Software Science - Parity Games!

Tom van Dijk
University of Twente, 2019

Acknowledgements

These slides were created in part based on earlier presentations by Tim Willemse, Wolfgang Schreiner and Nathanael Fijalkow.

Overview

- Lecture I
  - Labeled Transition Systems, Kripke Structures
  - The CTL*, CTL and LTL languages
  - The difference between CTL and LTL
    - Different intuition: properties of all runs vs branching structure
    - Incomparable in expressiveness
  - How to express common properties in LTL
  - Fixed points and the modal μ-calculus
  - Naive μ-calculus model checking
  - Translation of μ-calculus to parity games

Overview

- Lecture II
  - Concepts of “attractor computation”, “tangle”, “distraction”
  - Zielonka’s recursive algorithm
  - Priority promotion
  - Tangle learning
- Lecture III
  - Small progress measures algorithm
  - Universal trees and the succinct progress measures algorithm
  - “Ordered” progress measures algorithm
- Lecture IV
  - Strategy iteration
  - Fixed point iteration

Outline

Transition Systems

The behaviour of a system is modelled by a graph consisting of:

- nodes, representing states of the system
  (e.g. the value of a program counter, variables, registers, etc.)
- edges, representing state transitions of the system
  (e.g. events, input/output actions, internal computations)

Information can be put in states or on transitions (or both):

- Kripke Structures (KS)
  Information on states, called atomic propositions
- Labelled Transition Systems (LTS)
  Information on edges, called action labels

Transition Systems

Transition system $\mathcal{M} = \langle S, S_0, \mathcal{A}ct, R, L \rangle$ over set *AP* of atomic propositions:

- S is a set of states
- S_0 is a set of initial states (or s_0 is a single initial state)
- $\mathcal{A}ct$ is a set of action labels
- R is a labelled transition relation: $R \subseteq S \times \mathcal{A}ct \times S$
- L is a labelling: $L \in S \to 2^{AP}$

Notation: $s \xrightarrow{a} t$ denotes $(s, a, t) \in R$

Special cases:

- Kripke Structures: $\mathcal{A}ct$ is a singleton (only one transition relation)
- Labelled Transition Systems: *AP* is empty

Temporal Logics

We want to reason about transition systems, i.e., to specify system properties, behavior, etc.

- Reachability graph: starting from s_0, the system runs evolve
- Consider the reachability graph as an infinite computation tree
  - Different tree nodes may denote occurrences of the same state
  - Every path in this tree is infinite
  - Temporal logic CTL reasons about the computation tree
- Consider the reachability graph as a set of system runs
  - Same state may occur multiple times (in one or in different runs)
  - Temporal logic LTL reasons about each run

Computation Trees versus System Runs

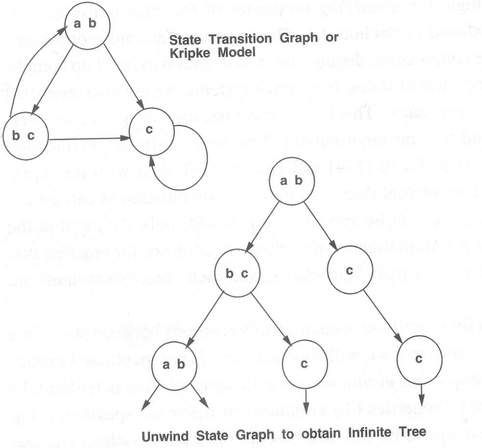


Figure 3.1
Computation trees.

Edmund Clarke et al: "Model Checking", 1999.

Set of system runs:
$[a, b] \to c \to c \to \ldots$
$[a, b] \to [b, c] \to c \to \ldots$
$[a, b] \to [b, c] \to [a, b] \to \ldots$
$[a, b] \to [b, c] \to [a, b] \to \ldots$
$\ldots$

Temporal Logics: CTL*

CTL* is the Full Computation Tree Logic

- CTL* formulae express properties over states or paths
- CTL* has the following temporal operators, which are used to express **properties of paths**: neXt, Future, Globally, Until, Weak Until, Strong Release (M), Release

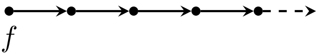
X f	f holds in the next state also: $\bigcirc$
F f	f holds somewhere (eventually) also: $\Diamond$
G f	f holds everywhere also: $\Box$
f **U** g	g holds eventually, and f in all preceding states
f **W** g	$(\mathbf{G}\ f) \vee (f\ \mathbf{U}\ g)$
f **M** g	$g\ \mathbf{U}\ (f \wedge g)$
f **R** g	$(\mathbf{G}\ g) \vee (f\ \mathbf{M}\ g)$

Example

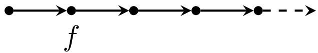
F G p versus **G F** p: *almost always* versus *infinitely often*

Temporal Logics: CTL*

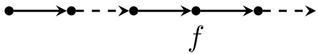
f



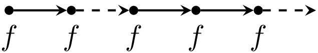
$\mathbf{X}\ f$



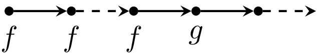
$\mathbf{F}\ f$



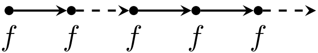
$\mathbf{G}\ f$



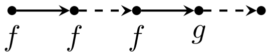
$f\ \mathbf{U}\ g$



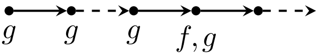
$f\ \mathbf{W}\ g$



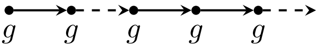
OR



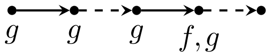
$f\ \mathbf{M}\ g$



$f\ \mathbf{R}\ g$



OR



Temporal Logics: CTL*

CTL* consists of:

- Atomic propositions (AP)
- Boolean connectives: $\neg$ (not), $\vee$ (or), $\wedge$ (and)
- Temporal operators (on paths)
- Path quantifiers (on states)

Path quantifiers are capable of expressing properties on a system's branching structure:

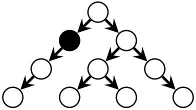
for All paths versus there Exists a path

Path quantifiers have the following intuitive meaning:

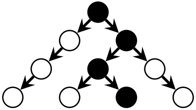
- **A** f: f holds for all paths from this state
- **E** f: f holds for at least one path from this state

Temporal Logics: CTL and LTL

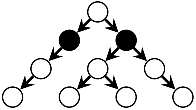
E X black



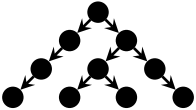
E G black



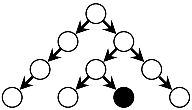
A X black



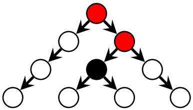
A G black



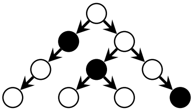
E F black



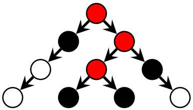
E red **U** black



A F black



A red **U** black



Temporal Logics: CTL*

CTL* state formulae ($\mathcal{S}$) and path formulae ($\mathcal{P}$) are defined simultaneously by induction:

$$
\begin{array}{rcl}
\mathcal{S} & ::= & \text{true} \mid \text{false} \mid AP \mid \neg\mathcal{S} \mid \mathcal{S} \wedge \mathcal{S} \mid \mathcal{S} \vee \mathcal{S} \mid \mathbf{E}\ \mathcal{P} \mid \mathbf{A}\ \mathcal{P} \\
\mathcal{P} & ::= & \mathcal{S} \mid \neg\mathcal{P} \mid \mathcal{P} \wedge \mathcal{P} \mid \mathcal{P} \vee \mathcal{P} \mid \mathbf{X}\ \mathcal{P} \mid \mathbf{F}\ \mathcal{P} \mid \mathbf{G}\ \mathcal{P} \mid \\
 & & \mathcal{P}\ \mathbf{U}\ \mathcal{P} \mid \mathcal{P}\ \mathbf{R}\ \mathcal{P} \mid \mathcal{P}\ \mathbf{W}\ \mathcal{P} \mid \mathcal{P}\ \mathbf{M}\ \mathcal{P}
\end{array}
$$

Summarising:

- State formulae ($\mathcal{S}$) are:
  - constants true and false and atomic propositions (basis)
  - Boolean combinations of state formulae
  - quantified path formulae
- Path formulae ($\mathcal{P}$) are:
  - state formulae (basis)
  - Boolean combinations of path formulae
  - temporal combinations of path formulae

Temporal Logics: CTL*

The semantics of CTL* state formulae and path formulae is defined relative to a fixed Kripke Structure $\mathcal{M} = \langle S, S_0, R, L \rangle$ over *AP*:

For state formulae:

$s \models \text{true}$

$s \not\models \text{false}$

$s \models p$	iff	$p \in L(s)$
$s \models \neg f$	iff	$s \not\models f$
$s \models f \wedge g$	iff	$s \models f$ and $s \models g$
$s \models f \vee g$	iff	$s \models f$ or $s \models g$
$s \models \mathbf{E}\ f$	iff	$\exists \pi \in \text{path}(s)\ .\ \pi \models f$
$s \models \mathbf{A}\ f$	iff	$\forall \pi \in \text{path}(s)\ .\ \pi \models f$

Temporal Logics: CTL*

The semantics of CTL* state formulae and path formulae is defined relative to a fixed Kripke Structure $\mathcal{M} = \langle S, S_0, R, L \rangle$ over *AP*:

For path formulae:

$\pi \models f$	iff	$\pi(0) \models f$	(if f is a state formula)
$\pi \models \neg f$	iff	$\pi \not\models f$	
$\pi \models f \wedge g$	iff	$\pi \models f$ and $\pi \models g$	
$\pi \models f \vee g$	iff	$\pi \models f$ or $\pi \models g$	
$\pi \models \mathbf{X}\ f$	iff	$\pi^1 \models f$	
$\pi \models \mathbf{F}\ f$	iff	$\exists i\ .\ \pi^i \models f$	
$\pi \models \mathbf{G}\ f$	iff	$\forall i\ .\ \pi^i \models f$	
$\pi \models f\ \mathbf{U}\ g$	iff	$\exists i\ .\ \pi^i \models g$ and $\forall j < i\ .\ \pi^j \models f$	
$\pi \models f\ \mathbf{W}\ g$	iff	$\pi \models \mathbf{G}\ f$ or $\pi \models f\ \mathbf{U}\ g$	
$\pi \models f\ \mathbf{M}\ g$	iff	$\exists i\ .\ \pi^i \models g$ and $\forall j \leq i\ .\ \pi^j \models f$	
$\pi \models f\ \mathbf{R}\ g$	iff	$\pi \models \mathbf{G}\ f$ or $\pi \models f\ \mathbf{M}\ g$	

Two simpler sublogics of CTL* are defined

CTL: Computation Tree Logic

$\phi, \psi \quad ::= \quad \text{true} \mid \neg\phi \mid AP \mid \phi \wedge \psi \mid \mathbf{EX}\ \phi \mid \mathbf{EG}\ \phi \mid \mathbf{E}(\phi\ \mathbf{U}\ \psi)$
(derived: false, $\vee$, **EF**, **EW**, **EM**, **ER**, **AX**, **AG**, **AF**, **AU**, **AW**, **AM**, **AR**)

CTL expressions: $\mathbf{AG}\ \mathbf{EF}\ p$, $\mathbf{E}\ p\ \mathbf{U}\ (\mathbf{E}\ \mathbf{X}\ q)$;
syntactically not in CTL: $\mathbf{A}\ \mathbf{F}\ \mathbf{G}\ p$, $\mathbf{A}\ \mathbf{X}\ \mathbf{X}\ p$, $\mathbf{E}(p\ \mathbf{U}\ (\mathbf{X}\ q))$
Question: $\mathbf{A}\ \mathbf{X}\ \mathbf{X}\ p \stackrel{?}{\equiv} \mathbf{AX}\ \mathbf{AX}\ p$

LTL: Linear Time Logic

$\phi, \psi \quad ::= \quad \text{true} \mid \neg\phi \mid AP \mid \phi \wedge \psi \mid \mathbf{X}\ \phi \mid (\phi\ \mathbf{U}\ \psi)$
(derived: false, $\vee$, **F**, **G**, **W**, **M**, **R**)

LTL expressions: $\mathbf{F}\ \mathbf{G}\ p$, $(\neg(\mathbf{G}\ \mathbf{F}\ p) \vee \mathbf{F}\ q)$;
syntactically not in LTL: $\mathbf{A}\ \mathbf{F}\ \mathbf{A}\ \mathbf{G}\ p$, $\mathbf{A}\ \mathbf{G}\ \mathbf{E}\ \mathbf{F}\ p$
Question: $\mathbf{A}\ \mathbf{F}\ \mathbf{G}\ p \stackrel{?}{\equiv} \mathbf{A}\ \mathbf{F}\ \mathbf{A}\ \mathbf{G}\ p$

Branching versus Linear Time Logic

We use temporal logic to specify a formula ϕ.

- Model checking question: $\mathcal{M} \models \phi$ ("ϕ holds in system $\mathcal{M}$").
- Branching time logic (CTL)
  - $\mathcal{M} \models \phi \;\Leftrightarrow\; \forall s_0 \in S_0 \;.\; s_0 \models \phi$
  - ϕ is evaluated on the computation tree of s_0.
- Linear time logic (LTL)
  - $\mathcal{M} \models \phi \;\Leftrightarrow\; \pi \models \phi$ for every run π of $\mathcal{M}$.
  - ϕ is evaluated on all paths of the computation tree originating in s_0.

Branching versus Linear Time Logic

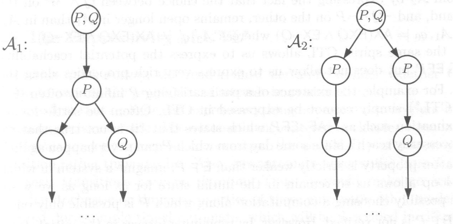


Fig. 2.4. Two automata, indistinguishable for PLTL

B. Berard et al: "Systems and Software Verification", 2001.

- Linear time logic: both systems have the same runs.
  - Thus every formula has same truth value in both systems.
- Branching time logic: the systems have different computation trees.
  - Take formula $\mathbf{AX}(\mathbf{EX}\ Q \wedge \mathbf{EX}\ \neg Q)$.
  - True for left system, false for right system.

The two variants of temporal logic have different expressive power.

Branching versus Linear Time Logic

Is one temporal logic variant more expressive than the other one?

- CTL formula: $\mathbf{AG}(\mathbf{EF}\ \phi)$.
  - "In every run, it is at any time still possible that later ϕ will hold".
  - Property cannot be expressed by any LTL logic formula.
- LTL formula: $\Diamond\Box\phi$ (i.e. $\mathbf{FG}\ \phi$).
  - "In every run, there is a moment from which on ϕ holds forever.".
  - Naive translation $\mathbf{AFG}\ \phi$ is not a CTL formula.
    - $\mathbf{G}\ \phi$ is a path formula, but $\mathbf{F}$ expects a state formula!
  - Translation $\mathbf{AFAG}\ \phi$ expresses a stronger property (see next page).
  - Property cannot be expressed by any CTL formula.

None of the two variants is strictly more expressive than the other one; no variant can express every system property.

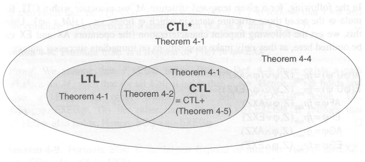
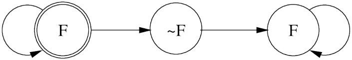
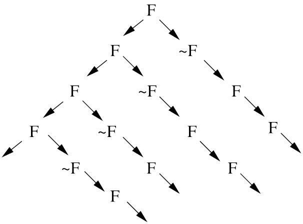


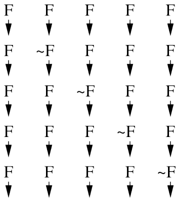
Fig. 4-8. Expressiveness of CTL*, CTL+, CTL and LTL

Thomas Kropf: "Introduction to Formal Hardware Verification", 1999.

Proof that $\mathbf{AFAG}\ F$ (CTL) is different from $\Diamond\Box F$ (LTL).







AFAG F <=> false

<>[] F <=> true

In every run, there is a moment when it is guarantueed that from now on F holds forever.

In every run, there is a moment from which on F holds forever.

Linear Time Logic

Why using linear time logic (LTL) for system specifications?

- LTL has many advantages:
  - LTL formulas are easier to understand.
    - Reasoning about computation paths, not computation trees.
    - No explicit path quantifiers used.
  - LTL can express most interesting system properties.
    - Invariance, guarantee, response, ... (see later).
  - LTL can express fairness constraints (see later).
    - CTL cannot do this.
    - But CTL can express resettability (which LTL cannot).
- LTL has also some disadvantages:
  - LTL is strictly less expressive than other specification languages.
    - CTL^* or μ-calculus.
  - Asymptotic complexity of model checking is higher.
    - LTL: exponential in size of formula; CTL: linear in size of formula.
    - In practice the number of system states dominates the checking time.

Frequently Used LTL Patterns

In practice, most temporal formulas are instances of particular patterns.

Pattern	Pronounced	Name
$\mathbf{G}\ \phi$	always ϕ	invariance
$\mathbf{F}\ \phi$	eventually ϕ	guarantee
$\mathbf{G}\ \mathbf{F}\ \phi$	ϕ holds infinitely often	recurrence
$\mathbf{F}\ \mathbf{G}\ \phi$	eventually ϕ holds permanently	stability
$\mathbf{G}\ (\phi \Rightarrow \mathbf{F}\ \psi)$	always, if ϕ holds, then eventually ψ holds	response
$\mathbf{G}\ (\phi \Rightarrow (\psi\ \mathbf{U}\ \chi))$	always, if ϕ holds, then ψ holds until χ holds	precedence

Typically, there are at most two levels of nesting of temporal operators.

