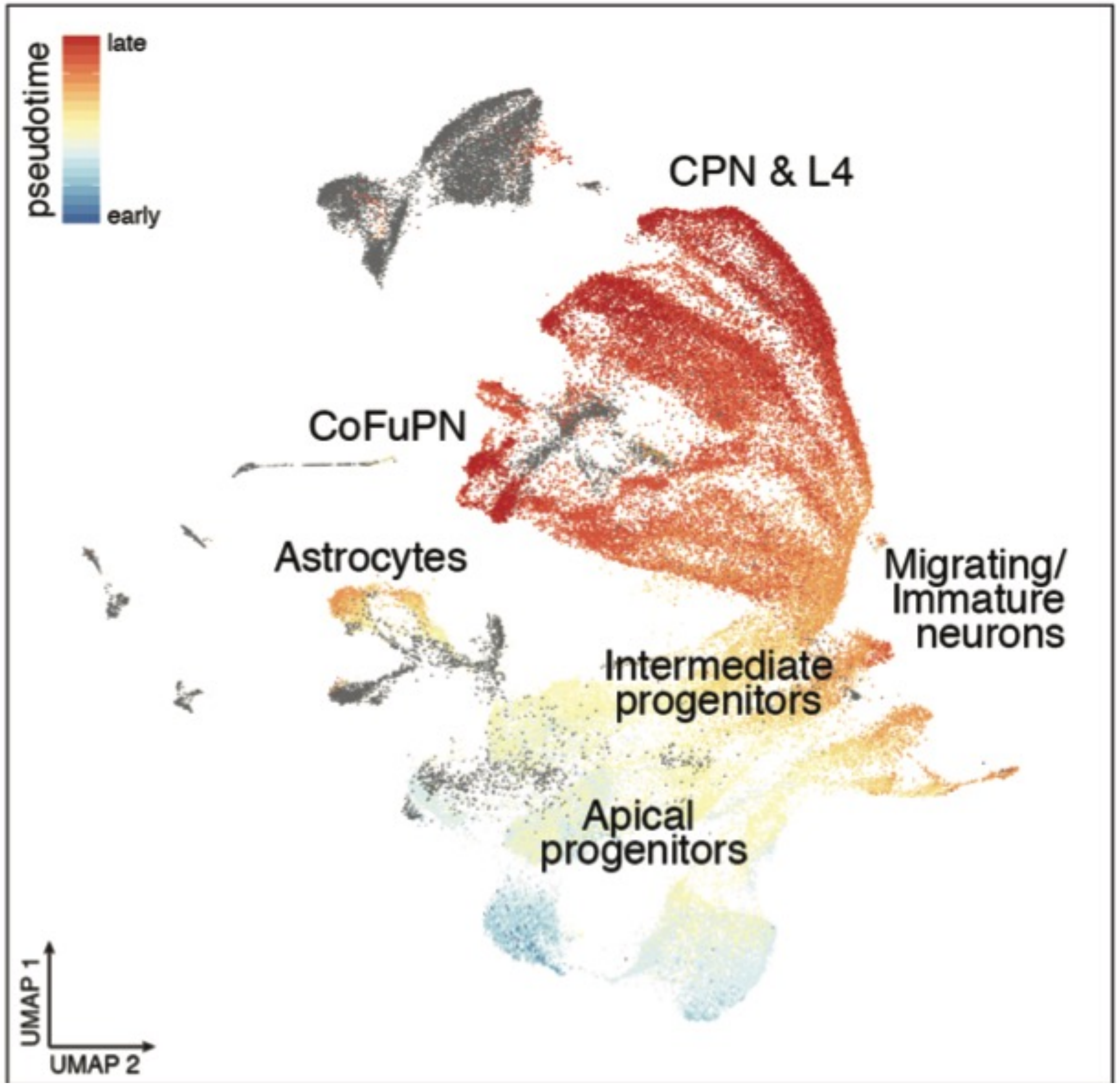
Boolean modelling of biological processes

