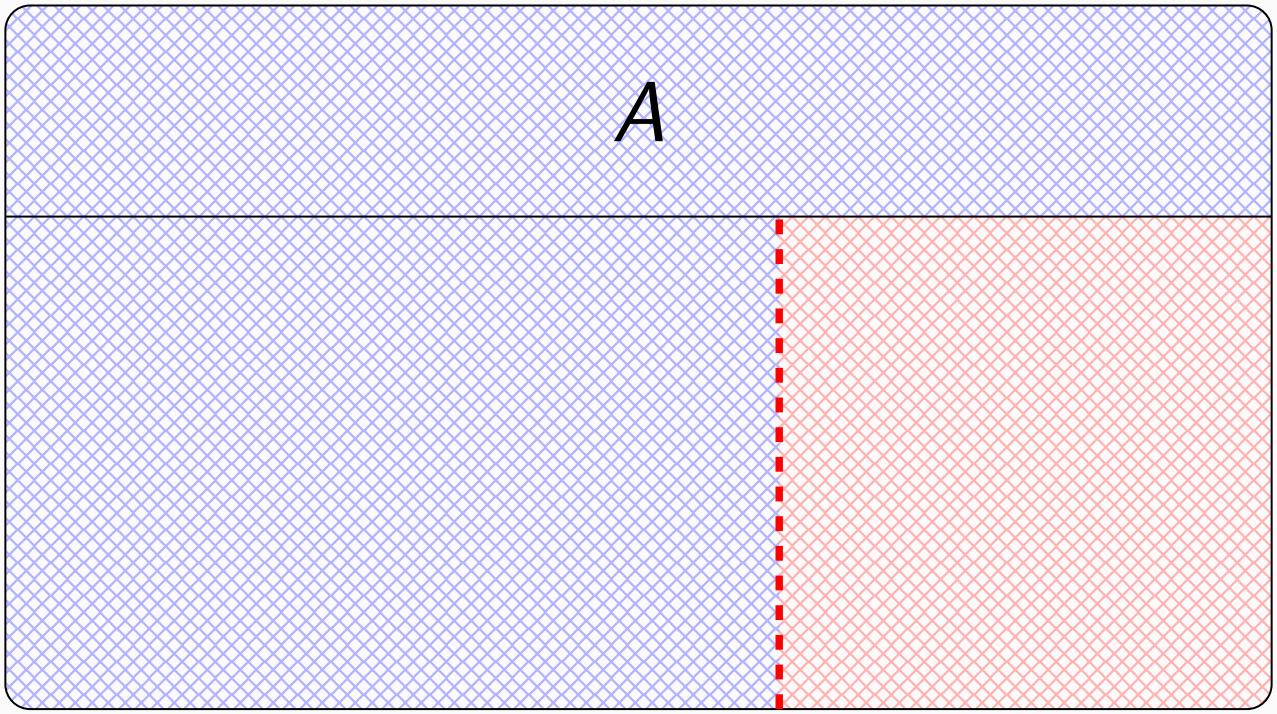
Recursively solve $\mathcal{G} \setminus A$



RECURSIVE STRUCTURE

Task: solve the game recursively

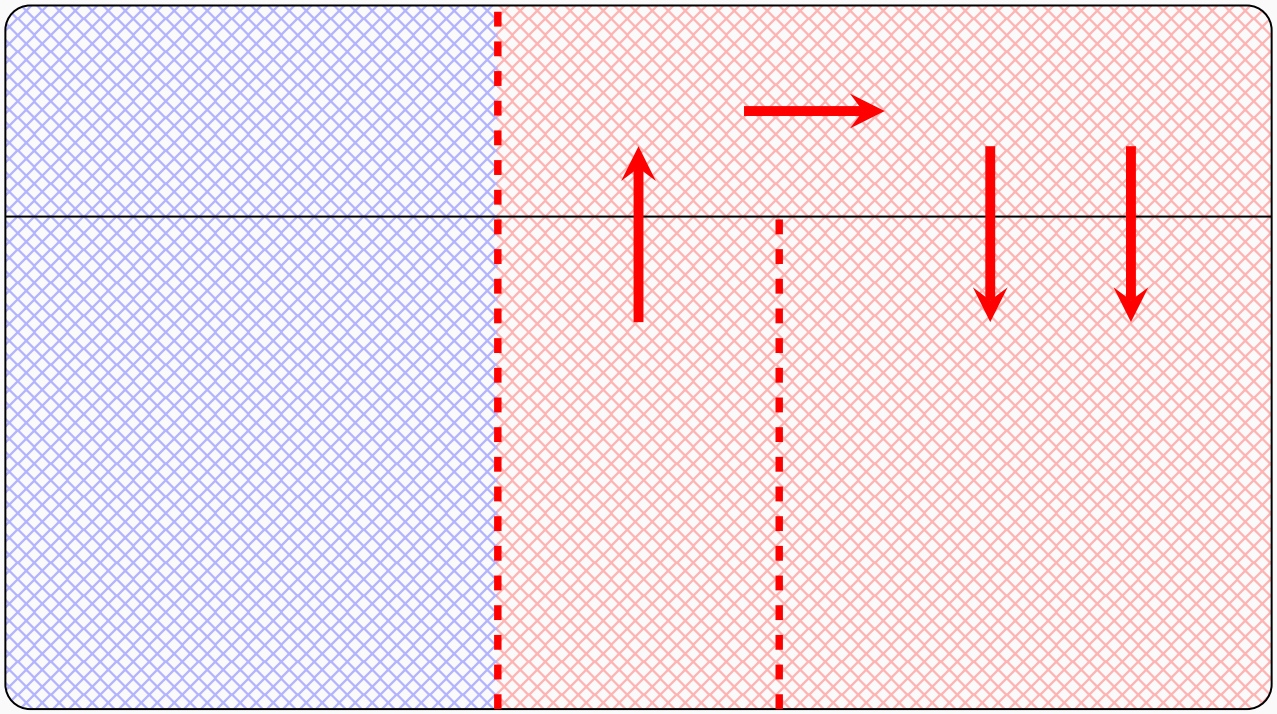
Attract from blue to red (if nothing attracted, then done!)



RECURSIVE STRUCTURE

Task: solve the game recursively

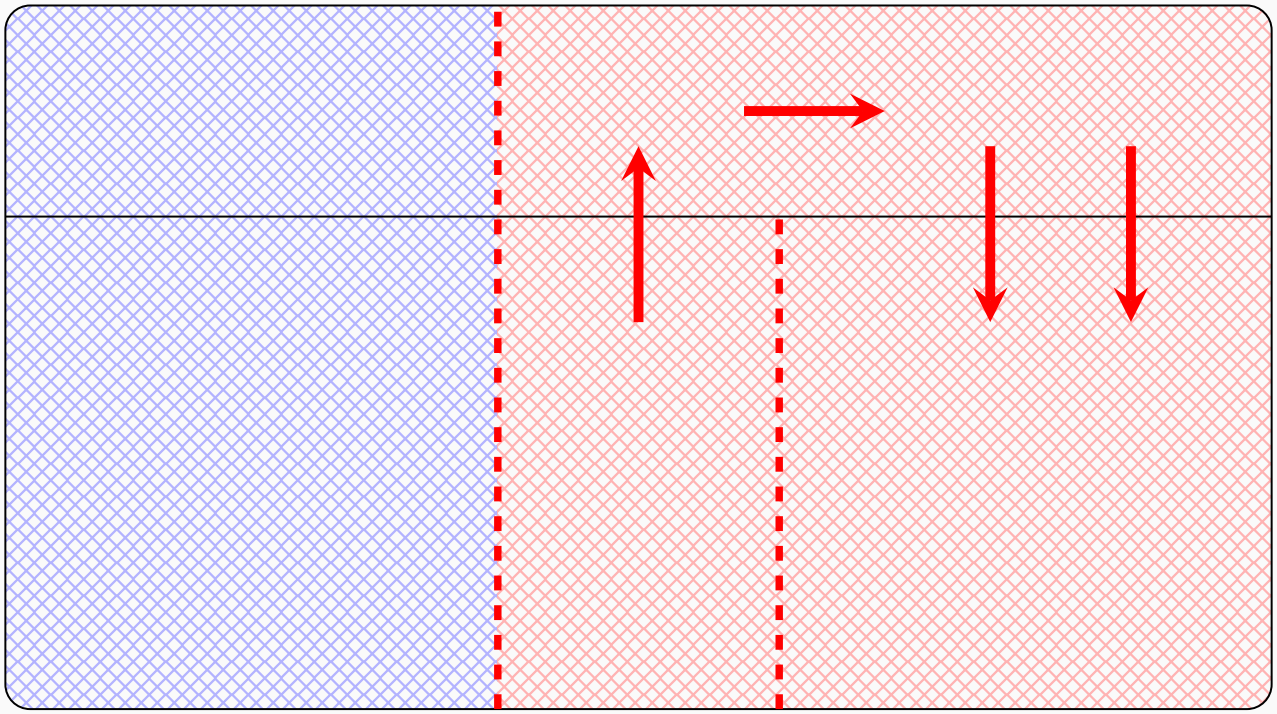
Attract from blue to red (if nothing attracted, then done!)



RECURSIVE STRUCTURE

Task: solve the game recursively

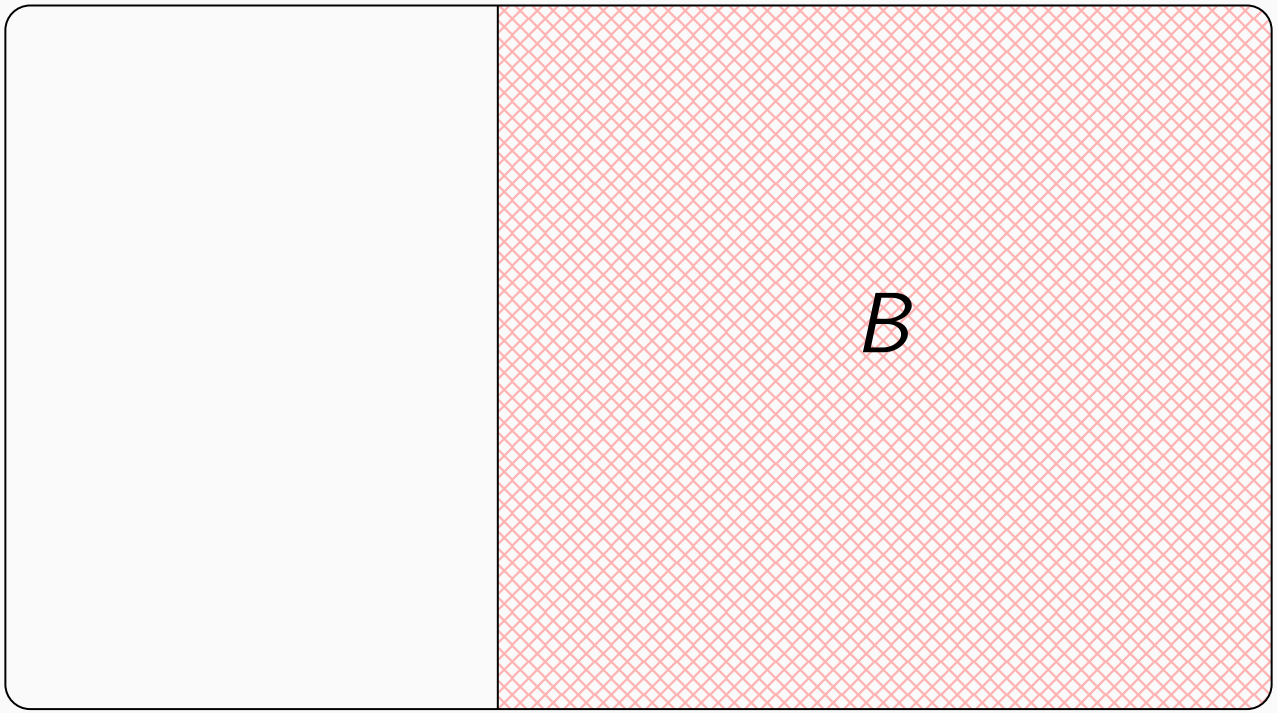
Set $B :=$ all vertices attracted to red



RECURSIVE STRUCTURE

Task: solve the game recursively

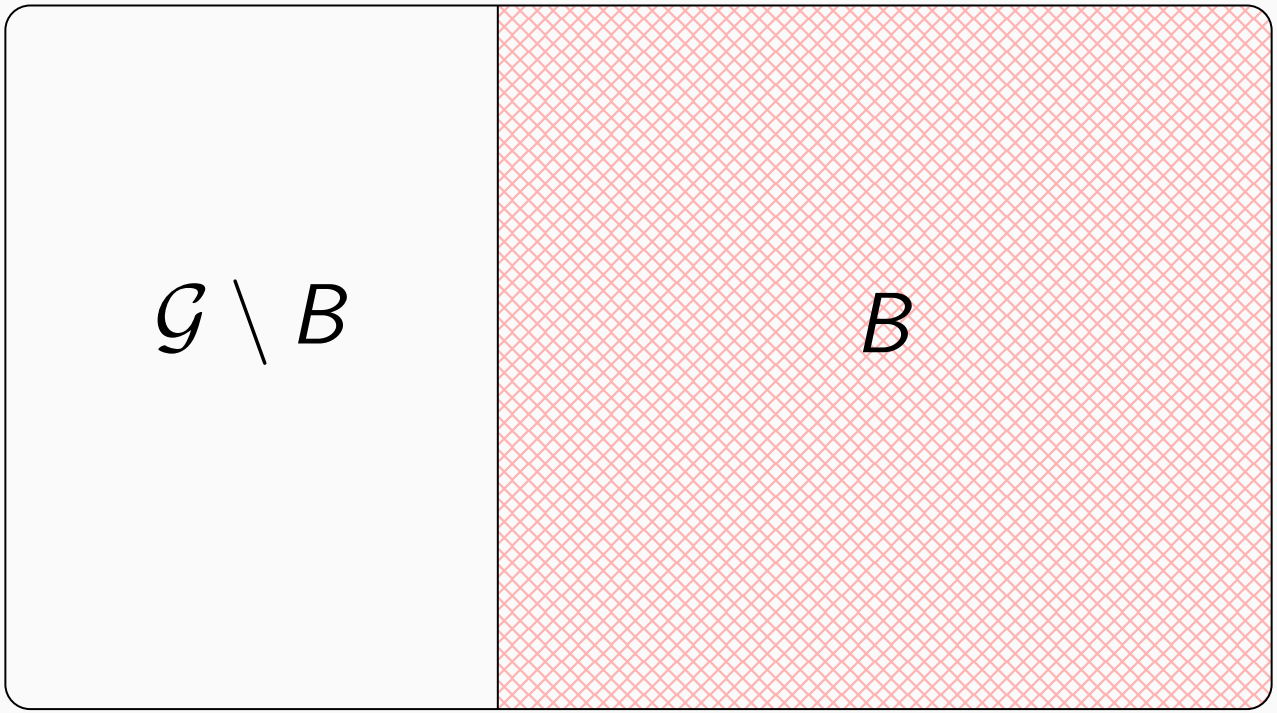
Set $B :=$ all vertices attracted to red



RECURSIVE STRUCTURE

Task: solve the game recursively

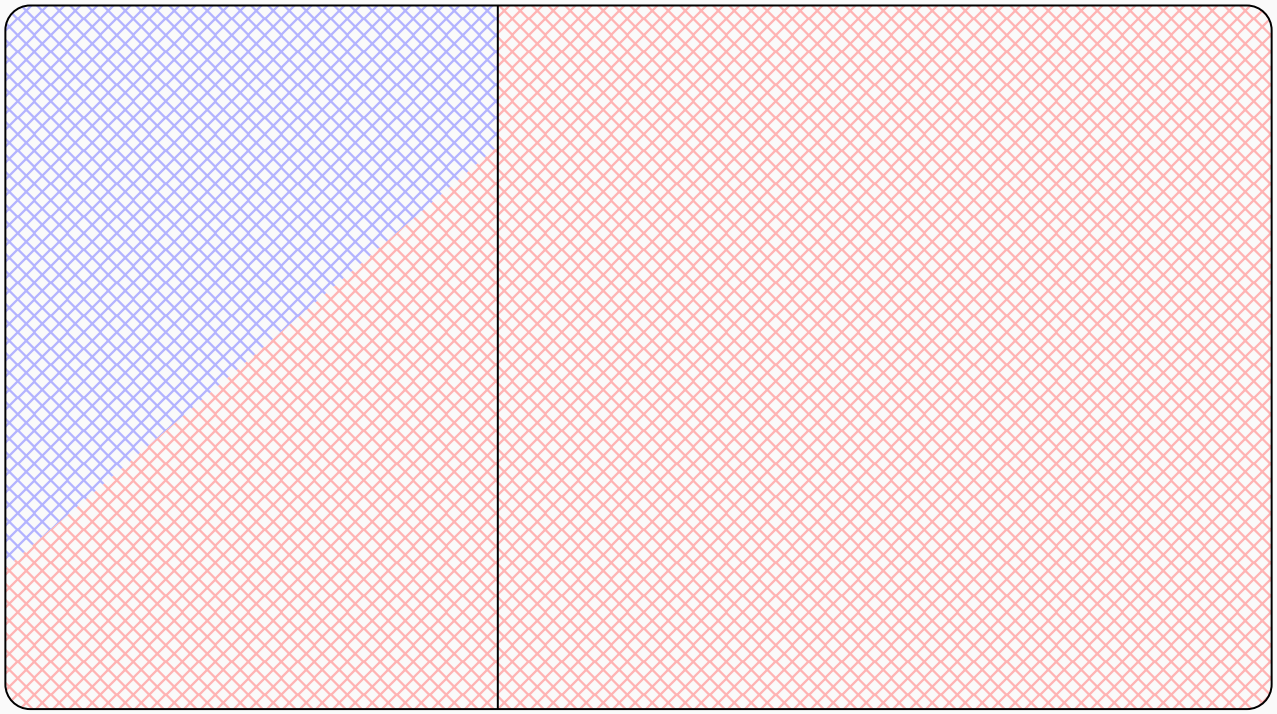
Recursively solve $\mathcal{G} \setminus B$



RECURSIVE STRUCTURE

Task: solve the game recursively

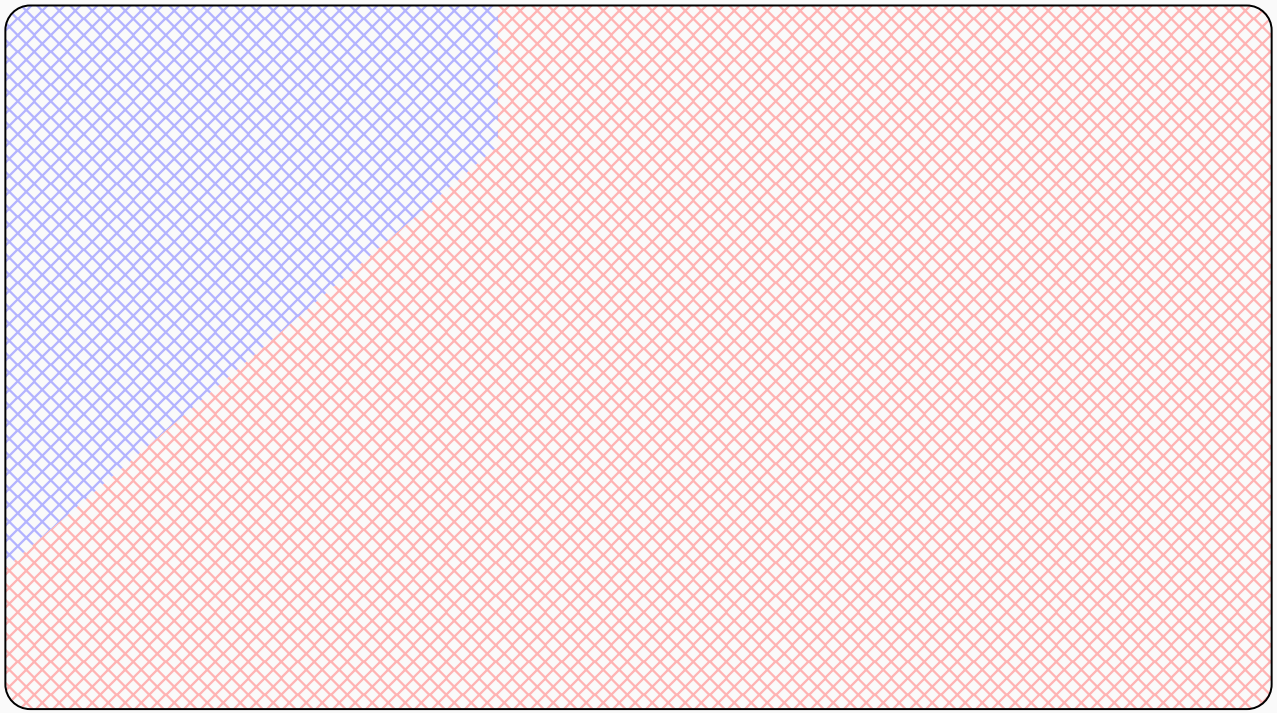
Recursively solve $\mathcal{G} \setminus B$



RECURSIVE STRUCTURE

Task: solve the game recursively

Game solved!



ZIELONKA'S RECURSIVE ALGORITHM

```
1  def zielonka(𝒢):
2      if 𝒢 = ∅ :
3          return ∅, ∅                                   // empty game
4      α ← pr(𝒢) mod 2                                   // winner of highest priority
5      Z ← pr^{-1}(pr(𝒢))                                // vertices of highest priority
6      A ← attract(𝒢, Z, V, α)                           // attracted to highest priority
7      W●, W⬟ ← zielonka(𝒢 \ A)                          // recursive solution
8      B ← attract(𝒢, W_ᾱ, V, ᾱ)                         // check if opponent attracts
9      if B = W_ᾱ :
10         W_α ← W_α ∪ A                                 // A is won by α
11     else:
12         W●, W⬟ ← zielonka(𝒢 \ B)                      // recompute remainder
13         W_ᾱ ← W_ᾱ ∪ B                                 // B is won by ᾱ
14     return W●, W⬟
```

ZIELONKA'S RECURSIVE ALGORITHM

Computing strategy

- Strategy is computed by attractor
  - Every attracted α-vertex u to some v in the set: strategy is $u \to v$
- Special case: α-vertices of the original target set
  - Pick any successor in winning region as strategy
- Implementation: use only a single strategy array, reset the strategy of target vertices before attracting

ZIELONKA'S RECURSIVE ALGORITHM

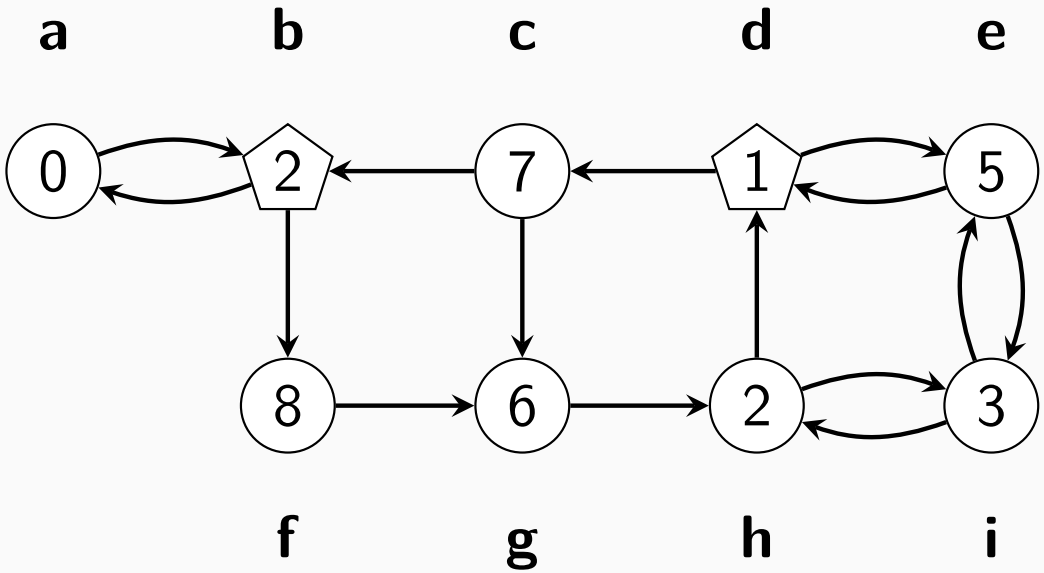
```
1  def zielonka(𝒢):
2      if 𝒢 = ∅ :
3          return ∅, ∅                                          // empty game
4      α ← pr(𝒢) mod 2                                  // winner of highest priority
5      Z ← pr^{-1}(pr(𝒢))                             // vertices of highest priority
6      A, σ_A ← attract(𝒢, Z, V, α, ∅)          // attracted to highest priority
7      W_●, W_⬟, σ_●, σ_⬟ ← zielonka(𝒢 \ A)                  // recursive solution
8      B, σ_ᾱ ← attract(𝒢, W_ᾱ, V, ᾱ, σ_ᾱ)          // check if opponent attracts
9      if B = W_ᾱ :
10         W_α ← W_α ∪ A                                        // A is won by α
11         σ_α ← σ_α ∪ σ_A ∪ ((z ∈ Z) ↦ pick(E(z) ∩ W_α))
12     else:
13         W_●, W_⬟, σ_●, σ_⬟ ← zielonka(𝒢 \ B)          // recompute remainder
14         W_ᾱ ← W_ᾱ ∪ B                                        // B is won by ᾱ
15         σ_ᾱ ← σ_ᾱ ∪ σ_B
16     return W_●, W_⬟, σ_●, σ_⬟
```

ZIELONKA'S RECURSIVE ALGORITHM

Another perspective

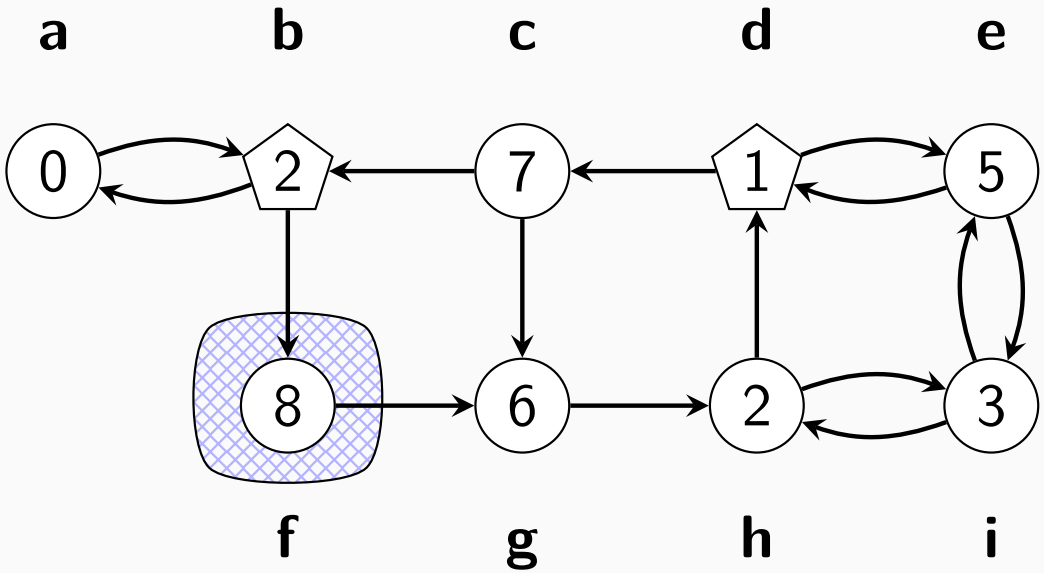
- Decompose top-down into regions
- Going back up, attract from the player's regions to the opponent's lower regions
- If so, reset and recompute the player's lower regions

ZIELONKA'S ALGORITHM



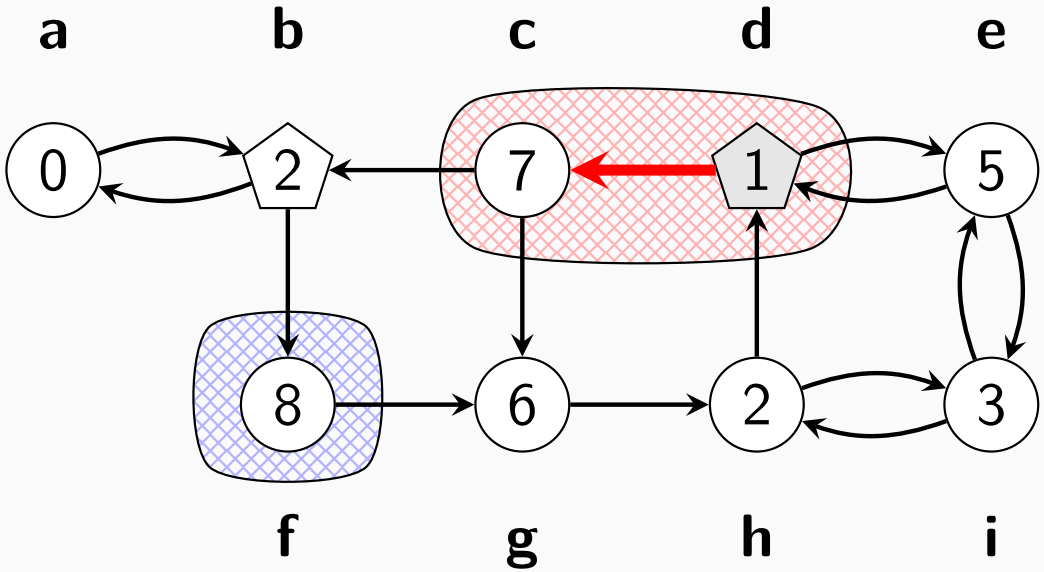
- We start by attracting to 8 for player Even.

ZIELONKA'S ALGORITHM



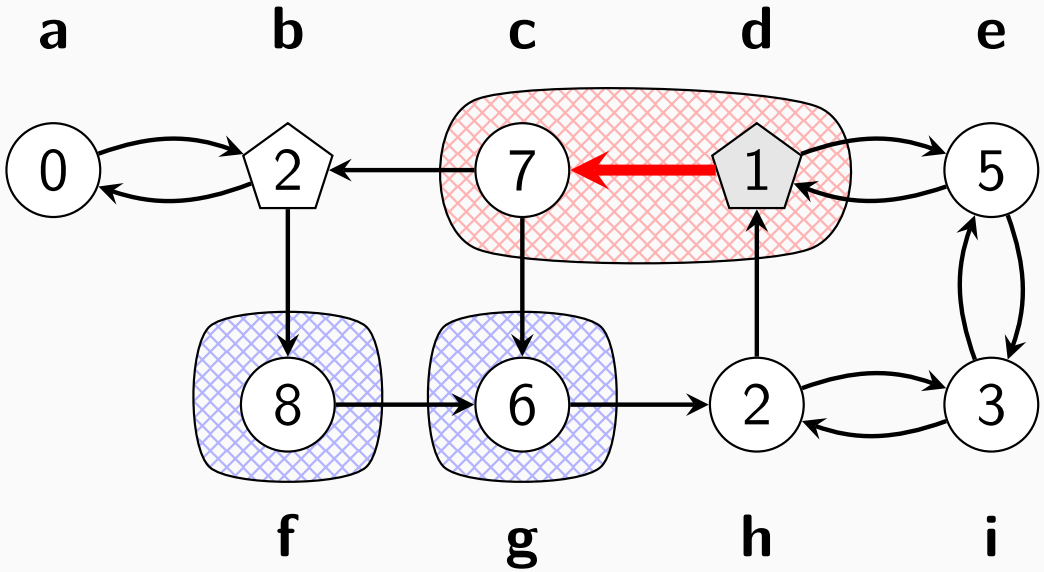
- After region 8 (player Even).
- Continue (recursively) with region 7.