Examples

- Mutual exclusion: **G** $\neg(pc_1 = C \wedge pc_2 = C)$.
  - Alternatively: $\neg$**F** $(pc_1 = C \wedge pc_2 = C)$.
  - Never both components are simultaneously in the critical region.
- No starvation: $\forall i :$ **G** $(pc_i = W \Rightarrow \Diamond pc_i = R)$.
  - Always, if component i waits for a response, it eventually receives it.
- No deadlock: **G** $\neg\forall i : pc_i = W$.
  - Never all components are simultaneously in a wait state W.
- Precedence: $\forall i :$ **G** $(pc_i \neq C \Rightarrow (pc_i \neq C \ \mathbf{U} \ lock = i))$.
  - Always, if component i is out of the critical region, it stays out until it receives the shared lock variable (which it eventually does).
- Partial correctness: **G** $(pc = L \Rightarrow C)$.
  - Always if the program reaches line L, the condition C holds.
- Termination: $\forall i :$ **F** $(pc_i = T)$.
  - Every component eventually terminates.

Example

If event a occurs, then b must occur before c can occur (a run $\ldots, a, (\neg b)^*, c, \ldots$ is illegal).

Example

If event a occurs, then b must occur before c can occur (a run $\ldots, a, (\neg b)^*, c, \ldots$ is illegal).

- First idea (wrong): $a \Rightarrow \ldots$
  - Every run $d, \ldots$ becomes legal.

Example

If event a occurs, then b must occur before c can occur (a run $\ldots, a, (\neg b)^*, c, \ldots$ is illegal).

- First idea (wrong): $a \Rightarrow \ldots$
  - Every run $d, \ldots$ becomes legal.
- Next idea (correct): **G** $(a \Rightarrow \ldots)$

Example

If event a occurs, then b must occur before c can occur (a run $\ldots, a, (\neg b)^*, c, \ldots$ is illegal).

- First idea (wrong): $a \Rightarrow \ldots$
  - Every run $d, \ldots$ becomes legal.
- Next idea (correct): **G** $(a \Rightarrow \ldots)$
- First attempt (wrong): **G** $(a \Rightarrow (b$ **U** $c))$
  - Run $a, b, \neg b, c, \ldots$ is illegal.

Example

If event a occurs, then b must occur before c can occur (a run $\ldots, a, (\neg b)^*, c, \ldots$ is illegal).

- First idea (wrong): $a \Rightarrow \ldots$
  - Every run $d, \ldots$ becomes legal.
- Next idea (correct): $\mathbf{G}\ (a \Rightarrow \ldots)$
- First attempt (wrong): $\mathbf{G}\ (a \Rightarrow (b\ \mathbf{U}\ c))$
  - Run $a, b, \neg b, c, \ldots$ is illegal.
- Second attempt (better): $\mathbf{G}\ (a \Rightarrow (\neg c\ \mathbf{U}\ b))$
  - Run $a, \neg c, \neg c, \neg c, \ldots$ is illegal.

Example

If event a occurs, then b must occur before c can occur (a run $\ldots, a, (\neg b)^*, c, \ldots$ is illegal).

- First idea (wrong): $a \Rightarrow \ldots$
  - Every run $d, \ldots$ becomes legal.
- Next idea (correct): $\mathbf{G}\ (a \Rightarrow \ldots)$
- First attempt (wrong): $\mathbf{G}\ (a \Rightarrow (b\ \mathbf{U}\ c))$
  - Run $a, b, \neg b, c, \ldots$ is illegal.
- Second attempt (better): $\mathbf{G}\ (a \Rightarrow (\neg c\ \mathbf{U}\ b))$
  - Run $a, \neg c, \neg c, \neg c, \ldots$ is illegal.
- Third attempt (correct): $\mathbf{G}\ (a \Rightarrow (\neg c\ \mathbf{W}\ b))$

Example

If event a occurs, then b must occur before c can occur (a run $\ldots, a, (\neg b)^*, c, \ldots$ is illegal).

- First idea (wrong): $a \Rightarrow \ldots$
  - Every run $d, \ldots$ becomes legal.
- Next idea (correct): **G** $(a \Rightarrow \ldots)$
- First attempt (wrong): **G** $(a \Rightarrow (b$ **U** $c))$
  - Run $a, b, \neg b, c, \ldots$ is illegal.
- Second attempt (better): **G** $(a \Rightarrow (\neg c$ **U** $b))$
  - Run $a, \neg c, \neg c, \neg c, \ldots$ is illegal.
- Third attempt (correct): **G** $(a \Rightarrow (\neg c$ **W** $b))$

Think in terms of allowed/prohibited sequences.

LTL Expansion Laws

Basic LTL expansion laws (e.g. for unfolding)

$$
\begin{array}{lcl}
\mathbf{F}\,\phi & \equiv & \phi \lor \mathbf{X}\,(\mathbf{F}\,\phi) \\
\mathbf{G}\,\phi & \equiv & \phi \land \mathbf{X}\,(\mathbf{G}\,\phi) \\
\phi\,\mathbf{U}\,\psi & \equiv & \psi \lor (\phi \land \mathbf{X}\,(\phi\,\mathbf{U}\,\psi)) \\
\phi\,\mathbf{W}\,\psi & \equiv & \psi \lor (\phi \land \mathbf{X}\,(\phi\,\mathbf{W}\,\psi)) \\
\phi\,\mathbf{M}\,\psi & \equiv & \psi \land (\phi \lor \mathbf{X}\,(\phi\,\mathbf{M}\,\psi)) \\
\phi\,\mathbf{R}\,\psi & \equiv & \psi \land (\phi \lor \mathbf{X}\,(\phi\,\mathbf{R}\,\psi))
\end{array}
$$

Notice the recursion

LTL Expansion Laws

Basic LTL expansion laws (e.g. for unfolding)

$$
\begin{array}{lcl}
\mathbf{F}\,\phi & \equiv & \phi \lor \mathbf{X}\,(\mathbf{F}\,\phi) \\
\mathbf{G}\,\phi & \equiv & \phi \land \mathbf{X}\,(\mathbf{G}\,\phi) \\
\phi\,\mathbf{U}\,\psi & \equiv & \psi \lor (\phi \land \mathbf{X}\,(\phi\,\mathbf{U}\,\psi)) \\
\phi\,\mathbf{W}\,\psi & \equiv & \psi \lor (\phi \land \mathbf{X}\,(\phi\,\mathbf{W}\,\psi)) \\
\phi\,\mathbf{M}\,\psi & \equiv & \psi \land (\phi \lor \mathbf{X}\,(\phi\,\mathbf{M}\,\psi)) \\
\phi\,\mathbf{R}\,\psi & \equiv & \psi \land (\phi \lor \mathbf{X}\,(\phi\,\mathbf{R}\,\psi))
\end{array}
$$

Notice the recursion

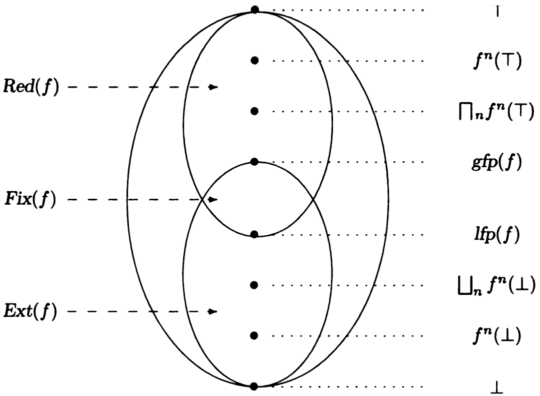
Think of **F**, **G**, **U**, **W**, **M**, **R** as specialized recursive operators.
What if we could have more powerful (arbitrary) recursions?

Outline

Background: Fixed-points

Reductive
$f(x) \sqsubseteq x$

Extensive
$x \sqsubseteq f(x)$



Tarski-Knaster theorem

A monotonic function $f : L \to L$ on a complete lattice L has a *greatest fixed point* (gfp) and a *least fixed point* (lfp).

$$\mathrm{gfp}(f) = \bigsqcup\{x \in L \mid x \sqsubseteq f(x)\} \; = \; \bigsqcup\{Ext(f)\} \in \mathrm{Fix}(f)$$

$$\mathrm{lfp}(f) = \bigsqcap\{x \in L \mid f(x) \sqsubseteq x\} \; = \; \bigsqcap\{Red(f)\} \in \mathrm{Fix}(f)$$

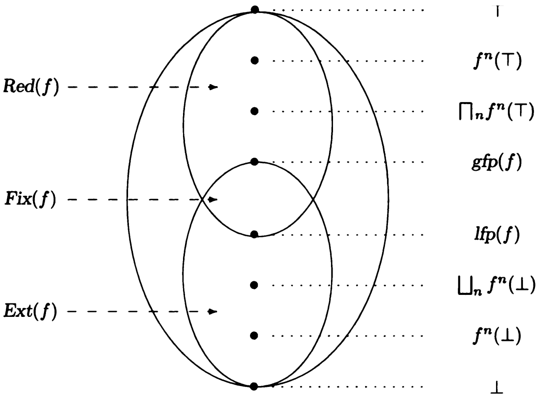
Background: Fixed-points

Reductive

$f(x) \sqsubseteq x$

Extensive

$x \sqsubseteq f(x)$



Kleene fixed-point theorem

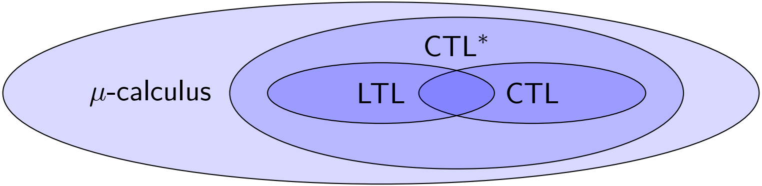
$$\begin{array}{lcccl} \text{gfp} & = & f^\infty(\top) & = & \bigsqcap_{n\geq 0} f^n(\top) \\ \text{lfp} & = & f^\infty(\bot) & = & \bigsqcup_{n\geq 0} f^n(\bot) \end{array}$$

$$\bot \sqsubseteq f(\bot) \sqsubseteq f(f(\bot)) \sqsubseteq \ldots \sqsubseteq \text{lfp}(f)$$
$$\sqsubseteq \text{gfp}(f) \sqsubseteq \ldots \sqsubseteq f(f(\top)) \sqsubseteq f(\top) \sqsubseteq \top$$

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

Some notation and terminology:

- The μ-calculus introduces variables representing sets of states.
- An occurrence of X is bound by a surrounding fixed point symbol μX or νX. Unbound occurrences of X are called free.
- A formula is closed if it has no free variables, otherwise it is called open
- A valuation $\mathcal{V}\colon \mathit{Var} \to 2^S$ interprets the free variables as sets of states.
- $\mathcal{V}[X := Q]$ is a valuation like $\mathcal{V}$, but X is set to Q
- The semantics of a μ-calculus formula ϕ is a set of states

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

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

(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

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

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

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

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

μ-calculus: Example (1/3)

$\mu X.[a]X$ represent states with no infinite sequences of a-transitions

$$\mu^0 X.[a]X = \emptyset \qquad \text{false}$$

$$\mu^1 X.[a]X = [a]\emptyset$$

$$= \{s \in S \mid \forall t.\ s \xrightarrow{a} t \rightarrow t \models \emptyset\}$$

since no t satisfies $\emptyset$, the right hand side (RHS) of $\rightarrow$ is false;

thus the left hand side (LHS) of $\rightarrow$ cannot be true.

This represents states with no outgoing a-transitions

$$\mu^2 X.[a]X = [a]T$$

where $T = \mu^1 X.[a]X$ are states with no outgoing a-transitions

Thus μ^2 means states with no aa-paths

μ-calculus: Example (2/3)

$\nu X.p \wedge [a]X$ is informally analogous to LTL **G** p

$$\nu^0 X.p \wedge [a]X = S \qquad \text{true}$$

$$\nu^1 X.p \wedge [a]X = p \wedge [a]S$$

Intersection between all nodes satisfying p (LHS of $\wedge$)

and all nodes (RHS of $\wedge$)

$$\nu^2 X.p \wedge [a]X = p \wedge [a]T$$

Where $T = \nu^1 X.p \wedge [a]X$ are all nodes that satisfy p

Thus μ^2 is the intersection between all nodes that satisfy p

and all nodes that have an outgoing edge labeled a

to a node that satisfies p

All nodes that satisfy p and whose descendants that are reachable through a-transitions also satisfy p.

μ-calculus: Example (3/3)

$\mu X.p \vee (\langle a \rangle True \wedge [a]X)$ is informally analogous to LTL **F** p

$\mu^0 X.p \vee (\langle a \rangle True \wedge [a]X) = \emptyset$

$\mu^1 X.p \vee (\langle a \rangle True \wedge [a]\emptyset) = p \vee (\langle a \rangle True \wedge [a]\emptyset)$

$\langle a \rangle True$ is the set of states with an outer a-transition

$[a]\emptyset$ is the set of states with no outgoing a-transition

Therefore, intersection $\wedge$ is empty

and the formula boils down to the set of states satisfying p

$\mu^2 X.p \vee (\langle a \rangle True \wedge [a]T) = p \vee (\langle a \rangle True \wedge [a]T)$

where $T = \mu^1$ which means nodes satisfying p

$[a]T$ are nodes whose children reachable via a-transitions satisfy p

Thus either p is satisfied, or it is satisfied via a node reachable through an a-transitions, or via an aa-transition, or via an a^n-transition.

Note

- 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
- See also Chapter 26 of the Handbook of Model Checking

Alternation Depth

Nesting Depth: maximum number of nested fixed points

$ND(f)$	$:=$	0	for $f \in \{p, \neg p, X\}$
$ND(ⓐ f)$	$:=$	$ND(f)$	for ⓐ $\in \{[a], \langle a \rangle\}$
$ND(f \square g)$	$:=$	$max(ND(f), ND(g))$	for $\square \in \{\wedge, \vee\}$
$ND(^{\mu}_{\nu} X.f)$	$:=$	$1 + ND(f)$	for $^{\mu}_{\nu} \in \{\mu, \nu\}$

Example: $ND\Big((\mu X_1.\ \nu X_2.\ X_1 \vee X_2) \wedge (\mu X_3.\ \mu X_4.\ (X_3 \wedge \mu X_5.\ p \vee X_5))\Big)$

Alternation Depth

Nesting Depth: maximum number of nested fixed points

$ND(f)$	$:=$	0	for $f \in \{p, \neg p, X\}$
$ND(ⓐ f)$	$:=$	$ND(f)$	for ⓐ $\in \{[a], \langle a \rangle\}$
$ND(f \Box g)$	$:=$	$max(ND(f), ND(g))$	for $\Box \in \{\wedge, \vee\}$
$ND(^{\mu}_{\nu} X.f)$	$:=$	$1 + ND(f)$	for $^{\mu}_{\nu} \in \{\mu, \nu\}$

Example:

$$ND\Big((\mu X_1.\ \nu X_2.\ X_1 \vee X_2) \wedge (\mu X_3.\ \mu X_4.\ (X_3 \wedge \mu X_5.\ p \vee X_5))\Big) = 3$$

X_3, X_4 and X_5 have no alternation between fixed point signs

Alternation Depth

Alternation Depth: number of alternating fixed points

$AD(f)$	$:=$	0	for $f \in \{p, \neg p, X\}$
$AD(ⓐ f)$	$:=$	$AD(f)$	for ⓐ $\in \{[a], \langle a \rangle\}$
$AD(f \Box g)$	$:=$	$max(AD(f), AD(g))$	for $\Box \in \{\wedge, \vee\}$
$AD(\mu X.f)$	$:=$	$1 + max\{AD(g) \mid g$ is a ν-subformula of $f\}$	
$AD(\nu X.f)$	$:=$	$1 + max\{AD(g) \mid g$ is a μ-subformula of $f\}$	

Examples:

$$AD\Big((\mu X_1.\ \nu X_2.\ X_1 \vee X_2) \wedge (\mu X_3.\mu X_4.\ (X_3 \wedge \mu X_5.p \vee X_5))\Big)$$

$$AD\Big((\mu X_1.\ \nu X_2.\ X_1 \vee X_2) \wedge (\mu X_3.\nu X_4.\ (X_3 \wedge \mu X_5.p \vee X_5))\Big)$$

Alternation Depth

Alternation Depth: number of alternating fixed points

$$\begin{array}{rcll} AD(f) & := & 0 & \text{for } f \in \{p, \neg p, X\} \\ AD(ⓐf) & := & AD(f) & \text{for } ⓐ \in \{[a], \langle a \rangle\} \\ AD(f \square g) & := & max(AD(f), AD(g)) & \text{for } \square \in \{\wedge, \vee\rangle\} \\ AD(\mu X.f) & := & 1 + max\{AD(g) \mid g \text{ is a } \nu\text{-subformula of } f\} & \\ AD(\nu X.f) & := & 1 + max\{AD(g) \mid g \text{ is a } \mu\text{-subformula of } f\} & \end{array}$$

Examples:

$$AD\Big((\mu X_1.\ \nu X_2.\ X_1 \vee X_2) \wedge (\mu X_3.\mu X_4.\ (X_3 \wedge \mu X_5.p \vee X_5))\Big) = 2$$

$$AD\Big((\mu X_1.\ \nu X_2.\ X_1 \vee X_2) \wedge (\mu X_3.\nu X_4.\ (X_3 \wedge \mu X_5.p \vee X_5))\Big) = 3$$

X_5 does not depend on X_3 and X_4

Alternation Depth

Dependent Alternation Depth (dAD): number of alternating fixed points, such that the innermost fixed point depends on the outermost.