Samuel Pastva

samuel.pastva@ist.ac.at

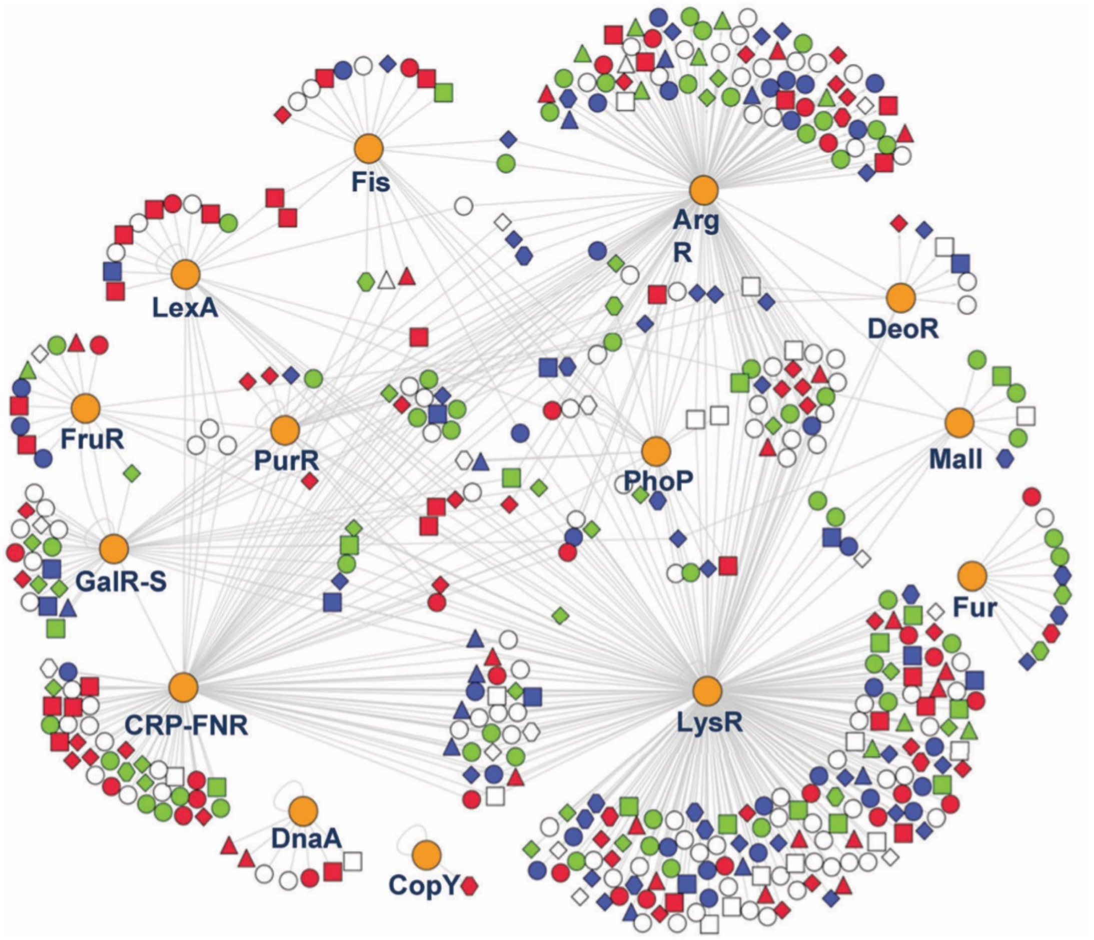
The sequencing boom

- Modern single-cell sequencing enables observations orders of magnitude more precise than 10-20 years ago.
- Activity of thousands of genes across thousands of cells, tissues and mutations.
- How do we rigorously use this data to understand complex biological systems?



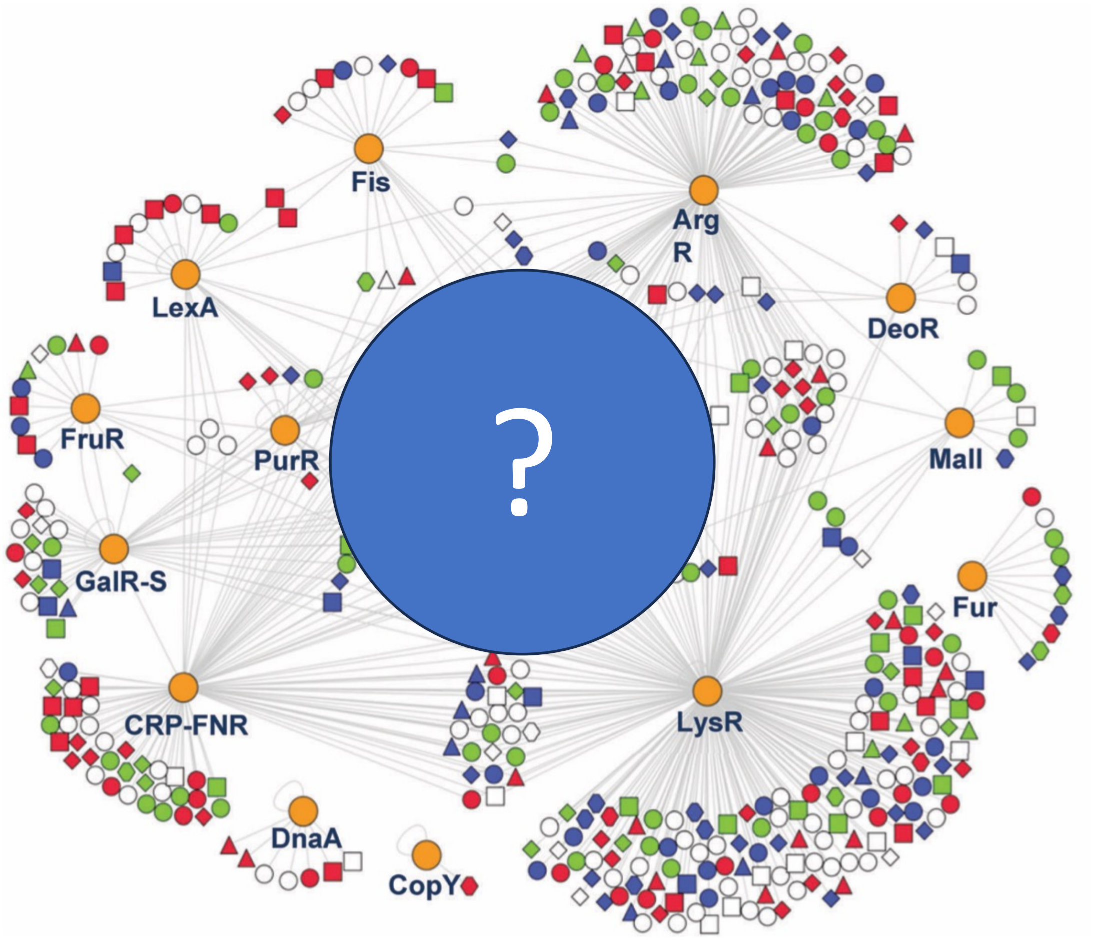
Mechanistic modelling

- Mechanistic models:
  - Grounded in explainable biochemical principles.
- “Black box” model learns to answer questions.
- “Mechanistic” model helps to design new questions.
- Boolean networks:
  - Simple, massively parallel programs emulating gene regulation.



Where are we going?

- Synthesis/inference:
  - What models fit observed data?
  - Bonus round: what does it even mean to fit data?
- Selection/identifiability:
  - Which candidate model is the ”best”?
  - How to design experiments to improve the candidate set?
  - Can we learn something from an incomplete model?
- BDDs / ASP / SMT / SAT
- As always… scalability…



Formal Methods for Safe and Trustworthy Probabilistic Systems

Djordje Zikelic

Institute of Science and Technology Austria

2023/2024

SMU
SINGAPORE MANAGEMENT UNIVERSITY

Formal verification

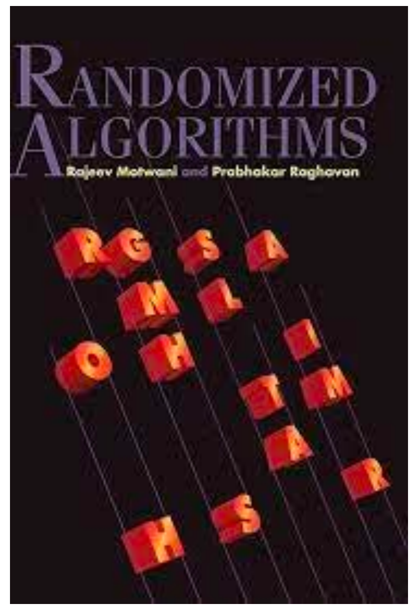
Formal controller synthesis

Applications

Formal verification

$x = 0$
while $x \geq 0$ **do**
  $r_1 := \mathit{Uniform}([-1, 0.5])$
  $x := x + r_1$
  if $x \geq 100$ **then**
    $r_2 := \mathit{Uniform}([-1, 2])$
    $x := x + r_2$