ZIELONKA'S ALGORITHM



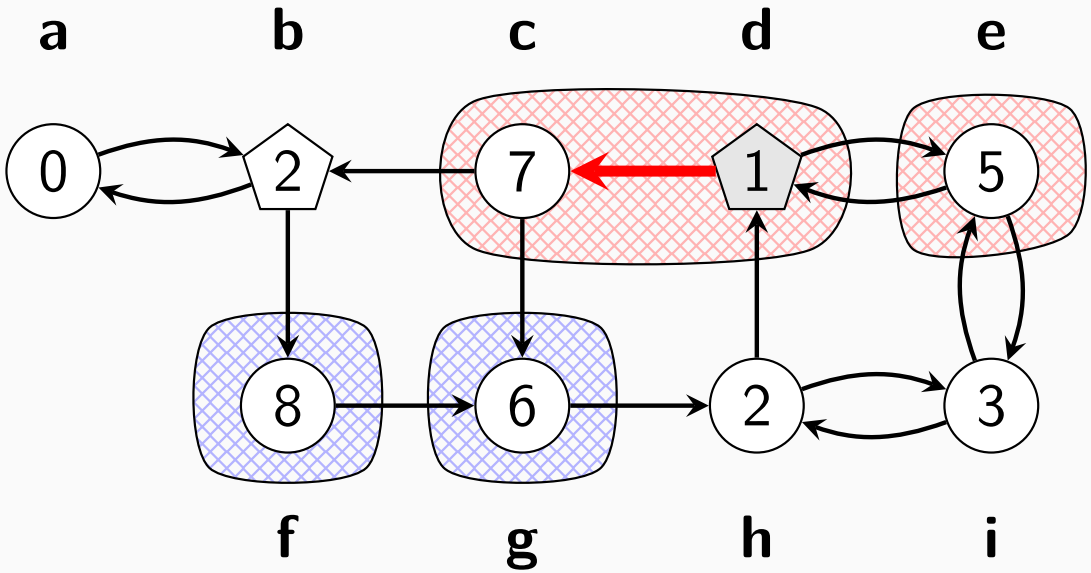
- After regions 8 (player Even) and 7 (player Odd).
- Continue (recursively) with region 6.

ZIELONKA'S ALGORITHM



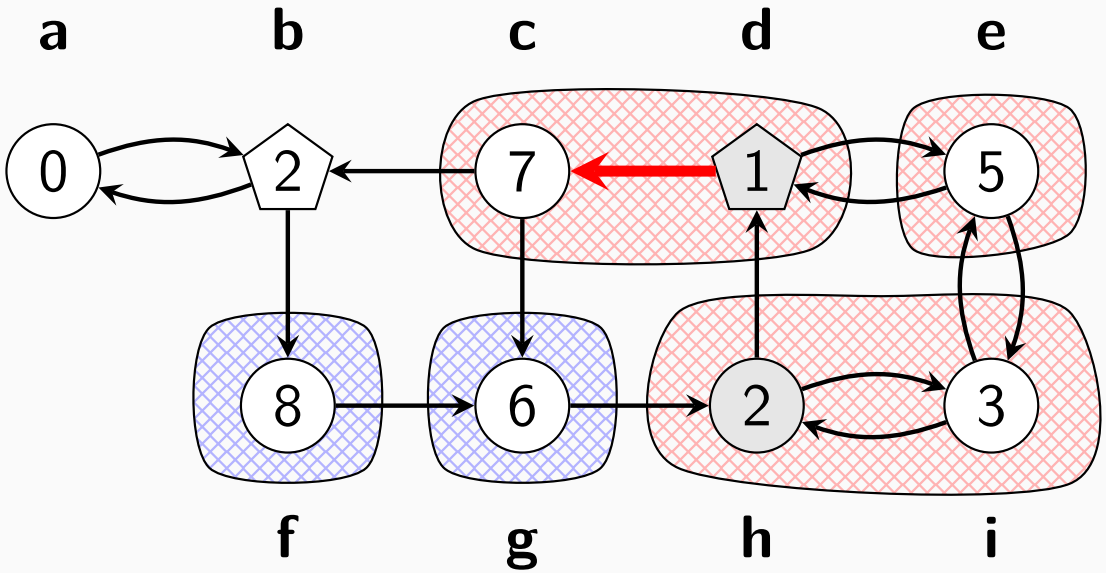
- After regions 8, 7 and 6.
- Continue (recursively) with region 5.

ZIELONKA'S ALGORITHM



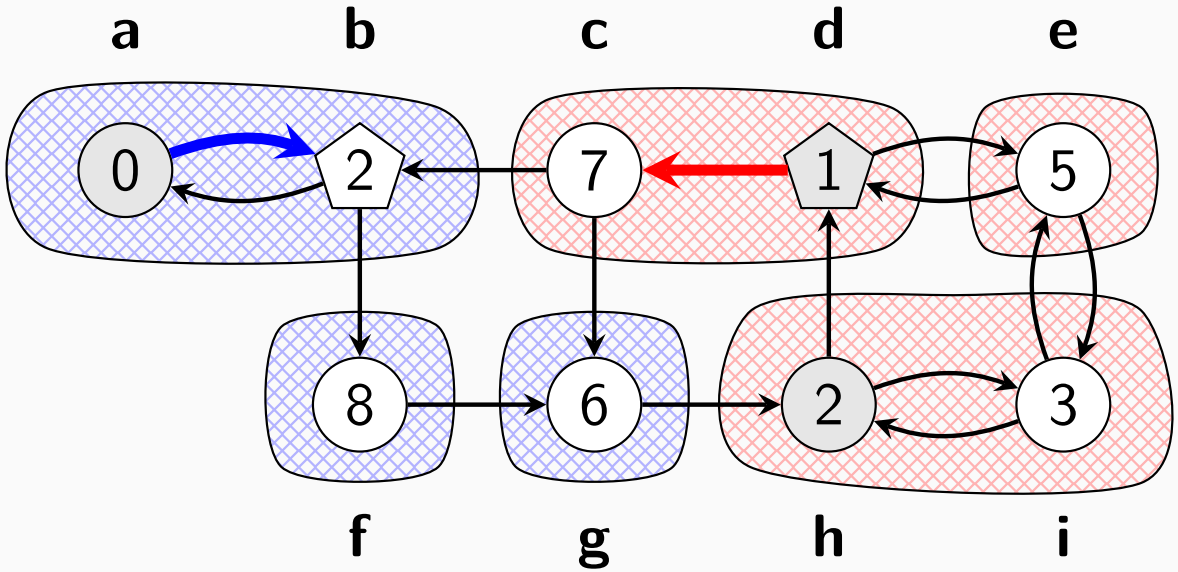
- After regions 8, 7, 6 and 5.
- Continue (recursively) with region 3.

ZIELONKA'S ALGORITHM



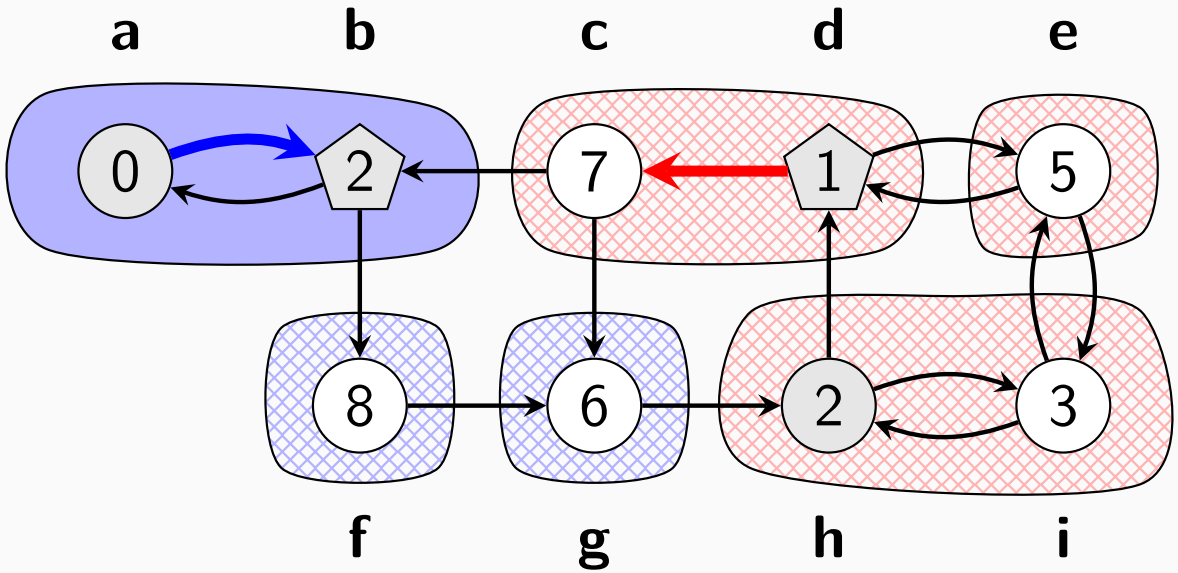
- After regions 8, 7, 6, 5 and 3.
- Now remains just region 2.

ZIELONKA'S ALGORITHM



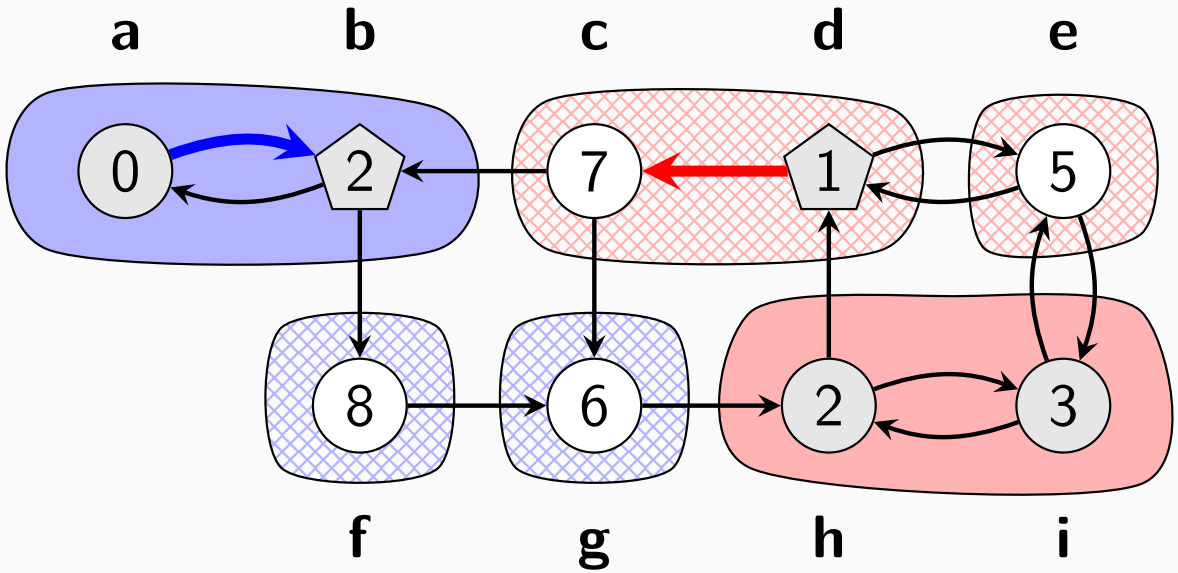
- Game is partitioned fully, now go up in the recursion.
- Up in region 2, does the lower opponent's winning region attract?
- Region 2: no (because the subgame is empty).

ZIELONKA'S ALGORITHM



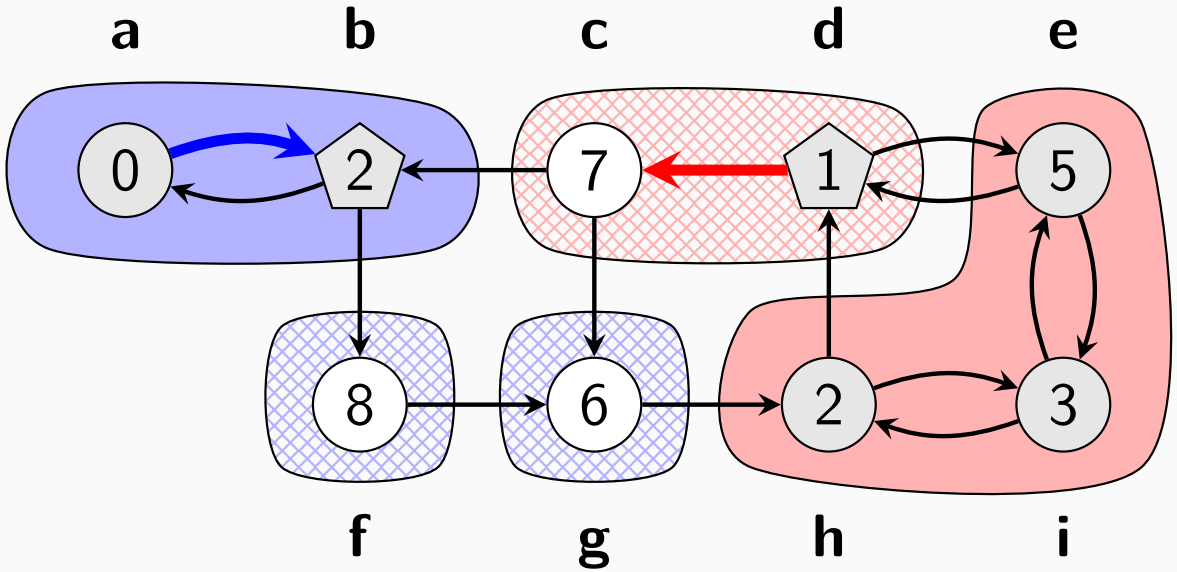
- Up in region 3: does the lower Even region attract from region 3?

ZIELONKA'S ALGORITHM



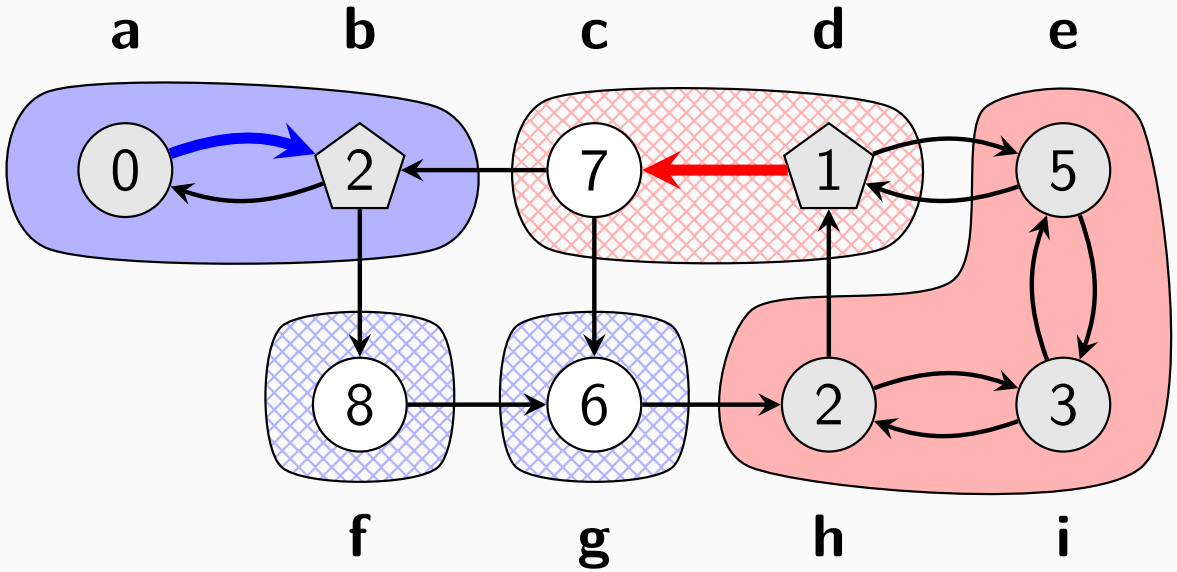
- Up in region 5: does the lower Even region attract from region 5?

ZIELONKA'S ALGORITHM



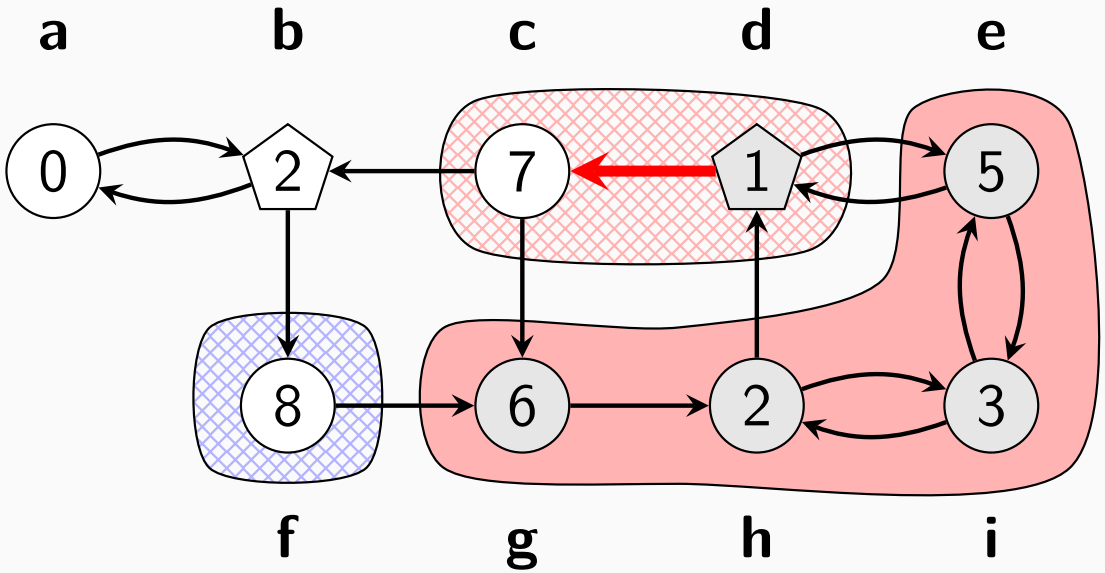
- Up in region 6: does the lower Odd region attract from 6?

ZIELONKA'S ALGORITHM



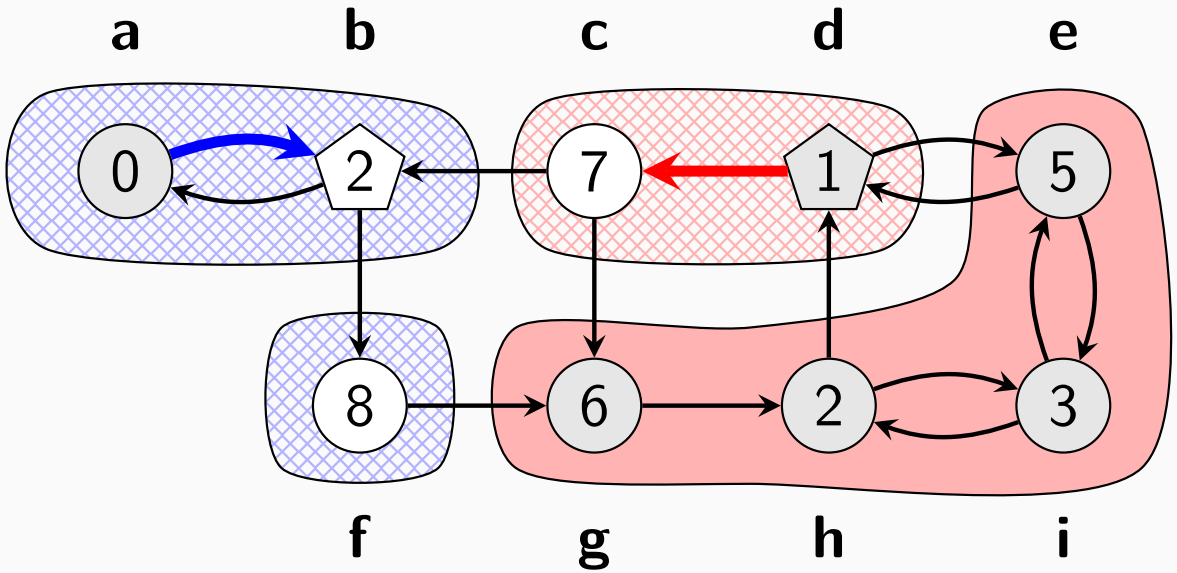
- Up in region 6: does the lower Odd region attract from 6?
- Yes: the lower Odd region attracts vertex **g**.
- So: attract to Odd, reset and recompute lower.

ZIELONKA'S ALGORITHM



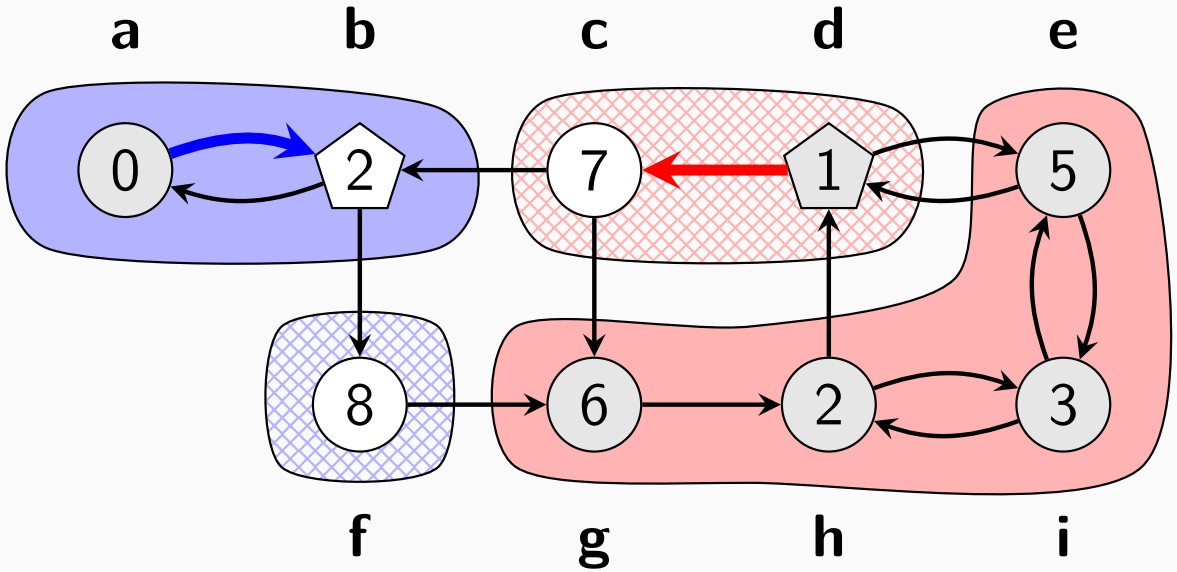
- Vertex **g** is attracted to the Odd region.
- So now recompute the (remainder of the) lower regions of Even.

ZIELONKA'S ALGORITHM



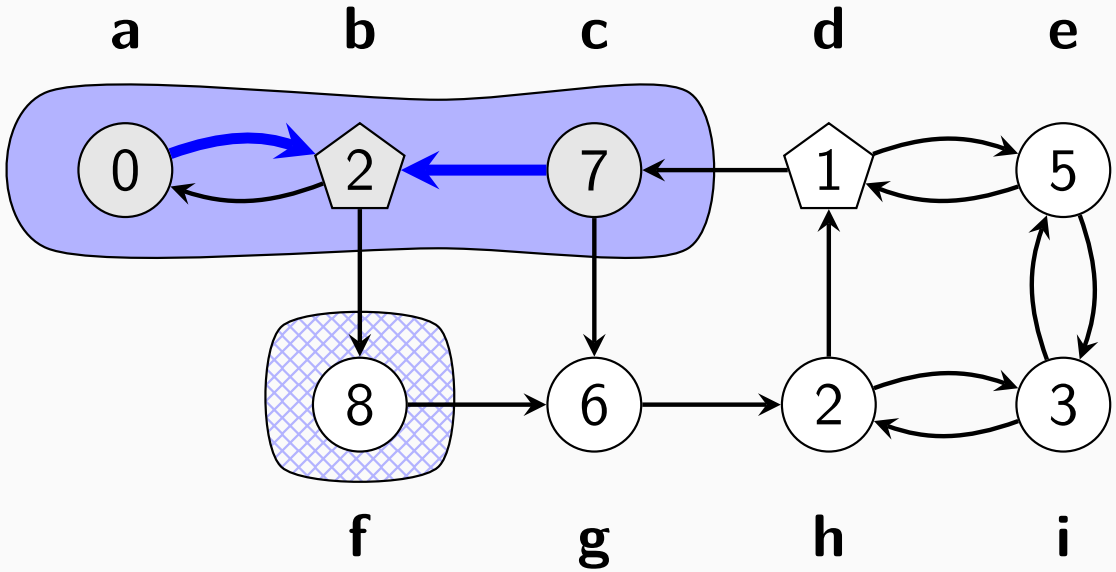
- Vertex **g** is attracted to the Odd region.
- So now recompute the (remainder of the) lower regions of Even.
- Recompute region 2...

ZIELONKA'S ALGORITHM



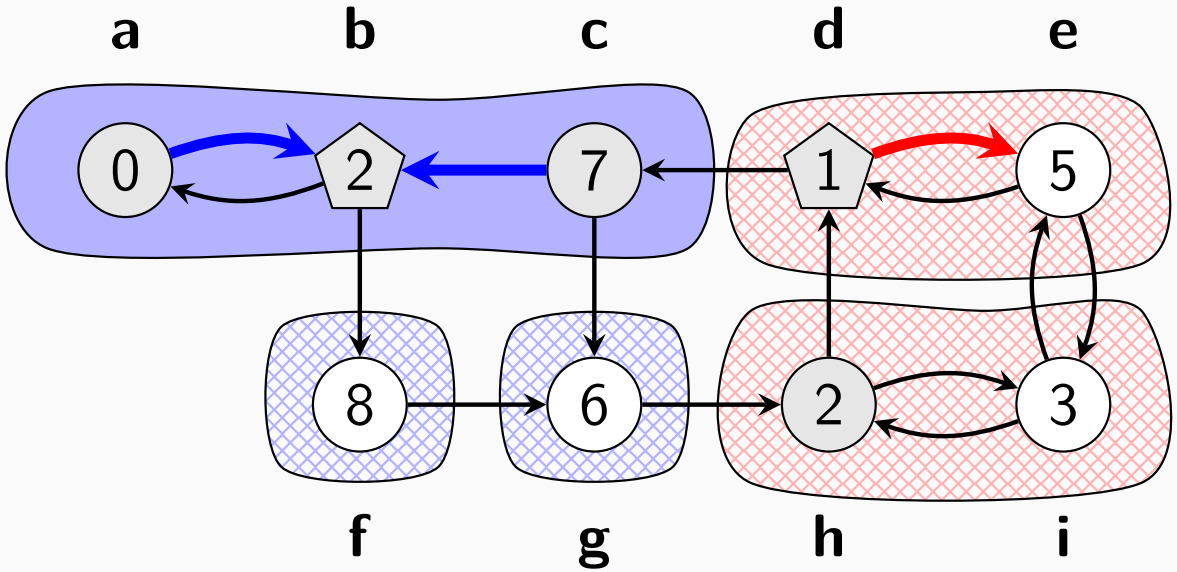
- Vertex **g** is attracted to the Odd region.
- So now recompute the (remainder of the) lower regions of Even.
- Recompute region 2... not attracted to lower Odd...
- Next: region 7: does the lower Even region attract from 7?
- Yes, the lower Even region attracts vertex **c**.
- Attract to Even, reset lower!

ZIELONKA'S ALGORITHM



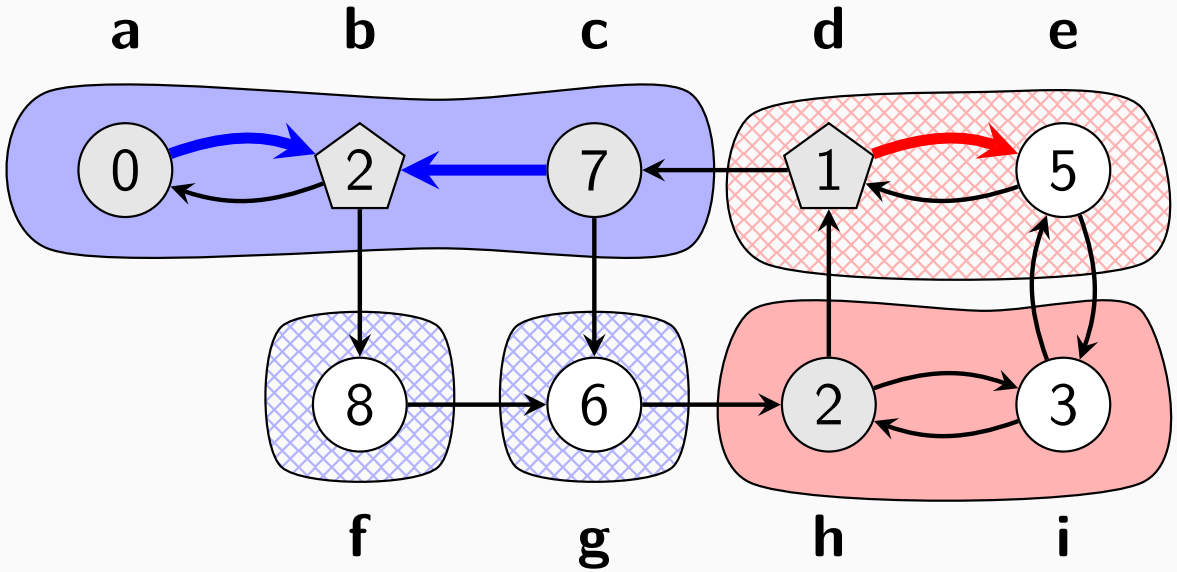
- Vertex **c** is attracted to the Even region.
- Recompute the remainder of the lower regions of Odd.

ZIELONKA'S ALGORITHM



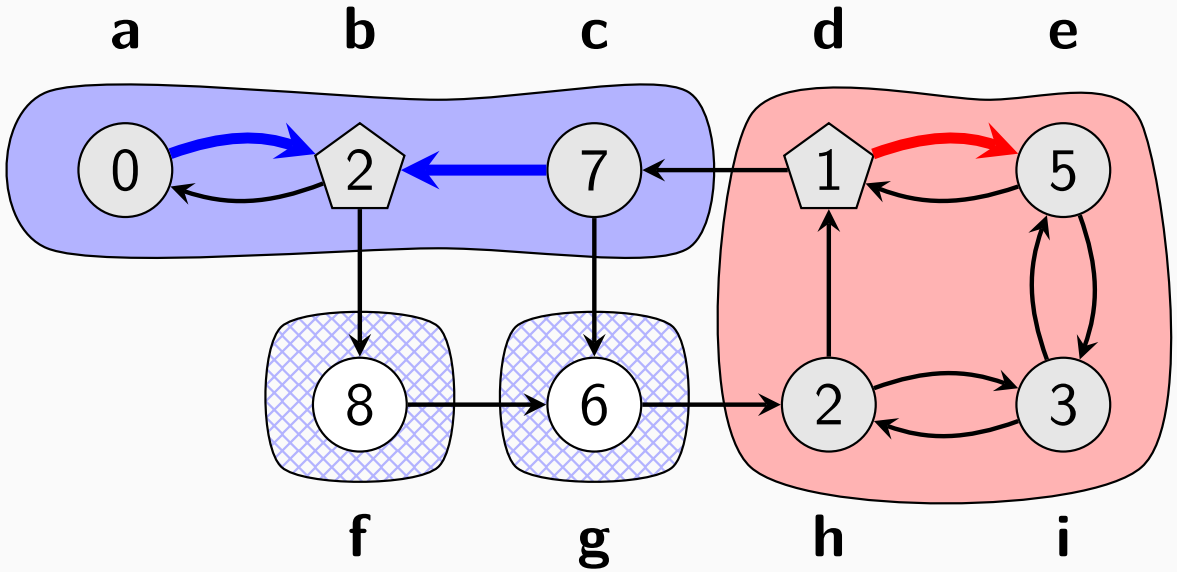
- Partition the remainder into regions 6, 5 and 3.
- Up in region 3: no attraction from Even.
- Up in region 5: no attraction from Even.
- Up in region 6: the lower Odd regions attract **g** again!