The definition of dAD is identical to AD, except for

$$\boxed{\begin{array}{lcl} dAD(\mu X.f) & := & max(dAD(f), \\ & & \quad 1 + max\{dAD(g) \mid \\ & & \qquad\qquad g \text{ is a } \nu\text{-subformula of } f \text{ and } X \text{ occurs in } g\} \\ dAD(\nu X.f) & := & max(dAD(f), \\ & & \quad 1 + max\{dAD(g) \mid \\ & & \qquad\qquad g \text{ is a } \mu\text{-subformula of } f \text{ and } X \text{ occurs in } g\} \end{array}}$$

Examples:

$$dAD\Big((\mu X_1.\ \nu X_2.\ X_1 \vee X_2) \wedge (\mu X_3.\mu X_4.\ (X_3 \wedge \mu X_5.p \vee X_5))\Big)$$

$$dAD\Big((\mu X_1.\ \nu X_2.\ X_1 \vee X_2) \wedge (\mu X_3.\nu X_4.\ (X_3 \wedge \mu X_5.p \vee X_5))\Big)$$

Alternation Depth

Dependent Alternation Depth (dAD): number of alternating fixed points, such that the innermost fixed point depends on the outermost.

The definition of dAD is identical to AD, except for

$$\boxed{\begin{array}{lcl} dAD(\mu X.f) & := & max(dAD(f), \\ & & \quad 1 + max\{dAD(g) \mid \\ & & \qquad g \text{ is a } \nu\text{-subformula of } f \text{ and } X \text{ occurs in } g\} \\ dAD(\nu X.f) & := & max(dAD(f), \\ & & \quad 1 + max\{dAD(g) \mid \\ & & \qquad g \text{ is a } \mu\text{-subformula of } f \text{ and } X \text{ occurs in } g\} \end{array}}$$

Examples:

$$dAD\Big((\mu X_1.\ \nu X_2.\ X_1 \vee X_2) \wedge (\mu X_3.\mu X_4.\ (X_3 \wedge \mu X_5.p \vee X_5))\Big) = 2$$

$$dAD\Big((\mu X_1.\ \nu X_2.\ X_1 \vee X_2) \wedge (\mu X_3.\nu X_4.\ (X_3 \wedge \mu X_5.p \vee X_5))\Big) = 2$$

Naive Algorithm

def eval(f):

  if $f = tt$ **: return** S

  elif $f = ff$ **: return** $\emptyset$

  elif $f = p$ **: return** $\{s \in S \mid p \in L(s)\}$

  elif $f = \neg p$ **: return** $\{s \in S \mid p \notin L(s)\}$

  elif $f = g_1 \wedge g_2$ **: return** $\mathsf{eval}(g_1) \cap \mathsf{eval}(g_2)$

  elif $f = g_1 \vee g_2$ **: return** $\mathsf{eval}(g_1) \cup \mathsf{eval}(g_2)$

  elif $f = [a]g$ **: return** $\{s \in S \mid \forall t \in S : s \xrightarrow{a} t \Rightarrow t \in \mathsf{eval}(g)\}$

  elif $f = \langle a \rangle g$ **: return** $\{s \in S \mid \exists t \in S : s \xrightarrow{a} t \wedge (t \in \mathsf{eval}(g))\}$

  elif $\ldots$ **:** $\ldots$

Naive Algorithm

```
def eval(f):
    if ... : ...
    elif f = X_i : return A[i]
    elif f = νX_i.g(X_i) :
        A[i] := S
        while A[i] changes :
            A[i] := eval(g)
        return A[i]
    elif f = μX_i.g(X_i) :
        A[i] := ∅
        while A[i] changes :
            A[i] := eval(g)
        return A[i]
```

Embedding CTL-formulae

Assume $\mathcal{A}ct = \{a\}$. There is a straightforward translation of CTL to the μ-calculus:

- $Tr(p) = p$
- $Tr(\neg f) = \neg Tr(f)$
- $Tr(f \wedge g) = Tr(f) \wedge Tr(g)$
- $Tr(\mathbf{E}\ \mathbf{X}\ f) = \langle a \rangle\ Tr(f)$
- $Tr(\mathbf{E}\ \mathbf{G}\ f) = \nu Y.(Tr(f) \wedge \langle a \rangle\ Y)$
- $Tr(\mathbf{E}\ [f\ \mathbf{U}\ g]) = \mu Y.(Tr(g) \vee (Tr(f) \wedge \langle a \rangle\ Y))$

Outline

Bird's Eye View

- Area: formal verification of systems
  - Verify if a system implements the specification
  - Synthesize a controller for an incomplete system that implements the specification
- "Does X have property p" as a game (or compute X such that...)
  - player 0 wants to prove this (or synthesize a controller)
  - player 1 wants to refute this
  - players make *choices*
- Interesting systems are often "reactive" (run forever)
  - when a car arrives, eventually the traffic light turns green
  - the reset button always works
  - "X is true until Y is true"
  - "X may not happen before Y"

Hence: properties regarding infinite runs of a finite-state system

Why do we want to solve parity games?

- Capture the expressive power of nested least and greatest fixpoint operators
- Equivalent (in polynomial time) to:
  - modal μ-calculus model-checking (CTL*, LTL...)
  - solving Boolean Equation Systems
- Backend for LTL model checking and LTL synthesis
  - important industrial applications (PSL, SVA)

Parity Games

Why do we want to solve parity games?

- Capture the expressive power of nested least and greatest fixpoint operators
- Equivalent (in polynomial time) to:
  - modal μ-calculus model-checking (CTL*, LTL...)
  - solving Boolean Equation Systems
- Backend for LTL model checking and LTL synthesis
  - important industrial applications (PSL, SVA)

Open question: Is solving parity games in **P**?

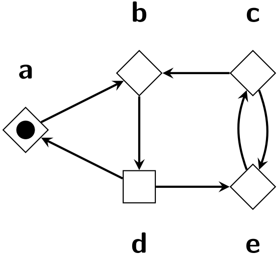
- It is in **UP** $\cap$ **co-UP** and **NP** $\cap$ **co-NP**
- It is believed a polynomial solution exists
- Hot topic! Recently: quasi-polynomial solution sparked great interest, several new algorithms that are all quasi-polynomial

(Incomplete list of) published algorithms

McNaughton/Zielonka	$\mathcal{O}(e \cdot n^d)$, $\mathcal{O}(2^n)$	1998
Small Progress Measures	$\mathcal{O}(d \cdot e \cdot (n/d)^{d/2})$	1998
Strategy Improvement	$\mathcal{O}(n \cdot e \cdot 2^e)$	2000
Dominion Decomposition	$\mathcal{O}(n^{\sqrt{n}})$	2006
Big Step	$\mathcal{O}(e \cdot n^{d/3})$	2007
APT	$\mathcal{O}(n^d)$	2016
Priority Promotion	$\Omega(2^{\sqrt{n}})$	2016
Quasi-Polynomial (multiple)	$\mathcal{O}(n^{6+\log d})$	2016 – 2018
Tangle Learning	$\Omega(2^{\sqrt{n}})$	2018
Recursive Tangle Learning	tbd	2018

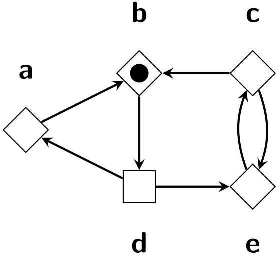
Parity Games

- A parity game is played on a directed graph
- Two players: Even $\diamond$ and Odd $\square$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



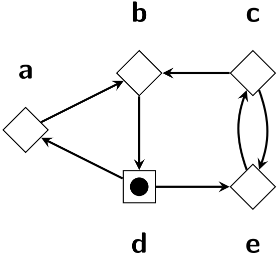
Parity Games

- A parity game is played on a directed graph
- Two players: Even $\Diamond$ and Odd $\Box$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



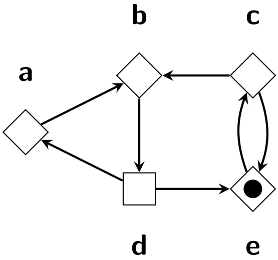
Parity Games

- A parity game is played on a directed graph
- Two players: Even $\diamond$ and Odd $\square$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



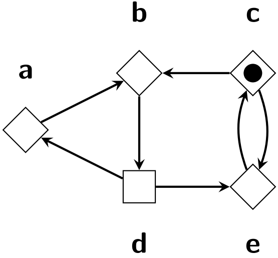
Parity Games

- A parity game is played on a directed graph
- Two players: Even $\Diamond$ and Odd $\Box$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



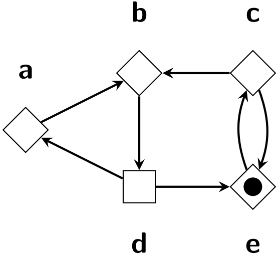
Parity Games

- A parity game is played on a directed graph
- Two players: Even $\diamond$ and Odd $\square$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



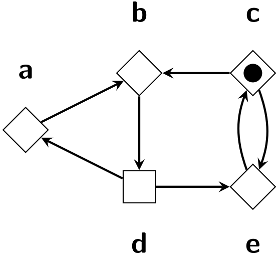
Parity Games

- A parity game is played on a directed graph
- Two players: Even $\Diamond$ and Odd $\Box$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



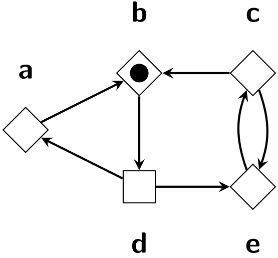
Parity Games

- A parity game is played on a directed graph
- Two players: Even ◇ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



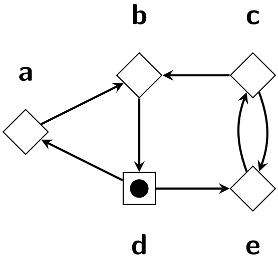
Parity Games

- A parity game is played on a directed graph
- Two players: Even $\Diamond$ and Odd $\Box$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



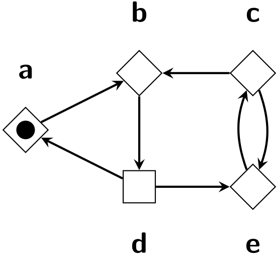
Parity Games

- A parity game is played on a directed graph
- Two players: Even ◇ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor

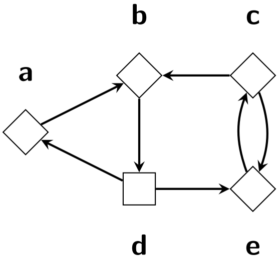


Parity Games

- A parity game is played on a directed graph
- Two players: Even $\Diamond$ and Odd $\Box$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor

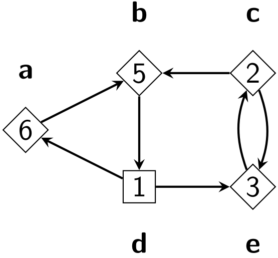


- A parity game is played on a directed graph
- Two players: Even ◇ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



How do we determine who wins a play?

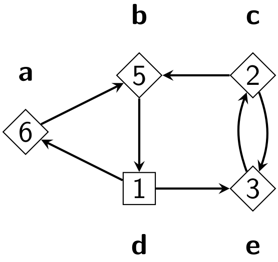
- A parity game is played on a directed graph
- Two players: Even $\Diamond$ and Odd $\Box$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



- Each vertex has a priority $\{0, 1, 2, \ldots, d\}$
- Highest priority seen infinitely often determines winner
- Player Even wins if this number is even

Parity Games

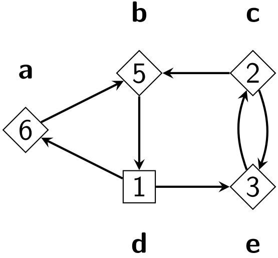
- A parity game is played on a directed graph
- Two players: Even $\Diamond$ and Odd $\Box$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



How do we determine who wins a vertex?

Parity Games

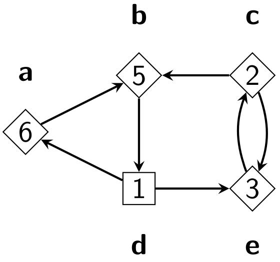
- A parity game is played on a directed graph
- Two players: Even $\Diamond$ and Odd $\Box$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



A player wins a vertex if it has a strategy
to win all plays from that vertex

Parity Games

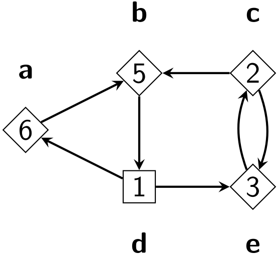
- A parity game is played on a directed graph
- Two players: Even $\diamond$ and Odd $\square$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



Which vertices are won by which player?

Parity Games

- A parity game is played on a directed graph
- Two players: Even $\Diamond$ and Odd $\Box$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



Player Odd wins all vertices with strategy $\{\mathbf{d} \to \mathbf{e}\}$

Parity Games

Known facts of parity games

- Some vertices are won by Even, some vertices are won by Odd
- The winner has a memoryless strategy to win

Memoryless strategy

"If I always play from v to w, then I win all plays from v"

Parity Games

Known facts of parity games

- Some vertices are won by Even, some vertices are won by Odd
- The winner has a memoryless strategy to win

Memoryless strategy

"If I always play from v to w, then I win all plays from v"

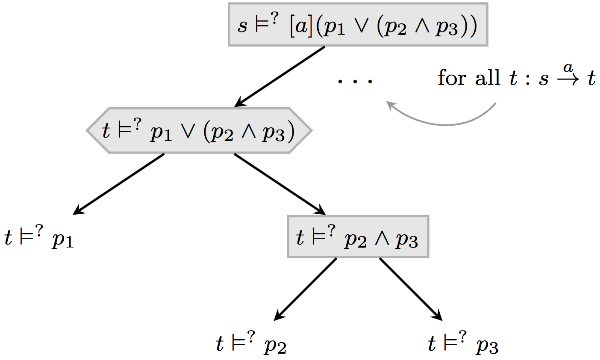
Solving a parity game

- Determine the winner of each vertex
- Compute the strategy for each player

Games and automata for verification and synthesis

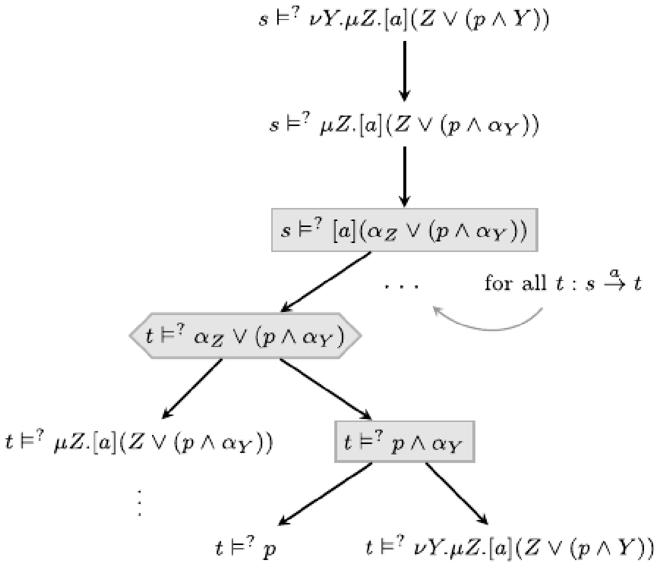
- Verification with automata
  1. Construct an automaton of the specification (typically negation)
  2. Cross-product with the Kripke Structure or LTS
  3. Solve resulting automaton (accept/reject)
     (typically produces a counterexample)
- Verification with games
  1. Construct a two-player game of the specification
     - one player tries to prove the specification (‘existential‘)
     - one player tries to violate the specification (‘universal‘)
  2. Cross-product with the Kripke Structure or LTS
  3. Solve resulting game
     (winner + strategy of winner.)
- Synthesis with games
  - strategy is an implementation of a controller (or a counterexample)
  - system + controller = guaranteed to implement the specification
  - (Actually used in practice to synthesize controllers for LTL properties!)

Model checking via parity games



- Adam picks t from $s \xrightarrow{a} t$ such that $t \not\models (p_1 \vee (p_2 \wedge p3)$
- Eve replies by showing that either $t \models p_1$ or that $t \models p_2$ and $t \models p_3$.

Model checking via parity games



Model checking via parity games

Create node (s, ψ) for every state s of $\mathcal{M}$ and every formula ψ in the closure of ϕ. Eve's goal is to show that a formula holds.