Probabilistic programs



Randomized algorithms

Formal controller synthesis

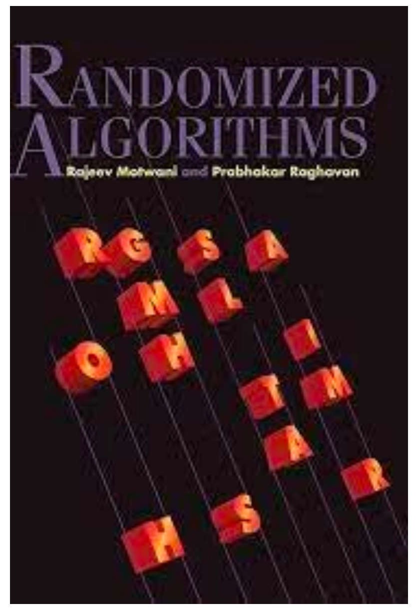
Applications

https://www.cambridge.org/core/books/randomized-algorithms/6A3E5CD760B0DDBA3794A100EE2843E8

Formal verification

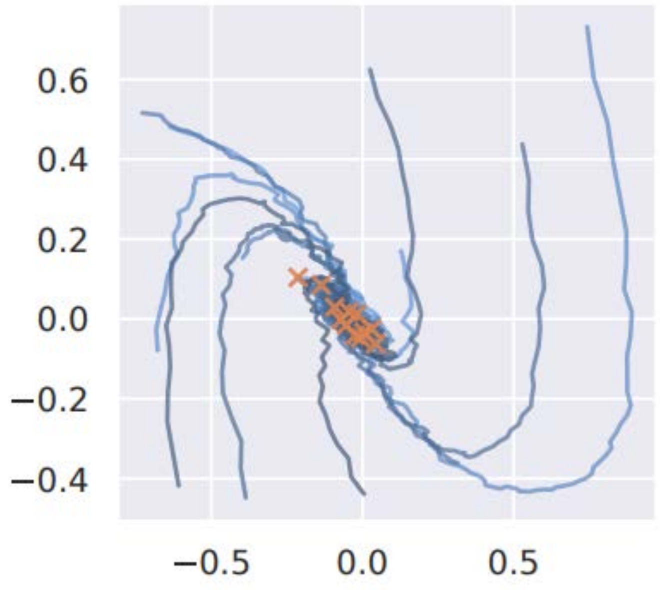
$x = 0$
while $x \geq 0$ **do**
  $r_1 := \mathit{Uniform}([-1, 0.5])$
  $x := x + r_1$
  if $x \geq 100$ **then**
    $r_2 := \mathit{Uniform}([-1, 2])$
    $x := x + r_2$

Probabilistic programs

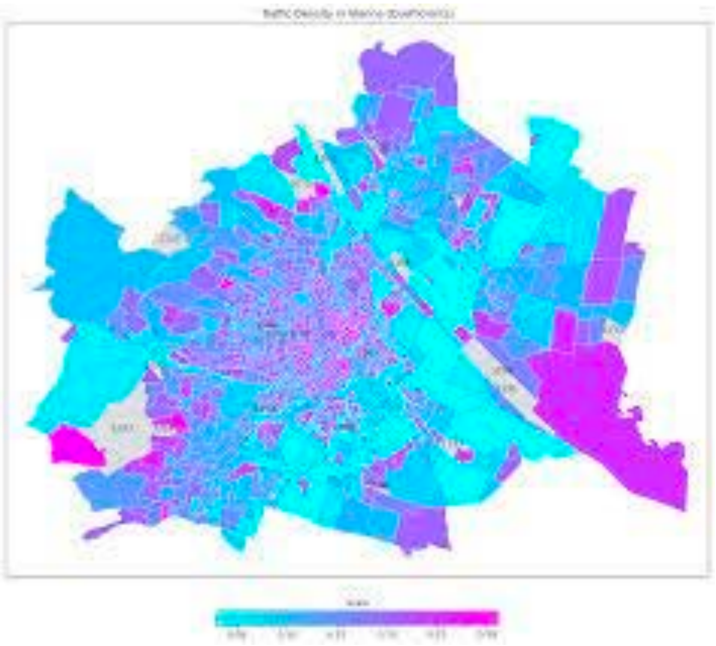


Randomized algorithms

Formal controller synthesis



Neurosymbolic methods



Distributional properties

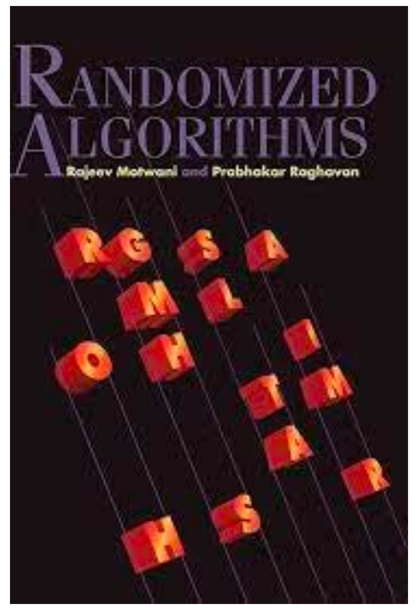
Applications

https://www.cambridge.org/core/books/randomized-algorithms/6A3E5CD760B0DDBA3794A100EE2843E8
https://towardsdatascience.com/modeling-traffic-density-of-the-city-of-vienna-c41480c35523?gi=942a7b186562