ZIELONKA'S ALGORITHM



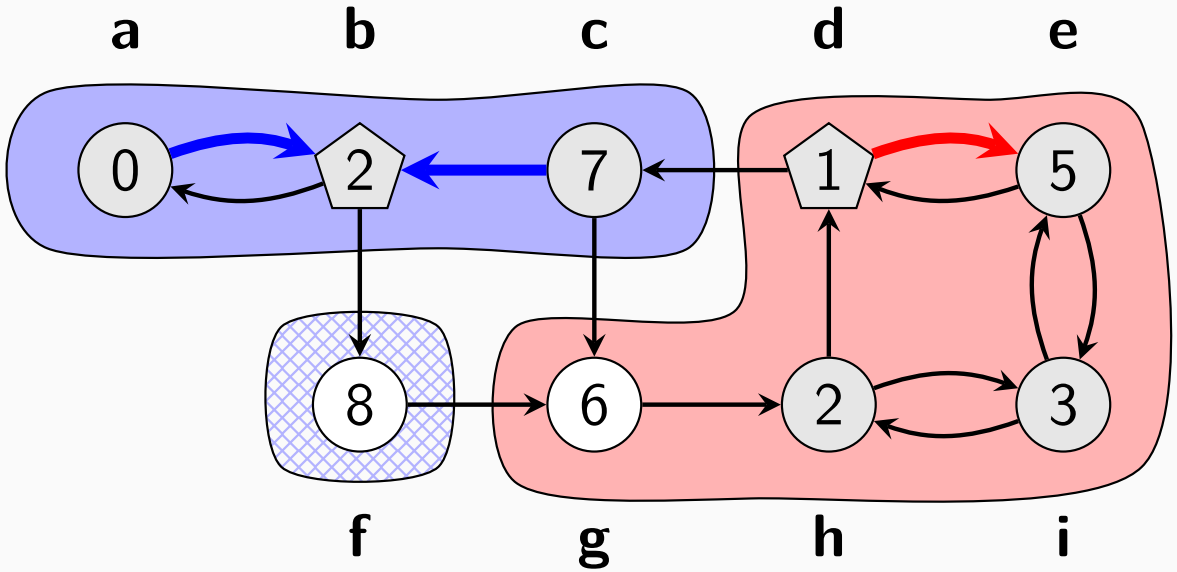
- Partition the remainder into regions 6, 5 and 3.
- Up in region 3: no attraction from Even.
- Up in region 5: no attraction from Even.
- Up in region 6: the lower Odd regions attract **g** again!

ZIELONKA'S ALGORITHM



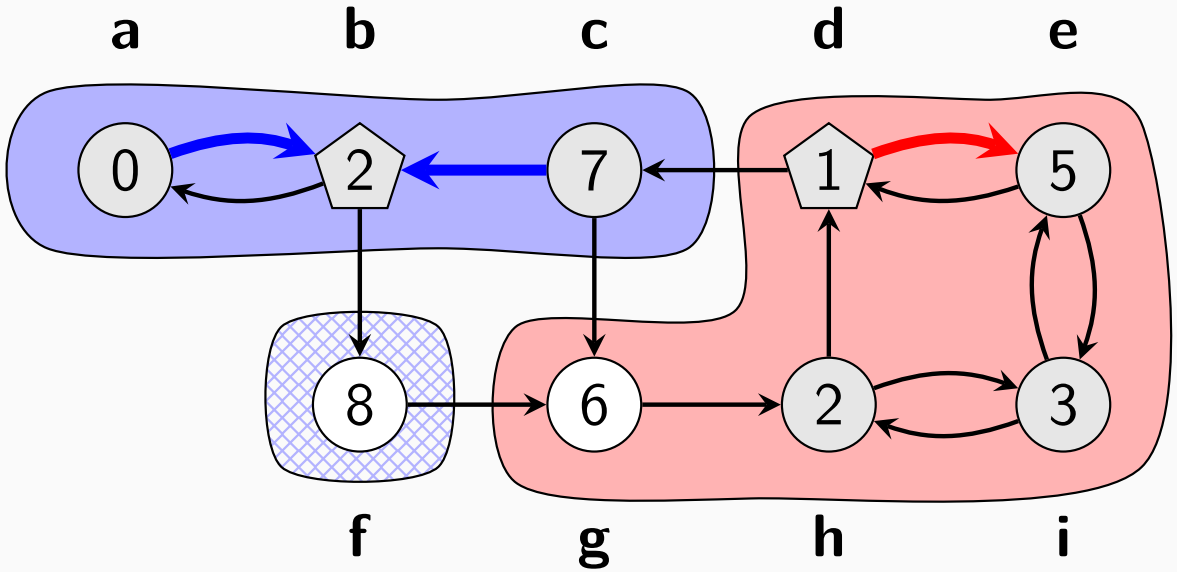
- Region 6: now the Odd region attracts vertex **g** again.
- Attract to Odd, reset the lower game.

ZIELONKA'S ALGORITHM



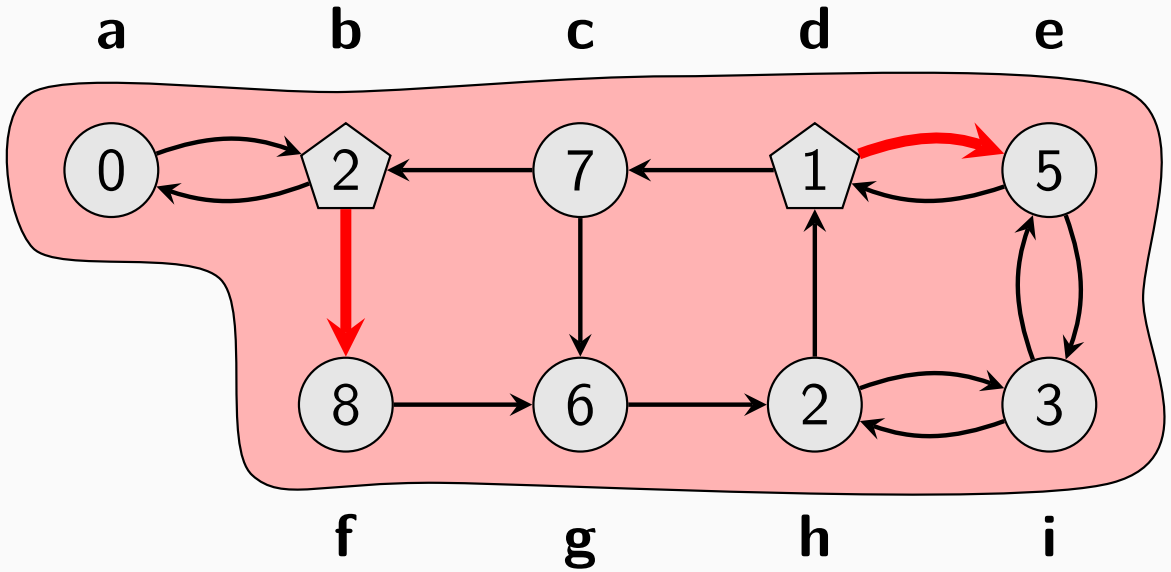
- Vertex **g** is attracted to the Odd region.
- Recursive game of 6 is empty.
- Up to region 8, does Odd now attract 8?
- Yes: vertex **f** is attracted to the Odd region.
- Attracting to Odd now attracts all remaining vertices...

ZIELONKA'S ALGORITHM



- Vertex **g** is attracted to the Odd region.
- Recursive game of 6 is empty.
- Up to region 8, does Odd now attract 8?

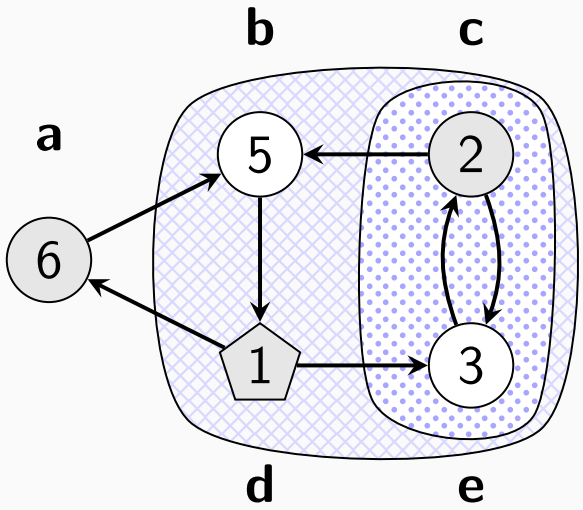
ZIELONKA'S ALGORITHM



- Final result, entire game won by player Odd.

Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.



A game with a 5-tangle and a 3-tangle

TANGLES

Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.

Definition

A tangle is

- a pair $T = (U, \sigma)$ where
  - $U \subseteq V$ is a nonempty set of vertices
  - $\sigma \colon U_\alpha \to U$ is a strategy for player $\alpha := \mathrm{pr}(U) \bmod 2$

  such that
- player α wins all cycles in the induced subgame $\mathcal{G}[U, \sigma]$
- the induced subgame $\mathcal{G}[U, \sigma]$ is strongly connected

TANGLES

Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.

Properties

- The opponent must escape (or lose inside the tangle)
- The opponent can freely choose any escape (strongly connected)
- A closed tangle (no escapes) is a winning region

TANGLES

Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.

Tangles are fundamental

- All algorithms implicitly reason about tangles:
  - every algorithm must deal with cycles and nested cyclic structures
  - if you conclude that the opponent cannot 'hide' in some subgame
  - then this **must** be because of tangles
- Most algorithms often explore the same tangle many times!
  (This can lead to repetitive behavior and exponential blowup!)

TANGLES

Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.

Tangle attractor

Because the opponent $\overline{\alpha}$ **must** escape the tangle, we can use tangles to attract all vertices of a tangle simultaneously.

Add to Z (exhaustively):

- all vertices of the player α that *can* play to Z
- all vertices of opponent $\overline{\alpha}$ that *must* play to Z
- all vertices in an α-tangle where all escapes lead to Z

Tangle learning

- Partition game into regions with tangle attractor
  - Attract to highest priority
  - Repeat with the remainder until nothing left

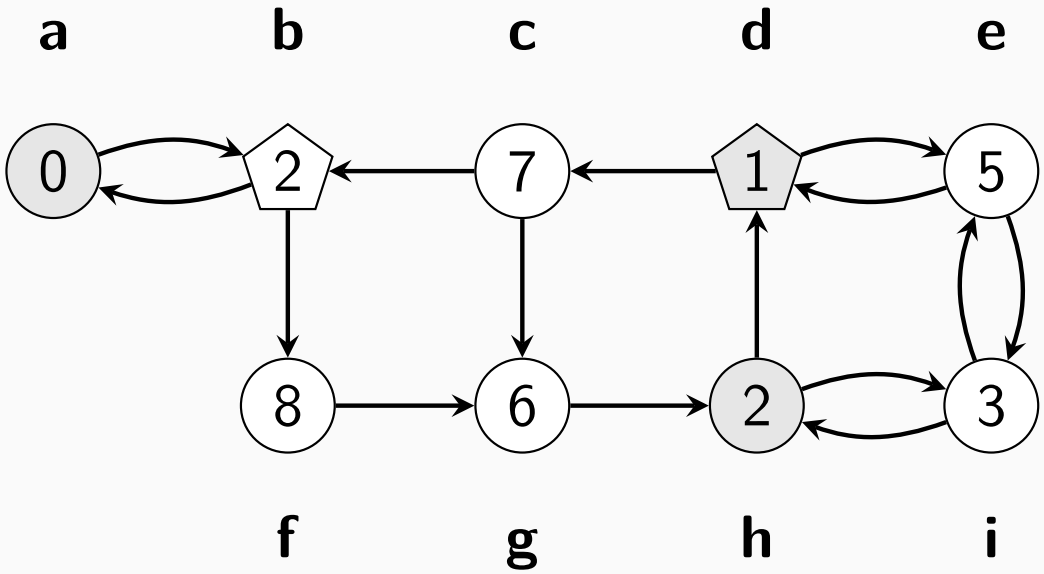
Tangle learning

- Partition game into regions with tangle attractor
  - Attract to highest priority
  - Repeat with the remainder until nothing left
- Every locally closed region contains a new tangle
  - Run Tarjan's SCC algorithm, from the top vertex, restricted to the attractor strategy
  - The result is the new tangle

Tangle learning

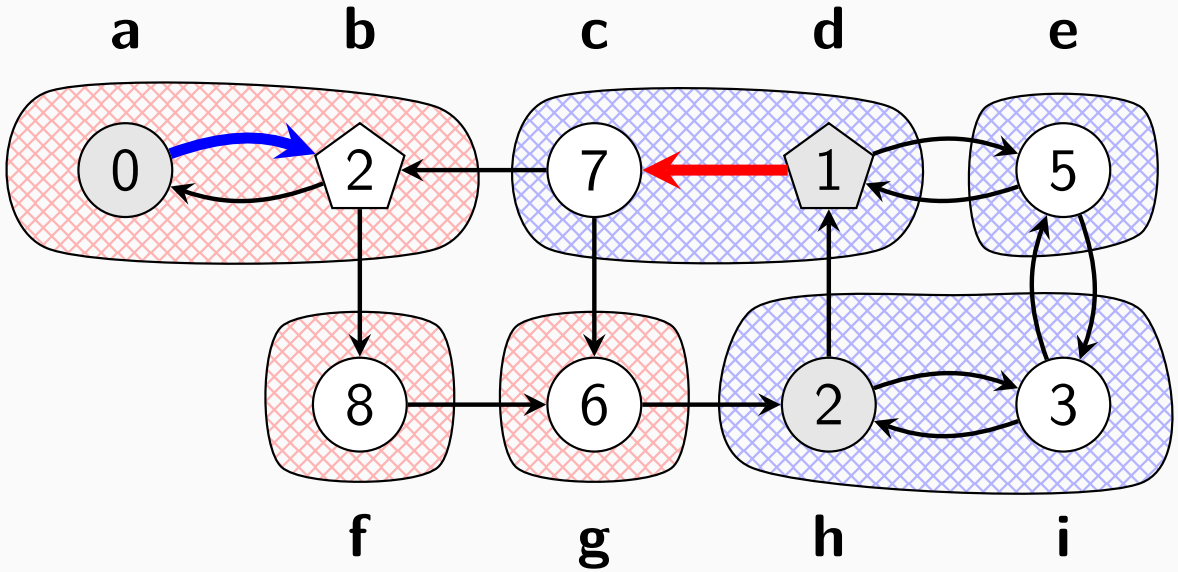
- Partition game into regions with tangle attractor
  - Attract to highest priority
  - Repeat with the remainder until nothing left
- Every locally closed region contains a new tangle
  - Run Tarjan's SCC algorithm, from the top vertex, restricted to the attractor strategy
  - The result is the new tangle
- Every closed tangle is a winning region (dominion)
  - After finding a dominion, maximize it with another attractor
  - Remove maximized dominions from the game

TANGLE LEARNING



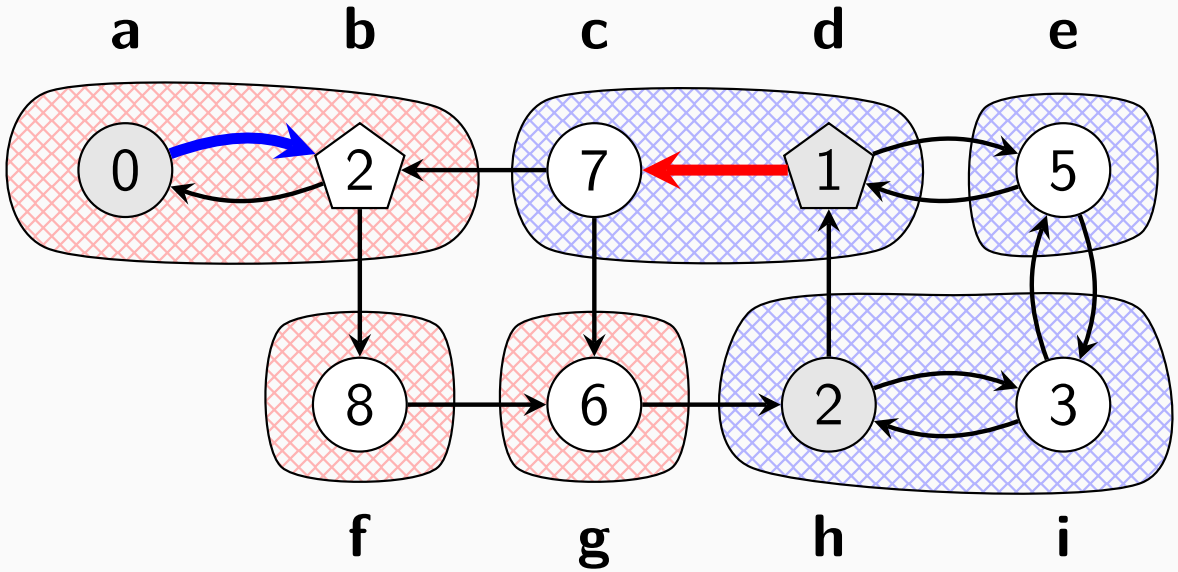
- First partition top-down into α-maximal regions

TANGLE LEARNING



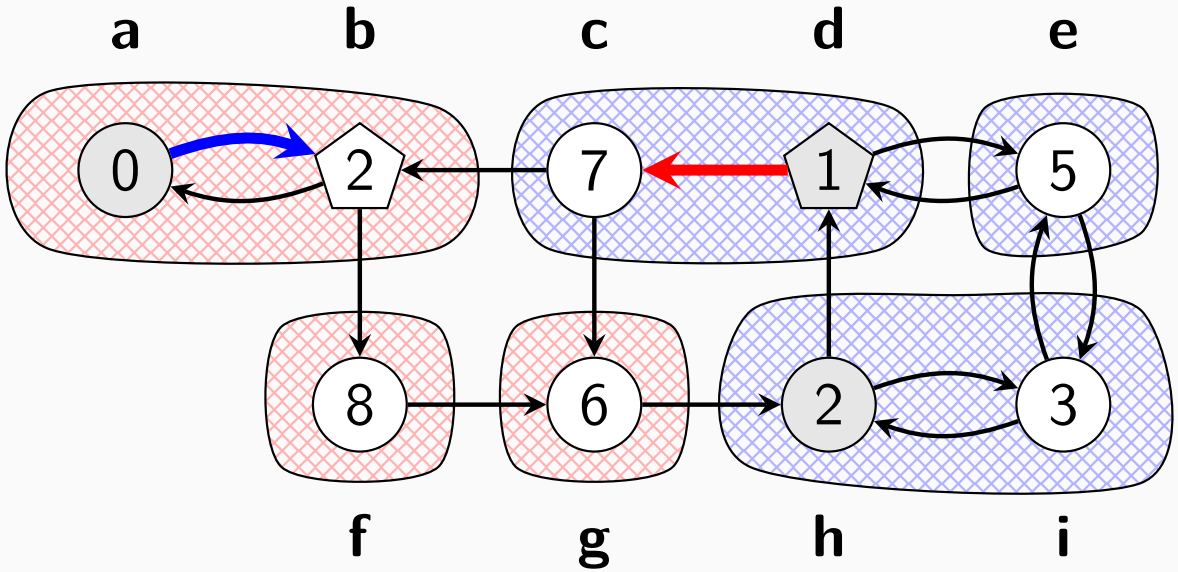
- Which regions are now closed?

TANGLE LEARNING



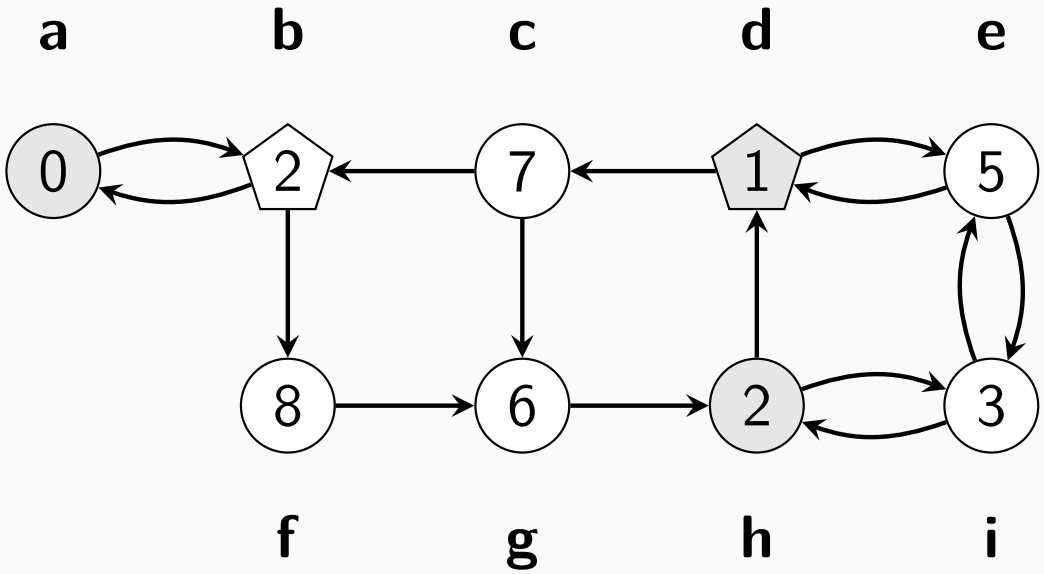
- Which regions are now closed?
- Regions 2 and 3 are locally closed (in their subgame)

TANGLE LEARNING



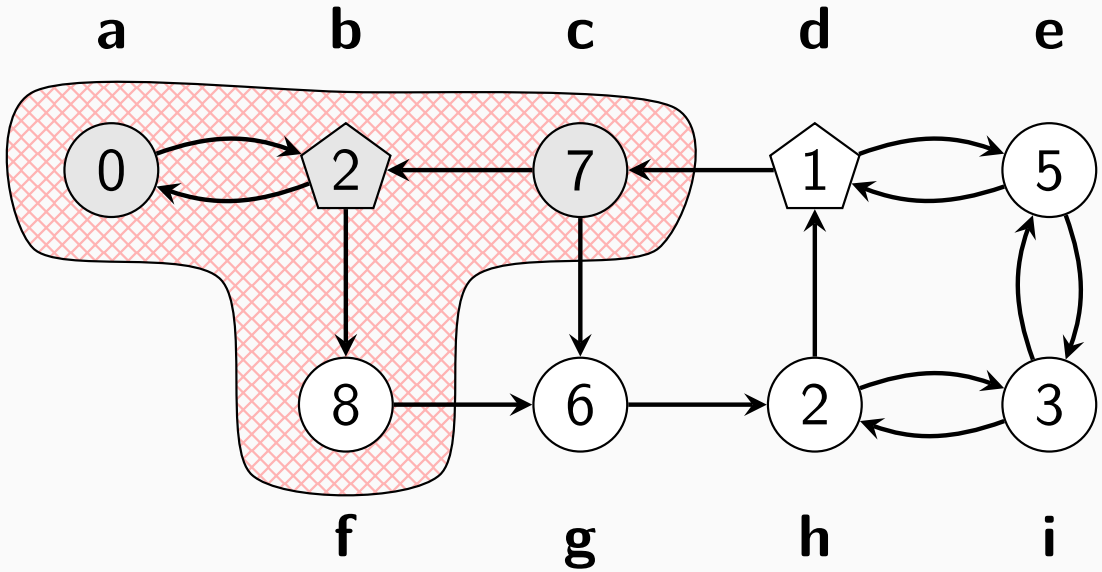
- Which regions are now closed?
- Regions 2 and 3 are locally closed (in their subgame)
- Tangle $(\{\mathbf{a}, \mathbf{b}\}, \{\mathbf{a} \to \mathbf{b}\})$ (attracted to 8)
- Tangle $(\{\mathbf{h}, \mathbf{i}\}, \emptyset)$ (attracted to 5)

TANGLE LEARNING



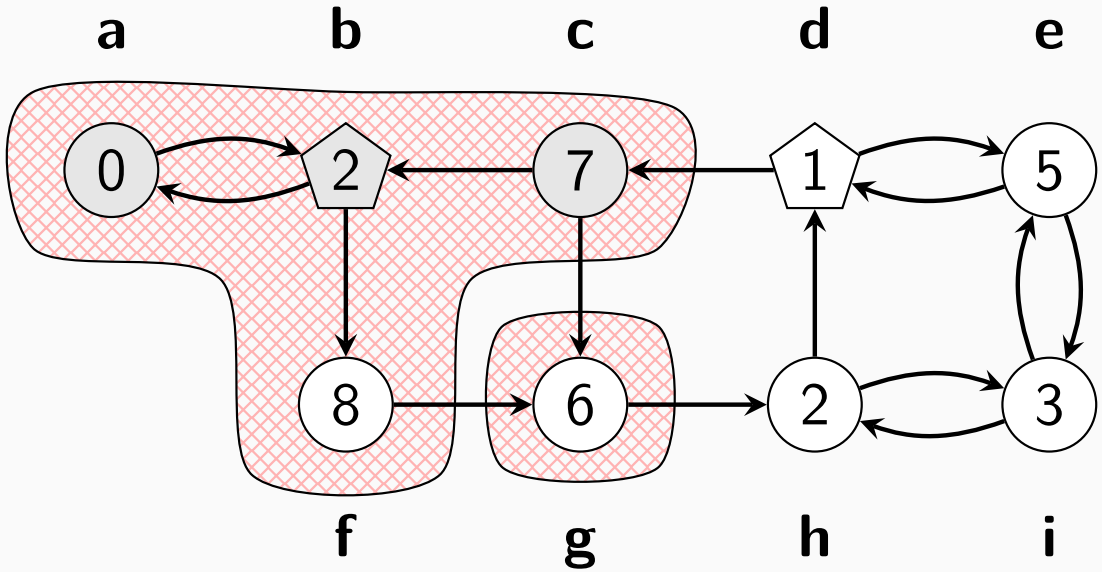
- Again partition top-down into α-maximal regions
- Now we have the two new tangles!

TANGLE LEARNING



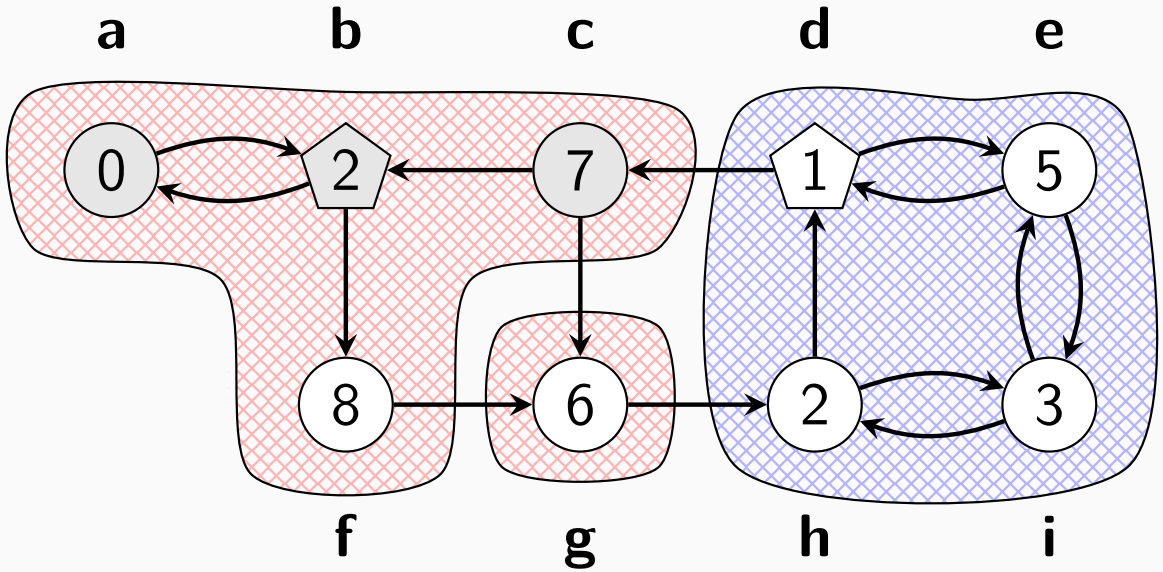
- Tangles: $\{\,\mathbf{a}, \mathbf{b}\,\}$ (for Even) and $\{\,\mathbf{h}, \mathbf{i}\,\}$ (for Odd)

TANGLE LEARNING



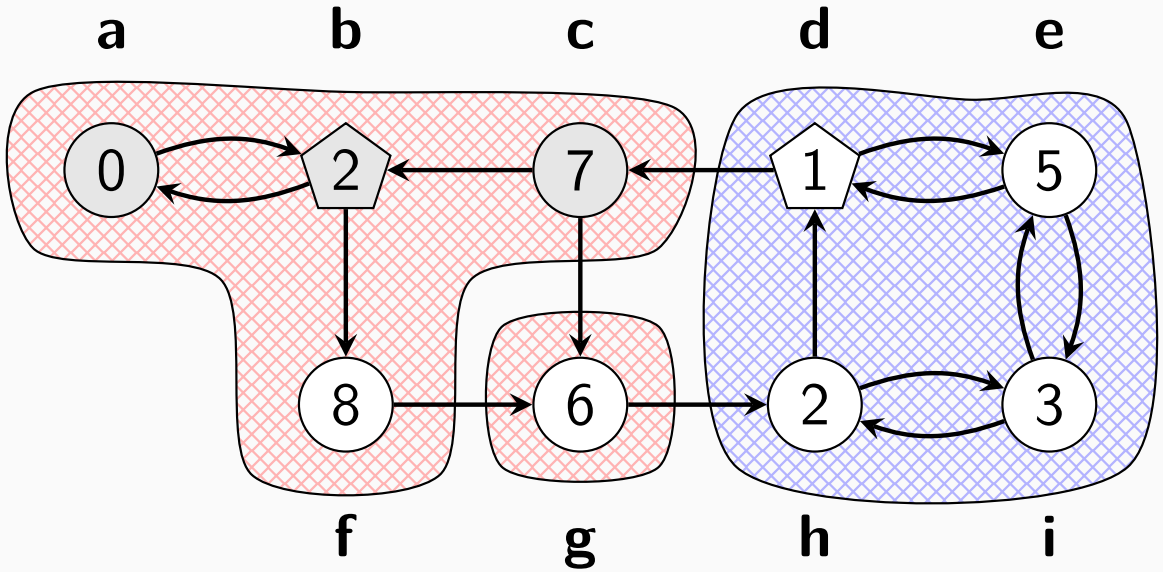
- Tangles: $\{\,\mathbf{a}, \mathbf{b}\,\}$ (for Even) and $\{\,\mathbf{h}, \mathbf{i}\,\}$ (for Odd)

TANGLE LEARNING



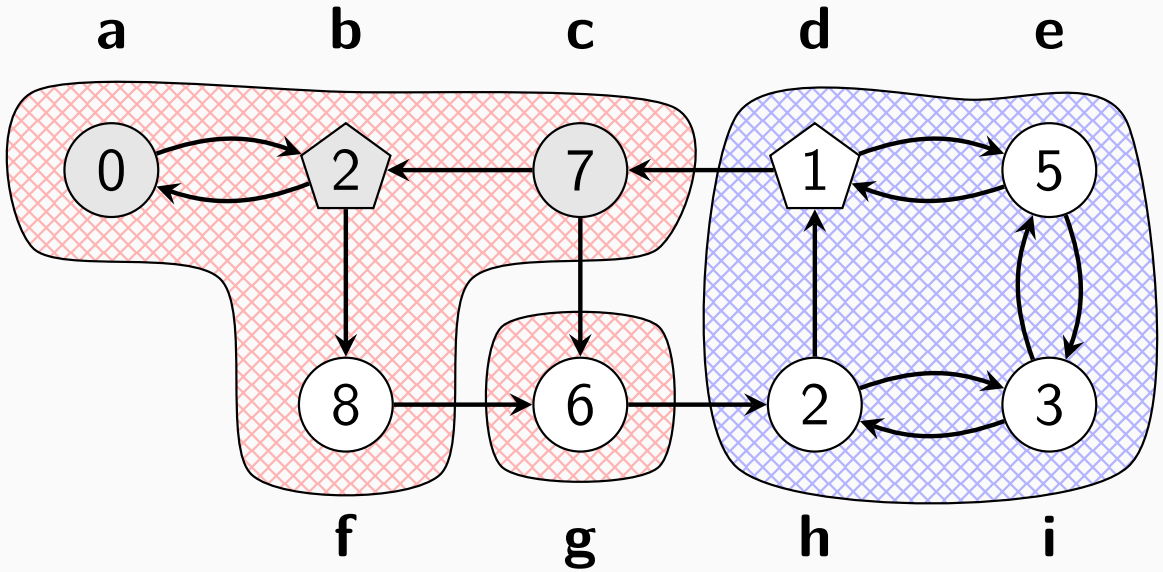
- Tangles: $\{\,\mathbf{a}, \mathbf{b}\,\}$ (for Even) and $\{\,\mathbf{h}, \mathbf{i}\,\}$ (for Odd)

TANGLE LEARNING



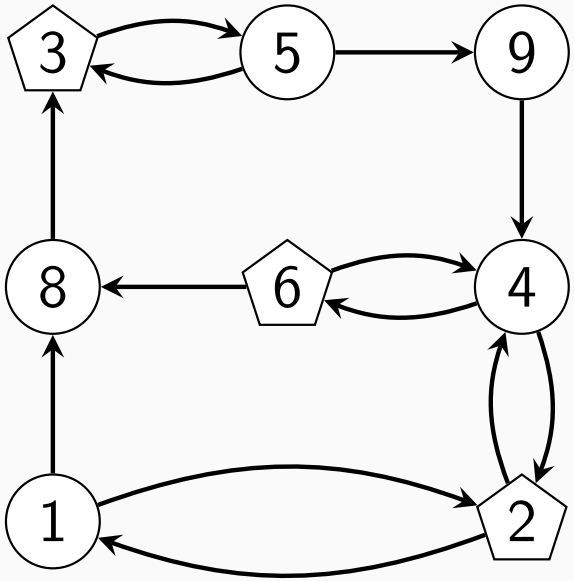
- Tangles: $\{\,\mathbf{a}, \mathbf{b}\,\}$ (for Even) and $\{\,\mathbf{h}, \mathbf{i}\,\}$ (for Odd)
- Which regions are now closed?