(s, p)	Eve wins if p holds in s, that is $s \vDash p$ Thus assign (s, p) to Adam and we put no transitions from it
$(s, \neg p)$	Same as (s, p) but reversing Adam and Eve's roles
$(s, \langle a \rangle \beta)$ $(s, [a]\beta)$	Connect to (t, β) for all t such that $s \xrightarrow{\mathsf{a}} t$ and assign $(s, [a]\beta)$ to Adam and $(s, \langle a \rangle \beta)$ to Eve
$(s, \mu X.\beta(X))$ $(s, \nu X.\beta(X))$	Connect to $(s, \beta(\mu X.\beta(X)))$ and to $(s, \beta(\nu X.\beta(X)))$ This corresponds to the intuition that a fixed-point is equivalent to its unfolding.

$$[\![\mu X.\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} = [\![\alpha[\mu X.\alpha / X]]\!]^{\mathcal{M}}_{\mathcal{V}}$$
$$[\![\nu X.\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} = [\![\alpha[\nu X.\alpha / X]]\!]^{\mathcal{M}}_{\mathcal{V}}$$

- Parity winning condition based on dependent alternation depth.
  - Priority $2 \cdot \lfloor \mathsf{dAD}(\phi)/2 \rfloor$ if ϕ is of the form $\nu X.\psi$
  - Priority $2 \cdot \lfloor \mathsf{dAD}(\phi)/2 \rfloor + 1$ if ϕ is of the form $\mu X.\psi$
  - Priority 0 otherwise

Oink

- Modern implementation of parity game algorithms
  - Zielonka's Algorithm (with optimizations; parallel)
  - Small progress measures (with optimizations)
  - Priority Promotion (different versions)
  - Strategy Improvement (parallel)
  - QPT progress measures
  - Succinct progress measures
  - Tangle learning
- The usual preprocessing algorithms
  - Inflation and compression
  - Remove self-loops
  - Detect 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("zlk");
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 $\eth$ is a tuple $(V_\Diamond, V_\Box, E, \mathsf{pr})$
- Vertices $V = V_\Diamond \cup V_\Box$ controlled by players Even and Odd
- Transitions $E\colon V \times V$ 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 $\mathsf{pr}\colon V \to \{0, 1, 2, \ldots, d\}$
- A path is a finite sequence $v_0 v_1 v_2 \ldots$ consistent with E
- 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 $\max(\mathsf{pr}(\inf \pi))$ is even

Notation for strategies

- A strategy for player α is a partial function $\sigma\colon V_\alpha \to V$ that assigns one successor to each vertex of player α.
- A path or play is consistent with σ if each v_i for which $\sigma(v_i)$ is defined, $v_{i+1} = \sigma(v_i)$.
- Plays(v, σ) 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 Plays(v, σ) 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.

Outline

1. Temporal logics CTL and LTL
2. The modal μ-calculus
3. Parity games
4. Attractor based algorithms
5. Fixed point based algorithms

Attractor computation

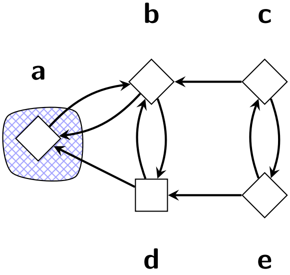
Compute all vertices from which player $\alpha \in \{\diamond, \square\}$ can ensure arrival in a given target set

Start with the target set A, then iteratively add vertices to A:

- All vertices of α with an edge to A
- All vertices of $\overline{\alpha}$ with only edges to A

Example of attractor computation

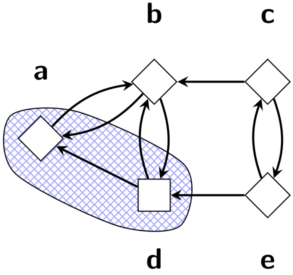
Computing the □-attractor to **a**



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

Example of attractor computation

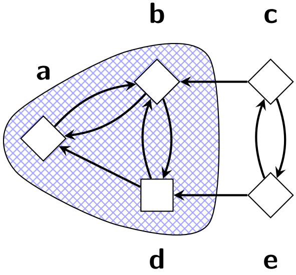
Computing the □-attractor to **a**



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

Example of attractor computation

Computing the □-attractor to **a**



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

Parity Game Algorithms

Roughly two types

- Local value iteration
  Based on locally improving the value of individual vertices by looking at their successors.
- Attractor-based
  Based on properties over sets of vertices computed with attractors.

Parity Game Algorithms

Attractor-based algorithms

- Partition the game into regions using attractors.
- Start with the highest priority (top-down).
- Each region is tentatively won by one player.
- Refine winning regions until dominion found.

Example: Zielonka's Recursive Algorithm (1998)

Attract higher regions downward after computing lower regions.
If your opponent attracts from your region, recompute your part.

Example: Priority Promotion (2016)

Merge regions upwards when the region is closed (in the subgame).
Then recompute lower regions.

Zielonka's recursive algorithm

def $\texttt{zielonka}(\Game)$:

  if $\Game = \emptyset$ **:**

    return $\emptyset, \emptyset$ // empty game

  $\alpha \leftarrow \mathsf{pr}(\Game) \bmod 2$ // winner of highest priority

  $Z \leftarrow \mathsf{pr}^{-1}(\mathsf{pr}(\Game))$ // vertices of highest priority

  $A \leftarrow \mathsf{Attr}^{\Game}_{\alpha}(Z)$ // attracted to highest priority

  $W_\diamond, W_\square \leftarrow \texttt{zielonka}(\Game \setminus A)$ // recursive solution

  $B \leftarrow \mathsf{Attr}^{\Game}_{\overline{\alpha}}(W_{\overline{\alpha}})$ // check if opponent attracts

  if $B = W_{\overline{\alpha}}$ **:**

    $W_\alpha \leftarrow W_\alpha \cup A$ // A is won by α

  else:

    $W_\diamond, W_\square \leftarrow \texttt{zielonka}(\Game \setminus B)$ // recompute remainder

    $W_{\overline{\alpha}} \leftarrow W_{\overline{\alpha}} \cup B$ // B is won by $\overline{\alpha}$

  return $W_\diamond, W_\square$

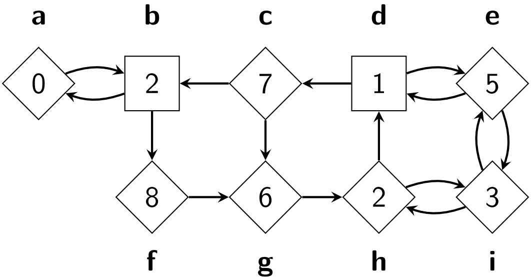
Zielonka's recursive algorithm

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 highest priority vertices before attracting

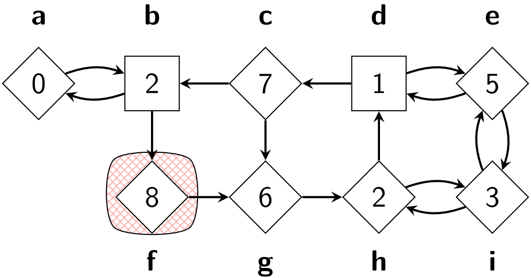
def zielonka($\Game$):
    if $\Game = \emptyset$ **:**
        return $\emptyset, \emptyset$ // empty game
    $\alpha \leftarrow \mathsf{pr}(\Game) \bmod 2$ // winner of highest priority
    $Z \leftarrow \mathsf{pr}^{-1}(\mathsf{pr}(\Game))$ // vertices of highest priority
    $A, \sigma_A \leftarrow \mathsf{Attr}_{\alpha}^{\Game}(Z)$ // attracted to highest priority
    $W_\Diamond, W_\Box, \sigma_\Diamond, \sigma_\Box \leftarrow$ zielonka($\Game \setminus A$) // recursive solution
    $B, \sigma_B \leftarrow \mathsf{Attr}_{\overline{\alpha}}^{\Game}(W_{\overline{\alpha}})$ // check if opponent attracts
    $\sigma_B \leftarrow \sigma_B \cup \sigma_{\overline{\alpha}}$ // add strategy of $W_{\overline{\alpha}}$
    if $B = W_{\overline{\alpha}}$ **:**
        $W_\alpha \leftarrow W_\alpha \cup A$ // A is won by α
        $\sigma_\alpha \leftarrow \sigma_\alpha \cup \sigma_A \cup \big((z \in Z) \mapsto \mathsf{pick}(E(z) \cap W_\alpha)\big)$
    else:
        $W_\Diamond, W_\Box, \sigma_\Diamond, \sigma_\Box \leftarrow$ zielonka($\Game \setminus B$) // recompute remainder
        $W_{\overline{\alpha}} \leftarrow W_{\overline{\alpha}} \cup B$ // B is won by $\overline{\alpha}$
        $\sigma_{\overline{\alpha}} \leftarrow \sigma_{\overline{\alpha}} \cup \sigma_B$
    return $W_\Diamond, W_\Box, \sigma_\Diamond, \sigma_\Box$

Zielonka's Algorithm



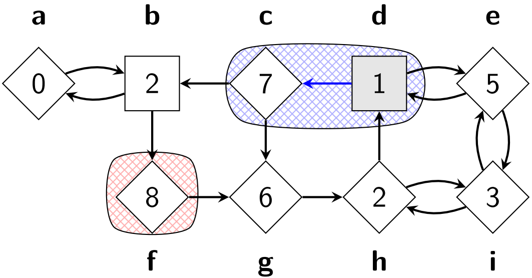
- We start by attracting to 8 for player Even.

Zielonka's Algorithm



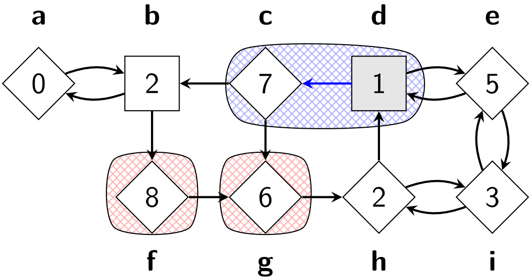
- After region 8 (player Even).
- Continue (recursively) with region 7.

Zielonka's Algorithm



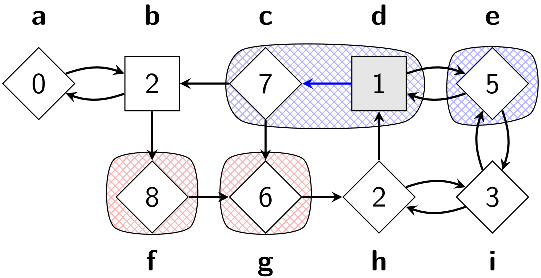
- After regions 8 (player Even) and 7 (player Odd).
- Continue (recursively) with region 6.

Zielonka's Algorithm



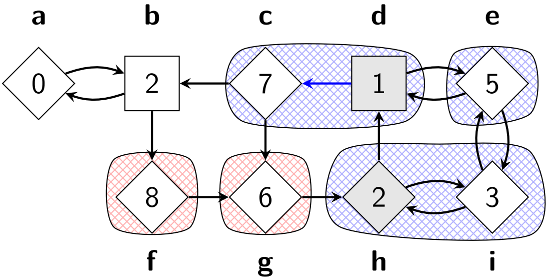
- After regions 8, 7 and 6.
- Continue (recursively) with region 5.

Zielonka's Algorithm



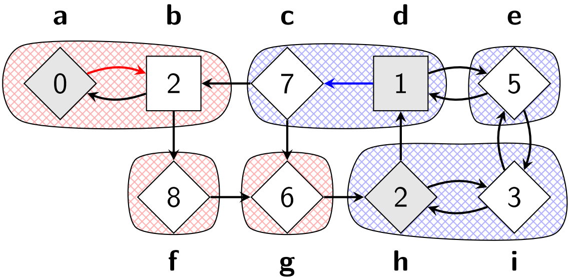
- After regions 8, 7, 6 and 5.
- Continue (recursively) with region 3.

Zielonka's Algorithm



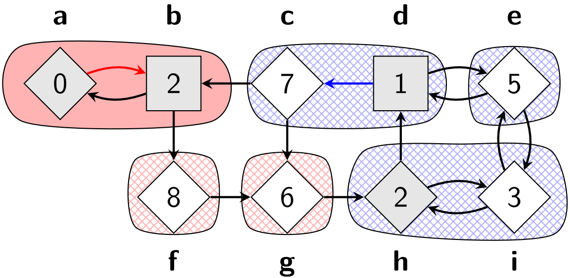
- After regions 8, 7, 6, 5 and 3.
- Now remains just region 2.

Zielonka's Algorithm



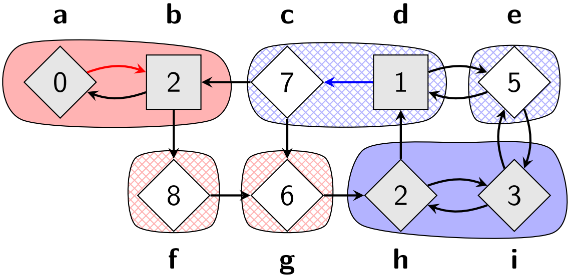
- Game is partitioned fully, now go up in the recursion.
- Up in region 2, does the lower opponent's winning region attract?
- Region 2: no (because the subgame is empty).

Zielonka's Algorithm



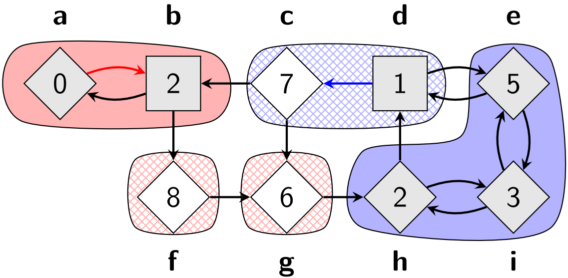
- Up in region 3: does the lower Even region attract from region 3?

Zielonka's Algorithm



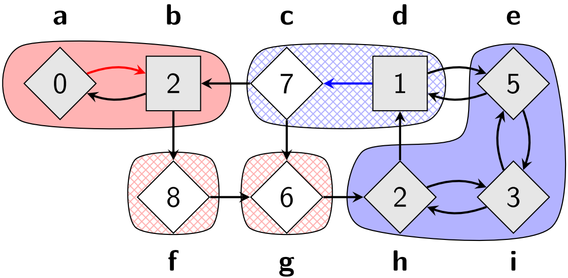
- Up in region 5: does the lower Even region attract from region 5?

Zielonka's Algorithm



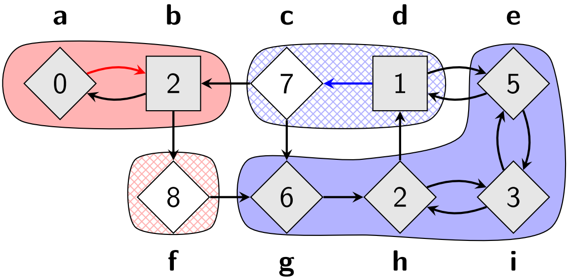
- Up in region 6: does the lower Odd region attract from 6?

Zielonka's Algorithm

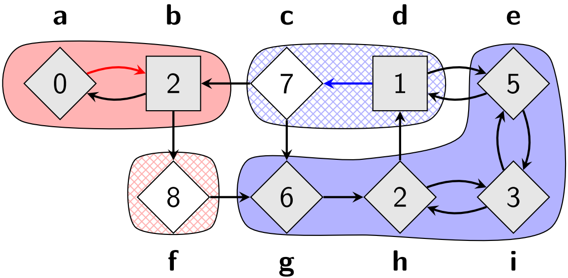


- Up in region 6: does the lower Odd region attract from 6?
- Yes: the lower Odd region attracts vertex **g**.

Zielonka's Algorithm

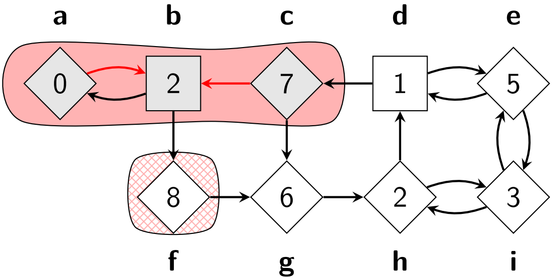


- Vertex **g** is attracted to the Odd region.
- So now recompute the (remainder of the) lower regions of Even.
  - Actually, nothing changes in the recursion.
- Up in region 7: does the lower Even region attract from 7?



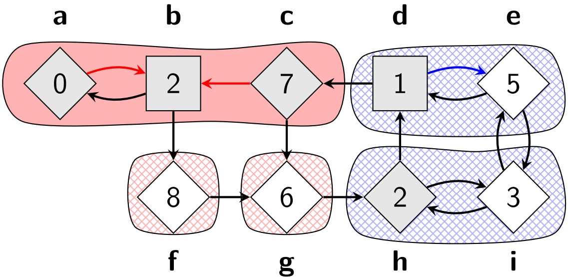
- Vertex **g** is attracted to the Odd region.
- So now recompute the (remainder of the) lower regions of Even.
  - Actually, nothing changes in the recursion.
- Up in region 7: does the lower Even region attract from 7?
- Yes, the lower Even region attracts vertex **c**.

Zielonka's Algorithm



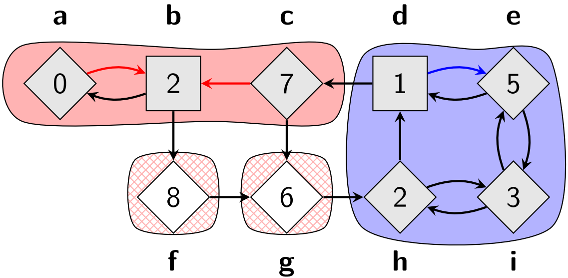
- Vertex **c** is attracted to the Even region.
- Recompute the remainder of the lower regions of Odd.