Formal verification

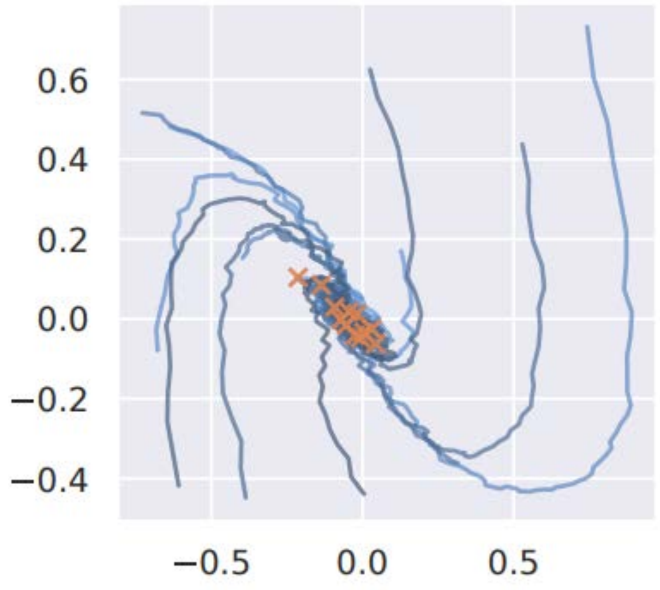
```
x = 0
while x ≥ 0 do
    r1 := Uniform([−1, 0.5])
    x := x + r1
    if x ≥ 100 then
        r2 := Uniform([−1, 2])
        x := x + r2
```

Probabilistic programs

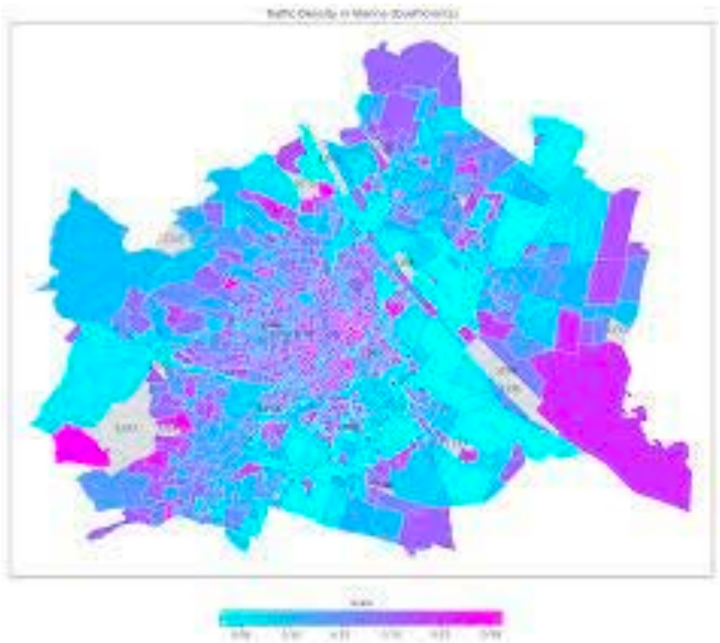


Randomized algorithms

Formal controller synthesis



Neurosymbolic methods



Distributional properties

Applications

Bidding games on graphs

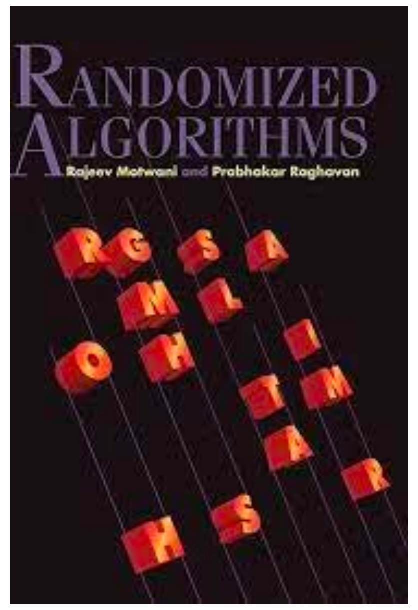
Blockchain protocols (very recent)

https://www.cambridge.org/core/books/randomized-algorithms/6A3E5CD760B0DDBA3794A100EE2843E8
https://towardsdatascience.com/modeling-traffic-density-of-the-city-of-vienna-c41480c35523?gi=942a7b186562
https://www.frommers.com/tips/airfare/upgrade-bidding-tips-how-to-game-airline-seat-auctions-so-youll-win
https://en.wikipedia.org/wiki/Bitcoin

Formal verification

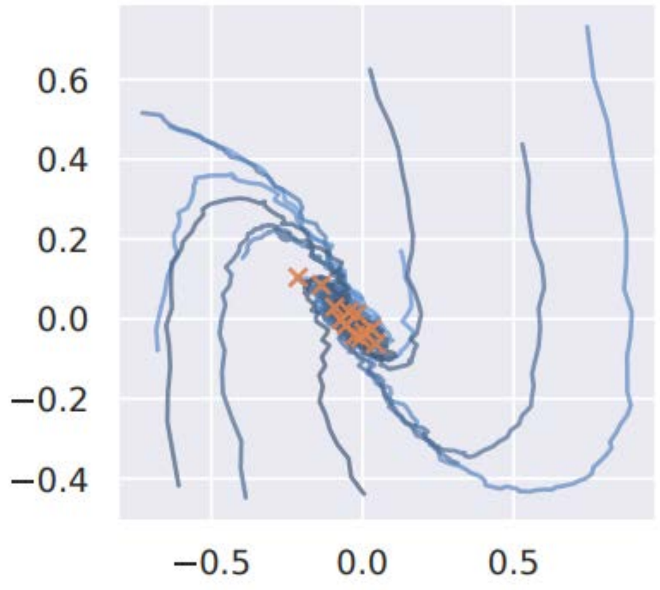
$x = 0$
while $x \geq 0$ **do**
  $r_1 := \mathit{Uniform}([-1, 0.5])$
  $x := x + r_1$
  if $x \geq 100$ **then**
    $r_2 := \mathit{Uniform}([-1, 2])$
    $x := x + r_2$