TANGLE LEARNING

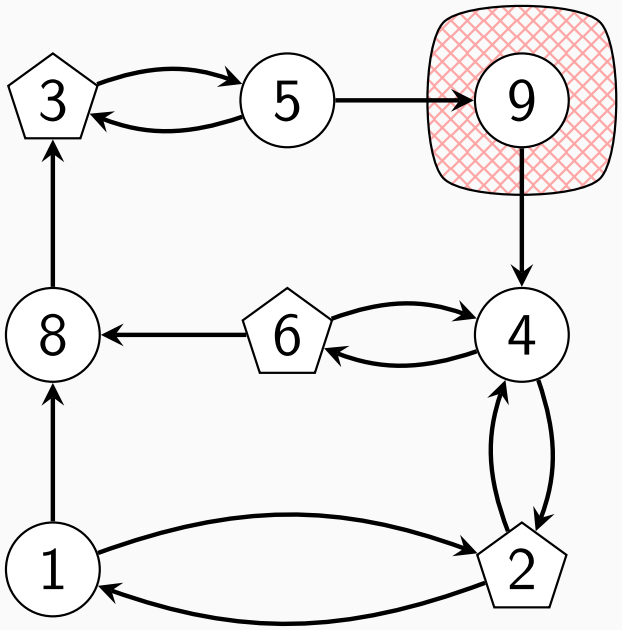


- Tangles: $\{\,\mathbf{a}, \mathbf{b}\,\}$ (for Even) and $\{\,\mathbf{h}, \mathbf{i}\,\}$ (for Odd)
- Which regions are now closed?
- Only closed region: 5
- One tangle, which is also a dominion

TANGLE LEARNING



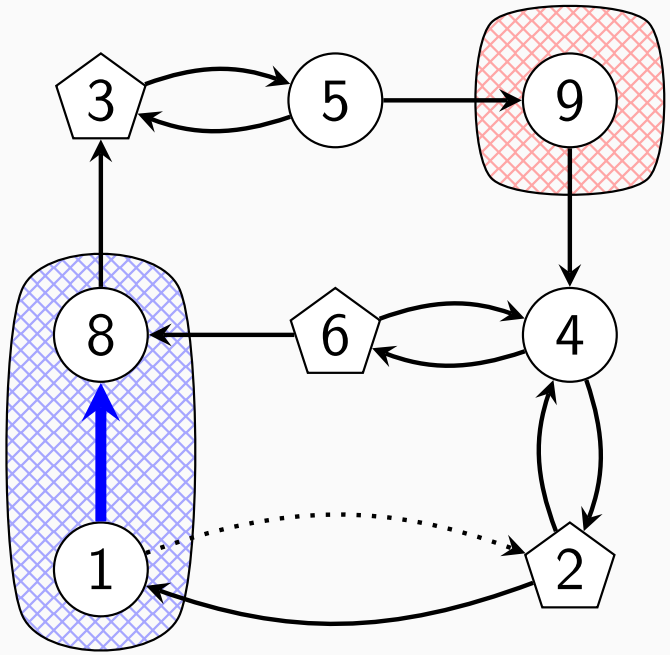
TANGLE LEARNING



Regions:

- 9 (open)

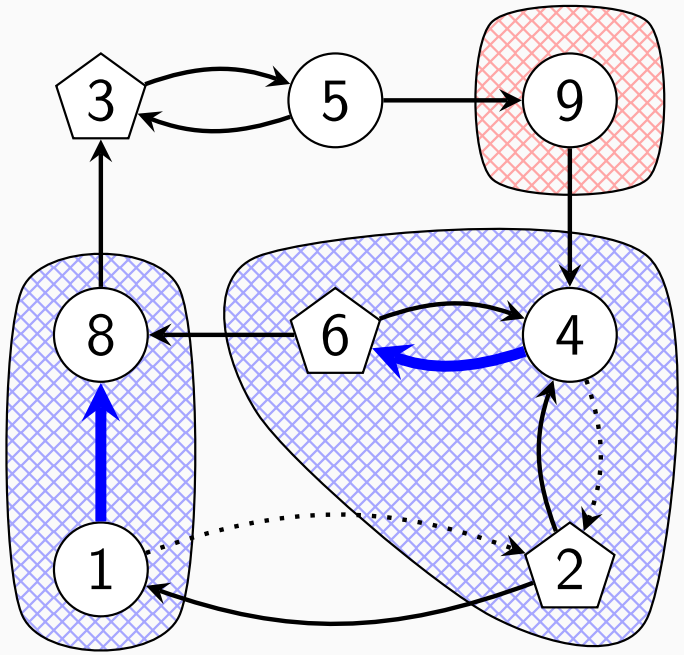
TANGLE LEARNING



Regions:

- 9 (open)
- 8 1 (open)

TANGLE LEARNING

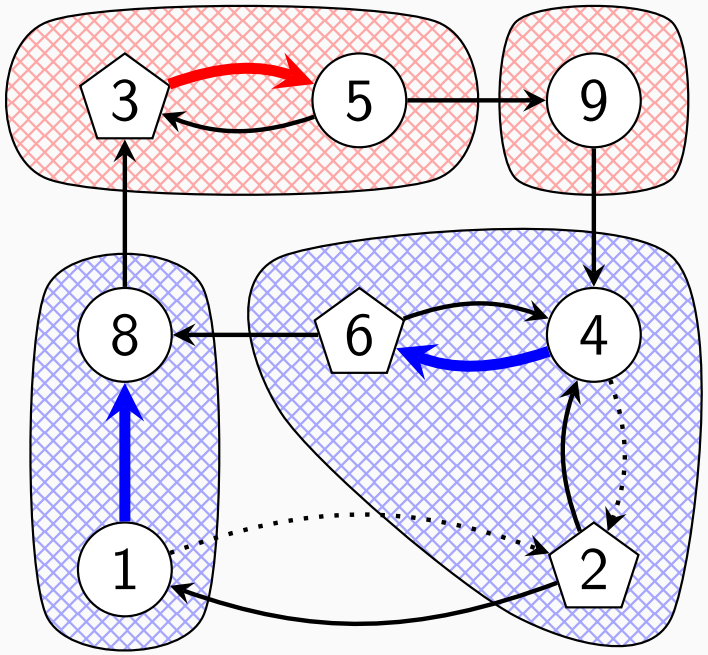


Regions:

- 9 (open)
- 8 1 (open)
- 6 4 2 (closed)

Learned tangles: $\{6, 4 \rightarrow 6\}$

TANGLE LEARNING

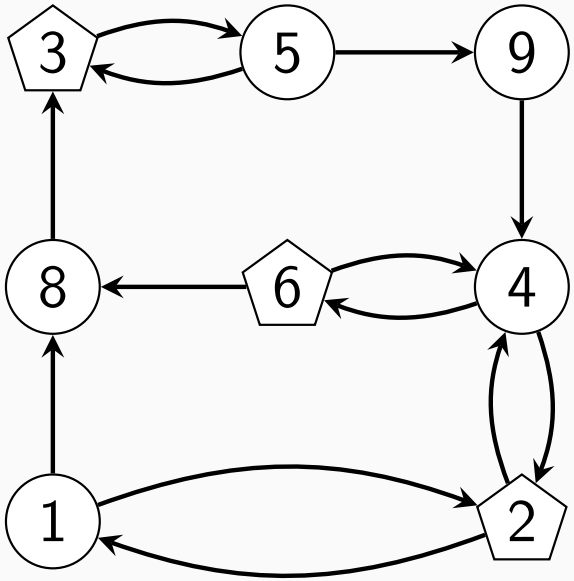


Regions:

- 9 (open)
- 8 1 (open)
- 6 4 2 (closed)
- 5 3 (closed)

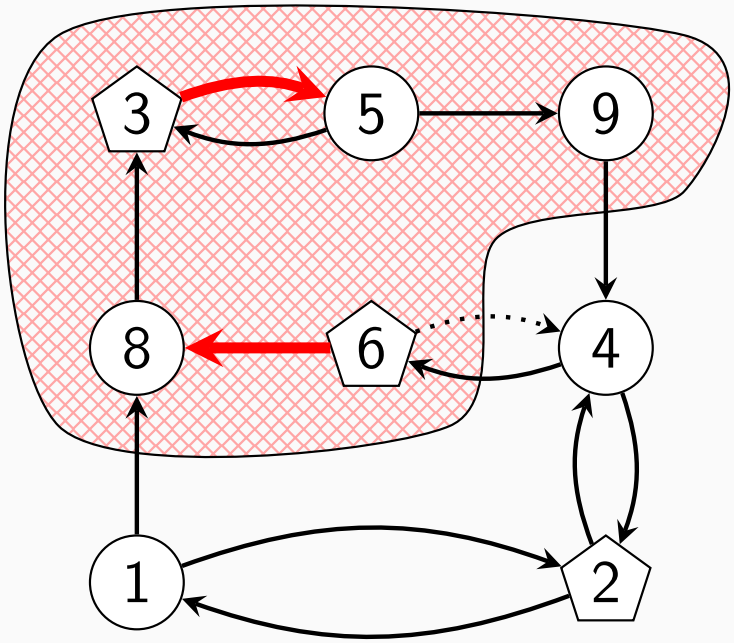
Learned tangles: $\{6, 4{\rightarrow}6\}$, $\{5, 3{\rightarrow}5\}$

TANGLE LEARNING



Learned tangles: $\{6, 4{\rightarrow}6\}$, $\{5, 3{\rightarrow}5\}$

TANGLE LEARNING

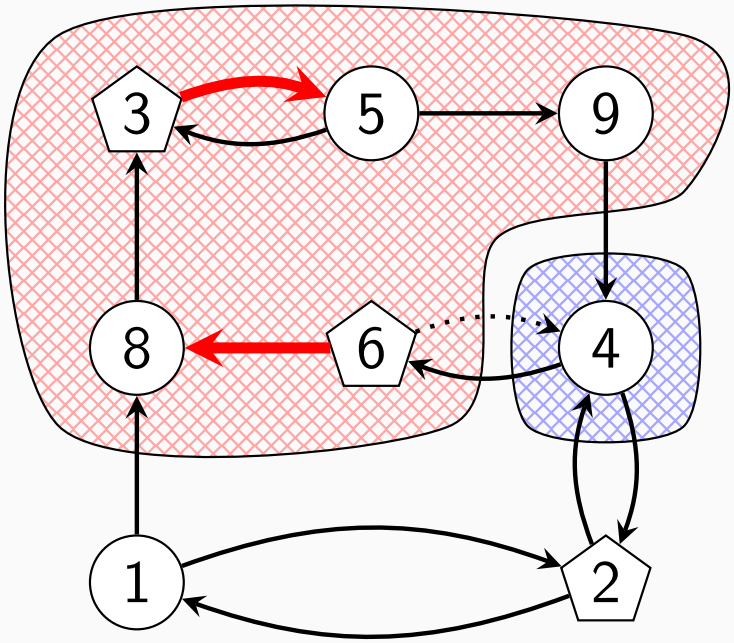


Regions:

- 9 5 3 8 6 (open)

Learned tangles: $\{6, 4{\rightarrow}6\}$, $\{5, 3{\rightarrow}5\}$

TANGLE LEARNING

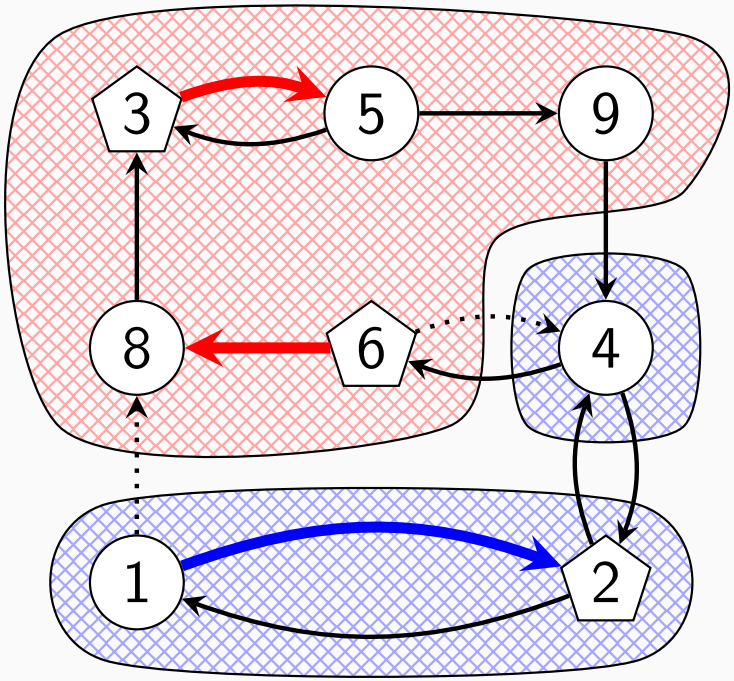


Regions:

- 9 5 3 8 6 (open)
- 4 (open)

Learned tangles: $\{6, 4{\rightarrow}6\}$, $\{5, 3{\rightarrow}5\}$

TANGLE LEARNING

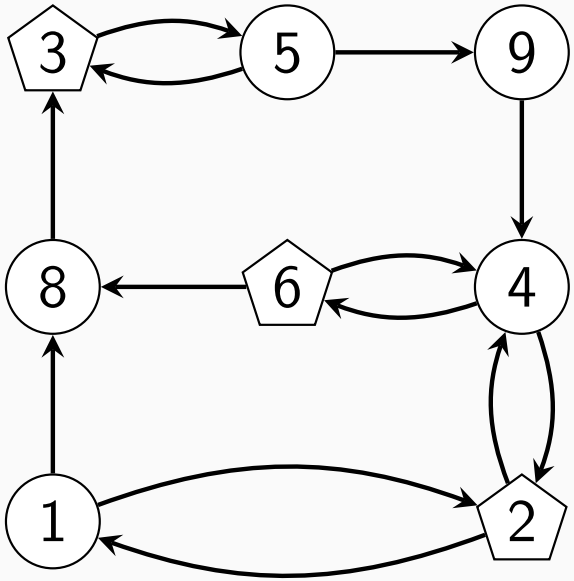


Regions:

- 9 5 3 8 6 (open)
- 4 (open)
- 2 1 (closed)

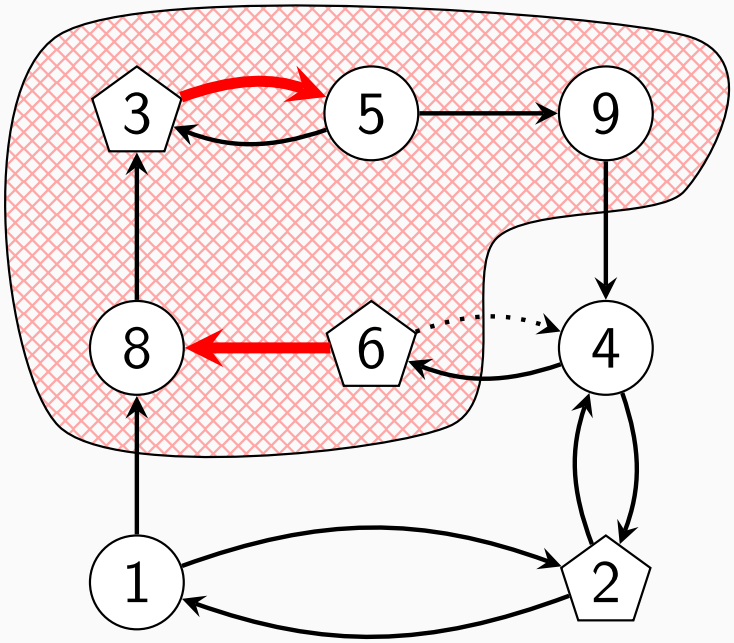
Learned tangles: $\{6, 4{\rightarrow}6\}$, $\{5, 3{\rightarrow}5\}$, $\{2, 1{\rightarrow}2\}$

TANGLE LEARNING



Learned tangles: $\{6, 4{\rightarrow}6\}$, $\{5, 3{\rightarrow}5\}$, $\{2, 1{\rightarrow}2\}$

TANGLE LEARNING

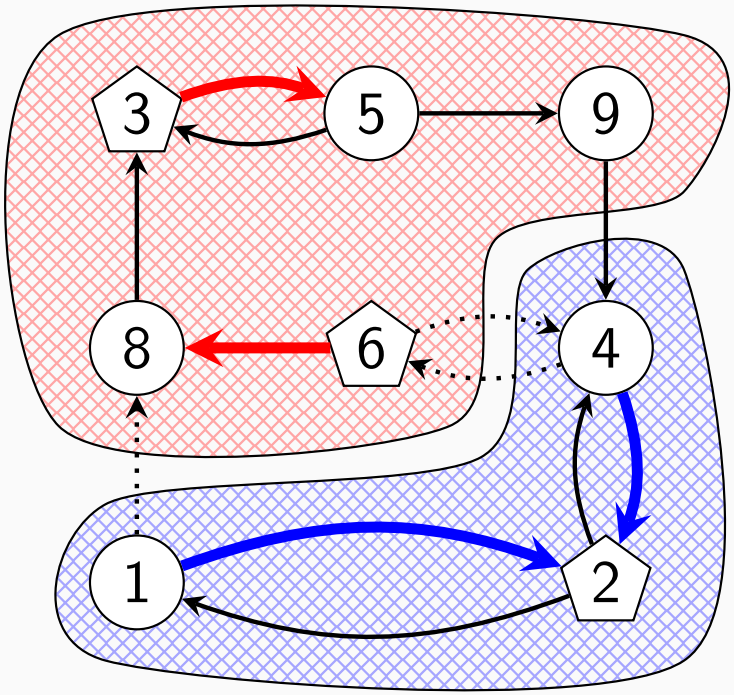


Regions:

- 9 5 3 8 6 (open)

Learned tangles: $\{6, 4{\rightarrow}6\}$, $\{5, 3{\rightarrow}5\}$, $\{2, 1{\rightarrow}2\}$

TANGLE LEARNING

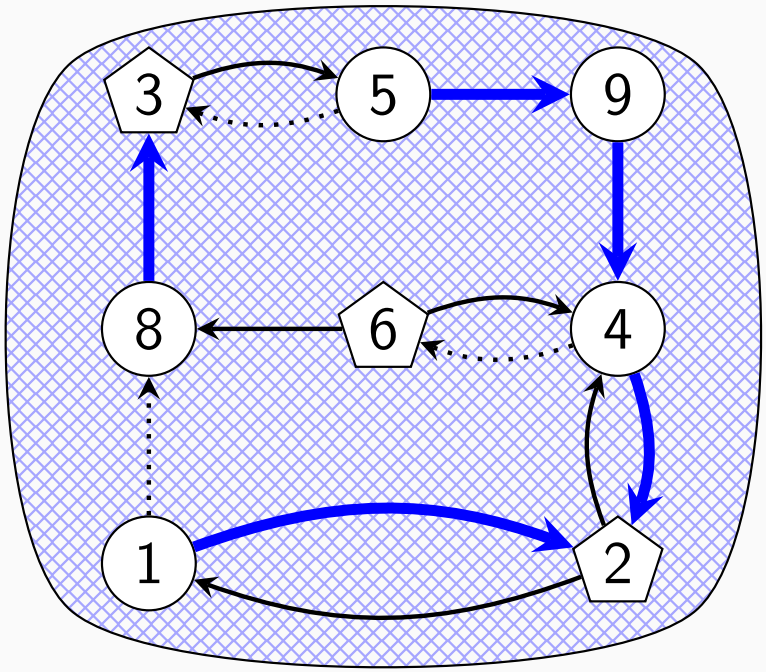


Regions:

- 9 5 3 8 6 (open)
- 4 2 1 (closed)

Learned tangles: $\{6, 4{\to}6\}$, $\{5, 3{\to}5\}$, $\{2, 1{\to}2\}$, $\underline{\{4{\to}2, 2, 1{\to}2\}}$

TANGLE LEARNING



Learned tangles: $\{6, 4{\rightarrow}6\}$, $\{5, 3{\rightarrow}5\}$, $\{2, 1{\rightarrow}2\}$, $\underline{\{4{\rightarrow}2, 2, 1{\rightarrow}2\}}$

TANGLE LEARNING

```
1  def tangle-attract(𝒢, 𝒯, Z, R, α, σ):
2      Q ← Z                                   // initialize with all target vertices
3      while Q ≠ ∅ :
4          v ← take(Q)                                          // take a v ∈ Q
5          foreach u : v ∈ E(u) do                        // check incoming edges
6              if u ∈ R ∧ u ∉ Z ∧ (u ∈ V_α ∨ E(u) ⊆ Z) :          // attract?
7                  Z ← Z ∪ {u}
8                  Q ← Q ∪ {u}
9              if u ∈ Z ∩ V_α ∧ u ∉ σ : σ[u] ← v                // set strategy?
10         foreach (U, τ) ∈ 𝒯_α : v ∈ E(U) do     // check adjacent tangles
11             if U ⊆ R ∪ Z ∧ E_T(U) ⊆ Z :                        // attract?
12                 U' ← U \ Z                    // only vertices not yet in Z
13                 Z ← Z ∪ U
14                 Q ← Q ∪ U
15                 σ ← σ ∪ (τ ∩ U')
16     return Z, σ
```

TANGLE LEARNING (1/2)

- `search` returns new tangles of $\mathcal{G}$, given known tangles $\mathcal{T}$.

```
1 def search(𝒢, 𝒯):
2     if 𝒢 = ∅ : return ∅
3     p ← pr(𝒢)                                          // highest priority
4     α ← pr(𝒢) mod 2                                    // player winning p
5     Z, σ ← tangle-attract(𝒢,𝒯,pr⁻¹(p),V,α,∅)                    // region
6     if Z_α = dom(σ) ∧ E(Z_ᾱ) ⊆ Z :                    // no open vertices?
7         return search(𝒢 \ Z, 𝒯) ∪ bottom-sccs(Z, σ)
8     else:                                               // open vertices?
9         return search(𝒢 \ Z, 𝒯)
```

TANGLE LEARNING (2/2)

1 **def tanglelearning($\mathcal{G}$):**
2 $W_{\bullet} \leftarrow \emptyset$, $\sigma_{\bullet} \leftarrow \emptyset$, $W_{\pentagon} \leftarrow \emptyset$, $\sigma_{\pentagon} \leftarrow \emptyset$, $\mathcal{T} \leftarrow \emptyset$
3 **while** $\mathcal{G} \neq \emptyset$ **:**
4 $Y \leftarrow$ **search**$(\mathcal{G}, \mathcal{T})$
5 $\mathcal{T} \leftarrow \mathcal{T} \cup \{ (U, \tau) \in Y \mid E_{\mathcal{T}}(U) \neq \emptyset \}$
6 $D \leftarrow \{ (U, \tau) \in Y \mid E_{\mathcal{T}}(U) = \emptyset \}$
7 **if** $D \neq \emptyset$ **:**
8 $D^{+}_{\bullet}, \sigma \leftarrow$**tangle-attract** $(\mathcal{G}, \mathcal{T}, \bigcup D_{\bullet}, V, \bullet, \bigcup D_{\bullet})$
9 $W_{\bullet} \leftarrow W_{\bullet} \cup D^{+}_{\bullet}$, $\sigma_{\bullet} \leftarrow \sigma_{\bullet} \cup \sigma$
10 $D^{+}_{\pentagon}, \sigma \leftarrow$**tangle-attract** $(\mathcal{G}, \mathcal{T}, \bigcup D_{\pentagon}, V, \pentagon, \bigcup D_{\pentagon})$
11 $W_{\pentagon} \leftarrow W_{\pentagon} \cup D^{+}_{\pentagon}$, $\sigma_{\pentagon} \leftarrow \sigma_{\pentagon} \cup \sigma$
12 $\mathcal{G} \leftarrow \mathcal{G} \setminus (D^{+}_{\bullet} \cup D^{+}_{\pentagon})$
13 $\mathcal{T} \leftarrow \mathcal{T} \cap \mathcal{G}$
14 **return** $W_{\bullet}, W_{\pentagon}, \sigma_{\bullet}, \sigma_{\pentagon}$

TANGLE LEARNING

Computing strategy

- Similar to priority promotion: compute strategy with the attractor, select any successor in the region for the highest priority vertices of α
- Store the σ of every tangle and use the stored σ as the strategy for α when attracting a tangle
- If a region is closed, then all α's vertices have a strategy
  - Suffices to look at the "target vertices" (highest priority)

Implementation notes

- Store tangle as interleaved list: $\alpha, u_0, \sigma(u_0), u_1, \sigma(u_1), \ldots, u_n, \sigma(u_n)$ (the player winning the tangle, followed by the vertices and the strategy)
- Store a separate list of escapes $E_T(U)$
- Don't get distracted by the mathematical notation... Put aside logic. Do what feels right.