Zielonka's Algorithm

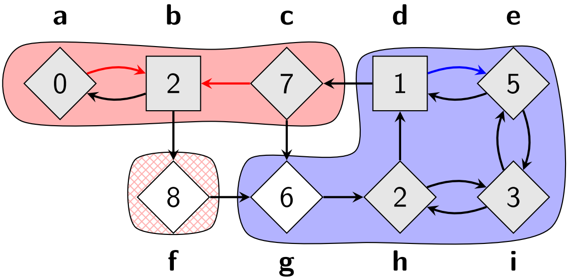


- Partition the remainder into regions 6, 5 and 3.
- Up in region 3: no attraction from Even.
- Up in region 5: no attraction from Even.
- Up in region 6: the lower Odd regions attract **g** again!

Zielonka's Algorithm

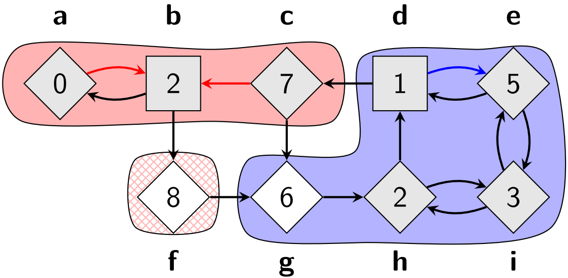


- Region 6: now the Odd region attracts vertex g again.



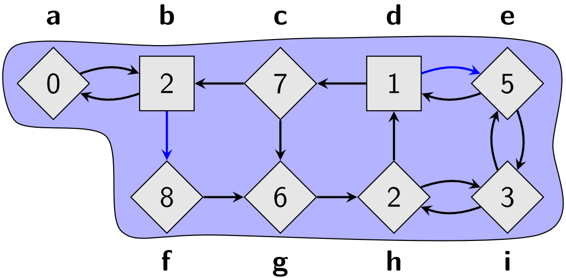
- Vertex **g** is attracted to the Odd region.
- Recursive game of 6 is empty.
- Up in region 8, does Odd now attract 8?

Zielonka's Algorithm



- Vertex **g** is attracted to the Odd region.
- Recursive game of 6 is empty.
- Up in region 8, does Odd now attract 8?
- But vertex **f** is attracted to the Odd region.
- Attracting at priority 8 attracts all vertices to player Odd.

Zielonka's Algorithm



- Final result, entire game won by player Odd.

Priority promotion

The main idea of priority promotion...

Region invariant

- In any region, the opponent either plays to a higher region of the player, or via the highest priority vertices.
- (Invariant holds for the regions of the "α-maximal partition")

Closed region

- A region of player α that is globally closed is a dominion of player α.
- A region of player α is locally closed iff the opponent can only escape to a higher region of player α.
- So: the opponent must escape to the lowest higher region.

$\Rightarrow$ Promote the region, i.e., merge the regions.

Priority promotion

def prioprom($\Game$):

    $\mathsf{r} \leftarrow V \mapsto \bot$ `// all vertices to ⊥`

    $p \leftarrow \mathsf{pr}(\Game)$ `// highest priority`

    while True **:**

        $\alpha \leftarrow p \bmod 2$ `// current player`

        $Z \leftarrow \{v \mid \mathsf{r}(v) \leq p\}$ `// current subgame`

        $A \leftarrow \mathsf{Attr}_{\alpha}^{\Game \cap Z}(\{v \in Z \mid \mathsf{r}(v) = p \vee \mathsf{pr}(v) = p\})$ `// attract`

        $C \leftarrow \{v \in A_{\alpha} \mid E(v) \cap A = \emptyset\}$ `// open α-vertices`

        $X \leftarrow E(A_{\overline{\alpha}}) \setminus A$ `// escapes`

        if $C \neq \emptyset \vee (X \cap Z) \neq \emptyset$ **:**

            $\mathsf{r} \leftarrow \mathsf{r}[A \mapsto p]$ `// set region`

            $p \leftarrow \mathsf{pr}(Z \setminus A)$ `// continue with next highest`

        elif $X \neq \emptyset$ **:**

            $p \leftarrow \min\{\mathsf{r}(v) \mid v \in X\}$ `// set p to lowest escape`

            $\mathsf{r} \leftarrow \mathsf{r}[A \mapsto p][\{v \mid \mathsf{r}(v) < p\} \mapsto \bot]$ `// merge and reset`

        else:

            return α, A `// dominion!`

Priority promotion

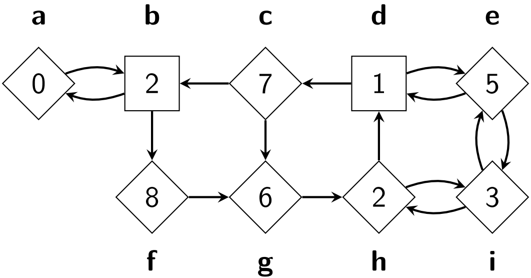
Notes

- The lowest region is always locally closed.
- Region resets only if at least 1 vertex promotes.
- This is sufficient to prove termination.
- Each call to `prioprom` computes a dominion of a player α.
- Attract for player α to the computed dominion, repeat until game solved.

Computing strategy

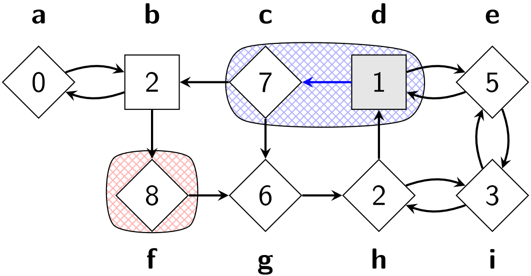
- Strategy is computed by attractor
  - Every attracted α-vertex u to some v in the set: strategy is $u \to v$
- Special case: α-vertices of the original target set
  - Pick any successor in result as strategy
- Implementation: use only a single strategy array, reset the strategy of highest priority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

Priority promotion



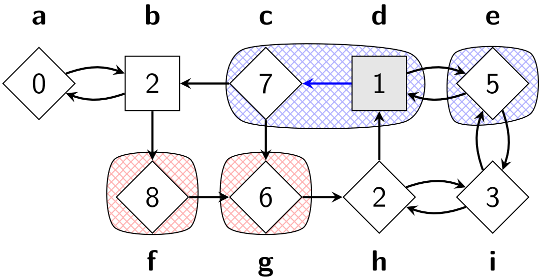
- We start by attracting to 8 for player Even, 7 for player Odd, etc.

Priority promotion



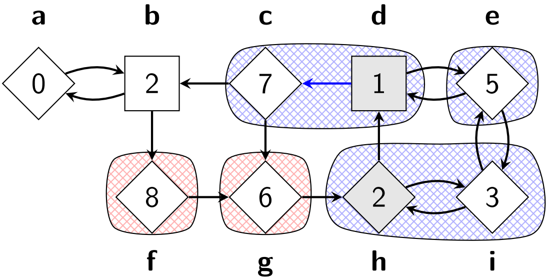
- After regions 8 (player Even) and 7 (player Odd).

Priority promotion



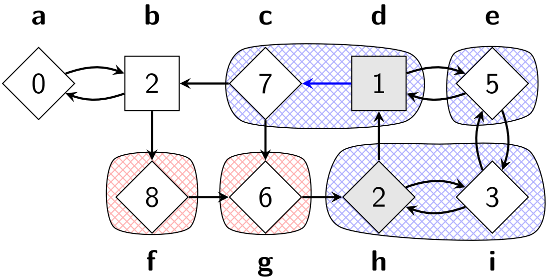
- After regions 6 and 5.

Priority promotion



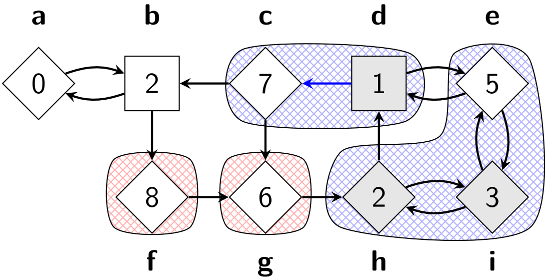
- After region 3, region 3 is now closed!
- **Note: region 2 would also be closed.**

Priority promotion



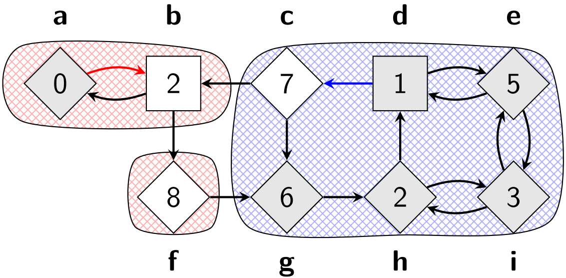
- After region 3, region 3 is now closed!
- **Note: region 2 would also be closed.**
- **The loser must escape to a higher region of the winner**.
- So promote 3 to 5.
- Meaning *the set* $\{\mathbf{h}, \mathbf{i}\}$ is attracted *as a whole* to region 5.

Priority promotion



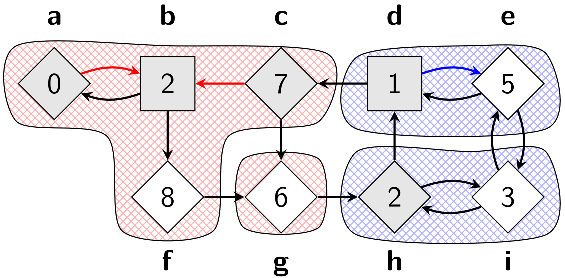
- Region 3 is promoted to region 5.
- Now region 5 is closed. (region 2 would also be closed)
- So promote 5 to 7.

Priority promotion



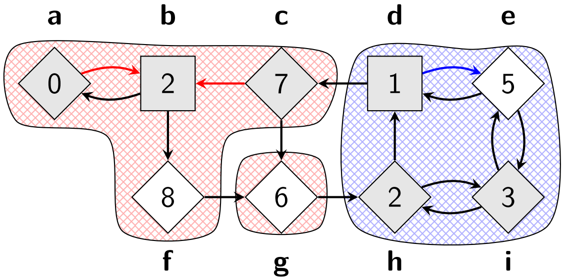
- After promotion, vertex **g** is attracted to region 7.
- Continue the partition with region 2...
- Region 2 is locally closed.
- So promote 2 to (the lowest escape) 8.

Priority promotion



- Region 2 is promoted to region 8.
- Meaning the set $\{$**a**$,$**b**$\}$ is attracted to region 8.
- Region 8 now also attracts vertex **c**!!
- Recompute the subgame...
- Now $\{$**h**$,$**i**$\}$ can be attracted to region 5 again.

Priority promotion



Region 5 is closed in the entire game.
Meaning that it is a dominion won by player Odd.

Priority promotion

Variations

- PP$+$: only reset regions of $\overline{\alpha}$.
- RP: only reset a region when the strategy of player α of the remaining vertices of the stored region leaves the region
- DP: "delayed promotion" strategy

Tangle

A tangle is:

- a (strongly connected) subgraph of a parity game,
- such that one player α has a strategy σ,
- such that the tangle restricted by σ is still strongly connected,
- and player α wins all plays (cycles) in the tangle.

Definition

A *p-tangle* is a nonempty set of vertices $U \subseteq V$ with $p = \mathrm{pr}(U)$, for which player $\alpha \equiv_2 p$ has a strategy $\sigma\colon U_\alpha \to U$, such that the graph (U, E'), with $E' := E \cap \big(\sigma \cup (U_{\overline{\alpha}} \times U)\big)$, is strongly connected and player α wins all cycles in (U, E').

Tangle Learning

Tangle

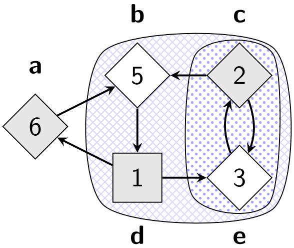
A tangle is a strongly connected subgraph for which one player has a strategy to win all cycles in the subgraph.

Properties

- Player α has a single strategy for every α-vertex.
- Player $\overline{\alpha}$ **must** escape (or lose).
- Player $\overline{\alpha}$ can reach all vertices of the tangle.
- Tangles have subtangles when player $\overline{\alpha}$ can avoid vertices.
- Every dominion is naturally composed of subtangles.

Tangle

A tangle is a strongly connected subgraph for which one player has a strategy to win all cycles in the subgraph.



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

Tangle learning

Tangle attractor

Because player $\overline{\alpha}$ **must** escape the tangle, we can use tangles to attract the vertices of a tangles together, if player $\overline{\alpha}$ can only escape to the attracting set.

- Add all $v \in V_\alpha \setminus A$ for which $E(v) \cap A \neq \emptyset$.
- Add all $v \in V_{\overline{\alpha}} \setminus A$ for which $E(v) \subseteq A$.
- Add all $\{ v \in V_T(t) \setminus A \mid t \in T_\alpha \}$ for which $E_T(t) \subseteq A$.

Tangle learning

- Partition game into α-maximal regions with tangle attractor.
- Add **bottom SCCs of closed regions** to the set of tangles.
- Repeat until a dominion is found, i.e., $E_T(t) = \emptyset$.

Tangle learning (1/2)

- `search` returns new tangles of $\Game$, given known tangles T.
- Note: store for each tangle its player α strategy.

def $\color{red}{\texttt{search}(\Game, T)}$**:**
    if $\Game = \emptyset$ **: return** $\emptyset$
    $p \leftarrow \mathsf{pr}(\Game)$, $\alpha \leftarrow \mathsf{pr}(\Game) \bmod 2$
    $Z, \sigma \leftarrow TAttr_{\alpha}^{\Game, T}(\mathsf{pr}^{-1}(p))$
    $O \leftarrow \{v \in Z_\alpha \mid E(v) \cap Z = \emptyset\} \cup \{v \in Z_{\overline{\alpha}} \mid E(v) \nsubseteq Z\}$
    if $O = \emptyset$ **:**
        return $\texttt{search}(\Game \setminus Z, T) \cup \texttt{bottom-sccs}(Z, \sigma)$
    else:
        return $\texttt{search}(\Game \setminus Z, T)$

Tangle learning (1/2)

- `search` returns new tangles of $\Game$, given known tangles T.
- Note: store for each tangle its player α strategy.

def $\color{red}{\texttt{search}(\Game, T)}$**:**

$\quad$ **if** $\Game = \emptyset$ **: return** $\emptyset$

$\quad$ $p \leftarrow \mathsf{pr}(\Game)$, $\alpha \leftarrow \mathsf{pr}(\Game) \bmod 2$

$\quad$ $Z, \sigma \leftarrow TAttr_{\alpha}^{\Game, T}(\mathsf{pr}^{-1}(p))$

$\quad$ $O \leftarrow \{v \in Z_\alpha \mid E(v) \cap Z = \emptyset\} \cup \{v \in Z_{\overline{\alpha}} \mid E(v) \not\subseteq Z\}$

$\quad$ **if** $O = \emptyset$ **:**

$\quad\quad$ **return** $\texttt{search}(\Game \setminus Z, T) \cup \texttt{bottom-sccs}(Z, \sigma)$

$\quad$ **else:**

$\quad\quad$ **return** $\texttt{search}(\Game \setminus Z, T) \cup \color{blue}{\texttt{search}(\Game \cap (Z \setminus TAttr_{\overline{\alpha}}^{\Game \cap Z, T}(O)), T)}$

The "recursive" variant of tangle learning

Tangle learning (2/2)

def `tanglelearning(`$\Game$`)`**:**

  $W_\Diamond \leftarrow \emptyset$, $\sigma_\Diamond \leftarrow \emptyset$, $W_\Box \leftarrow \emptyset$, $\sigma_\Box \leftarrow \emptyset$, $T \leftarrow \emptyset$

  while $\Game \neq \emptyset$ **:**

    $Y \leftarrow$ `search(`$\Game$`,`T`)`

    $T \leftarrow T \cup \{ t \in Y \mid E_T(t) \neq \emptyset \}$

    $D \leftarrow \{ t \in Y \mid E_T(t) = \emptyset \}$

    if $D \neq \emptyset$ **:**

      $D_\Diamond^+, \sigma \leftarrow \mathit{TAttr}_\Diamond^{\Game,T}(\bigcup D_\Diamond)$

      $W_\Diamond \leftarrow W_\Diamond \cup D_\Diamond^+$, $\sigma_\Diamond \leftarrow \sigma_\Diamond \cup \sigma$

      $D_\Box^+, \sigma \leftarrow \mathit{TAttr}_\Box^{\Game,T}(\bigcup D_\Box)$

      $W_\Box \leftarrow W_\Box \cup D_\Box^+$, $\sigma_\Box \leftarrow \sigma_\Box \cup \sigma$

      $\Game \leftarrow \Game \setminus (D_\Diamond^+ \cup D_\Box^+)$

      $T \leftarrow T \cap \Game$

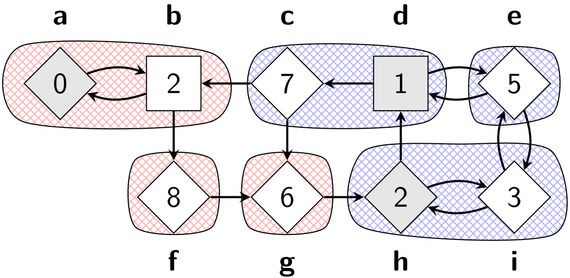
  return $W_\Diamond, W_\Box, \sigma_\Diamond, \sigma_\Box$

Tangle learning

Computing strategy

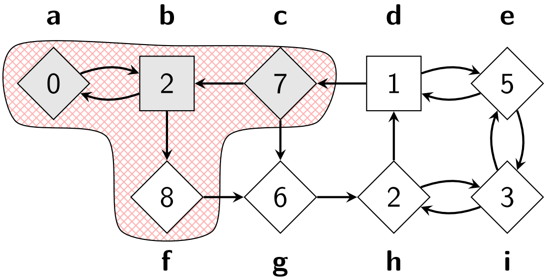
- Similar to priority promotion: compute strategy with the attractor, select any successor in the region for the highest priority vertices of α
- Store the σ of every tangle and use the stored σ as the strategy for α when attracting a tangle

Tangle learning



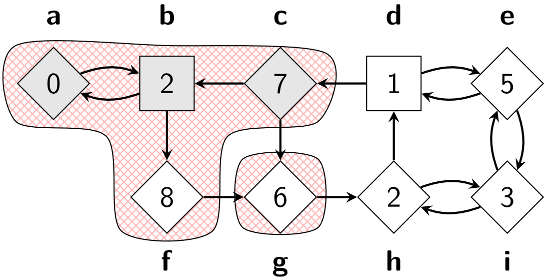
- After first partition into α-maximal regions.
- Regions 2 and 3 are closed (in their subgame).
- Tangle $\{\,\mathbf{a},\mathbf{b}\,\}$ attracted to 8.
- Tangle $\{\,\mathbf{h},\mathbf{i}\,\}$ attracted to 5.