Probabilistic programs

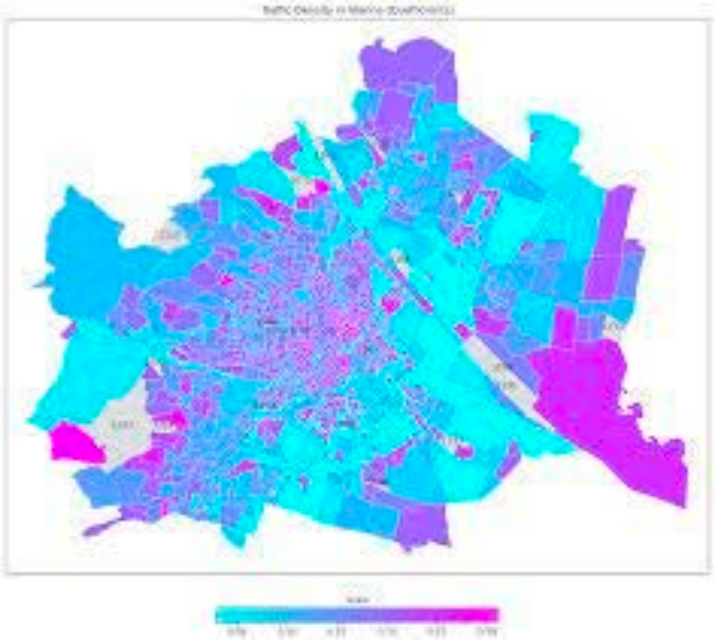


Randomized algorithms

Formal controller synthesis



Neurosymbolic methods



Distributional properties

Applications

Bidding games
on graphs

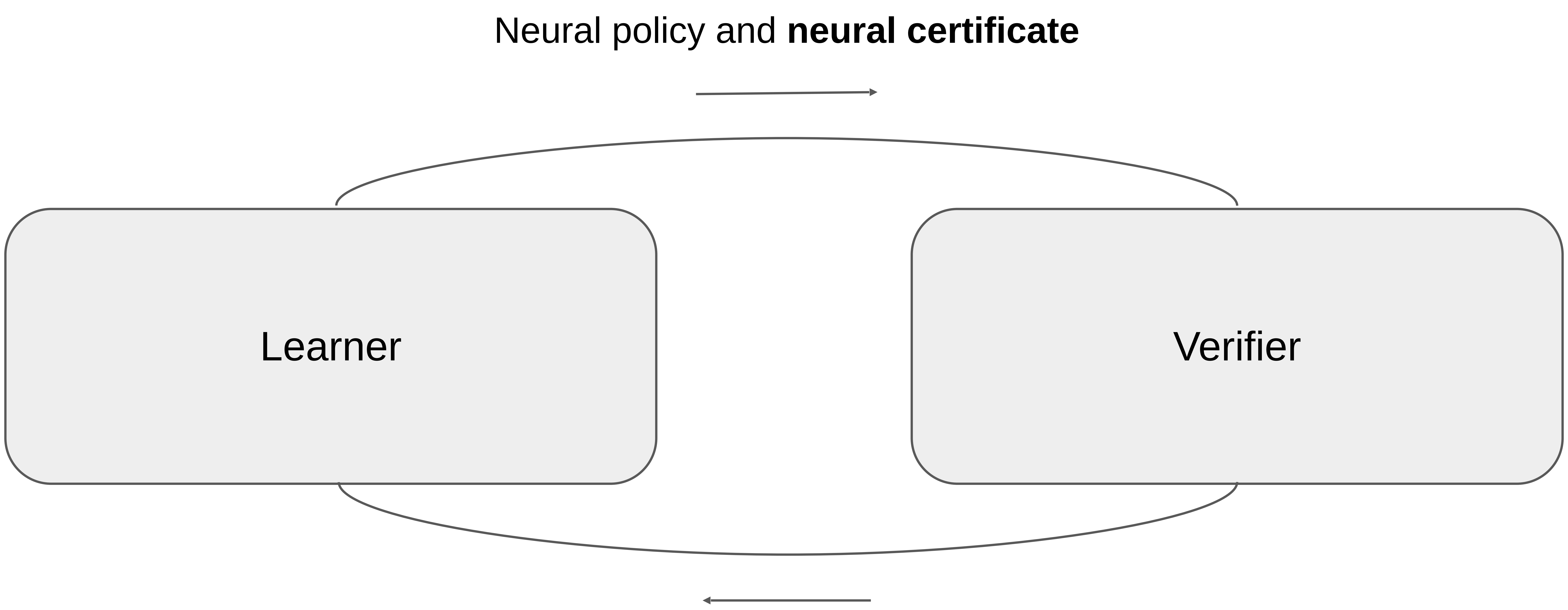
Blockchain protocols
(very recent)

https://www.cambridge.org/core/books/randomized-algorithms/6A3E5CD760B0DDBA3794A100EE2843E8
https://towardsdatascience.com/modeling-traffic-density-of-the-city-of-vienna-c41480c35523?gi=942a7b186562
https://www.frommers.com/tips/airfare/upgrade-bidding-tips-how-to-game-airline-seat-auctions-so-youll-win
https://en.wikipedia.org/wiki/Bitcoin

Why neurosymbolic methods, why formal?