OUTLINE

1. Background / Applications of parity games
   - Ω-automata
   - Modal μ-calculus
   - LTL synthesis
2. Parity game solving algorithms
   - Introduction
   - Decomposition based algorithms
   - Local algorithms
3. Quasi-polynomial algorithms
   - Universal trees and progress measures
   - Ordered progress measures
   - Quasi-polynomial recursive algorithm
4. Final remarks

VALUE ITERATION

Core idea of value iteration (1/2)

- Measure $\rho\colon V \to \mathbb{M}$ assign a value to every vertex from some domain $\mathbb{M}$, containing a special symbol $\top$.
- The measure represents how good is the "best" continuation?
  - "best" for one of the players, e.g., Even
  - symbol $\top$ means "winning for the player" (Even)
  - Even wants high values, Odd wants low values
- A monotone* function $\texttt{Prog}(m, p)$ that computes the value of playing from a vertex with priority p to a vertex with measure m
- ρ is the least parity game progress measure, if smallest ρ such that:

$$\forall v \in V\colon \rho(v) = \begin{cases} \max_{\sqsubset}\{\texttt{Prog}(\rho(w), \mathrm{pr}(v)) \mid w \in E(v)\} & v \in V_{\bigcirc} \\ \min_{\sqsubset}\{\texttt{Prog}(\rho(w), \mathrm{pr}(v)) \mid w \in E(v)\} & v \in V_{\pentagon} \end{cases}$$

*with respect to a special ordering (see later)

Core idea of value iteration (2/2)

- If ρ is the least parity game progress measure, then:
  - $W_{\bigcirc} = \{v \mid \rho(v) = \top\}$, $W_{\pentagon} = \{v \mid \rho(v) \neq \top\}$
  - if $v \in W_{\pentagon}$, then $\rho(v) = \texttt{Prog}\big(\rho(\sigma(w)), \mathsf{pr}(v)\big)$
    meaning: the winning strategy for Odd is the best continuation
  - no* winning strategy for Even
- It is a least fixed point: starting with $\bot$, update ρ until fixed point
- This is called lifting the measures
- *Idea*: this is like playing an "optimal" game backwards
  - Player Even finds better paths
  - Player Odd then selects the least bad option

*except with some extra effort: Gazda and Willemse, 2014

Outline 1/3

- Every vertex has a value a.k.a. measure from an ordered set M
  Meaning: how much player Odd "prefers" a vertex

Outline 1/3

- Every vertex has a value a.k.a. measure from an ordered set M
  Meaning: how much player Odd "prefers" a vertex
- Special top element $\top \in M$
  Meaning: definitely won by Odd

Outline 1/3

- Every vertex has a value a.k.a. measure from an ordered set M
  Meaning: how much player Odd "prefers" a vertex
- Special top element $\top \in M$
  Meaning: definitely won by Odd
- An edge (v, u) is μ-progressive if:
  - if $\mathsf{pr}(v)$ is even and $\mu(v) \geq_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) >_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) = \mu(u) = \top$

  Meaning: increase value when we see odd priorities, until $\top$

SMALL PROGRESS MEASURES

Outline 1/3

- Every vertex has a value a.k.a. measure from an ordered set M
  Meaning: how much player Odd "prefers" a vertex
- Special top element $\top \in M$
  Meaning: definitely won by Odd
- An edge (v, u) is μ-progressive if:
  - if $\mathsf{pr}(v)$ is even and $\mu(v) \geq_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) >_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) = \mu(u) = \top$

  Meaning: increase value when we see odd priorities, until $\top$
- A mapping $\mu \colon V \to M$ is an Even progress measure if:
  - for every even-controlled vertex: some edge is μ-progressive
  - for every odd-controlled vertex: all edges are μ-progressive
  - and it is the smallest μ where this condition holds

  Meaning: Even chooses lowest escape; Odd chooses best strategy

Outline 2/3

- A mapping $\mu\colon V \to M$ is an Even progress measure if:
  - every even-controlled vertex: some edge is μ-progressive
  - every odd-controlled vertex: all edges are μ-progressive
  - and it is the smallest μ where this condition holds
- An edge (v, u) is μ-progressive if:
  - if $\mathsf{pr}(v)$ is even and $\mu(v) \geq_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) >_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) = \mu(u) = \top$

Outline 2/3

- A mapping $\mu\colon V \to M$ is an Even progress measure if:
  - every even-controlled vertex: some edge is μ-progressive
  - every odd-controlled vertex: all edges are μ-progressive
  - and it is the smallest μ where this condition holds
- An edge (v, u) is μ-progressive if:
  - if $\mathsf{pr}(v)$ is even and $\mu(v) \geq_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) >_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) = \mu(u) = \top$
- Even progress measure is a least fixed point
  - Start with $\mu = V \to \bot$ (the lowest element of M)
  - Repeatedly "lift" value $\mu(v)$ to locally repair the condition
    - Even-controlled: find lowest progressive successor (the escape for Even)
    - Odd-controlled: find highest progressive successor (the strategy for Odd)

Outline 2/3

- A mapping $\mu\colon V \to M$ is an Even progress measure if:
  - every even-controlled vertex: some edge is μ-progressive
  - every odd-controlled vertex: all edges are μ-progressive
  - and it is the smallest μ where this condition holds
- An edge (v, u) is μ-progressive if:
  - if $\mathsf{pr}(v)$ is even and $\mu(v) \geq_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) >_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) = \mu(u) = \top$
- Even progress measure is a least fixed point
  - Start with $\mu = V \to \bot$ (the lowest element of M)
  - Repeatedly "lift" value $\mu(v)$ to locally repair the condition
    - Even-controlled: find lowest progressive successor (the escape for Even)
    - Odd-controlled: find highest progressive successor (the strategy for Odd)
- After computing the Even progress measure:
  - all $\top$ won by Odd, all non-$\top$ won by Even
  - strategy for Even: choose any μ-progressive edge

Outline 3/3

- Measures are tuples $\langle r_7, r_5, r_3, r_1 \rangle$ (with highest odd priority 7)
- Each $r_p = [0..n_p]$ with n_p the number of vertices with priority p
- Example: $M = (\{0\} \times \{0,1,2\} \times \{0\} \times \{0,1\}) \cup \{\top\}$ if there are 2 vertices of priority 5 and 1 vertex of priority 1.

Outline 3/3

- Measures are tuples $\langle r_7, r_5, r_3, r_1 \rangle$ (with highest odd priority 7)
- Each $r_p = [0..n_p]$ with n_p the number of vertices with priority p
- Example: $M = (\{0\} \times \{0,1,2\} \times \{0\} \times \{0,1\}) \cup \{\top\}$ if there are 2 vertices of priority 5 and 1 vertex of priority 1.
- A total order $<$ which is lexicographic:

$$\begin{aligned} \langle 1,0,0,0 \rangle &< \langle 1,0,0,1 \rangle \\ \langle 4,2,10,5 \rangle &< \langle 4,3,0,0 \rangle \end{aligned}$$

Outline 3/3

- Measures are tuples $\langle r_7, r_5, r_3, r_1 \rangle$ (with highest odd priority 7)
- Each $r_p = [0..n_p]$ with n_p the number of vertices with priority p
- Example: $M = (\{0\} \times \{0,1,2\} \times \{0\} \times \{0,1\}) \cup \{\top\}$ if there are 2 vertices of priority 5 and 1 vertex of priority 1.
- A total order $<$ which is lexicographic:

$$\begin{aligned} \langle 1,0,0,0 \rangle &< \langle 1,0,0,1 \rangle \\ \langle 4,2,10,5 \rangle &< \langle 4,3,0,0 \rangle \end{aligned}$$

- A p-truncation keeps only elements r_i with $i \geq p$:

$$\begin{aligned} \langle 1,2,3,2 \rangle|_2 &= \langle 1,2,3 \rangle \\ \langle 1,2,3,2 \rangle|_5 &= \langle 1,2 \rangle \\ \langle 1,2,3,2 \rangle|_8 &= \langle \rangle \end{aligned}$$

Outline 3/3

- Measures are tuples $\langle r_7, r_5, r_3, r_1 \rangle$ (with highest odd priority 7)
- Each $r_p = [0..n_p]$ with n_p the number of vertices with priority p
- Example: $M = (\{0\} \times \{0,1,2\} \times \{0\} \times \{0,1\}) \cup \{\top\}$ if there are 2 vertices of priority 5 and 1 vertex of priority 1.
- A total order $<$ which is lexicographic:

$$\begin{aligned} \langle 1,0,0,0 \rangle &< \langle 1,0,0,1 \rangle \\ \langle 4,2,10,5 \rangle &< \langle 4,3,0,0 \rangle \end{aligned}$$

- A p-truncation keeps only elements r_i with $i \geq p$:

$$\begin{aligned} \langle 1,2,3,2 \rangle|_2 &= \langle 1,2,3 \rangle \\ \langle 1,2,3,2 \rangle|_5 &= \langle 1,2 \rangle \\ \langle 1,2,3,2 \rangle|_8 &= \langle \rangle \end{aligned}$$

- Additional notation: $m_1 >_p m_2 \Leftrightarrow m_1|_p > m_2|_p$
  Compare p-truncated measures (or: compare until r_p)

The Prog function

Playing from a vertex with priority p to a vertex with measure m yields:

$$\texttt{Prog}(m, p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{aligned} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) &= \end{aligned}$$

The Prog function

Playing from a vertex with priority p to a vertex with measure m yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{array}{lcl} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) & = & \langle 3,2,3,2\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) & = & \end{array}$$

The Prog function

Playing from a vertex with priority p to a vertex with measure m yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{array}{lcl} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) & = & \langle 3,2,3,2\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) & = & \langle 3,2,3,3\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) & = & \end{array}$$

The Prog function

Playing from a vertex with priority p to a vertex with measure m yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{array}{lcl} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) & = & \langle 3,2,3,2\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) & = & \langle 3,2,3,3\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) & = & \langle 3,2,3,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) & = & \end{array}$$

The Prog function

Playing from a vertex with priority p to a vertex with measure m yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{array}{lcl} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) & = & \langle 3,2,3,2\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) & = & \langle 3,2,3,3\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) & = & \langle 3,2,3,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) & = & \langle 3,3,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) & = & \end{array}$$

The Prog function

Playing from a vertex with priority p to a vertex with measure m yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{array}{lcl} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) & = & \langle 3,2,3,2\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) & = & \langle 3,2,3,3\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) & = & \langle 3,2,3,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) & = & \langle 3,3,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) & = & \langle 3,2,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) & = & \end{array}$$

The Prog function

Playing from a vertex with priority p to a vertex with measure m yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{array}{lcl} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) & = & \langle 3,2,3,2\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) & = & \langle 3,2,3,3\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) & = & \langle 3,2,3,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) & = & \langle 3,3,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) & = & \langle 3,2,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) & = & \langle 3,3,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) & = & \end{array}$$

The Prog function

Playing from a vertex with priority p to a vertex with measure m yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{aligned}
\texttt{Prog}(\langle 3,2,3,2\rangle, 0) &= \langle 3,2,3,2\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 1) &= \langle 3,2,3,3\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 2) &= \langle 3,2,3,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 3) &= \langle 3,3,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 4) &= \langle 3,2,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 5) &= \langle 3,3,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 6) &= \langle 3,0,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 7) &= \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 8) &=
\end{aligned}$$

The Prog function

Playing from a vertex with priority p to a vertex with measure m yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{array}{lcl} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) & = & \langle 3,2,3,2\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) & = & \langle 3,2,3,3\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) & = & \langle 3,2,3,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) & = & \langle 3,3,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) & = & \langle 3,2,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) & = & \langle 3,3,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) & = & \langle 3,0,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) & = & \top \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) & = & \end{array}$$

The Prog function

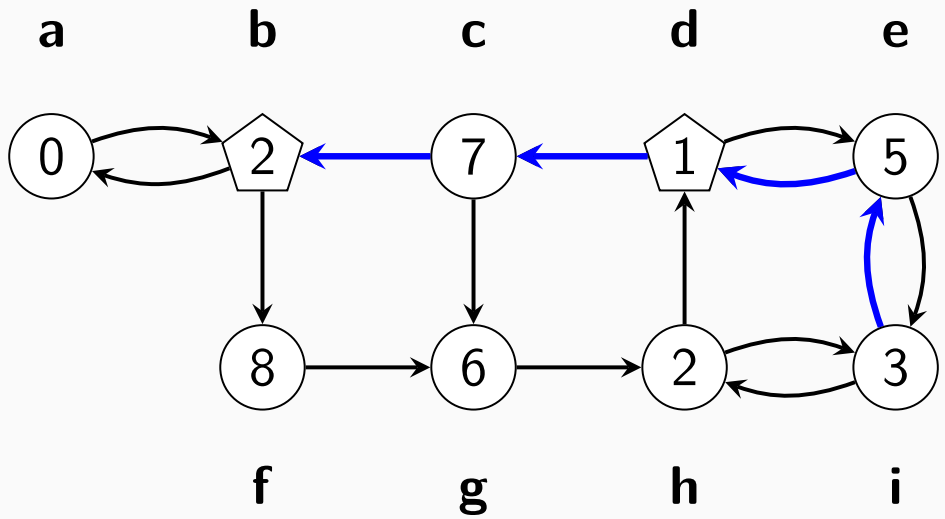
Playing from a vertex with priority p to a vertex with measure m yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

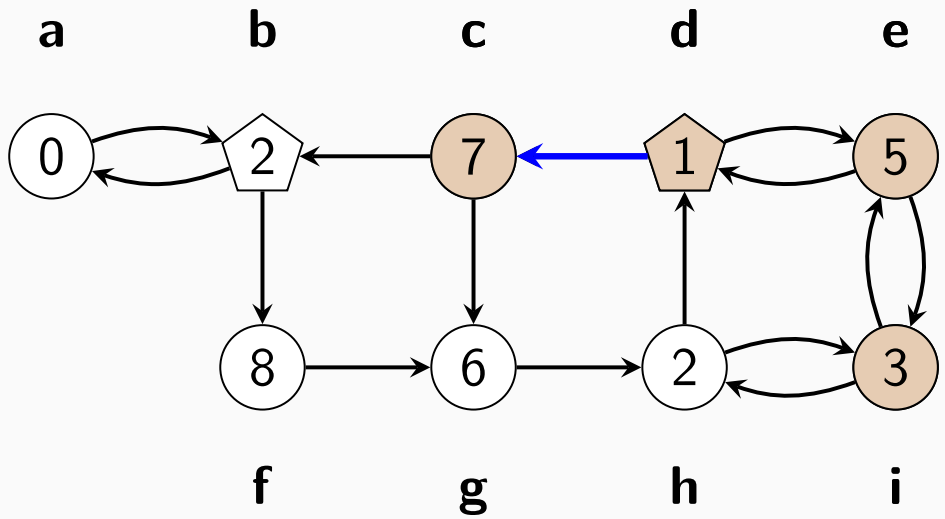
$$\begin{aligned}
\texttt{Prog}(\langle 3,2,3,2\rangle, 0) &= \langle 3,2,3,2\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 1) &= \langle 3,2,3,3\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 2) &= \langle 3,2,3,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 3) &= \langle 3,3,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 4) &= \langle 3,2,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 5) &= \langle 3,3,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 6) &= \langle 3,0,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 7) &= \top \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 8) &= \langle 0,0,0,0\rangle
\end{aligned}$$

SMALL PROGRESS MEASURES



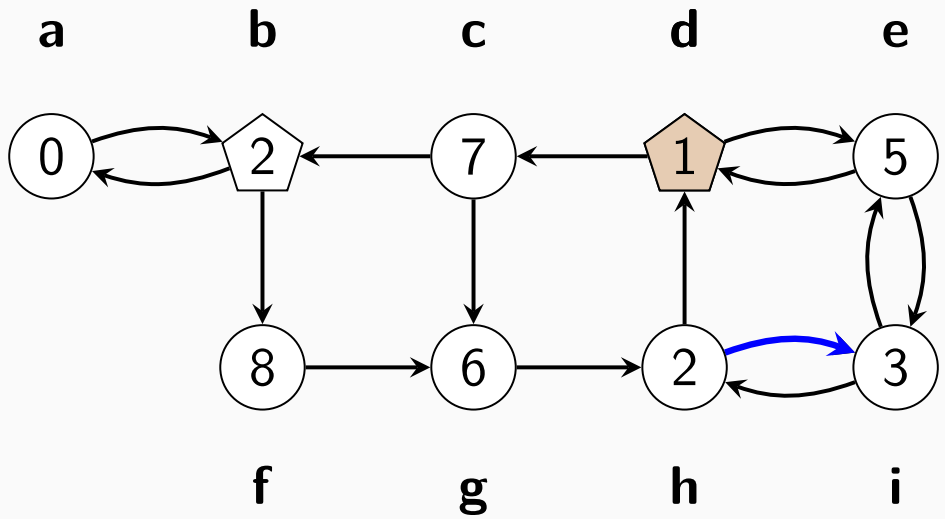
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 0,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,1\rangle$	1
e	$\langle 0,0,-,-\rangle$	to	$\langle 0,1,-,-\rangle$	5
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 0,-,-,-\rangle$	to	$\langle 0,-,-,-\rangle$	6
h	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
i	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,1,-\rangle$	3

SMALL PROGRESS MEASURES



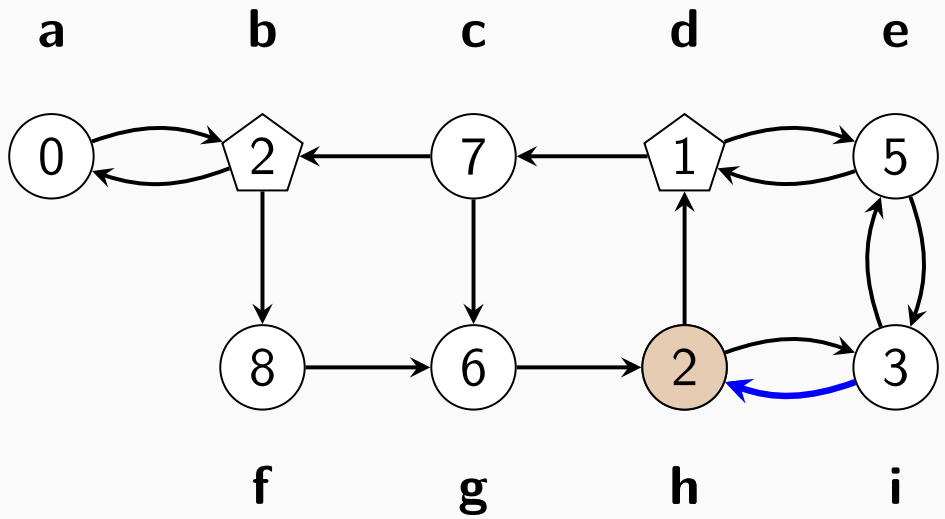
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 0,0,0,1\rangle$	to	$\langle 1,0,0,1\rangle$	17
e	$\langle 0,1,-,-\rangle$	to	$\langle 0,1,-,-\rangle$	5
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 0,-,-,-\rangle$	to	$\langle 0,-,-,-\rangle$	6
h	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
i	$\langle 0,0,1,-\rangle$	to	$\langle 0,0,1,-\rangle$	3

SMALL PROGRESS MEASURES



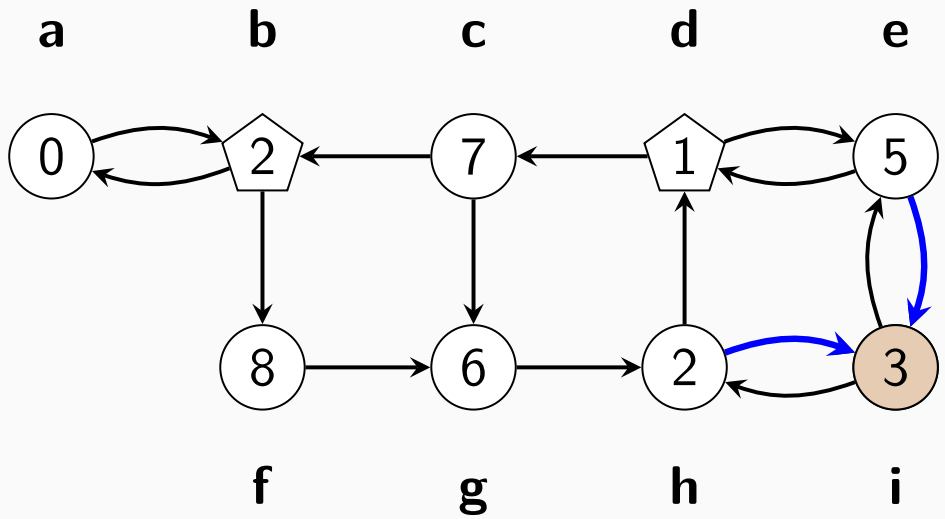
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 1,0,0,1\rangle$	to	$\langle 1,0,0,1\rangle$	17
e	$\langle 0,1,-,-\rangle$	to	$\langle 0,1,-,-\rangle$	5
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 0,-,-,-\rangle$	to	$\langle 0,-,-,-\rangle$	6
h	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,1,-\rangle$	23
i	$\langle 0,0,1,-\rangle$	to	$\langle 0,0,1,-\rangle$	3

SMALL PROGRESS MEASURES



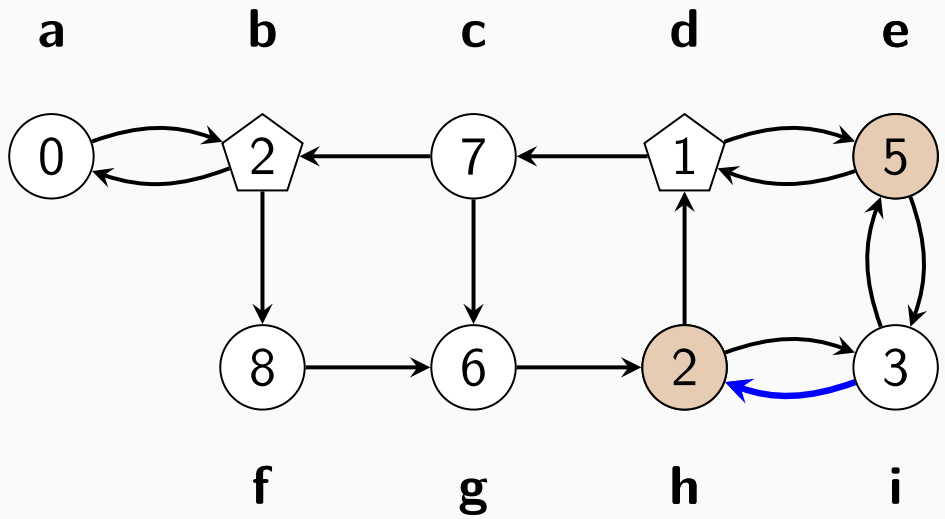
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 1,0,0,1\rangle$	to	$\langle 1,0,0,1\rangle$	17
e	$\langle 0,1,-,-\rangle$	to	$\langle 0,1,-,-\rangle$	5
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 0,-,-,-\rangle$	to	$\langle 0,-,-,-\rangle$	6
h	$\langle 0,0,1,-\rangle$	to	$\langle 0,0,1,-\rangle$	23
i	$\langle 0,0,1,-\rangle$	to	$\langle 0,1,0,-\rangle$	323

SMALL PROGRESS MEASURES



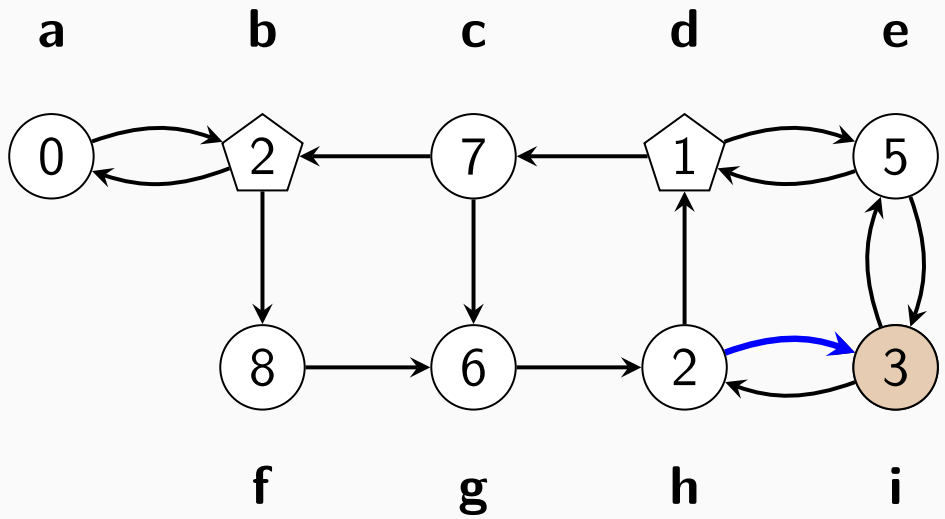
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 1,0,0,1\rangle$	to	$\langle 1,0,0,1\rangle$	17
e	$\langle 0,1,-,-\rangle$	to	$\langle 1,0,-,-\rangle$	5323
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 0,-,-,-\rangle$	to	$\langle 0,-,-,-\rangle$	6
h	$\langle 0,0,1,-\rangle$	to	$\langle 0,1,0,-\rangle$	2323
i	$\langle 0,1,0,-\rangle$	to	$\langle 0,1,0,-\rangle$	323

SMALL PROGRESS MEASURES



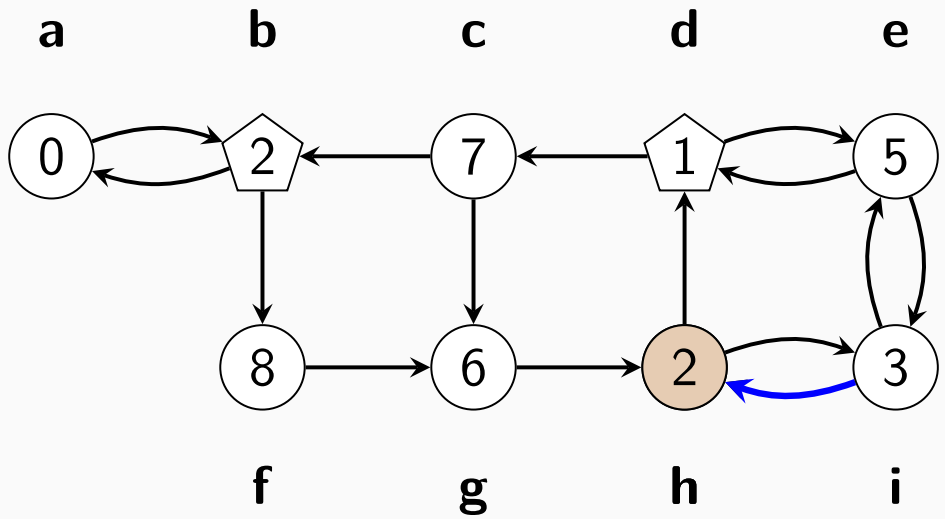
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 1,0,0,1\rangle$	to	$\langle 1,0,0,1\rangle$	17
e	$\langle 1,0,-,-\rangle$	to	$\langle 1,0,-,-\rangle$	5323
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 0,-,-,-\rangle$	to	$\langle 0,-,-,-\rangle$	6
h	$\langle 0,1,0,-\rangle$	to	$\langle 0,1,0,-\rangle$	2323
i	$\langle 0,1,0,-\rangle$	to	$\langle 0,1,1,-\rangle$	32323

SMALL PROGRESS MEASURES



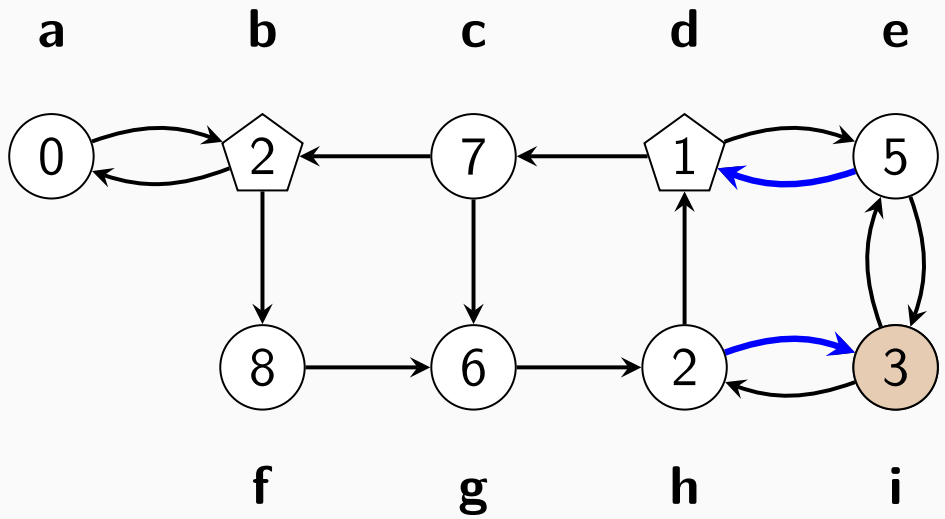
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 1,0,0,1\rangle$	to	$\langle 1,0,0,1\rangle$	17
e	$\langle 1,0,-,-\rangle$	to	$\langle 1,0,-,-\rangle$	5323
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 0,-,-,-\rangle$	to	$\langle 0,-,-,-\rangle$	6
h	$\langle 0,1,0,-\rangle$	to	$\langle 0,1,1,-\rangle$	232323
i	$\langle 0,1,1,-\rangle$	to	$\langle 0,1,1,-\rangle$	32323

SMALL PROGRESS MEASURES



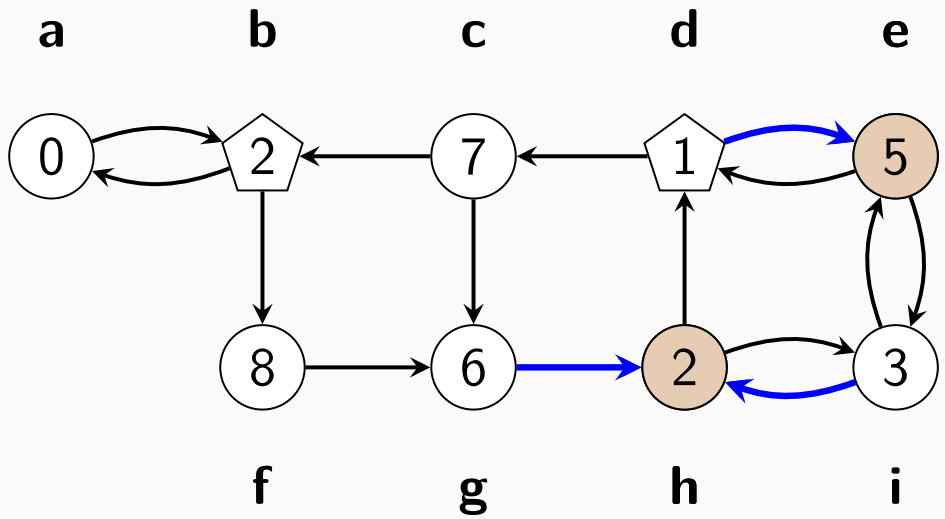
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 1,0,0,1\rangle$	to	$\langle 1,0,0,1\rangle$	17
e	$\langle 1,0,-,-\rangle$	to	$\langle 1,0,-,-\rangle$	5323
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 0,-,-,-\rangle$	to	$\langle 0,-,-,-\rangle$	6
h	$\langle 0,1,1,-\rangle$	to	$\langle 0,1,1,-\rangle$	232323
i	$\langle 0,1,1,-\rangle$	to	$\langle 1,0,0,-\rangle$	3232323

SMALL PROGRESS MEASURES



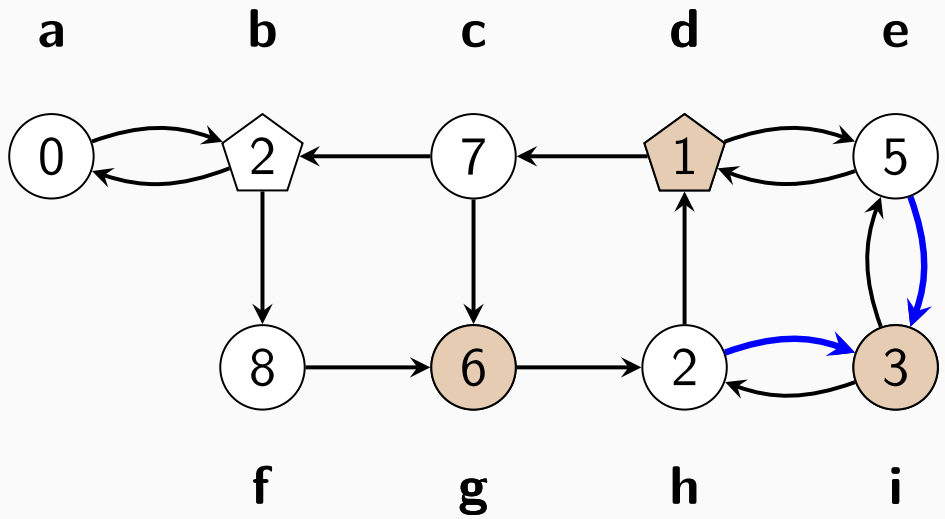
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 1,0,0,1\rangle$	to	$\langle 1,0,0,1\rangle$	17
e	$\langle 1,0,-,-\rangle$	to	$\langle 1,1,-,-\rangle$	517
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 0,-,-,-\rangle$	to	$\langle 0,-,-,-\rangle$	6
h	$\langle 0,1,1,-\rangle$	to	$\langle 1,0,0,-\rangle$	23232323
i	$\langle 1,0,0,-\rangle$	to	$\langle 1,0,0,-\rangle$	3232323