Tangle learning

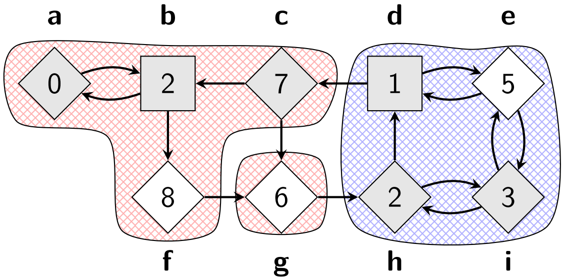


- Tangles: $\{\mathbf{a},\mathbf{b}\}$ (2) and $\{\mathbf{h},\mathbf{i}\}$ (3).
- After tangle attractor to 8...

Tangle learning

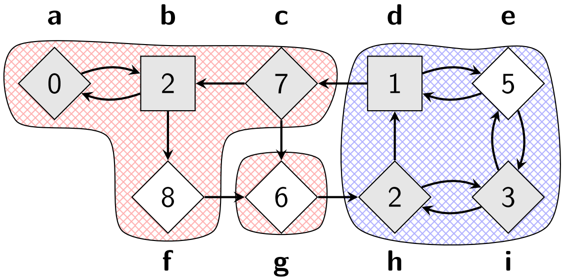


- Tangles: $\{\,$**a**$,$**b**$\,\}$ (2) and $\{\,$**h**$,$**i**$\,\}$ (3).
- After tangle attractor to 6...



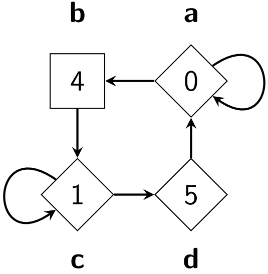
- Tangles: $\{\mathbf{a},\mathbf{b}\}$ (2) and $\{\mathbf{h},\mathbf{i}\}$ (3).
- After tangle attractor to 5...

Tangle learning



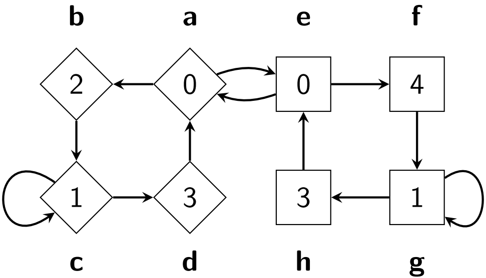
- Only closed region: 5.
- One tangle, which is also a dominion.

Distractions



- Vertex **b** is a distraction for player Even.
- Learn opponent tangles to attract the distractions.
- Tangle $\{\mathbf{c}\}$ is attracted to region 5.
- Now vertex **a** is not distracted by vertex **b**.

Distractions



- First round: tangle $\{\mathbf{c}\}$ (attracts distraction **b**).
- Second round: tangle $\{\mathbf{a}, \mathbf{e}\}$ (attracts distraction **h**).
- Third round: tangle $\{\mathbf{g}\}$ (dominion).

Outline

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:

$$\forall v \in V : \rho(v) = \begin{cases} \max_{\sqsubset}\{\texttt{Prog}(\rho(w), \mathsf{pr}(v)) \mid w \in E(v)\} & v \in V_{\diamond} \\ \min_{\sqsubset}\{\texttt{Prog}(\rho(w), \mathsf{pr}(v)) \mid w \in E(v)\} & v \in V_{\Box} \end{cases}$$

*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:
  - $W_\Diamond = \{v \mid \rho(v) = \top\}$, $W_\Box = \{v \mid \rho(v) \neq \top\}$
  - if $v \in W_\Box$, 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

Small progress measures

Even measures

- Measures are tuples $\langle e_6, e_4, e_2, e_0 \rangle$ (with highest even priority 6)
- Each $e_p = [0..n_p]$ with n_p the number of vertices with priority p
- Example: $\mathbb{M} = (\{0\} \times \{0,1,2\} \times \{0\} \times \{0,1\}) \cup \{\top\}$
- A total order $\sqsubset$ which is lexicographic: $m_1 \sqsubset m_2$ iff there is a highest unequal priority z and $m_1(z) < m_2(z)$ (and $\top = \top$)

$$\begin{array}{rcl} \langle 1,0,0,0 \rangle & \sqsubset & \langle 1,0,0,1 \rangle \\ \langle 4,2,10,5 \rangle & \sqsubset & \langle 4,3,0,0 \rangle \end{array}$$

Odd measures

- Same, but with the odd priorities

Small progress measures

Even measures

- Measures are tuples $\langle e_6, e_4, e_2, e_0 \rangle$ (with highest even priority 6)
- A p-truncation keeps only elements $\geq p$:

$$\begin{array}{rcl} \langle 1,2,3,2 \rangle|_1 & = & \langle 1,2,3 \rangle \\ \langle 1,2,3,2 \rangle|_4 & = & \langle 1,2 \rangle \\ \langle 1,2,3,2 \rangle|_7 & = & \varepsilon \end{array}$$

- Notation: $m_1 \sqsupset_p m_2 \equiv m_1|_p \sqsupset m_2|_p$
- An edge $v \to u$ is progressive if $\rho(v) \sqsupseteq_{\mathsf{pr}(v)} \rho(u)$ if v is odd and $\rho(v) \sqsupset_{\mathsf{pr}(v)} \rho(u)$ if v is even
- ρ is a progress measure if:
  - for every vertex of Even, some outgoing edge is progressive in ρ
  - for every vertex of Odd, every outgoing edge is progressive in ρ

(remember: we are interested in the least progress measure)

The `Prog` function

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

$$\mathsf{Prog}(m,p) := \begin{cases} \min\{\, m' \in \mathbb{M} \mid m' \sqsupset_p m \,\} & p \text{ is even} \\ \min\{\, m' \in \mathbb{M} \mid m' \sqsupseteq_p m \,\} & p \text{ is odd} \end{cases}$$

Example (with highest value 3 for all elements):

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

Small progress measures

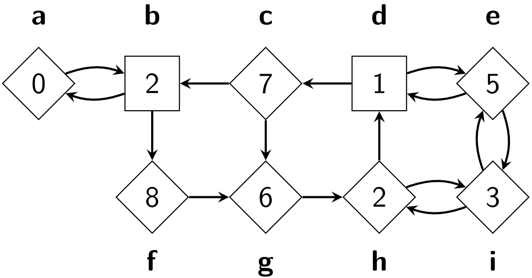
Operational interpretation [Gazda, Willemse, 2015]

- p-dominated stretches: how often priority p is encountered before a higher priority
- Example: play $\underline{00}10\underline{2}1\underline{2}0\underline{2}321\underline{4}2\underline{6}5\underline{6}201$ 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
- If priority p is seen more than n_p times, there must be a cycle!

Operational interpretation [Van Dijk, 2018]

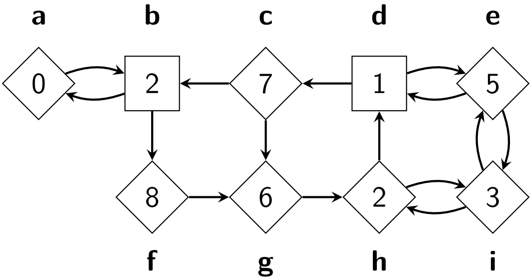
- *Notice the "overflow mechanism"!*
  If priority p overflows, our optimal path contains a cycle of priority p.
  Keep increasing the measure until the opponent "escapes"
  (Compare to priority promotion / tangles!)

Small progress measures



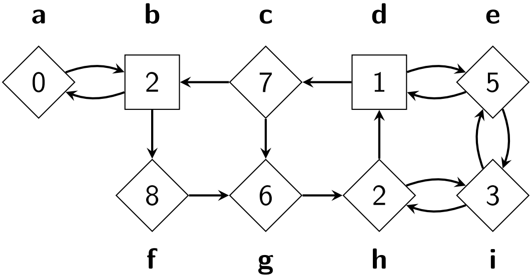
a	$\langle 0,0,0,0,0\rangle$	to	$\langle 0,0,0,0,1\rangle$	0
b	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	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



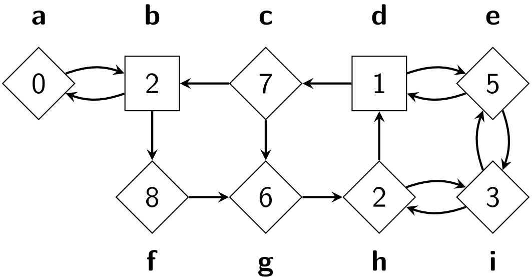
a	$\langle 0,0,0,0,1\rangle$	to	$\langle 0,0,0,1,1\rangle$	02
b	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,1,-\rangle$	2
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	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



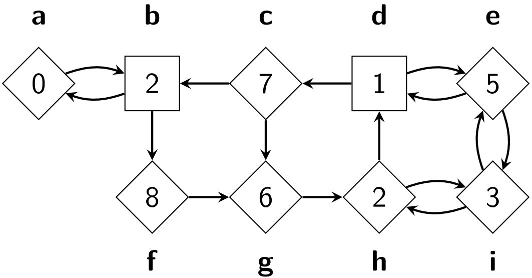
a	$\langle 0,0,0,1,1\rangle$	to	$\langle 0,0,0,1,1\rangle$	02
b	$\langle 0,0,0,1,-\rangle$	to	$\langle 0,0,0,2,-\rangle$	202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	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



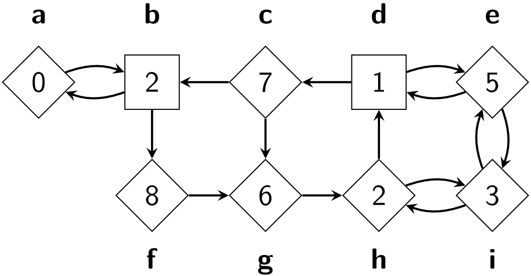
a	$\langle 0,0,0,1,1\rangle$	to	$\langle 0,0,0,2,1\rangle$	0202
b	$\langle 0,0,0,2,-\rangle$	to	$\langle 0,0,0,2,-\rangle$	202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	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



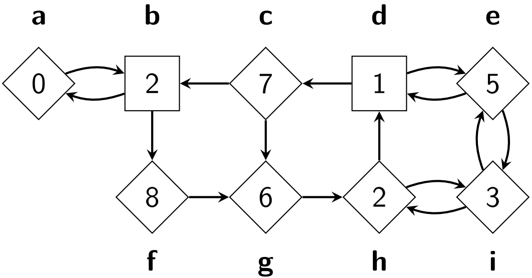
a	$\langle 0,0,0,2,1\rangle$	to	$\langle 0,0,0,2,1\rangle$	0202
b	$\langle 0,0,0,2,-\rangle$	to	$\langle 0,1,0,0,-\rangle$	20202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	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



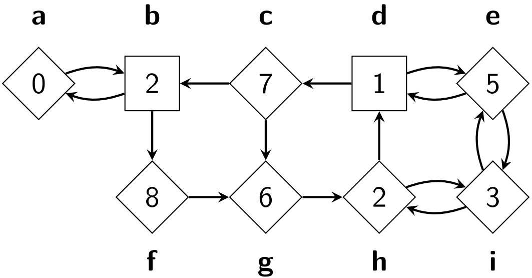
a	$\langle 0,0,0,2,1\rangle$	to	$\langle 0,1,0,0,1\rangle$	020202
b	$\langle 0,1,0,0,-\rangle$	to	$\langle 0,1,0,0,-\rangle$	20202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	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



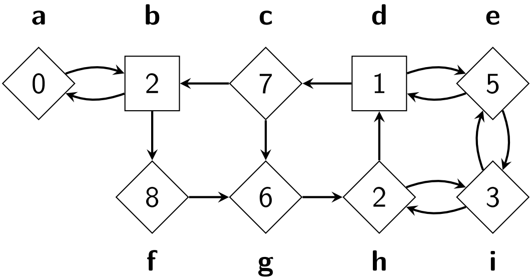
a	$\langle 0,1,0,0,1\rangle$	to	$\langle 0,1,0,0,1\rangle$	020202
b	$\langle 0,1,0,0,-\rangle$	to	$\langle 0,1,0,1,-\rangle$	2020202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	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



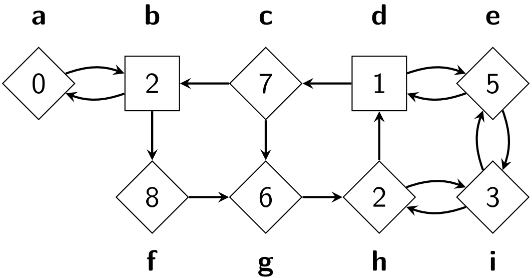
a	$\langle 0,1,0,0,1 \rangle$	to	$\color{blue}\langle 0,1,0,1,1 \rangle$	02020202
b	$\langle 0,1,0,1,- \rangle$	to	$\langle 0,1,0,1,- \rangle$	2020202
c	$\langle 0,-,-,-,- \rangle$	to	$\langle 0,-,-,-,- \rangle$	7
d	$\langle 0,0,0,0,- \rangle$	to	$\langle 0,0,0,0,- \rangle$	1
e	$\langle 0,0,-,-,- \rangle$	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



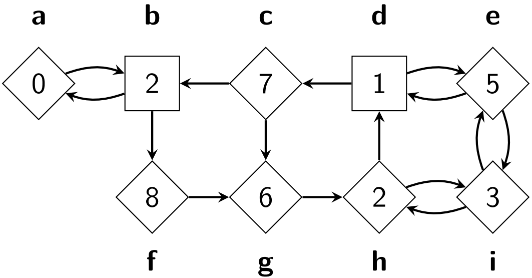
a	$\langle 0,1,0,1,1\rangle$	to	$\langle 0,1,0,1,1\rangle$	02020202
b	$\langle 0,1,0,1,-\rangle$	to	$\langle 0,1,0,2,-\rangle$	202020202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	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



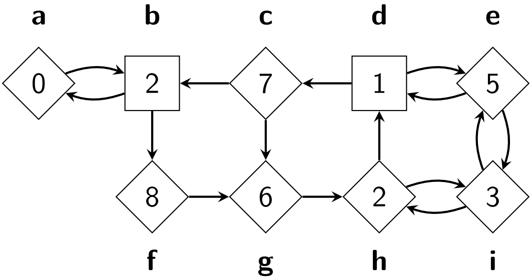
a	$\langle 0,1,0,1,1\rangle$	to	$\langle 0,1,0,2,1\rangle$	0202020202
b	$\langle 0,1,0,2,-\rangle$	to	$\langle 0,1,0,2,-\rangle$	20202020
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	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



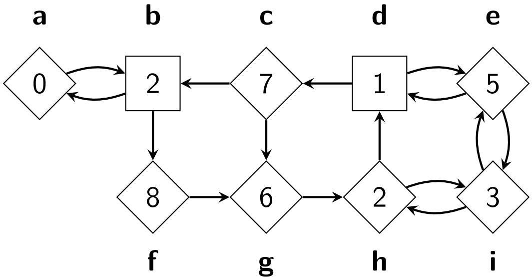
a	$\langle 0,1,0,2,1\rangle$	to	$\langle 0,1,0,2,1\rangle$	0202020202
b	$\langle 0,1,0,2,-\rangle$	to	$\langle 1,0,0,0,-\rangle$	20202020202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 0,-,-,-,-\rangle$	7
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	to	$\langle 0,0,-,-,-\rangle$	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



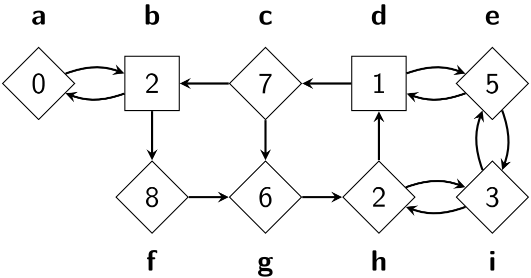
a	$\langle 0,1,0,2,1\rangle$	to	$\langle 1,0,0,0,1\rangle$	020202020202
b	$\langle 1,0,0,0,-\rangle$	to	$\langle 1,0,0,0,-\rangle$	20202020202
c	$\langle 0,-,-,-,-\rangle$	to	$\langle 1,-,-,-,-\rangle$	720202020202
d	$\langle 0,0,0,0,-\rangle$	to	$\langle 0,0,0,0,-\rangle$	1
e	$\langle 0,0,-,-,-\rangle$	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