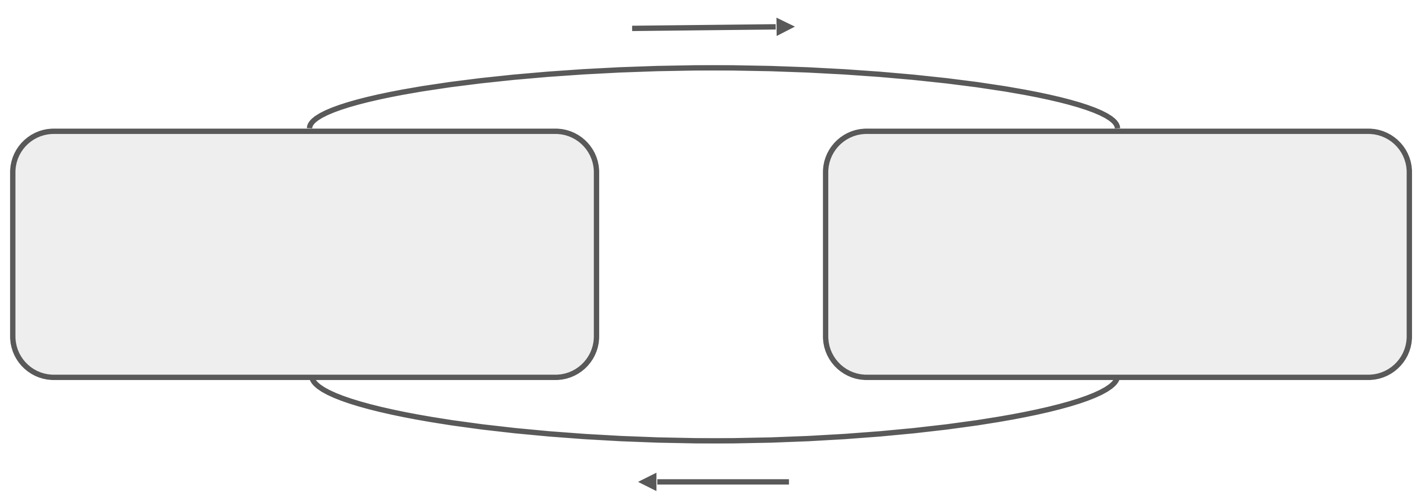
Safety-critical applications require formal correctness guarantees

Learner-verifier framework [1,2,3]



[1] Chang, Roohi, Gao. *Neural Lyapunov Control.* NeurIPS 2019
[2] Ravanbakhsh, Sankaranarayanan. *Learning Control Lyapunov Functions from Counterexamples and Demonstrations.* Autonomous Robots 2019
[3] Abate, Ahmed, Giacobbe, Peruffo. *Formal Synthesis of Lyapunov Neural Networks.* IEEE Control Systems Letters 2020

Learner-verifier framework

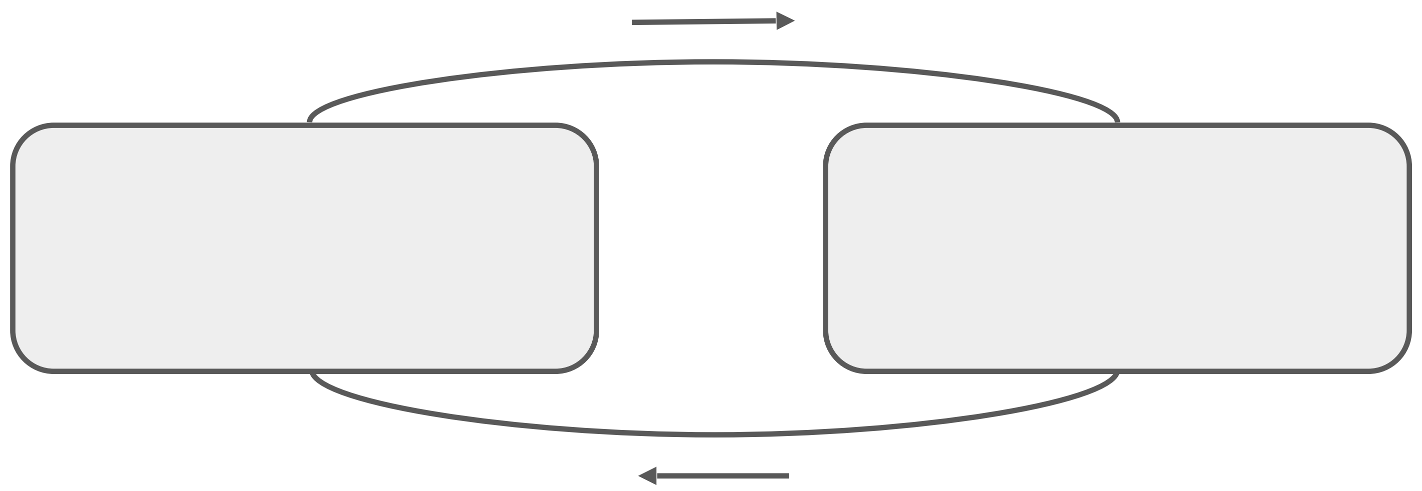


What are learnable certificates for stochastic systems?

How to learn these certificates?

How to formally verify these certificates?

Learner-verifier framework



Results*

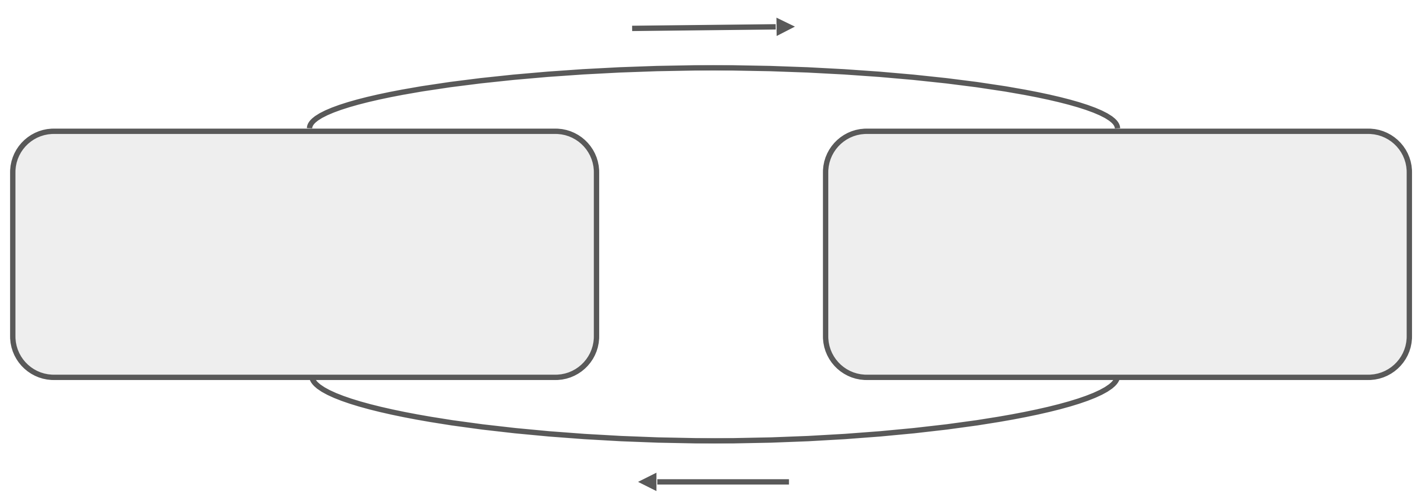
Neural martingales as formal certificates