SMALL PROGRESS MEASURES



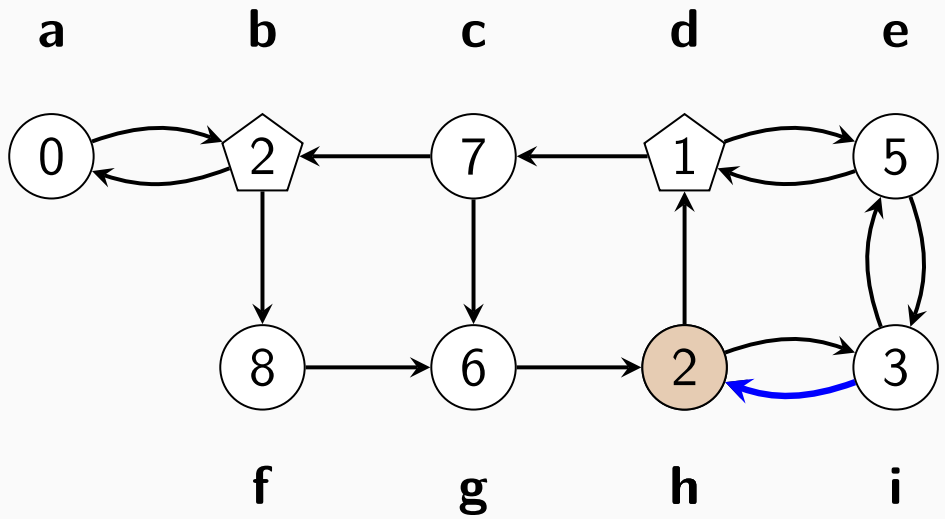
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 1,0,0,1\rangle$	to	$\langle 1,1,0,1\rangle$	1517
e	$\langle 1,1,-,-\rangle$	to	$\langle 1,1,-,-\rangle$	517
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 0,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	623232323
h	$\langle 1,0,0,-\rangle$	to	$\langle 1,0,0,-\rangle$	23232323
i	$\langle 1,0,0,-\rangle$	to	$\langle 1,0,1,-\rangle$	323232323

SMALL PROGRESS MEASURES



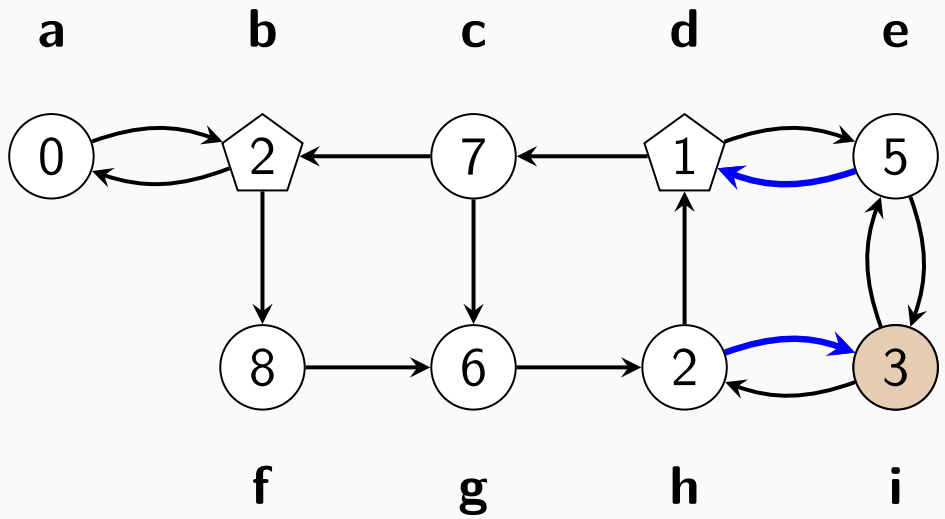
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 1,1,0,1\rangle$	to	$\langle 1,1,0,1\rangle$	1517
e	$\langle 1,1,-,-\rangle$	to	$\langle 1,1,-,-\rangle$	5323232323
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	623232323
h	$\langle 1,0,0,-\rangle$	to	$\langle 1,0,1,-\rangle$	2323232323
i	$\langle 1,0,1,-\rangle$	to	$\langle 1,0,1,-\rangle$	323232323

SMALL PROGRESS MEASURES



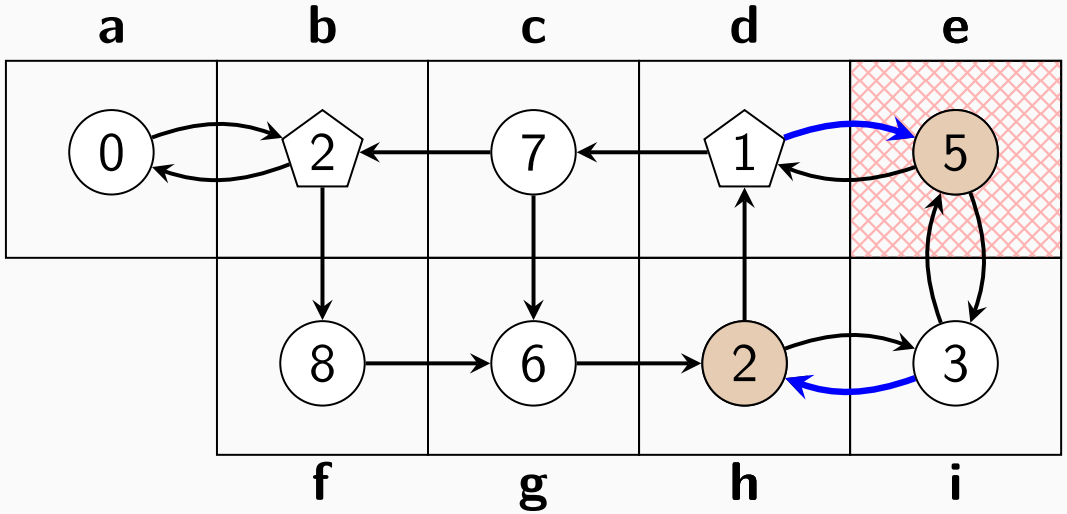
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 1,1,0,1\rangle$	to	$\langle 1,1,0,1\rangle$	1517
e	$\langle 1,1,-,-\rangle$	to	$\langle 1,1,-,-\rangle$	5323232323
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	623232323
h	$\langle 1,0,1,-\rangle$	to	$\langle 1,0,1,-\rangle$	2323232323
i	$\langle 1,0,1,-\rangle$	to	$\langle 1,1,0,-\rangle$	32323232323

SMALL PROGRESS MEASURES



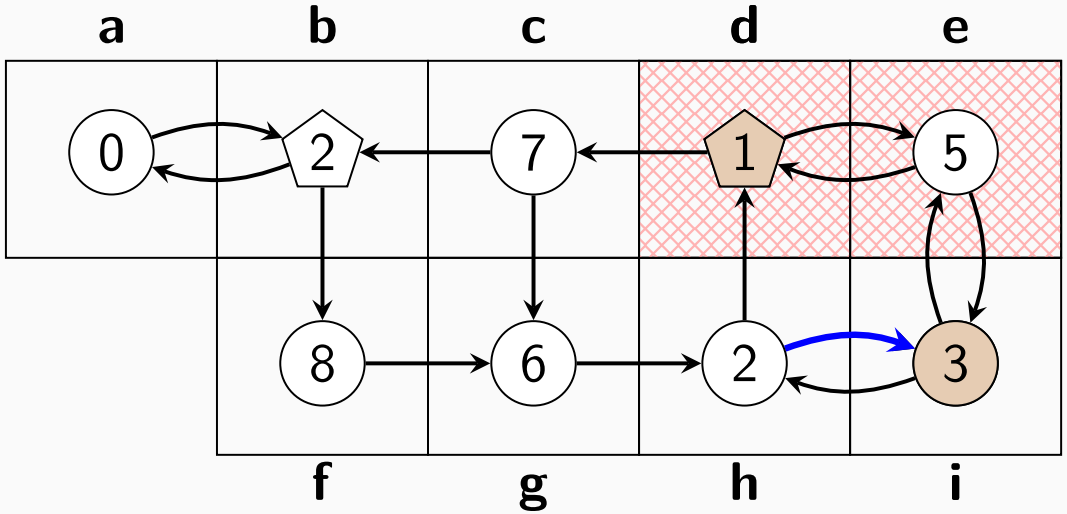
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 1,1,0,1\rangle$	to	$\langle 1,1,0,1\rangle$	1517
e	$\langle 1,1,-,-\rangle$	to	$\top$	51517
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	623232323
h	$\langle 1,0,1,-\rangle$	to	$\langle 1,1,0,-\rangle$	232323232323
i	$\langle 1,1,0,-\rangle$	to	$\langle 1,1,0,-\rangle$	32323232323

SMALL PROGRESS MEASURES



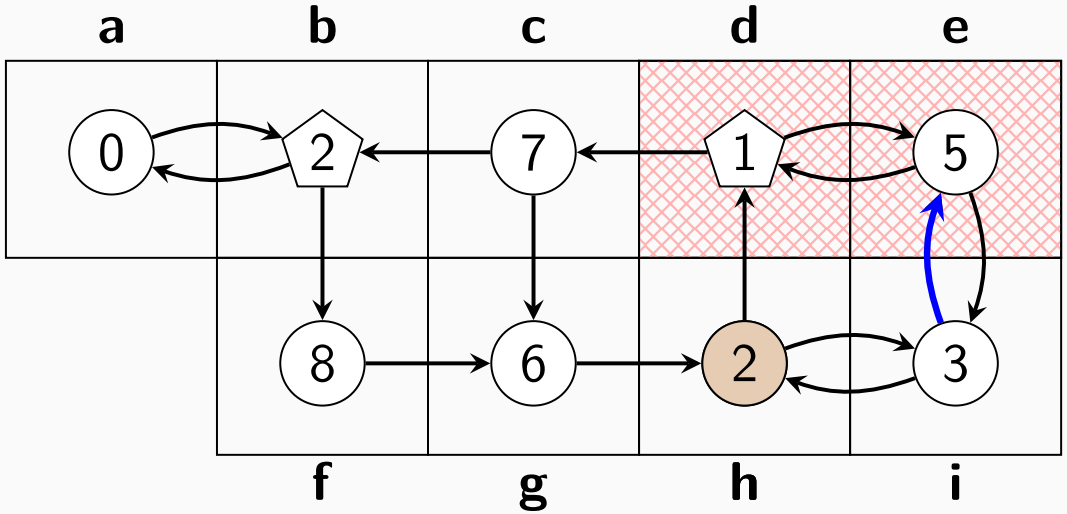
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\langle 1,1,0,1\rangle$	to	$\top$	151517
e	$\top$	to	$\top$	51517
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	623232323
h	$\langle 1,1,0,-\rangle$	to	$\langle 1,1,0,-\rangle$	232323232323
i	$\langle 1,1,0,-\rangle$	to	$\langle 1,1,1,-\rangle$	3232323232323

SMALL PROGRESS MEASURES



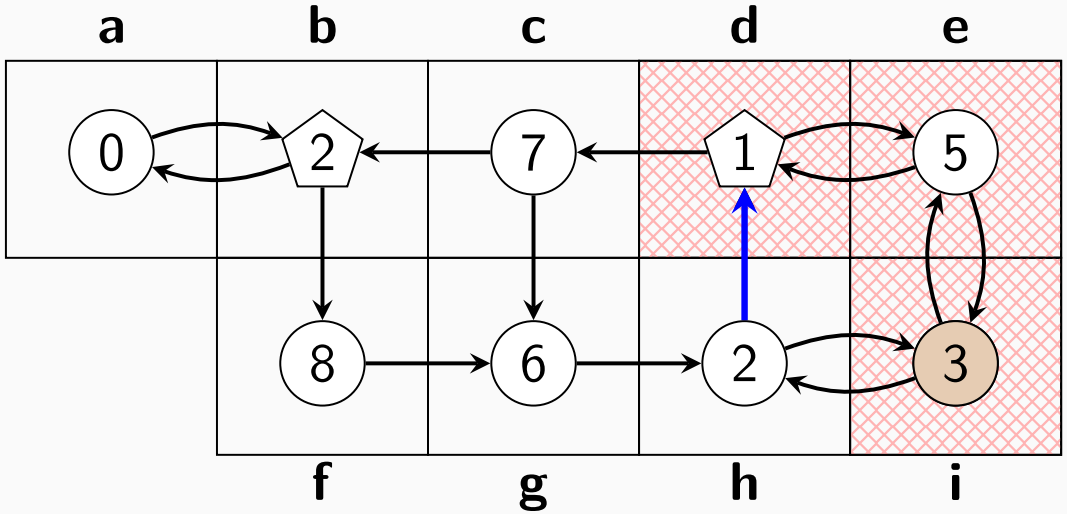
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\top$	to	$\top$	151517
e	$\top$	to	$\top$	51517
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	623232323
h	$\langle 1,0,1,-\rangle$	to	$\langle 1,1,1,-\rangle$	23232323232323
i	$\langle 1,1,1,-\rangle$	to	$\langle 1,1,1,-\rangle$	3232323232323

SMALL PROGRESS MEASURES



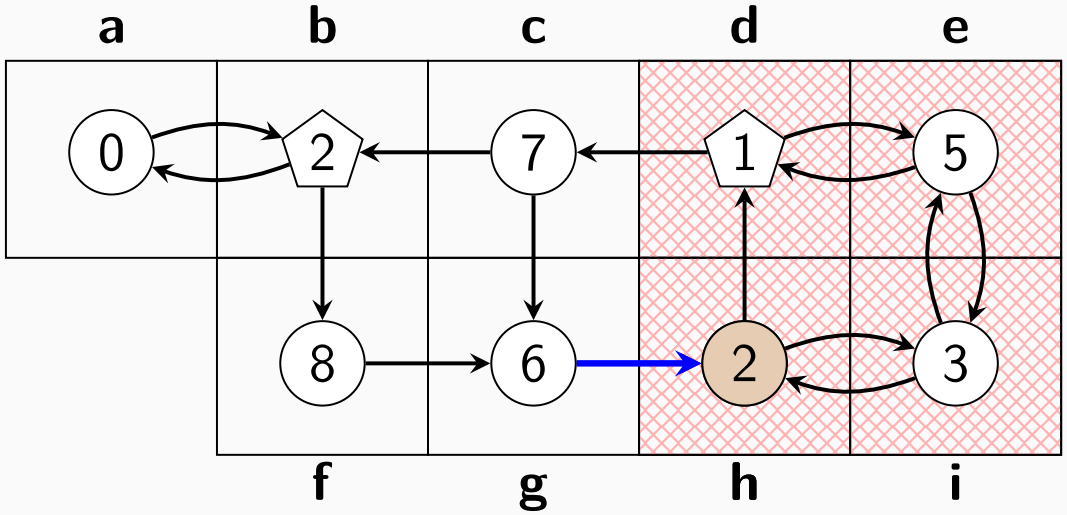
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\top$	to	$\top$	151517
e	$\top$	to	$\top$	51517
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	623232323
h	$\langle 1,1,1,-\rangle$	to	$\langle 1,1,1,-\rangle$	23232323232323
i	$\langle 1,1,1,-\rangle$	to	$\top$	351517

SMALL PROGRESS MEASURES



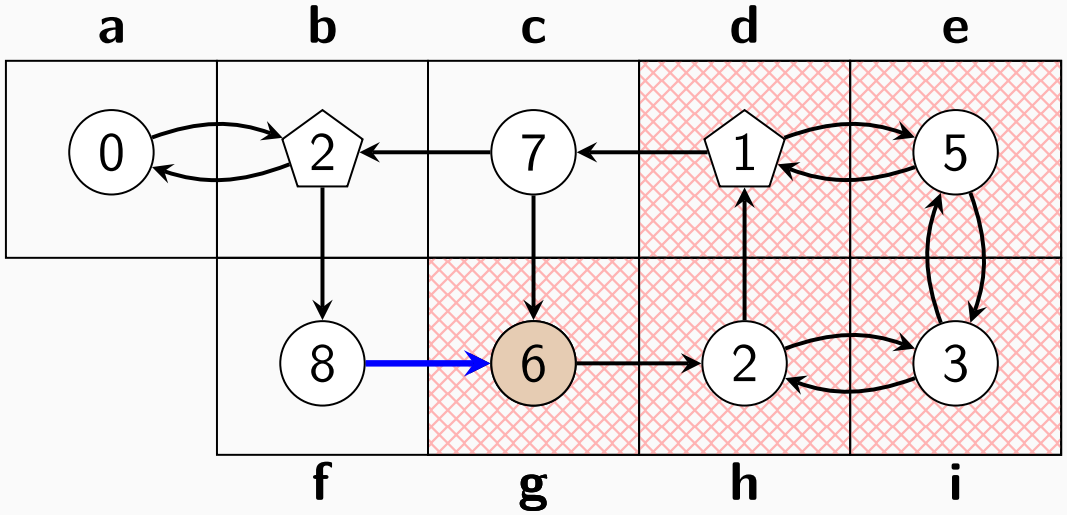
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\top$	to	$\top$	151517
e	$\top$	to	$\top$	51517
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	623232323
h	$\langle 1,1,1,-\rangle$	to	$\top$	2151517
i	$\top$	to	$\top$	351517

SMALL PROGRESS MEASURES



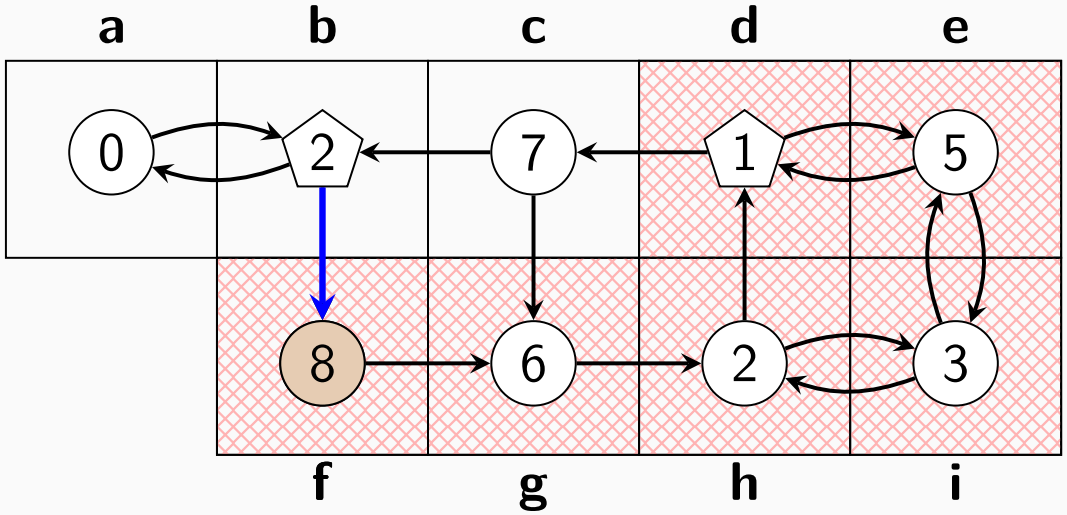
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\top$	to	$\top$	151517
e	$\top$	to	$\top$	51517
f	$\langle -,-,-,-\rangle$	to	$\langle -,-,-,-\rangle$	8
g	$\langle 1,-,-,-\rangle$	to	$\top$	62151517
h	$\top$	to	$\top$	2151517
i	$\top$	to	$\top$	351517

SMALL PROGRESS MEASURES



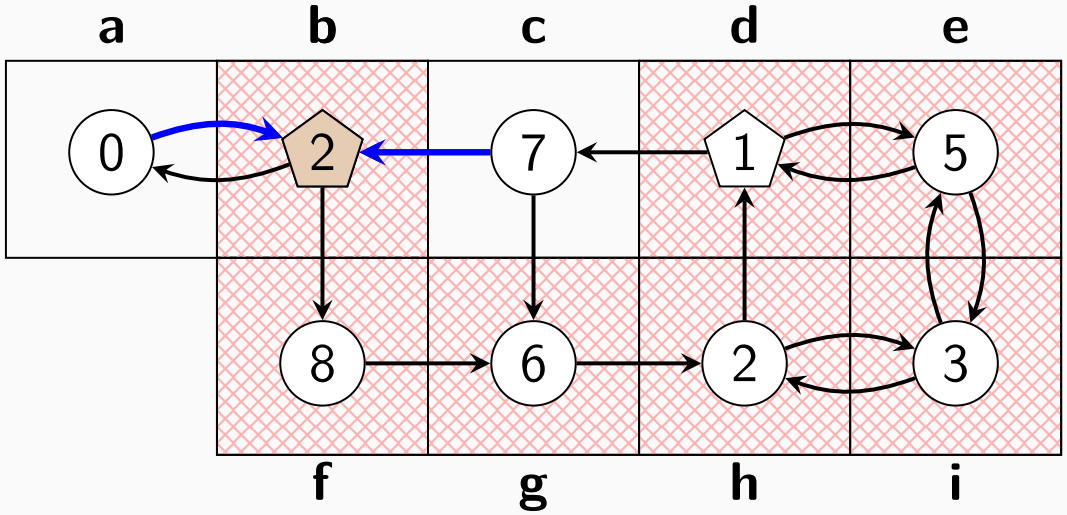
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\langle 0,0,0,-\rangle$	2
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\top$	to	$\top$	151517
e	$\top$	to	$\top$	51517
f	$\langle -,-,-,-\rangle$	to	$\top$	862151517
g	$\top$	to	$\top$	62151517
h	$\top$	to	$\top$	2151517
i	$\top$	to	$\top$	351517

SMALL PROGRESS MEASURES



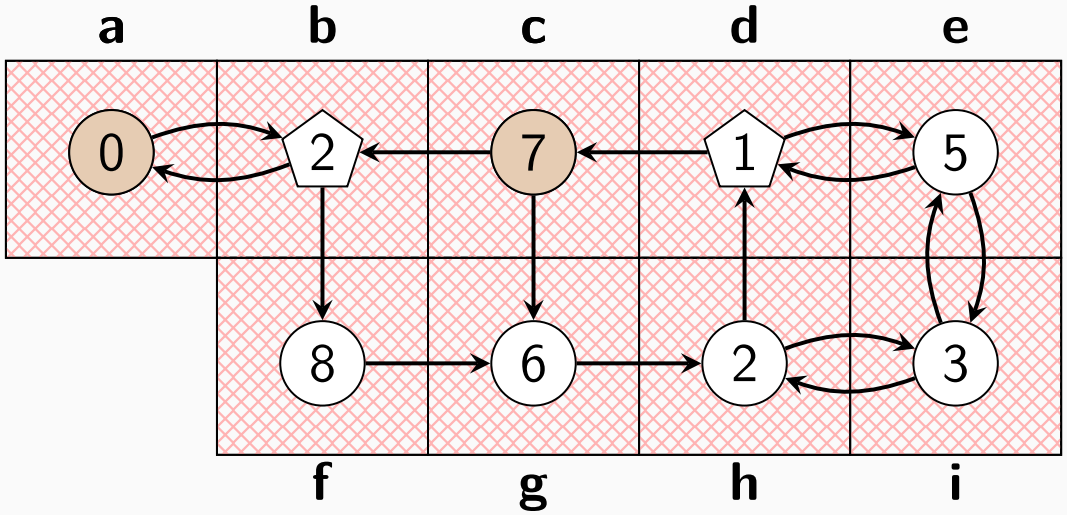
a	$\langle 0,0,0,0\rangle$	to	$\langle 0,0,0,0\rangle$	0
b	$\langle 0,0,0,-\rangle$	to	$\top$	2862151517
c	$\langle 1,-,-,-\rangle$	to	$\langle 1,-,-,-\rangle$	7
d	$\top$	to	$\top$	151517
e	$\top$	to	$\top$	51517
f	$\top$	to	$\top$	862151517
g	$\top$	to	$\top$	62151517
h	$\top$	to	$\top$	2151517
i	$\top$	to	$\top$	351517

SMALL PROGRESS MEASURES



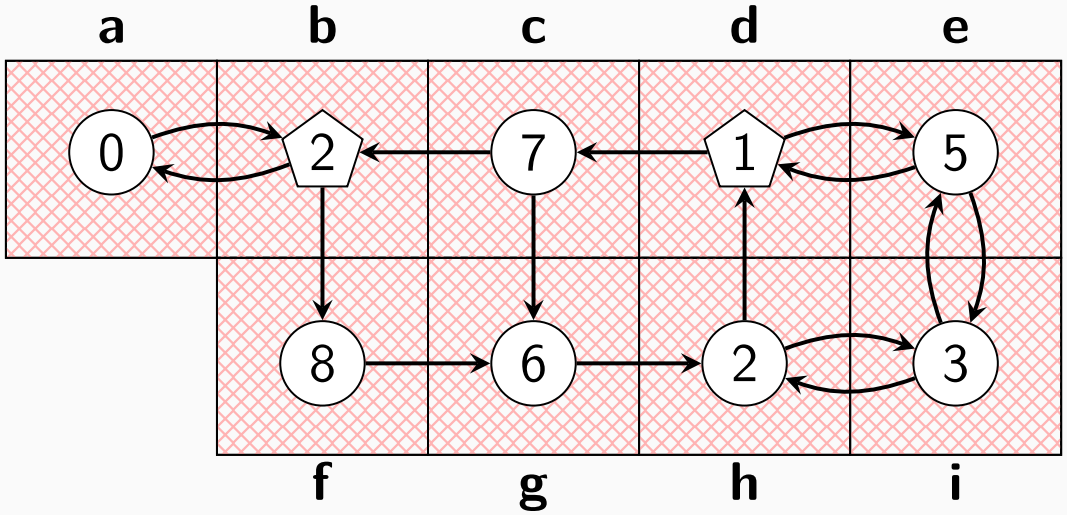
a	$\langle 0,0,0,0\rangle$	to	$\top$	02862151517
b	$\top$	to	$\top$	2862151517
c	$\langle 1,-,-,-\rangle$	to	$\top$	762151517
d	$\top$	to	$\top$	151517
e	$\top$	to	$\top$	51517
f	$\top$	to	$\top$	862151517
g	$\top$	to	$\top$	62151517
h	$\top$	to	$\top$	2151517
i	$\top$	to	$\top$	351517

SMALL PROGRESS MEASURES



a	$\top$	to	$\top$	02862151517
b	$\top$	to	$\top$	2862151517
c	$\top$	to	$\top$	762151517
d	$\top$	to	$\top$	151517
e	$\top$	to	$\top$	51517
f	$\top$	to	$\top$	862151517
g	$\top$	to	$\top$	62151517
h	$\top$	to	$\top$	2151517
i	$\top$	to	$\top$	351517

SMALL PROGRESS MEASURES



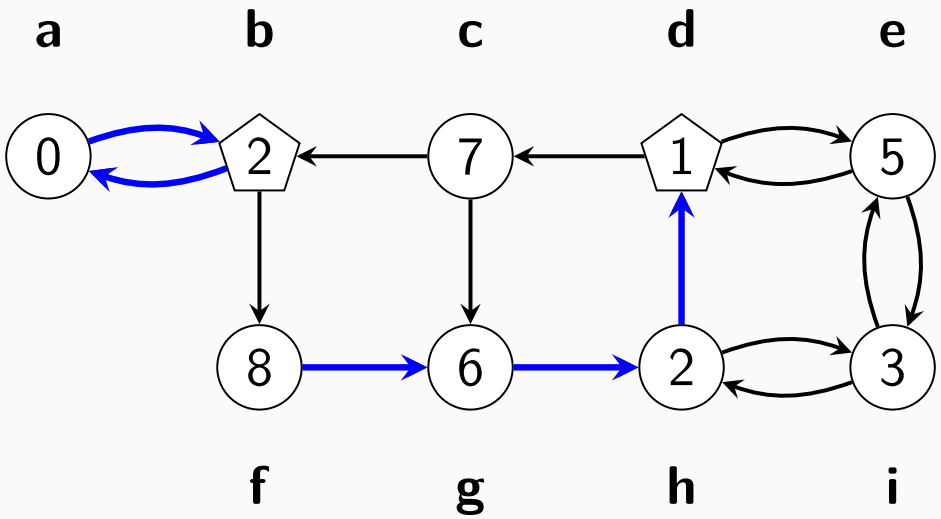
- All vertices are won by Odd
- No strategy for Even

Odd progress measure

To get a strategy for Odd, we need an Odd progress measure

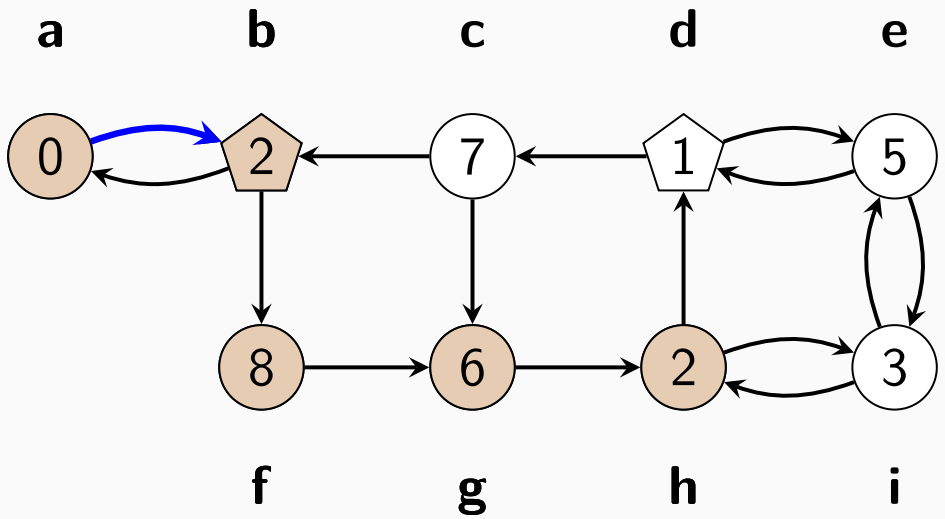
- Now we record how good a vertex is for player Even;
- $\top$ means won by player Even
- A mapping $\mu\colon V \to M$ is an Odd progress measure if:
  - every even-controlled vertex: all edges are μ-progressive
  - every odd-controlled vertex: some edge is μ-progressive
- An edge (v, u) is now μ-progressive if:
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) \geq_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is even and $\mu(v) >_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is even and $\mu(v) = \mu(u) = \top$

SMALL PROGRESS MEASURES



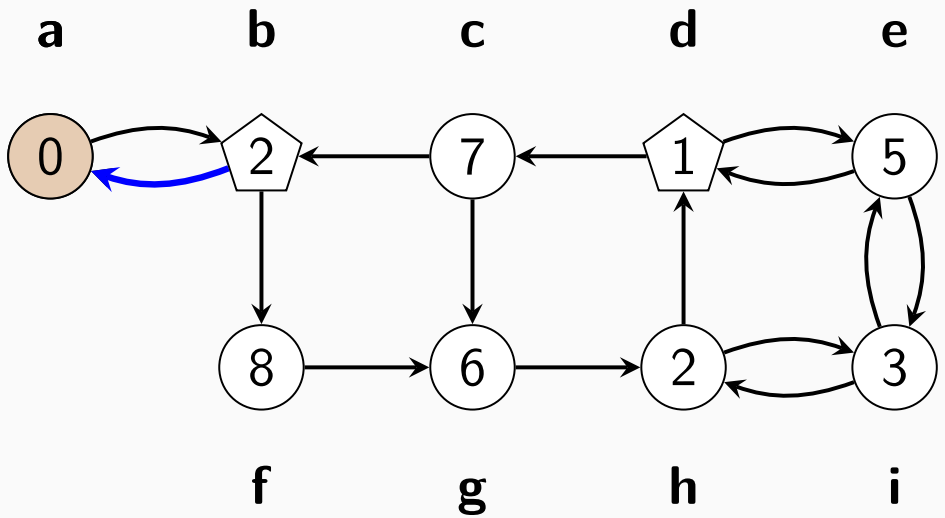
a	$\langle 0,0,0,0,0\rangle$	to	$\langle 0,0,0,0,1\rangle$	0
b	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 0,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