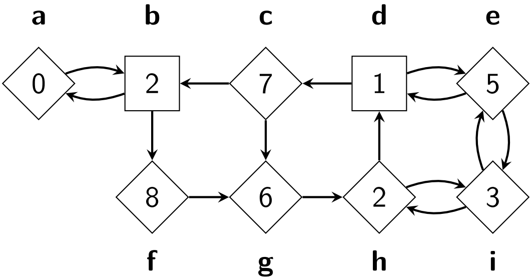
a	$\langle 1,0,0,0,1 \rangle$	to	$\langle 1,0,0,0,1 \rangle$	020202020202
b	$\langle 1,0,0,0,- \rangle$	to	$\langle 1,0,0,1,- \rangle$	28
c	$\langle 1,-,-,-,- \rangle$	to	$\langle 1,-,-,-,- \rangle$	728
d	$\langle 0,0,0,0,- \rangle$	to	$\langle 0,0,0,0,- \rangle$	1
e	$\langle 0,0,-,-,- \rangle$	to	$\langle 0,0,-,-,- \rangle$	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	$\langle 1, 0, 0, 1, 1 \rangle$	**b**
b	$\langle 1, 0, 0, 1, - \rangle$	**f**
c	$\langle 1, -, -, -, - \rangle$	**b**
d	$\langle 0, 0, 0, 0, - \rangle$	**e**
e	$\langle 0, 0, -, -, - \rangle$	**d**/**i**
f	$\langle 1, -, -, -, - \rangle$	**g**
g	$\langle 0, 1, -, -, - \rangle$	**h**
h	$\langle 0, 0, 0, 1, - \rangle$	**d**/**i**
i	$\langle 0, 0, 0, -, - \rangle$	**h**/**e**

- All vertices are won by Odd
  No vertices are lifted to $\top$
- Strategy for Odd
  - from **b** to **f**
  - from **d** to **e**

def spm($\circlearrowleft$)**:**

    $\rho \leftarrow V \mapsto \langle 0, \ldots, 0 \rangle$

    while $\rho(v) \sqsubset \mathsf{Lift}(\rho, v)$ for some v **:** $\rho \leftarrow \rho[v \mapsto \mathsf{Lift}(\rho, v)]$

    $W_\Diamond \leftarrow \{v \mid \rho(v) = \top\}$

    $W_\Box \leftarrow \{v \mid \rho(v) \neq \top\}$

    $\sigma_\Box \leftarrow (v \in W_\Box \cap V_\Box) \mapsto \mathsf{pick}(\{u \in E(v) \mid \rho(v) = \mathsf{Prog}(\rho(w), \mathsf{pr}(v))\})$

    return $W_\Diamond, W_\Box$

$$\mathsf{Lift}(\rho, v) \quad := \quad \begin{cases} \max_\sqsubset \{\mathsf{Prog}(\rho(w), \mathsf{pr}(v)) \mid w \in E(v)\} & v \in V_\Diamond \\ \min_\sqsubset \{\mathsf{Prog}(\rho(w), \mathsf{pr}(v)) \mid w \in E(v)\} & v \in V_\Box \end{cases}$$

$$\mathsf{Prog}(m, p) \quad := \quad \begin{cases} \min\{\, m' \in \mathbb{M} \mid m' \sqsupset_p m \,\} & p \text{ is even} \\ \min\{\, m' \in \mathbb{M} \mid m' \sqsupseteq_p m \,\} & p \text{ is odd} \end{cases}$$

Small progress measures

def spm($\Game$):

  $\rho \leftarrow V \mapsto \langle 0, \ldots, 0 \rangle$

  $Z \leftarrow V$ // use a queue or a stack

  while $Z \neq \emptyset$ **:**

    $v \leftarrow \mathsf{pick}(Z)$

    $Z \leftarrow Z \setminus \{v\}$

    if $\rho(v) \sqsubset \mathsf{Lift}(\rho, v)$ **:**

      $\rho \leftarrow \rho[v \mapsto \mathsf{Lift}(\rho, v)]$

      $Z \leftarrow Z \cup E^{-1}(v)$

  $W_\Diamond \leftarrow \{v \mid \rho(v) = \top\}$

  $W_\Box \leftarrow \{v \mid \rho(v) \neq \top\}$

  $\sigma_\Box \leftarrow (v \in W_\Box \cap V_\Box) \mapsto \mathsf{pick}(\{u \in E(v) \mid \rho(v) = \mathsf{Prog}(\rho(w), \mathsf{pr}(v))\})$

  return $W_\Diamond, W_\Box$

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!)
- When lifting an **even priority vertex** to $\top$, decrease n_p by 1
- Also compute odd measures (strategy for Even)
- **Advanced technique**: occasionally, compute the attractor to vertices in Z, any vertex not attracted and not $\top$ is won by the other player!
- Preprocessing: use **compression** and **SCC-decomposition** and **self-loop solving**.

Measures as tree navigation paths

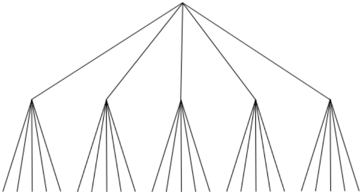
Core idea

- A tuple $\langle 4, 2, 3 \rangle$ can be a *navigation path* of a tree
- Follow branch 4, then branch 2, then branch 3
- Then:
  - 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
  - the exact numbers (4, 2, 3) are not important!
  - what matters is the order
- Example: (1,2), (0,2), (1,1), (1,2), (2,1), (0,1), (1,0)
  - Draw tree corresponding to this set of navigation paths
  - Notice how the labels of the tree are irrelevant, only the order matters

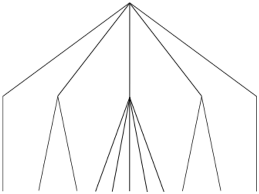
Measures as tree navigation paths

Universal trees (see explanation by Fijalkow 2018)

A (n,h)-universal tree is a tree that can embed all trees of height h and with n leaves.



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

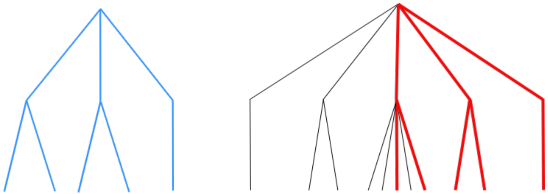


A (5,2)-universal tree of size 11

Measures as tree navigation paths

Universal trees

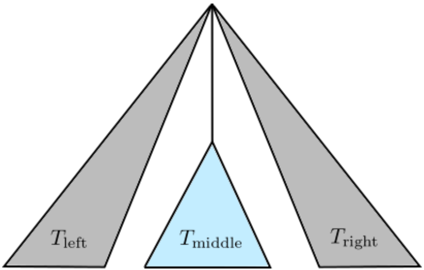
A (n,h)-universal tree is a tree that can embed all trees with height h and n leaves.



The tree on the left is embedded into the universal tree

Measures as tree navigation paths

Universal trees

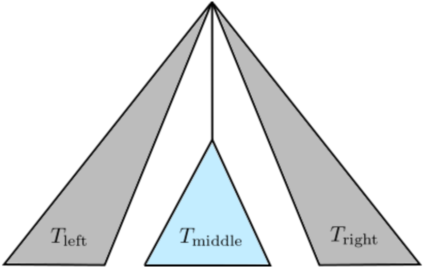


Simple algorithm:

- Split tree in three parts: Left, Middle, Right
- Such that $|Left| < n/2$ and $|Right| < n/2$
- Repeat left/right to obtain all branches, and repeat recursively...

Measures as tree navigation paths

Universal trees



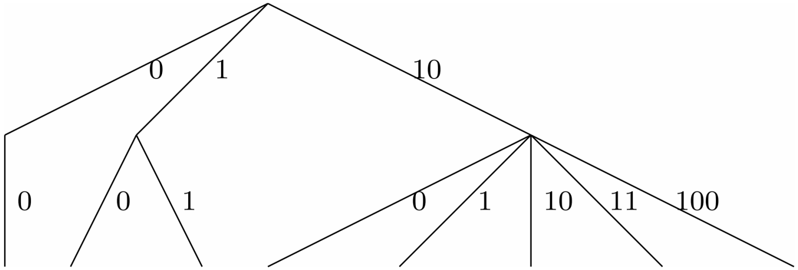
Tree encoding:

- Instead of $\langle 4, 2, 3 \rangle$, encode as a tuple of bitstrings
- For example $\langle 100, 010, 011 \rangle$

Universal trees

Tree encoding:

- Instead of $\langle 4, 2, 3 \rangle$, encode as a tuple of bitstrings
- For example $\langle 100, 10, 11 \rangle$



Measures as tree navigation paths

Universal trees

Succinct tree encoding:

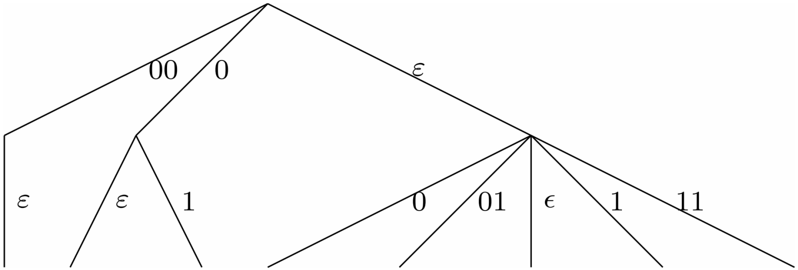
- Encode as a tuple of bitstrings (empty allowed)
- Order on bits: $0 \sqsubset \varepsilon \sqsubset 1$
- Order on bitstrings: 0s $\sqsubset$ s $\sqsubset$ 1s
  Example: $00 \sqsubset 0, 0 \sqsubset 01, 1 \sqsubset 10$
- Order on tuples: lexicographic, and *shorter prefix* is lower
  Example: $\langle 01, \varepsilon \rangle \sqsubset \langle 01, \varepsilon, 00 \rangle$, but $\langle 01, \varepsilon, 000 \rangle \sqsubset \langle 1000, \varepsilon \rangle$

Measures as tree navigation paths

Universal trees

Succinct tree encoding:

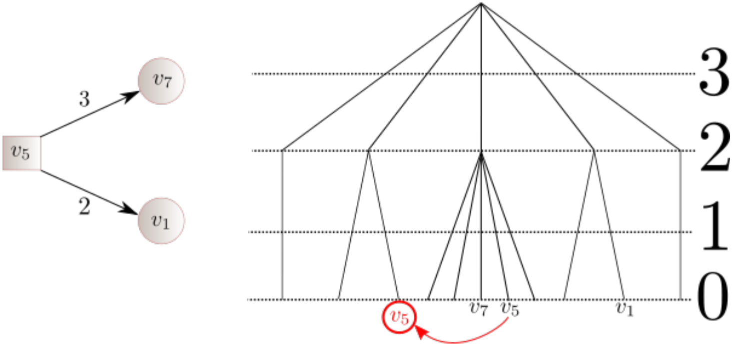
- Prefix Left with 0, Right with 1, Middle with ε.
- For example $\langle 100, 10, 11 \rangle$



- Maximum bitstring length: 2 bits

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 $v_5 \triangleleft_3 v_7$ and $v_5 \triangleleft_2 v_1$

Succinct progress measures

Implementation notes

- Implementation is complicated.
- Core idea is the same: keep lifting vertices to the smallest higher measure, either the maximum (player Even) or the minimum (player Odd)

"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: $1\underline{2}1314\!\!\!\!\underline{}\,23\underline{2}15\underline{6}3\underline{2}1\underline{2}$
- _ means "no such witness"
- 7 means a witness, but starting with odd 7
- 6 means a witness, startng with even 6

"Ordered" progress measures

Update rules

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

Problem

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

“Ordered” progress measures

Implementation notes

- See paper by Fearnley et al on arXiv
- See `qpt.cpp` in Oink
- It's complicated...

Winner-controller winning cycles

Simple algorithm to find trivial winning regions

Algorithm

- For every vertex v that is controlled by player $\alpha := \mathsf{pr}(v) \bmod 2$
- $Z, \sigma :=$ attract vertices in $\{u \in V_\alpha \mid \mathsf{pr}(u) \leq \mathsf{pr}(v)\}$ to v
  - Just backward DFS from v via α-vertices with $\leq$ priority
- If Z is closed (v is reached), then Z is an α-dominion with strategy σ; maximize Z by attracting from the entire game to Z and remove from the game

There are more optimal algorithms, employing SCC reductions, etc. See also Maks Verver's MSc Thesis "Practical Improvements to Parity Game Solving" and fatal attractors of [Huth, Kwo, Piterman, 2014]

Strategy iteration

Strategy improvement/iteration overview

- Originates from policy iteration algorithms for Markov decision processes and similar algorithms for stochastic games.
- First parity game specific algorithm by Vöge and Jurdzinski in 2000
- Later numerous modified versions
  - better "best response" computation
  - smarter strategy selection heuristics (hoping to find one requiring polynomially many changes)
  - learning snares (kind of tangles): Fearnley 2011
- Suitable for parallel computation (e.g. van de Pol and Weber; Kandziora (2009) and Van de Berg (2010) on the Playstation 3; various GPU and multi-core implementations)

Strategy iteration

Core idea of strategy iteration

- Both players have a total strategy
  - strategy σ for all $v \in V_\Diamond$
  - strategy τ for all $v \in V_\Box$
- These induce a single play π for every $v \in V$
- Every play π ends in a cycle
- Play profile $\rho\colon V \to \mathbb{M}$ assigns a value to v based on π
- The value represents how optimal are current strategies σ and τ?
- Keep improving strategies until fixed point
  - Odd computes the best response to σ
  - Even uses ρ to improve the strategy σ <u>once</u>
  - Repeat
- Why improve against the best response? Because then each time you improve σ, you know that Odd could not find a better response

Strategy iteration

Algorithm

1. Start with some σ for player 0
2. Compute the best response τ for player 1
   - Traditional approach: Bellman-Ford shortest path algorithm
   - [Fearnley 2017] proposes: use strategy iteration to compute τ:
     1. Start with some τ for player 1 (e.g. previous τ)
     2. Compute play profiles and switchable edges
     3. Select switchable edges for the next τ
     4. Repeat until no more switchable edges
3. Compute the play profiles and the switchable edges (that would locally improve the valuation) for player 0
4. Select switchable edges for the next σ
   - Different proposed switching rules (can we do it in P iterations)
5. Repeat from step 2 until no more switchable edges

Strategy iteration

Play profiles

- Relevance order $<$ (value is priority):
  - $u < v \quad \Leftrightarrow \quad \mathsf{pr}(u) < \mathsf{pr}(v)$
  - $\max_<(V) =$ highest priority vertex
- Reward order $\prec$ (value as seen from player 0):
  - $V_+ = \{v \mid \mathsf{pr}(v) \text{ is even}\} \qquad V_- = \{v \mid \mathsf{pr}(v) \text{ is odd}\}$
  - $u \prec v \quad \Leftrightarrow \quad (u < v \wedge v \in V_+) \vee (v < u \wedge u \in V_-)$
  - $P \prec Q \quad \Leftrightarrow \quad P \neq Q \wedge max_<(P \triangle Q) \in (Q \triangle V_-)$
    - highest vertex in symmetric difference is in Q and even
    - highest vertex in symmetric difference is in P and odd
- Reward order $\prec$ (alternative formulation)
  - $\mathsf{rew}(v) := \mathsf{pr}(v) \times (-1)^{\mathsf{pr}(v)}$ (that is: negate if $\mathsf{pr}(v)$ is odd)
  - $u \prec v \quad \Leftrightarrow \quad \mathsf{rew}(u) < \mathsf{rew}(v)$

Strategy iteration

Play profiles [VJ00]

- Relevance order $<$ and reward order $\prec$
- Original play profile of [Vöge, Jurdzinski 2000]: tuple $\langle u, P, e \rangle$
  - u_π is most relevant vertex in the loop of π: $u_\pi = max_<(\inf(\pi))$
  - P_π is the set of vertices more relevant than u_π in π (seen once in the prefix of u_π)
  - e_π is the number of vertices in π before u_π
- $\langle u, P, e \rangle \prec \langle v, Q, f \rangle \quad \Leftrightarrow \quad \begin{cases} u \prec v \quad \vee \\ (u = v \wedge P \prec Q) \quad \vee \\ (u = v \wedge P = Q \wedge v \in V_- \wedge e < f) \quad \vee \\ (u = v \wedge P = Q \wedge v \in V_+ \wedge e > f) \end{cases}$
- A strategy is optimal in vertex v if it selects the $\prec$-maximal successor in $E(v)$ for player 0 (or $\prec$-minimal for player 1)
- A strategy is optimal if it is optimal for all vertices

Strategy iteration

Play profiles [F17]

- Modify σ: now player Even is also allowed to *halt* the play (if the continuation is not favorable)
- Initially σ is $\perp$ (halt) for all Even's vertices
- Result: now every infinite play (cycle) is won by Even!
  - because otherwise Even would halt to avoid the losing cycle
  - except if Odd can win a cycle without any vertices of Even
- Requires preprocessing: remove winner-controlled winning cycles of Odd
  - or maybe: let Even force Odd vertices to halt instead...
- Play profile: $\top$ if π is infinite; otherwise $\langle e_d, e_{d-1}, \ldots, e_1, e_0 \rangle$ with $e_p = |\{v \in \pi \mid \mathsf{pr}(v) = p\}|$, i.e., count how often each priority p is encountered in the finite path π
- Profile $X \prec Y$ if the highest different priority p is either even and $X(p) < Y(p)$ or odd and $X(p) > Y(p)$; also $X \prec \top$ for all $X \neq \top$