Learner-verifier loop for neural policies + martingales

(reachability [AAAI’22], reach-avoidance [AAAI’23], stability [ATVA’23], compositional reasoning [NeurIPS’23], Bayesian neural networks [NeurIPS’21])

*Joint work with Mathias Lechner, Krish, Tom, Matin Ansaripour, Abhinav Verma

Learner-verifier framework



Results*

Neural martingales as formal certificates

Learner-verifier loop for neural policies + martingales

(reachability [AAAI'22], reach-avoidance [AAAI'23], stability [ATVA'23], compositional reasoning [NeurIPS'23], Bayesian neural networks [NeurIPS'21])

What's next?

Richer specifications

Compositional reasoning about systems, neural policies and neural certificates

Scaling to larger systems

*Joint work with Mathias Lechner, Krish, Tom, Matin Ansaripour, Abhinav Verma

Custom Theory Reasoning
Clemens Eisenhofer

TU Wien, Austria

SMT solvers

Satisfiability Modulo Theories (*SMT*) solvers support reasoning in (fragments of) first-order logic:

SMT solvers

Satisfiability Modulo Theories (*SMT*) solvers support reasoning in (fragments of) first-order logic:

- SMT-solvers can reason natively in a wide range of theories: Integers, arrays, strings, bit-vectors, ADTs, . . .

SMT solvers

Satisfiability Modulo Theories (*SMT*) solvers support reasoning in (fragments of) first-order logic:

- SMT-solvers can reason natively in a wide range of theories: Integers, arrays, strings, bit-vectors, ADTs, ...

$\Rightarrow$ Essential component in automated software/hardware/protocol verification.

SMT solvers

Satisfiability Modulo Theories (*SMT*) solvers support reasoning in (fragments of) first-order logic:

```
int32 i1, i2;
...
assume(i1 > 0);
arr[0] = 1;
arr[i1 + i2] = 2;
assert(arr[0] = 1);
```

SMT solvers

Satisfiability Modulo Theories (*SMT*) solvers support reasoning in (fragments of) first-order logic:

```
int32 i1, i2;
...
assume(i1 > 0);
arr[0] = 1;
arr[i1 + i2] = 2;
assert(arr[0] = 1);
```

$$\Rightarrow \begin{array}{c} \ldots \wedge \\ i1 > 0 \wedge \\ arr_1 = store(arr_0, 0, 1) \wedge \\ arr_2 = store(arr_1, i1 + i2, 2) \wedge \\ select(arr_2, 0) \neq 1 \end{array}$$

SMT solvers

Satisfiability Modulo Theories (*SMT*) solvers support reasoning in (fragments of) first-order logic:

```
int32 i1, i2;
...
assume(i1 > 0);
arr[0] = 1;
arr[i1 + i2] = 2;
assert(arr[0] = 1);
```

$\Rightarrow$

$$\begin{gathered} \ldots \wedge \\ i1 > 0 \wedge \\ arr_1 = store(arr_0, 0, 1) \wedge \\ arr_2 = store(arr_1, i1 + i2, 2) \wedge \\ select(arr_2, 0) \neq 1 \end{gathered}$$

$\Rightarrow$

$$\begin{gathered} array_0 \mapsto \langle 0, \ldots, 0 \rangle, \\ array_1 \mapsto \langle 1, \ldots, 0 \rangle, \\ array_2 \mapsto \langle 2, \ldots, 0 \rangle, \\ i1 \mapsto 2^{31}, \\ i2 \mapsto 2^{31} \end{gathered}$$

SMT solvers

Satisfiability Modulo Theories (*SMT*) solvers support reasoning in (fragments of) first-order logic:

```
int32 i1, i2;
...
assume(i1 > 0);
arr[0] = 1;
arr[i1 + i2] = 2;
assert(arr[0] = 1);
```

$\Rightarrow$

$$\begin{aligned} &\ldots \wedge \\ i1 &> 0 \wedge \\ arr_1 &= store(arr_0, 0, 1) \wedge \\ arr_2 &= store(arr_1, i1 + i2, 2) \wedge \\ &select(arr_2, 0) \neq 1 \end{aligned}$$

$\Rightarrow$

$$\begin{aligned} array_0 &\mapsto \langle 0, \ldots, 0 \rangle, \\ array_1 &\mapsto \langle 1, \ldots, 0 \rangle, \\ array_2 &\mapsto \langle 2, \ldots, 0 \rangle, \\ i1 &\mapsto 2^{31}, \\ i2 &\mapsto 2^{31} \end{aligned}$$

The solver has efficient procedures for dealing with $>$, $+$, *select*, and *store*.

My Current Research

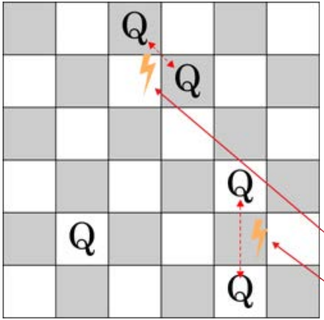
- Custom theory reasoning (“user-propagation”) in Z3

My Current Research

- Custom theory reasoning ("user-propagation") in Z3

My Current Research

- Custom theory reasoning ("user-propagation") in Z3



```
fixed(ast, value) :

  queenY = queenToY(ast)
  queenX = value

  if (queenX ≥ board)
    conflict({ ast })
    return

  foreach (fixed in alreadyFixedVars)
    otherX = model[fixed]
    otherY = queenToY(fixed)

    if (|queenX - otherX| = |queenY - otherY|)
      conflict({ ast, fixed })
    else if (queenX = otherX)
      conflict({ ast, fixed })
```

My Current Research

- Custom theory reasoning ("user-propagation") in Z3
  - Solving combinatorial problems by oracles, lazy axioms, efficient custom theories
    - Improve reasoning time
    - Less memory required