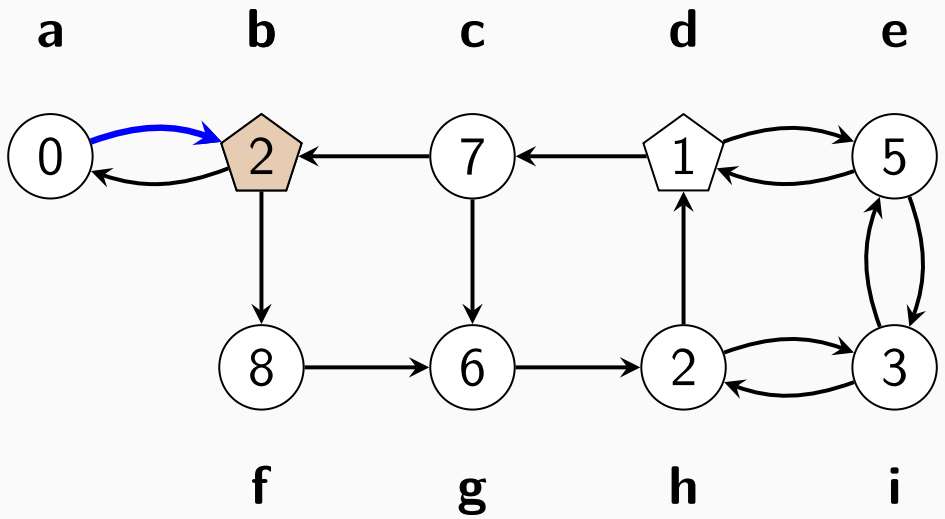
a	$\langle 0,0,0,0,1\rangle$	to	$\langle 0,0,0,1,1\rangle$	02
b	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



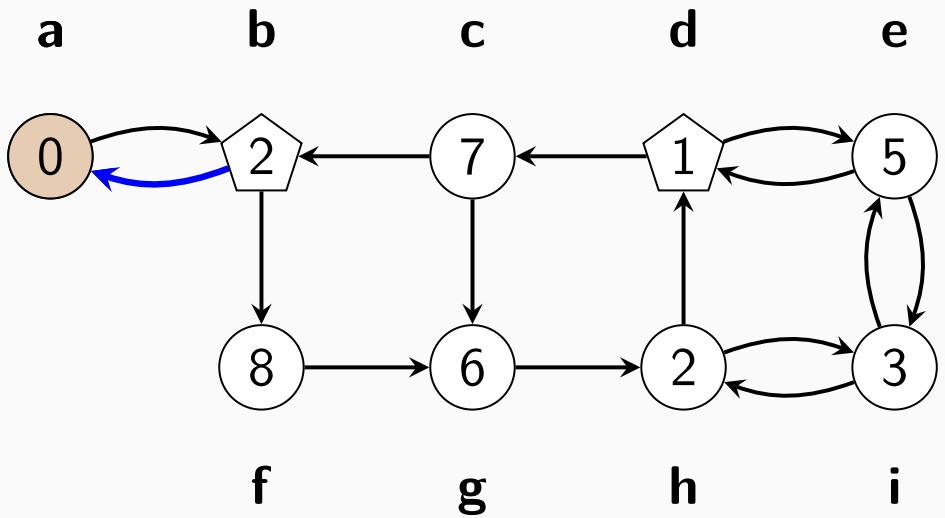
a	$\langle 0,0,0,1,1\rangle$	to	$\langle 0,0,0,1,1\rangle$	02
b	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,2,-\rangle$	202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



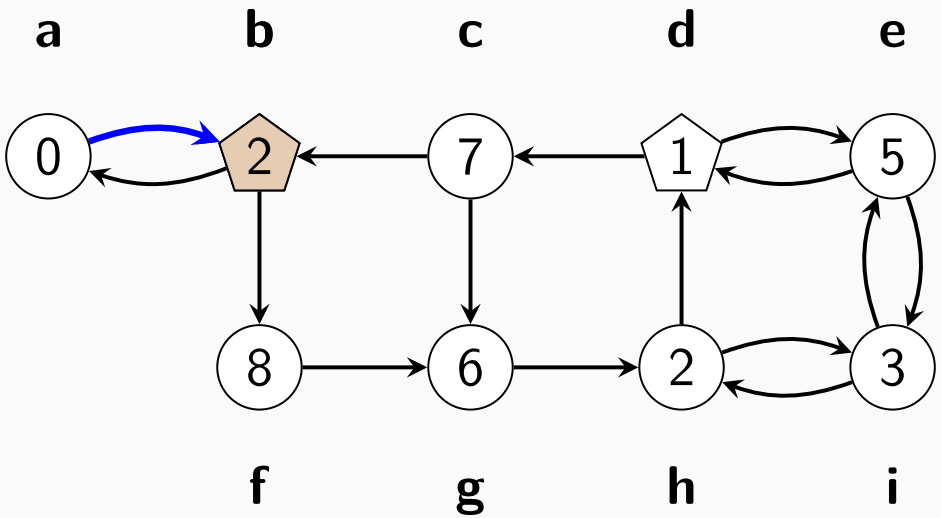
a	$\langle 0,0,0,1,1\rangle$	to	$\langle 0,0,0,2,1\rangle$	0202
b	$\langle 0,0,0,2,-\rangle$	to	$\langle 0,0,0,2,-\rangle$	202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



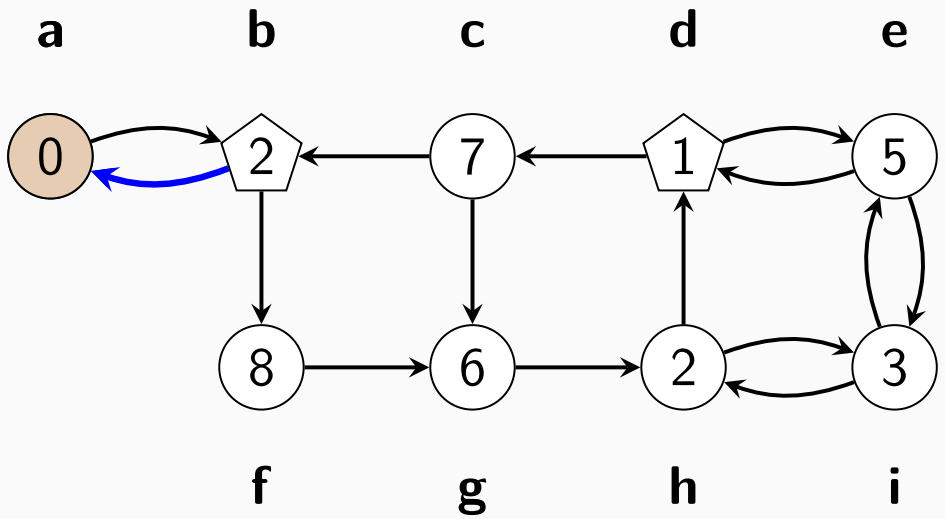
a	$\langle 0,0,0,2,1\rangle$	to	$\langle 0,0,0,2,1\rangle$	0202
b	$\langle 0,0,0,2,-\rangle$	to	$\langle 0,1,0,0,-\rangle$	20202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



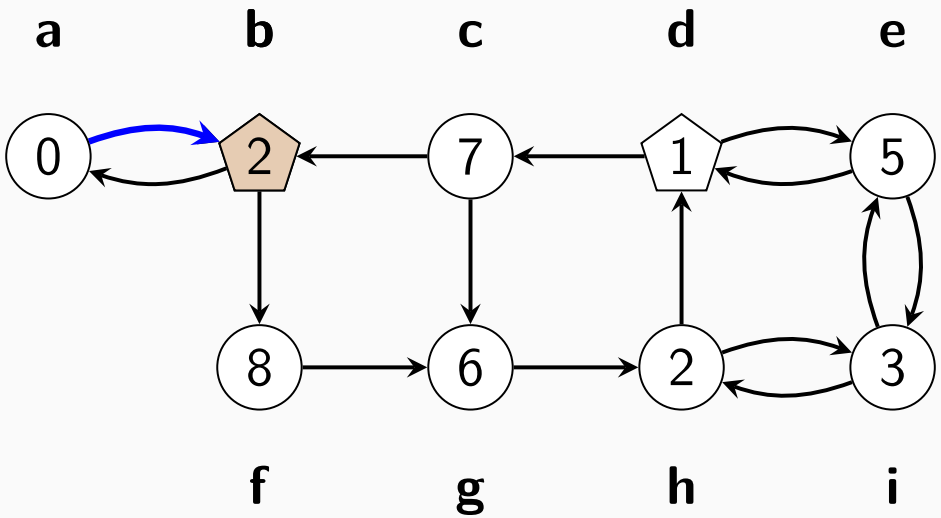
a	$\langle 0,0,0,2,1\rangle$	to	$\langle 0,1,0,0,1\rangle$	020202
b	$\langle 0,1,0,0,-\rangle$	to	$\langle 0,1,0,0,-\rangle$	20202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



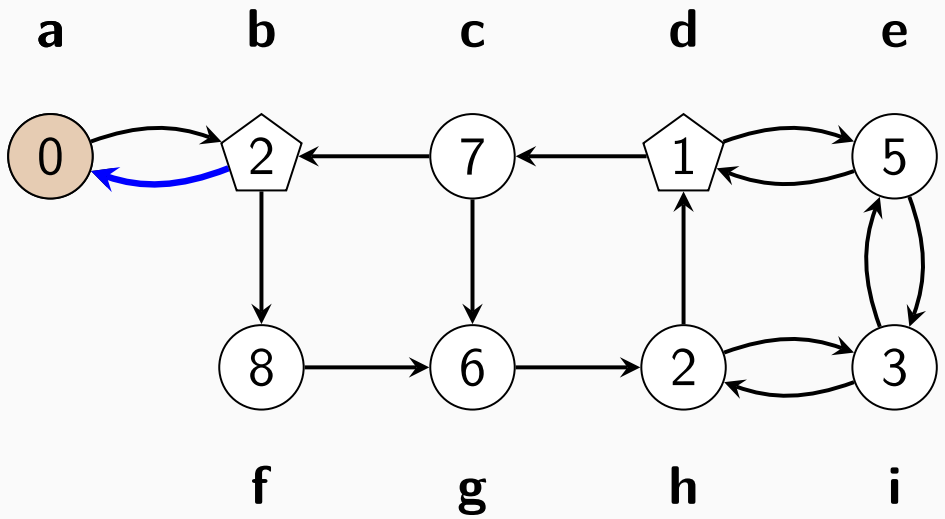
a	$\langle 0,1,0,0,1\rangle$	to	$\langle 0,1,0,0,1\rangle$	020202
b	$\langle 0,1,0,0,-\rangle$	to	$\langle 0,1,0,1,-\rangle$	2020202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



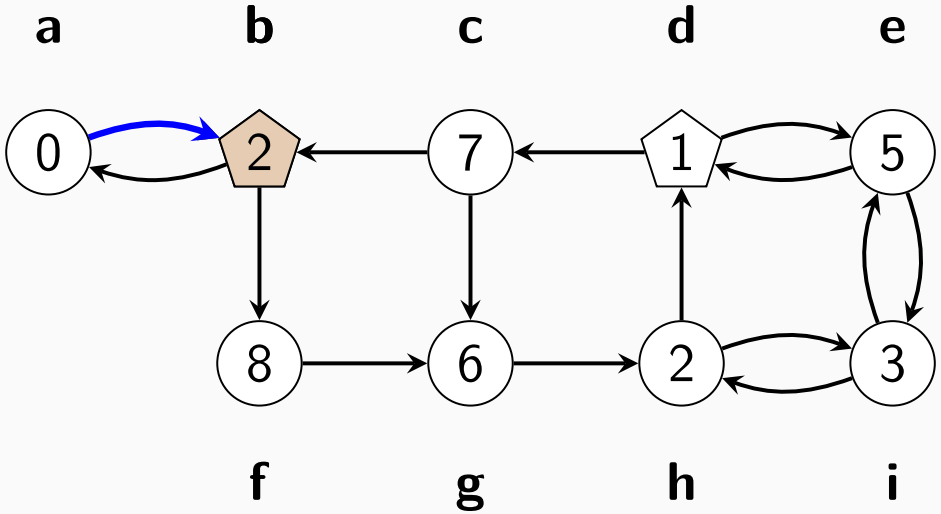
a	$\langle 0,1,0,0,1\rangle$	to	$\langle 0,1,0,1,1\rangle$	02020202
b	$\langle 0,1,0,1,-\rangle$	to	$\langle 0,1,0,1,-\rangle$	2020202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



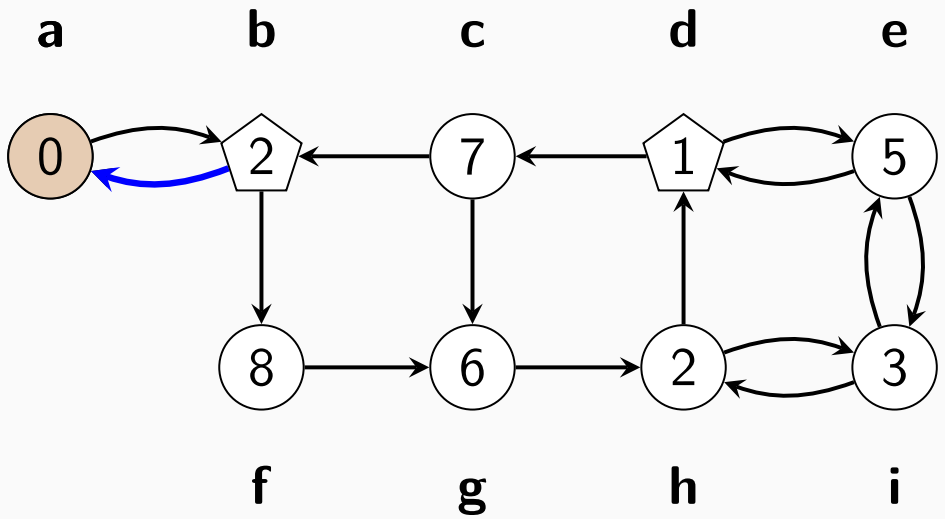
a	$\langle 0,1,0,1,1\rangle$	to	$\langle 0,1,0,1,1\rangle$	02020202
b	$\langle 0,1,0,1,-\rangle$	to	$\langle 0,1,0,2,-\rangle$	202020202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



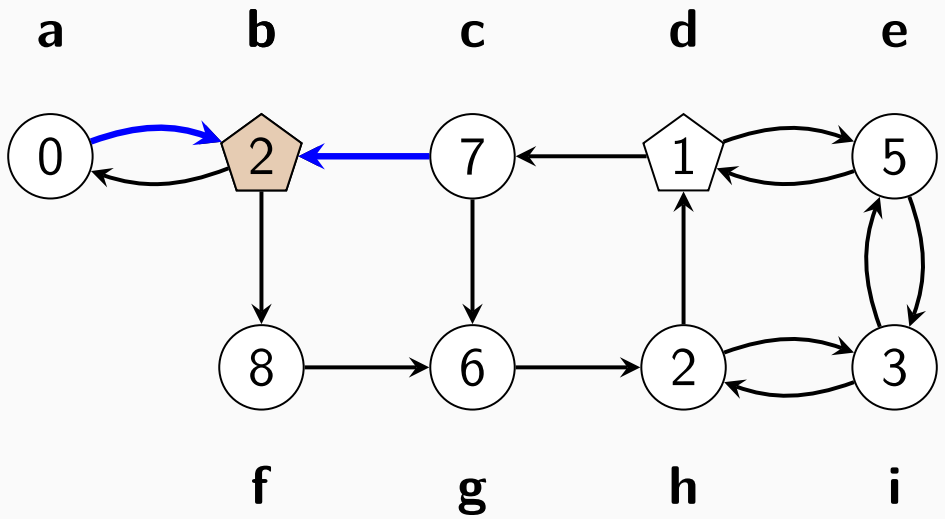
a	$\langle 0,1,0,1,1\rangle$	to	$\langle 0,1,0,2,1\rangle$	0202020202
b	$\langle 0,1,0,2,-\rangle$	to	$\langle 0,1,0,2,-\rangle$	202020202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



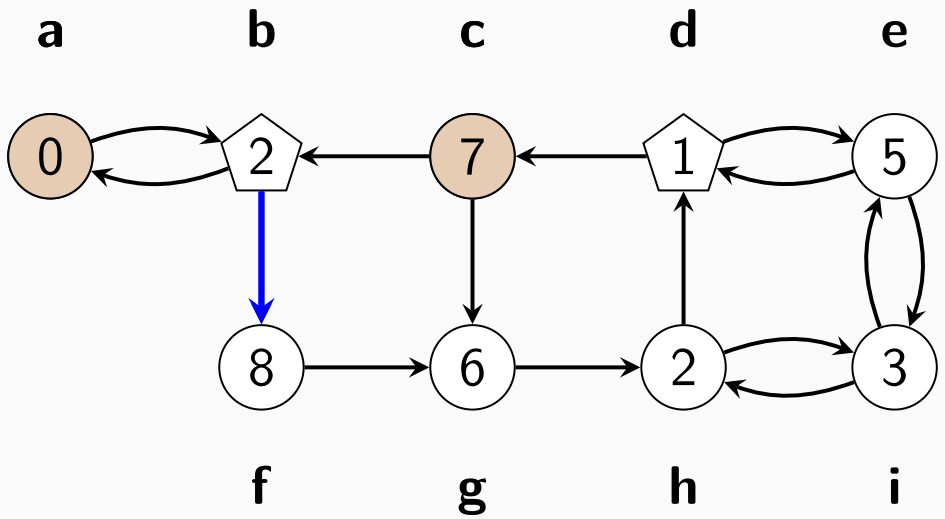
a	$\langle 0,1,0,2,1\rangle$	to	$\langle 0,1,0,2,1\rangle$	0202020202
b	$\langle 0,1,0,2,-\rangle$	to	$\langle 1,0,0,0,-\rangle$	20202020202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



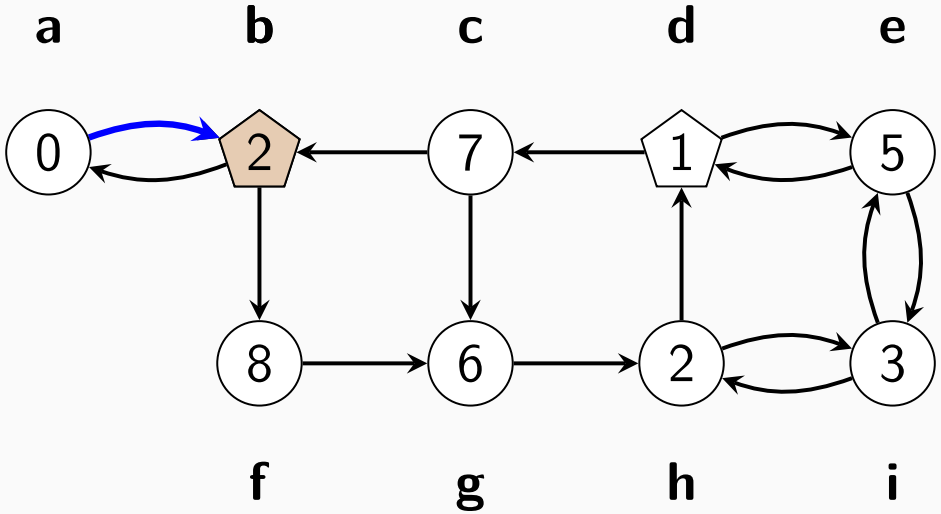
a	$\langle 0,1,0,2,1\rangle$	to	$\langle 1,0,0,0,1\rangle$	02020202020
b	$\langle 1,0,0,0,-\rangle$	to	$\langle 1,0,0,0,-\rangle$	20202020202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	720202020202
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



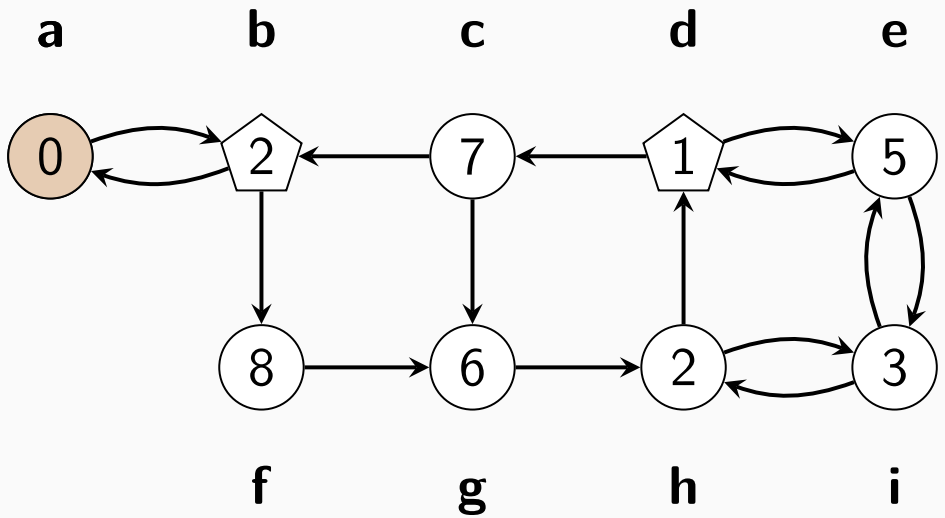
a	$\langle 1,0,0,0,1\rangle$	to	$\langle 1,0,0,0,1\rangle$	020202020202
b	$\langle 1,0,0,0,-\rangle$	to	$\langle 1,0,0,1,-\rangle$	28
c	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	728
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



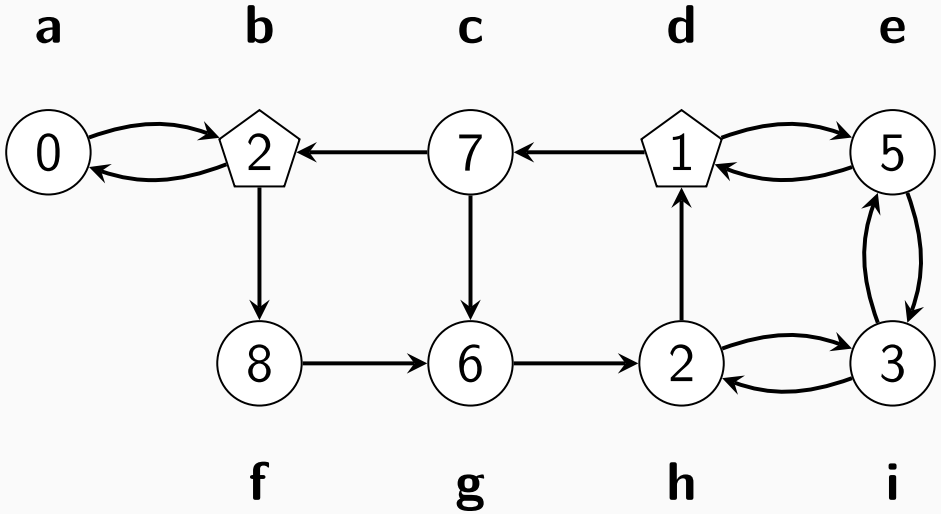
a	$\langle 1,0,0,0,1\rangle$	to	$\langle 1,0,0,1,1\rangle$	028
b	$\langle 1,0,0,1,-\rangle$	to	$\langle 1,0,0,1,-\rangle$	28
c	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	728
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



a	$\langle 1,0,0,1,1\rangle$	to	$\langle 1,0,0,1,1\rangle$	028
b	$\langle 1,0,0,1,-\rangle$	to	$\langle 1,0,0,1,-\rangle$	28
c	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	728
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	5
f	$\langle 1,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	8
g	$\langle 0,1,-,-,-\rangle$	to	$\langle 0,1,-,-,-\rangle$	6
h	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
i	$\langle 0,0,0,-,-\rangle$	to	$\langle 0,0,0,-,-\rangle$	3

SMALL PROGRESS MEASURES



a	$\langle 1, 0, 0, 1, 1 \rangle$
b	$\langle 1, 0, 0, 1, - \rangle$
c	$\langle 1, -, -, -, - \rangle$
d	$\langle 0, 0, 0, 0, - \rangle$
e	$\langle 0, 0, -, -, - \rangle$
f	$\langle 1, -, -, -, - \rangle$
g	$\langle 0, 1, -, -, - \rangle$
h	$\langle 0, 0, 0, 1, - \rangle$
i	$\langle 0, 0, 0, -, - \rangle$

- No vertices are won by Even
- Strategy for Odd
  - from **b** to **f**
    (to **a** not μ-progressive!)
  - from **d** to **e**
    (to **c** not μ-progressive!)

SMALL PROGRESS MEASURES

Operational interpretation [Gazda, Willemse, 2015]

- p-dominated stretches: how often is priority p encountered before a higher priority
- Example: play 0010212023214265 6201 corresponds to $\langle 2, 1, 3, 2 \rangle$
  - priority 0 is seen $2\times$ before a higher priority
  - priority 2 is seen $3\times$ before a higher priority
  - priority 4 is seen $1\times$ before a higher priority
  - priority 6 is seen $2\times$ before a higher priority

Operational interpretation [Van Dijk, 2018]

- Value represents the "statistic of the best continuation"
- *Notice the "overflow mechanism"!* If priority p is seen more than n_p times, there must be a cycle! If priority p overflows, our optimal path contains a cycle of priority p. Keep increasing the measure until the opponent is forced to escape...

SMALL PROGRESS MEASURES

1 **def spm($\mathcal{G}, \alpha$):**
2 $\quad \mu \leftarrow V \mapsto \langle 0, \ldots, 0 \rangle$
3 $\quad Q \leftarrow \{v \in V \mid \mathrm{pr}(v) \bmod 2 = \overline{\alpha}\}$ // use a queue or a stack
4 $\quad$ **while** $Q \neq \emptyset$ **:**
5 $\quad\quad v \leftarrow \mathrm{take}(Q)$ // take a $v \in Q$
6 $\quad\quad$ **if** $\mu(v) < \mathrm{Lift}(\mu, v)$ **:** // can be lifted?
7 $\quad\quad\quad \mu[v] \leftarrow \mathrm{Lift}(\mu, v)$ // update $\mu(v)$
8 $\quad\quad\quad Q \leftarrow Q \cup E^{-1}(v)$ // add predecessors to Q
9 $\quad W_{\overline{\alpha}} \leftarrow \{v \mid \mu(v) = \top\}$
10 $\quad W_\alpha \leftarrow \{v \mid \mu(v) \neq \top\}$
11 $\quad \sigma_\alpha \leftarrow (W_\alpha \cap V_\alpha) \mapsto \mathrm{pick}(\{u \in E(v) \mid \mu(v) = \mathrm{Prog}(\mu(u), \mathrm{pr}(v))\})$
12 $\quad$ **return** $W_\alpha, W_{\overline{\alpha}}, \sigma_\alpha$

$$\mathrm{Lift}(\mu, v) := \begin{cases} \min_{(v,w)\in E} \mathrm{Prog}(\mu(w), \mathrm{pr}(v)) & v \in V_\alpha \\ \max_{(v,w)\in E} \mathrm{Prog}(\mu(w), \mathrm{pr}(v)) & v \in V_{\overline{\alpha}} \end{cases}$$

$$\mathrm{Prog}(m, p) := \begin{cases} \min\{m' \in M \mid m' \geq_p m\} & p \text{ is } \alpha\text{'s priority} \\ \min\{m' \in M \mid m' >_p m\} & p \text{ is } \overline{\alpha}\text{'s priority} \end{cases}$$

SMALL PROGRESS MEASURES

Implementation notes

- use a queue or stack to store "to do" vertices
- after lifting a vertex, add its predecessors to the queue (only once!)
  - a simple bit array can take care of this
- when lifting an $\overline{\alpha}$ **priority vertex** to $\top$, decrease n_p by 1
- the most difficult part is implementing $<$ and `Prog`

OUTLINE

1. Background / Applications of parity games
   - Ω-automata
   - Modal μ-calculus
   - LTL synthesis
2. Parity game solving algorithms
   - Introduction
   - Decomposition based algorithms
   - Local algorithms
3. Quasi-polynomial algorithms
   - Universal trees and progress measures
   - Ordered progress measures
   - Quasi-polynomial recursive algorithm
4. Final remarks

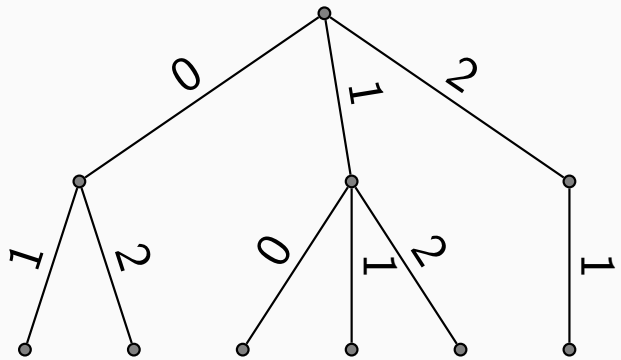
OUTLINE

1. Background / Applications of parity games
   - Ω-automata
   - Modal μ-calculus
   - LTL synthesis
2. Parity game solving algorithms
   - Introduction
   - Decomposition based algorithms
   - Local algorithms
3. Quasi-polynomial algorithms
   - Universal trees and progress measures
   - Ordered progress measures
   - Quasi-polynomial recursive algorithm
4. Final remarks

MEASURES AS TREE NAVIGATION PATHS

Core idea: value space is a tree

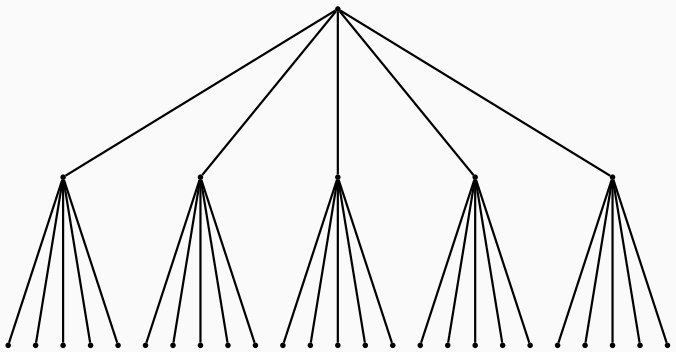
- A tuple $\langle 4, 2, 3 \rangle$ can be a *navigation path* of a tree
- Follow branch 4, then branch 2, then branch 3
- The set of measures form a tree with n leaves and $\lceil d/2 \rceil$ height
- The measures essentially encode the current order between vertices
- Example: (1,2), (0,2), (1,1), (1,2), (2,1), (0,1), (1,0)



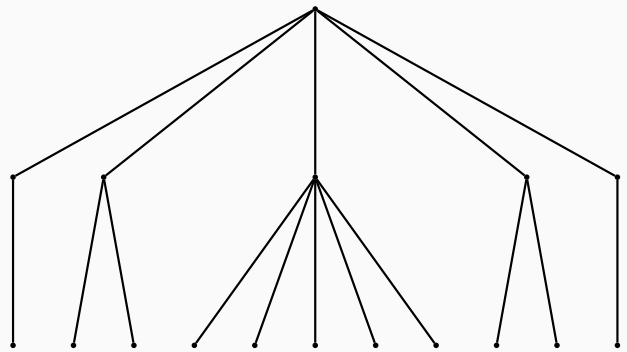
UNIVERSAL TREES

Universal trees (e.g. Fijalkow 2018)

A (n,h)-universal tree can embed all trees of height h and width n.