Strategy iteration

Compute using a backward search from vertices where Even halts

def compute-valuations(⅁, σ, τ)**:**
  $\theta \leftarrow \sigma \cup \tau$ // for easier notation
  $Z \leftarrow \theta^{-1}(\perp)$ // where Even halts
  $\rho \leftarrow (V \mapsto \top)$ // initialize
  while $Z \neq \emptyset$ **:**
    $v \leftarrow \mathsf{pop}(Z)$ // pop any v from Z
    $m \leftarrow \begin{cases} \langle 0, \ldots, 0 \rangle & \theta(v) = \perp \\ \rho(\theta(v)) & \text{otherwise} \end{cases}$ // get successor profile
    $m(\mathsf{pr}(v)) \leftarrow m(\mathsf{pr}(v)) + 1$ // update profile
    $\rho(v) \leftarrow m$ // set profile of v
    $Z \leftarrow Z \cup \theta^{-1}(v)$ // add predecessors to Z
  return ρ

Implementation note

Two stages: first compute θ^{-1}, then do the backward search

Switching rule Greedy All Switches

Extend ρ with a valuation of $\bot$; define ρ over sets; define Best_α as the set of successors of v with the optimal profile for player α; define $\mathsf{GreedyAll}_\alpha$ to update the strategy with a switchable edge (if current strategy is not optimal)

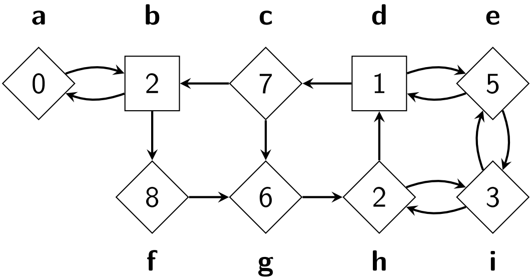
$$\begin{aligned}
\rho_\bot &:= \rho \cup \{\bot \mapsto \langle 0, \ldots, 0\rangle\} \\
\rho_\bot(X) &:= \{\rho_\bot(x) \mid x \in X\} \\
\mathsf{Best}_\Box(\supset, \rho, v) &:= \{u \in E(v) \mid \rho(u) = \min\nolimits_\prec \rho(E(v))\} \\
\mathsf{Best}_\Diamond(\supset, \rho, v) &:= \{u \in E(v) \cup \{\bot\} \mid \rho(u) = \max\nolimits_\prec \rho(E(v) \cup \{\bot\})\} \\
\mathsf{GreedyAll}_\alpha(\supset, \sigma, \rho) &:= V_\alpha \mapsto \begin{cases} \sigma(v) & \sigma(v) \in \mathsf{Best}_\alpha(\supset, \rho, v) \\ \mathsf{pick}(\mathsf{Best}_\alpha(\supset, \rho, v)) & \text{otherwise} \end{cases}
\end{aligned}$$

```
1  def si(⅁):
2      σ ← (V◇ ↦ ⊥), τ ← random strategy for Odd
3      repeat
4          repeat
5              ρ ← compute-valuations(⅁, σ, τ)
6              τ ← GreedyAll□(⅁, τ, ρ)
7          until τ is unchanged
8          σ ← GreedyAll◇(⅁, σ, ρ)
9      until σ is unchanged
10     return W◇, W□, σ, τ where W◇ ← {v | ρ(v) = ⊤}, W□ ← V \ W◇
```

Implementation note

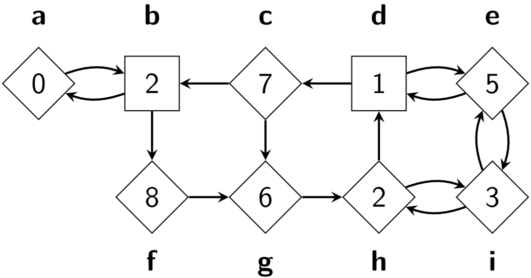
After line 7, any vertex v with $\rho(v) = \top$, can be added to $W_\Diamond$ already and does not need to be improved anymore; any vertex remaining in the end is then won by Odd

Strategy iteration



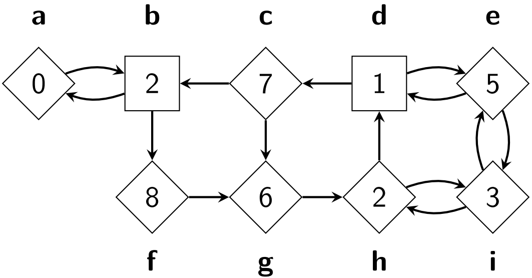
σ: **a** $\rightarrow \perp$, **c** $\rightarrow \perp$, **e** $\rightarrow \perp$, **f** $\rightarrow \perp$, **g** $\rightarrow \perp$, **h** $\rightarrow \perp$, **i** $\rightarrow \perp$

Strategy iteration



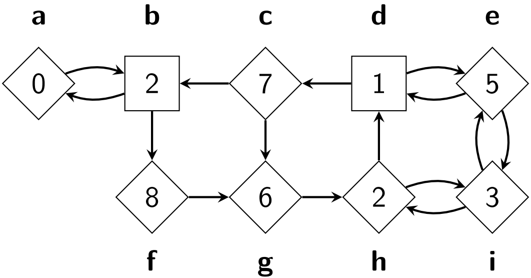
σ: **a** $\rightarrow \perp$, **c** $\rightarrow \perp$, **e** $\rightarrow \perp$, **f** $\rightarrow \perp$, **g** $\rightarrow \perp$, **h** $\rightarrow \perp$, **i** $\rightarrow \perp$
best response τ: **b** $\rightarrow$ **a** and **d** $\rightarrow$ **c**

Strategy iteration



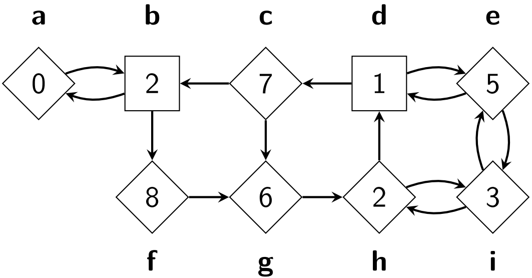
σ: $\mathbf{a} \to \bot$, $\mathbf{c} \to \bot$, $\mathbf{e} \to \bot$, $\mathbf{f} \to \bot$, $\mathbf{g} \to \bot$, $\mathbf{h} \to \bot$, $\mathbf{i} \to \bot$
best response τ: $\mathbf{b} \to \mathbf{a}$ and $\mathbf{d} \to \mathbf{c}$
σ: $\boldsymbol{a} \to \boldsymbol{b}$, $\boldsymbol{c} \to \boldsymbol{g}$, $\mathbf{e} \to \bot$, $\boldsymbol{f} \to \boldsymbol{g}$, $\boldsymbol{g} \to \boldsymbol{h}$, $\mathbf{h} \to \bot$, $\mathbf{i} \to \bot$

Strategy iteration



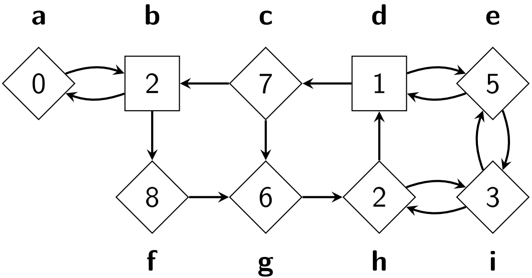
σ: $\mathbf{a} \to \bot$, $\mathbf{c} \to \bot$, $\mathbf{e} \to \bot$, $\mathbf{f} \to \bot$, $\mathbf{g} \to \bot$, $\mathbf{h} \to \bot$, $\mathbf{i} \to \bot$
best response τ: $\mathbf{b} \to \mathbf{a}$ and $\mathbf{d} \to \mathbf{c}$
σ: $\boldsymbol{a} \to \boldsymbol{b}$, $\boldsymbol{c} \to \boldsymbol{g}$, $\mathbf{e} \to \bot$, $\boldsymbol{f} \to \boldsymbol{g}$, $\boldsymbol{g} \to \boldsymbol{h}$, $\mathbf{h} \to \bot$, $\mathbf{i} \to \bot$
best response τ: $\boldsymbol{b} \to \boldsymbol{f}$ and $\mathbf{d} \to \mathbf{c}$

Strategy iteration



σ: **a** $\to \bot$, **c** $\to \bot$, **e** $\to \bot$, **f** $\to \bot$, **g** $\to \bot$, **h** $\to \bot$, **i** $\to \bot$
best response τ: **b** $\to$ **a** and **d** $\to$ **c**
σ: ***a*** $\to$ ***b***, ***c*** $\to$ ***g***, **e** $\to \bot$, ***f*** $\to$ ***g***, ***g*** $\to$ ***h***, **h** $\to \bot$, **i** $\to \bot$
best response τ: ***b*** $\to$ ***f*** and **d** $\to$ **c**
σ: **a** $\to$ **b**, ***c*** $\to$ ***b***, **e** $\to \bot$, **f** $\to$ **g**, **g** $\to$ **h**, **h** $\to \bot$, **i** $\to \bot$

Strategy iteration



σ: $\mathbf{a} \to \perp$, $\mathbf{c} \to \perp$, $\mathbf{e} \to \perp$, $\mathbf{f} \to \perp$, $\mathbf{g} \to \perp$, $\mathbf{h} \to \perp$, $\mathbf{i} \to \perp$

best response τ: $\mathbf{b} \to \mathbf{a}$ and $\mathbf{d} \to \mathbf{c}$

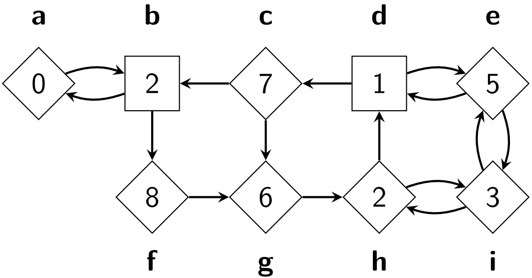
σ: $\boldsymbol{a} \to \boldsymbol{b}$, $\boldsymbol{c} \to \boldsymbol{g}$, $\mathbf{e} \to \perp$, $\boldsymbol{f} \to \boldsymbol{g}$, $\boldsymbol{g} \to \boldsymbol{h}$, $\mathbf{h} \to \perp$, $\mathbf{i} \to \perp$

best response τ: $\boldsymbol{b} \to \boldsymbol{f}$ and $\mathbf{d} \to \mathbf{c}$

σ: $\mathbf{a} \to \mathbf{b}$, $\boldsymbol{c} \to \boldsymbol{b}$, $\mathbf{e} \to \perp$, $\mathbf{f} \to \mathbf{g}$, $\mathbf{g} \to \mathbf{h}$, $\mathbf{h} \to \perp$, $\mathbf{i} \to \perp$

best response τ: $\mathbf{b} \to \mathbf{f}$ and $\boldsymbol{d} \to \boldsymbol{e}$

Strategy iteration



σ: **a** $\to \bot$, **c** $\to \bot$, **e** $\to \bot$, **f** $\to \bot$, **g** $\to \bot$, **h** $\to \bot$, **i** $\to \bot$
best response τ: **b** $\to$ **a** and **d** $\to$ **c**
σ: ***a*** $\to$ ***b***, ***c*** $\to$ ***g***, **e** $\to \bot$, ***f*** $\to$ ***g***, ***g*** $\to$ ***h***, **h** $\to \bot$, **i** $\to \bot$
best response τ: ***b*** $\to$ ***f*** and **d** $\to$ **c**
σ: **a** $\to$ **b**, ***c*** $\to$ ***b***, **e** $\to \bot$, **f** $\to$ **g**, **g** $\to$ **h**, **h** $\to \bot$, **i** $\to \bot$
best response τ: **b** $\to$ **f** and ***d*** $\to$ ***e***
Odd wins entire game with strategy τ

Fixed point iteration

Core idea

- We can solve μ-calculus model checking by solving the fixed points explicitly
- We can solve μ-calculus model checking by solving a parity game
- Here: we solve parity games by via a fixed point iteration
- Via weak alternating automata [Kupfermann, Vardi, 1998]
- `APT` implementation [Di Stasio, Murano, Perelli, Vardi, 2016]
- Via μ-calculus: [Bruse, Falk, Lange, 2014]
- Quite fast for games with low number of priorities

Fixed point iteration

Core idea

- "Using fixed points, update winning regions using a 1-step attractor"
- Record "distraction sets" $Z_p \subseteq V_p$ $\quad (V_p = \{v \mid \mathsf{pr}(v) = p\})$
- A vertex is a distraction if:
  - it has even priority and is won by Odd
  - it has odd priority and is won by Even
- Monotonically update Z_0, then Z_1, etc.
- When adding vertices to Z_p, reset $Z_{<p}$ to $\emptyset$

Fixed point iteration

Given some set of distracted vertices $Z = Z_0 \cup Z_1 \cup \cdots \cup Z_d$,

$$\mathsf{winner}(v, Z) \quad := \quad \begin{cases} \mathsf{pr}(v) \bmod 2 & v \notin Z \\ 1 - (\mathsf{pr}(v) \bmod 2) & v \in Z \end{cases}$$

$$\mathsf{next}(v, Z) \quad := \quad \begin{cases} 0 & v \in V_\Diamond \wedge \exists u \in E(v) : \mathsf{winner}(u, Z) = 0 \\ 1 & v \in V_\Diamond \wedge \forall u \in E(v) : \mathsf{winner}(u, Z) = 1 \\ 1 & v \in V_\Box \wedge \exists u \in E(v) : \mathsf{winner}(u, Z) = 1 \\ 0 & v \in V_\Box \wedge \forall u \in E(v) : \mathsf{winner}(u, Z) = 0 \end{cases}$$

Fixed point iteration

def fpi(⅁):

  $p \leftarrow 0$ // start with lowest priority

  $Z \leftarrow \emptyset$ // start with no distractions

  while $p \leq d$ **:**

    $Y \leftarrow \{v \in V_p \setminus Z \mid \mathsf{next}(v, Z) \neq \mathsf{pr}(v) \bmod 2\}$ // distractions

    if $Y \neq \emptyset$ **:**

      $Z \leftarrow Z \cup Y$ // update current fixed point Z_p

      $Z \leftarrow Z \setminus \{v \mid \mathsf{pr}(v) < p\}$ // reset all lower fixed points

      $p \leftarrow 0$ // continue with lowest priority

    else:

      $p \leftarrow p + 1$ // fixed point, continue higher

  return $W_\Diamond, W_\Box$ where $W_\Diamond \leftarrow \{v \mid \mathsf{winner}(v, Z) = 0\}$, $W_\Box \leftarrow V \setminus W_\Diamond$

Note: algorithm does not give a strategy (see [BFL14] for a method)!

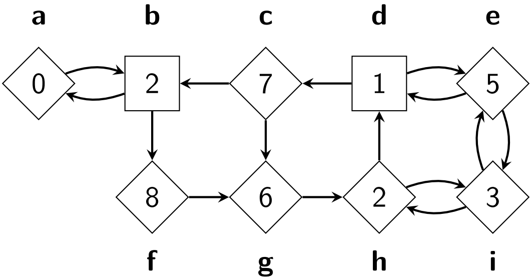
def fpi($\Game$):
    /* assume vertices are sorted by priority, $V(i)$ for ith vertex */
    $Z \leftarrow V \mapsto 0$ // start with no distractions
    $i \leftarrow 0$ // start with lowest vertex
    $p \leftarrow \mathsf{pr}(V(i))$ // start with lowest priority
    Chg $\leftarrow$ False // whether Z_p is updated
    while True **:**
        if $i = n \vee \mathsf{pr}(V(i)) \neq p$ **:**
            if Chg **:**
                $Z \leftarrow Z[\{v \mid \mathsf{pr}(v) < p\} \mapsto 0]$ // reset all lower vertices
                goto 3 // restart with lowest vertex
            elif $i = n$ **:**
                return $\{v \mid \mathsf{winner}(v, Z) = 0\}, \{v \mid \mathsf{winner}(v, Z) = 1\}$
            else:
                $p \leftarrow \mathsf{pr}(V(i))$ // Z_p not updated; continue
        else:
            if $\neg Z[i] \wedge \mathsf{next}(V(i), Z) \neq \mathsf{pr}(V(i)) \bmod 2$ **:**
                $Z[i] \leftarrow 1$ // ith vertex is distraction
                Chg $\leftarrow$ True // mark that Z_p is updated
            $i \leftarrow i + 1$

Fixed point iteration

Some notes...

- That was my own version of the fixed point algorithm
- To prove: that it is correct
- To show: that it is equivalent to [BFL14] and [KV98] and [dSMPV16]
- To study: whether [BFL14] also leads to a method of finding strategies
- Implementation can be a tight loop with Z implemented as a bit vector, and all vertices sorted by priority... going from low to high, and resetting all lower priority vertices plus restarting the loop whenever some $Z[v]$ is set

Fixed point iteration



h, **g** (reset **h**)
h, **c** (reset **h**, **g**)
h, **g** (reset **h**)
h, **f** (reset **h**, **g**, **c**)
b, **h**, **a**, **g** (reset **b**, **h**, **a**)
b, **h**, **a**