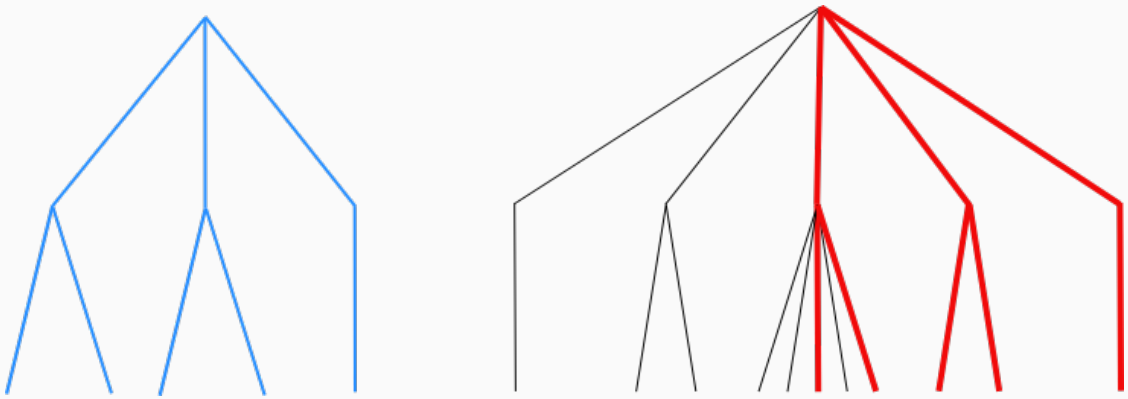
The naive (5,2)-universal tree of size 25



A (5,2)-universal tree of size 11

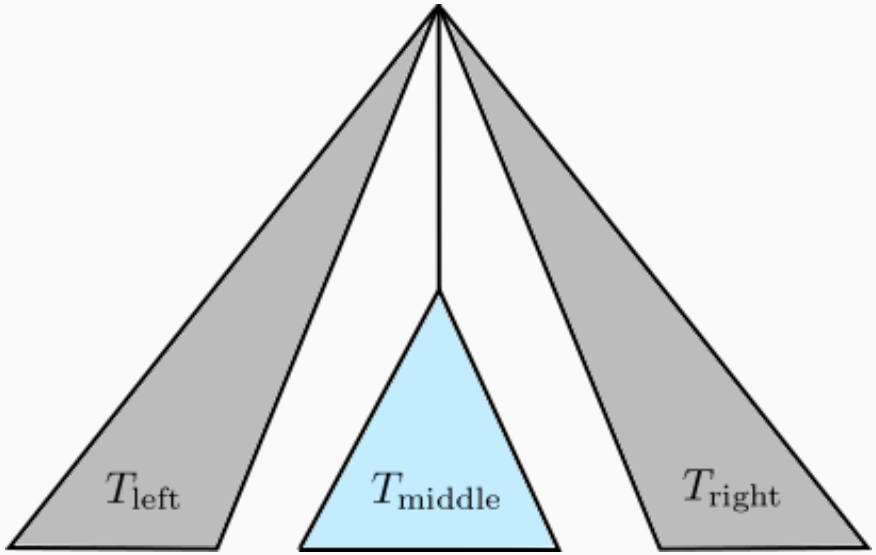
UNIVERSAL TREES

A (n,h)-universal tree can embed all trees with height h and width n.



The tree on the left is embedded into the universal tree

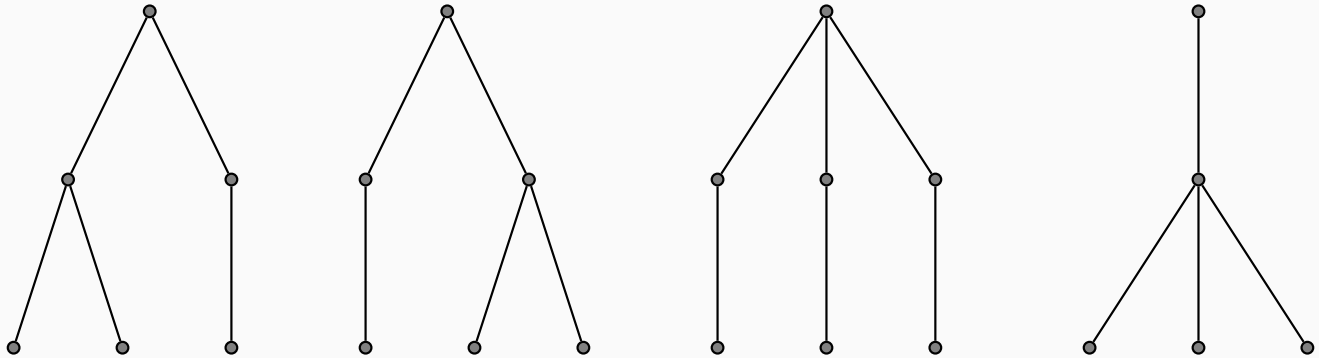
Construction



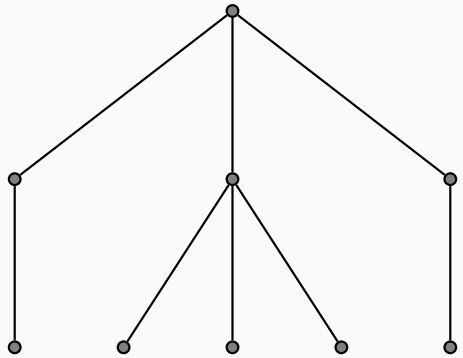
- Split tree in three parts: Left, Middle, Right
- Left: universal tree of height h and width $\lfloor n/2 \rfloor$
- Middle: universal tree of height $h-1$ and width n
- Right: universal tree of height h and width $n-1-\lfloor n/2 \rfloor$

UNIVERSAL TREES

All possible trees of height 2 and width 3:

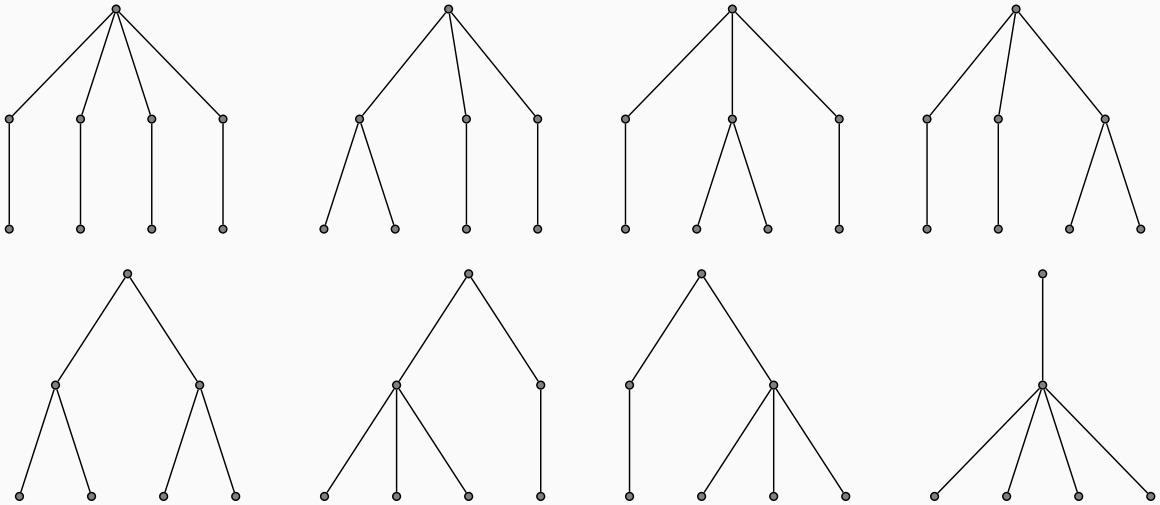


A universal tree of height 2 and width 3:

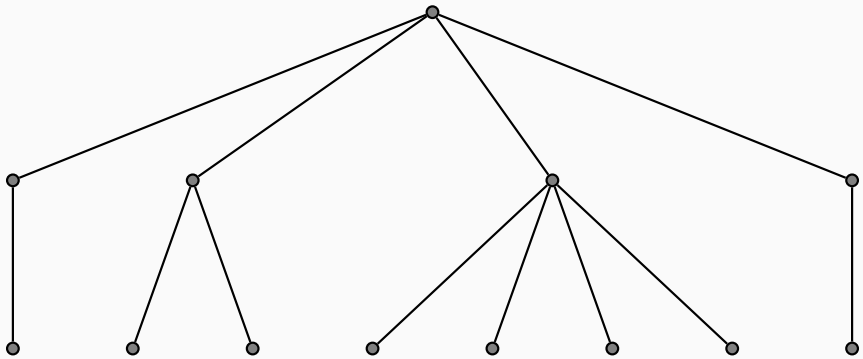


UNIVERSAL TREE

All possible trees of height 2 and width 4:

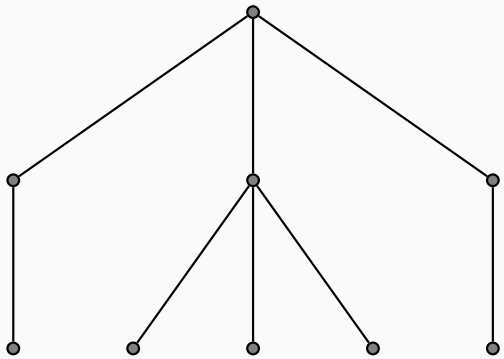


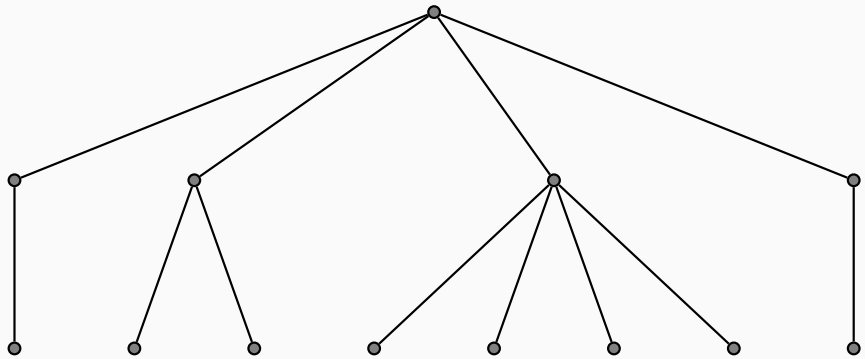
A universal tree of height 2 and width 4:

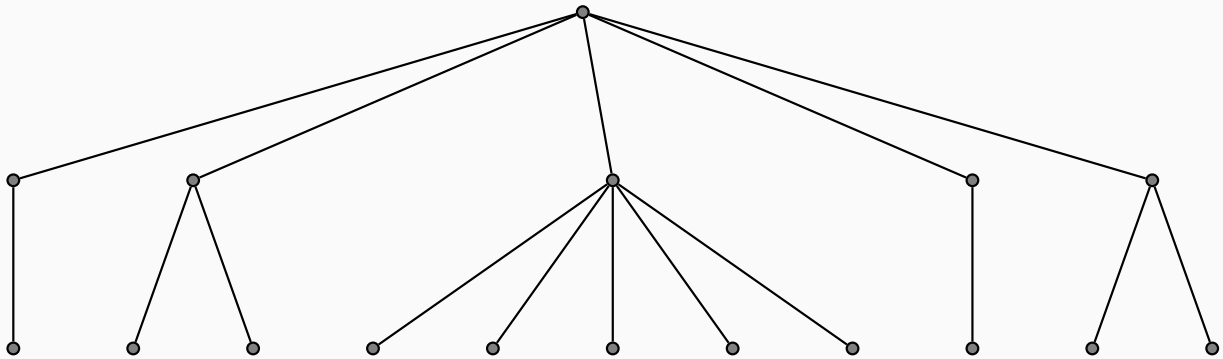


UNIVERSAL TREES

Trees for height 2 and widths 3, 4, 5







Size of universal trees

$$
\begin{array}{lcll}
f(0,h) & = & 0 & \text{for } h \geq 0 \\
f(n,0) & = & 1 & \text{for } n \geq 1 \\
f(n,h) & = & f(n,h-1) + f(\lfloor n/2 \rfloor, h) + f(n-1-\lfloor n/2 \rfloor, h) & \text{for } n,h \geq 1 \\
 & \leq & \mathcal{O}\left(n^{2.45-\varepsilon+\log_2\left(\frac{h}{\log_2(n)}\right)}\right) &
\end{array}
$$

With priorities on vertices, for $d = \omega(\log n)$, one can find a winner in a parity game with n vertices, m edges, and d priorities in time

$$\mathcal{O}\left(m \cdot n^{1.45+\log_2\left(\frac{d}{\log_2(n)}\right)} \cdot \frac{1}{d}\right)$$

(source: arXiv, Parys et al 2023)

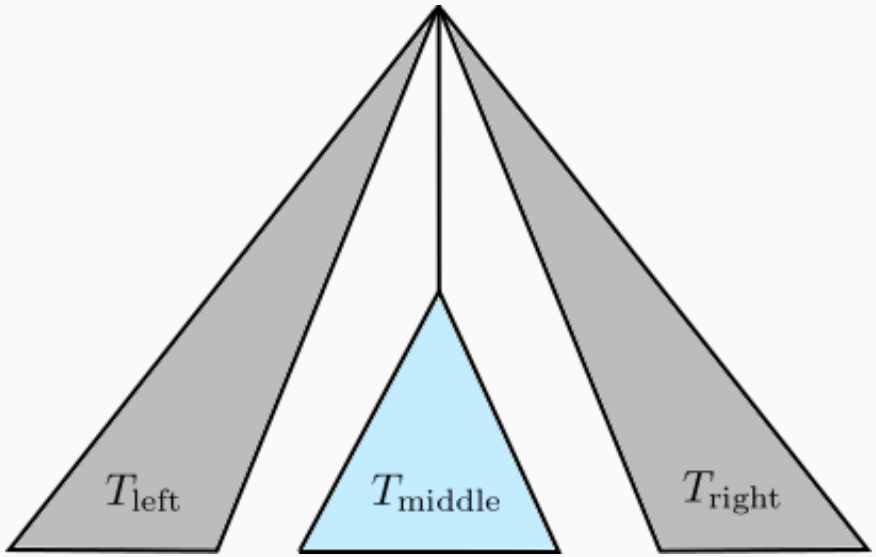
UNIVERSAL TREES

Remarks

- Lower bound of universal tree size is quasi-polynomial
- Many approaches to solving parity games exhibit universal trees
- See Wojciech Czerwinski, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdziński, Ranko Lazić, Pawel Parys: Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games

SUCCINCT TREE ENCODING

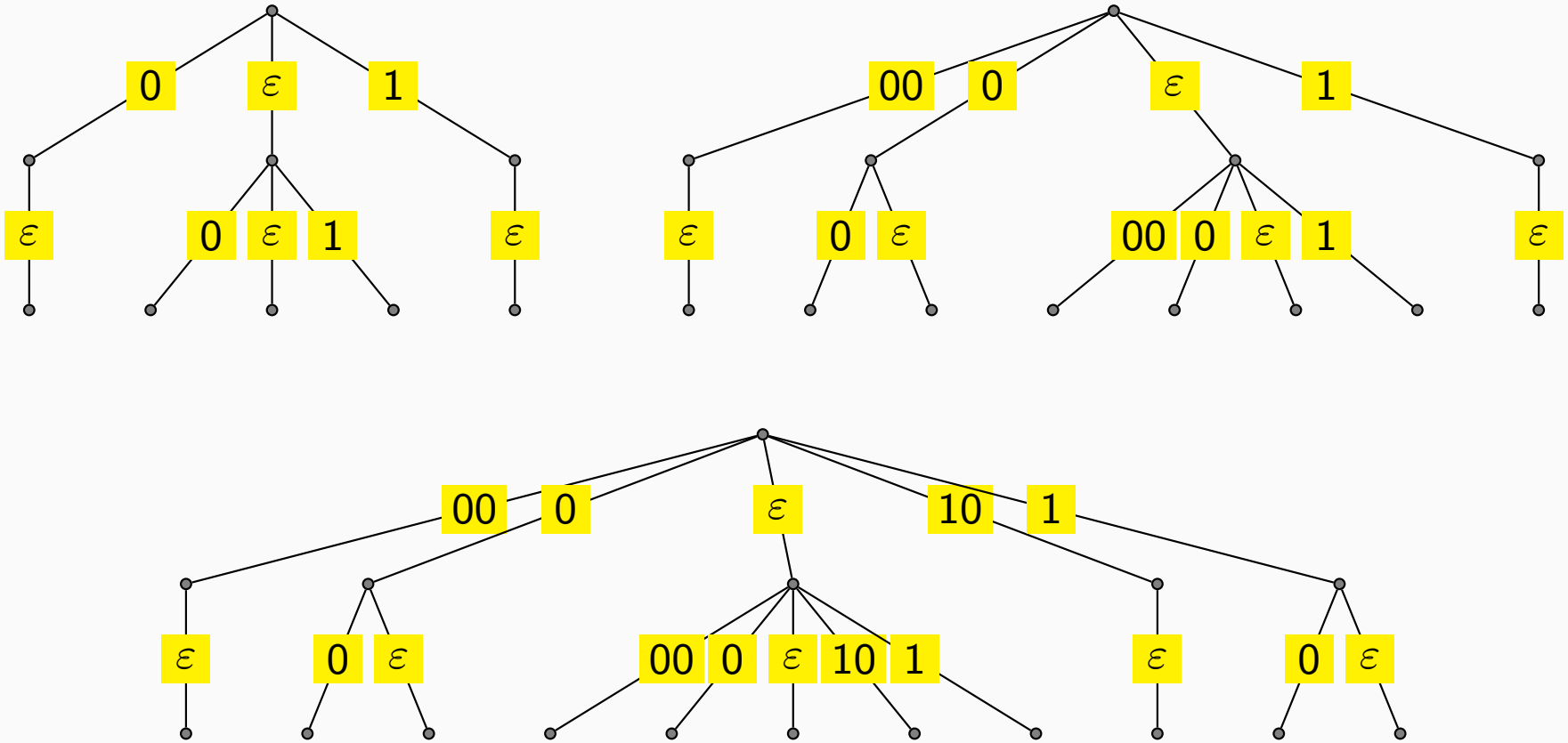
Tree encoding



- Instead of $\langle 4, 2, 3 \rangle$, encode as a tuple of bitstrings
- For example $\langle 100, 010, 011 \rangle$
- Encode Left with 0, Right with 1, Middle with ε.

SUCCINCT TREE ENCODING

Succinct tree encoding for height 2 and widths 3, 4, 5

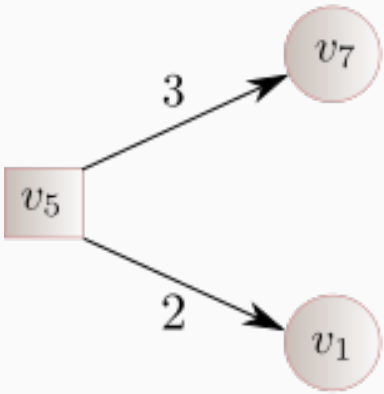


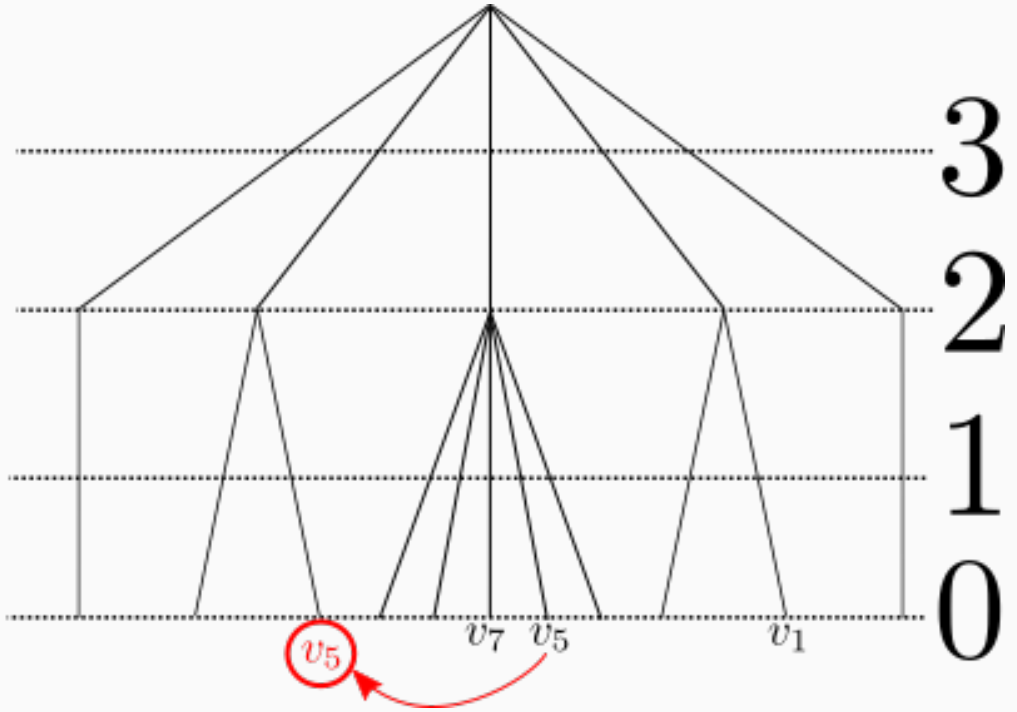
Implementation notes

- Core idea is the same: starting all vertices at the lowest measure, keep lifting vertices to the smallest higher measure, either the maximum (player Even) or the minimum (player Odd)
- Encode bitstrings by concatenating (skip ϵ) and denoting the depth of each bit
  Example: $\langle 00, \varepsilon, 1, 0 \rangle$ is 0010 with depths $0, 0, 2, 3$.

Universal trees

Lifting in the succinct tree encoding
(notice: slightly different notation here)





Example of lifting v_5: it is pushed to the left in order to satisfy $v_5 \lhd_3 v_7$ and $v_5 \lhd_2 v_1$

OUTLINE

1. Background / Applications of parity games
   - Ω-automata
   - Modal μ-calculus
   - LTL synthesis
2. Parity game solving algorithms
   - Introduction
   - Decomposition based algorithms
   - Local algorithms
3. Quasi-polynomial algorithms
   - Universal trees and progress measures
   - Ordered progress measures
   - Quasi-polynomial recursive algorithm
4. Final remarks

"ORDERED" PROGRESS MEASURES

Core idea

- Domain: $_ < 7 < 5 < 3 < 1 < 0 < 2 < 4 < 6$
- Tuples $\langle i_{32}, i_{16}, i_8, i_4, i_2, i_1 \rangle$ encode so-called i-witnesses
- An i_k-witness encodes the existence of a path where Even (or Odd) dominates k times
- Example: $1\underline{2}1\underline{2}1\underline{4}23\underline{2}15\underline{6}3\underline{2}1$
  - Path length 6
  - Tuple $\langle _, _, _, 4, 2, _ \rangle$, a i_2-witness followed by a i_4-witness
- $_$ means "no such witness"
- 7 means a witness, but prefixed by higher odd priority 7
- 6 means a witness, starting with even 6

"ORDERED" PROGRESS MEASURES

Update rules

- successor $\langle 7, _, _, _ \rangle$ and we see a 6: $\langle 7, _, _, 6 \rangle$
- successor $\langle 7, _, _, 6 \rangle$ and we see a 2: $\langle 7, _, 2, _ \rangle$
- successor $\langle 7, _, 2, _ \rangle$ and we see a 1: $\langle 7, _, 2, _ \rangle$
- successor $\langle 7, _, 2, _ \rangle$ and we see a 0: $\langle 7, _, 2, 0 \rangle$
- successor $\langle 7, _, 2, 0 \rangle$ and we see a 6: $\langle 7, 6, _, _ \rangle$
- successor $\langle 7, 6, _, _ \rangle$ and we see an 8: $\langle 8, _, _, _ \rangle$

Problem

- Not quite monotone.
- Solution: "antagonistic update". Given measure m and priority p, compute $\min\{\mathrm{Prog}(m', p) \mid m' \sqsupseteq m\}$

"ORDERED" PROGRESS MEASURES

Complexity

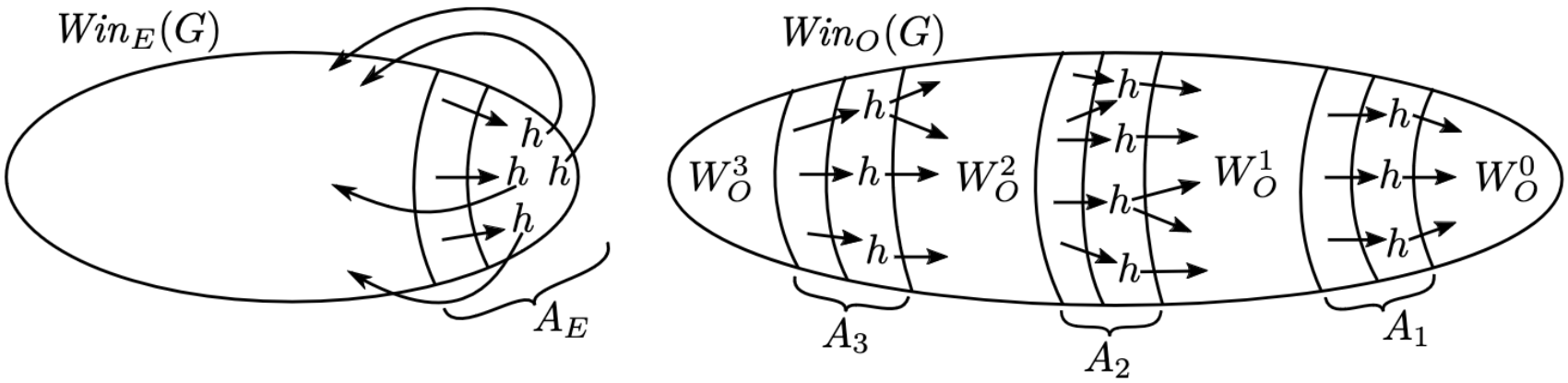
- Each vertex has a tuple of size $\lceil \log_2(n) \rceil$
- Each element of the tuple can have $d + 1$ values
- Therefore: complexity roughly $\mathcal{O}(n \cdot d^{\log_2(n)})$.

In the paper: $\mathcal{O}(m \cdot h \cdot n^{1.45-\varepsilon+\log_2(h)})$ where $h = \lceil 1 + d/\log(n) \rceil$

OUTLINE

1. Background / Applications of parity games
   - Ω-automata
   - Modal μ-calculus
   - LTL synthesis
2. Parity game solving algorithms
   - Introduction
   - Decomposition based algorithms
   - Local algorithms
3. Quasi-polynomial algorithms
   - Universal trees and progress measures
   - Ordered progress measures
   - Quasi-polynomial recursive algorithm
4. Final remarks

QUASI-POLYNOMIAL RECURSIVE ALGORITHM



- See: Parys P.: Parity Games: Zielonka's Algorithm in Quasi-polynomial Time. In: MFCS 2019.
- Based on the structure of the winning regions
- Each winning region consists of smaller "cores" that attract vertices with potentially higher priority. Example:
  - small winning region of priority 1 attracts vertices of priority ≥ 2
  - another odd winning region attracts more vertices of higher Even priority
  - etc.
- Observation 1: these winning regions are disjoint
- Observation 2: only 1 of these has size $> n/2$

QUASI-POLYNOMIAL RECURSIVE ALGORITHM

- Normal Zielonka's recursive algorithm has two recursive calls, both completely solve the subgames $\mathcal{G} \setminus A$ and $\mathcal{G} \setminus B$
- But we only really need a single recursive call with "full precision" and wrap it in at most n calls with "half precision"
  - Find opponent winning regions with size $\leq n/2$ until no more found
  - Find opponent winning regions with size $\leq n$, once
  - Find opponent winning regions with size $\leq n/2$ until no more found
- Algorithm runs in roughly $\mathcal{O}(n^{\log_2(n)})$

QUASI-POLYNOMIAL RECURSIVE ALGORITHM

```
1  def parys_●(G, p, n_●, n_⬠):
2      if G = ∅ ∨ n_● = 0 : return ∅                          // Even wins nothing
3      repeat
4          H ← G\attract(●, G, pr^{-1}(p))                    // compute subgame H
5          W_⬠ ← parys_⬠(H, p − 1, n_●, n_⬠/2)                // half precision
6          G ← G\attract(⬠, G, W_ᾱ)                           // let opponent attract
7      until W_⬠ = ∅
8      H ← G\attract(●, G, pr^{-1}(p))                        // compute subgame H
9      W_⬠ ← parys_⬠(H, p − 1, n_●, n_⬠)                      // full precision
10     G ← G\attract(⬠, G, W_ᾱ)                               // let opponent attract
11     repeat
12         H ← G\attract(●, G, pr^{-1}(p))                    // compute subgame H
13         W_⬠ ← parys_⬠(H, p − 1, n_●, n_⬠/2)                // half precision
14         G ← G\attract(⬠, G, W_ᾱ)                           // let opponent attract
15     until W_⬠ = ∅
16     return V_G                                             // return remaining nodes as won by Even
```

QUASI-POLYNOMIAL RECURSIVE ALGORITHM

Universal trees everywhere!

- Observation: Parys' recursive algorithm's call tree is two interleaved universal trees!
- Improvement: a recursive algorithm parameterised by a universal tree
- Complexity: $n^{\mathcal{O}\left(\log_2\left(1+\frac{d}{\log_2(n)}\right)\right)}$
- See: Lehtinen K., Parys P., Schewe S., Wojtczak D.: A Recursive Approach to Solving Parity Games in Quasipolynomial Time.
- See also: Jurdziński M., Morvan R., Thejaswini K . S.: Universal Algorithms for Parity Games and Nested Fixpoints

QUASI-POLYNOMIAL RECURSIVE ALGORITHM

```
1 def lehtinen●(𝒢, p, n●, n⬠):
2     if 𝒢 = ∅ ∨ n⬠ = 0 : return V_𝒢                          // Even wins all
3     𝒢₁ ←lehtinen●(𝒢, p, n●, ⌊n⬠/2⌋)        // left part of universal tree
4     Z ← {v ∈ 𝒢₁ | pr(v) = p}                // find all nodes of priority p
5     H ← 𝒢₁\attract●(●, 𝒢₁, Z)                  // compute remaining subgame
6     W⬠ ←lehtinen⬠(H, p − 1, n●, n⬠)            // solve subgame recursively
7     𝒢₂ ← 𝒢₁\attract⬠(⬠, 𝒢₁, W⬠)                  // remove won by opponent
8     𝒢₃ ←lehtinen●(𝒢₂, p, n●, ⌊n⬠/2⌋)       // right part of universal tree
9     return 𝒢₃                          // return remaining nodes as won by Even
```

OUTLINE

1. Background / Applications of parity games
   - Ω-automata
   - Modal μ-calculus
   - LTL synthesis
2. Parity game solving algorithms
   - Introduction
   - Decomposition based algorithms
   - Local algorithms
3. Quasi-polynomial algorithms
   - Universal trees and progress measures
   - Ordered progress measures
   - Quasi-polynomial recursive algorithm
4. Final remarks

FINAL REMARKS

- μ-calculus / parity games in P? A 40 year open problem
- Local algorithms were long a target for a PTIME algorithm
- Progress measures approach appears to have QP lower bound
- No clear way to improve upon QP algorithms
- But: they all have limitations
- Tangle learning is more flexible
  - Not limited to a strict order (like recursive algorithms)
  - Can reason about nested cyclic structures directly
- Most tangle learning variations so far have exponential counterexamples