My Current Research

- Custom theory reasoning ("user-propagation") in Z3
  - Solving combinatorial problems by oracles, lazy axioms, efficient custom theories
    - Improve reasoning time
    - Less memory required
  - Non-classical logics in SMT
    - e.g., $\top \sqsubseteq (\Diamond_r.a < 1 \land \Diamond_r.a > 1)$ ($\mathcal{ALC}$)

My Current Research

- Custom theory reasoning ("user-propagation") in Z3
  - Solving combinatorial problems by oracles, lazy axioms, efficient custom theories
    - Improve reasoning time
    - Less memory required
  - Non-classical logics in SMT
    - e.g., $\top \sqsubseteq (\Diamond_r.a < 1 \land \Diamond_r.a > 1)$ ($\mathcal{ALC}$)
  - Theorem proving via weird calculi in SMT
    - e.g., $\{\{P(x)\}; \{P(a), \neg P(x) \lor P(f(x)), \neg P(f(f(a)))\}, \emptyset\}$ (Connection Calculus)

My Current Research

- Custom theory reasoning ("user-propagation") in Z3
  - Solving combinatorial problems by oracles, lazy axioms, efficient custom theories
    - Improve reasoning time
    - Less memory required
  - Non-classical logics in SMT
    - e.g., $\top \sqsubseteq (\Diamond_r.a < 1 \wedge \Diamond_r.a > 1)$ ($\mathcal{ALC}$)
  - Theorem proving via weird calculi in SMT
    - e.g., $\{\{P(x)\}; \{P(a), \neg P(x) \vee P(f(x)), \neg P(f(f(a)))\}, \emptyset\}$ (Connection Calculus)
  - *New (Nielson) string solver as theory extension*
    - $"a" ++ x = x ++ "b"$

My Current Research

- Custom theory reasoning ("user-propagation") in Z3
  - Solving combinatorial problems by oracles, lazy axioms, efficient custom theories
    - Improve reasoning time
    - Less memory required
  - Non-classical logics in SMT
    - e.g., $\top \sqsubseteq (\Diamond_r.a < 1 \wedge \Diamond_r.a > 1)$ ($\mathcal{ALC}$)
  - Theorem proving via weird calculi in SMT
    - e.g., $\{\{P(x)\}; \{P(a), \neg P(x) \vee P(f(x)), \neg P(f(f(a)))\}, \emptyset\}$ (Connection Calculus)
  - *New (Nielson) string solver as theory extension*
    - $"a" ++ x = x ++ "b"$

Applying SMT Propagation to "Everything"

Interface Theory for Security and Privacy

Ana Oliveira da Costa
Institute of Science and Technology Austria (ISTA)

October 9, 2023

Designing Secure Systems

We need to consider:

- Multiple architectural layers.
- Sub-systems developed by different teams.
- Heterogeneous components.
- Interaction between cyber and physical components.

Designing Secure Systems

We need to consider:

- Multiple architectural layers.
- Sub-systems developed by different teams.
- Heterogeneous components.
- Interaction between cyber and physical components.

$$\Downarrow$$

Contract-based design.

Interface Theory

Luca de Alfaro and Thomas A. Henzinger. *Interface theories for component-based design.* (2001)

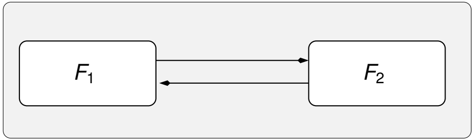
$\langle \mathbb{I}, \preceq, \sim, \otimes \rangle$ where $\preceq$ is *refinement*, $\sim$ is *compatibility*, and $\otimes$ is *composition*.

Interface Theory

Luca de Alfaro and Thomas A. Henzinger. *Interface theories for component-based design.* (2001)

$\langle \mathbb{I}, \preceq, \sim, \otimes \rangle$ where $\preceq$ is *refinement*, $\sim$ is *compatibility*, and $\otimes$ is *composition*.

Composition ($\otimes$)



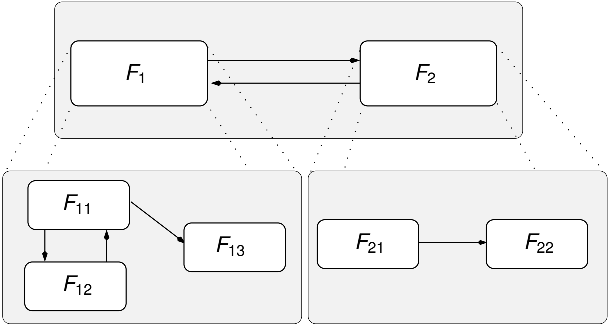
Interface Theory

Luca de Alfaro and Thomas A. Henzinger. *Interface theories for component-based design.* (2001)

$\langle \mathbb{I}, \preceq, \sim, \otimes \rangle$ where $\preceq$ is *refinement*, $\sim$ is *compatibility*, and $\otimes$ is *composition*.

Composition ($\otimes$)

Refinement ($\preceq$)

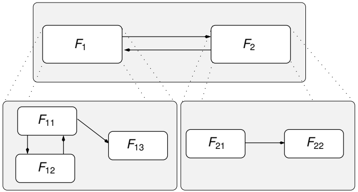


Interface Theory

Incremental Design: Composition only requires knowledge about the parts being composed.

Interface Theory

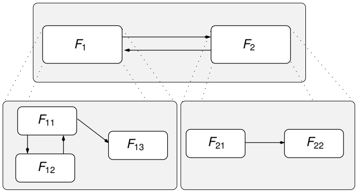
Incremental Design: Composition only requires knowledge about the parts being composed.



Interface Theory

Incremental Design: Composition only requires knowledge about the parts being composed.

If $F \sim G$ and $F \otimes G \sim H$, then $G \sim H$ and $F \sim G \otimes H$.

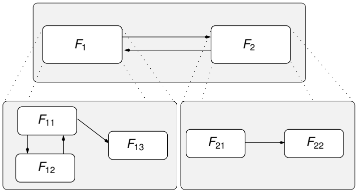


Interface Theory

Incremental Design: Composition only requires knowledge about the parts being composed.

If $F \sim G$ and $F \otimes G \sim H$, then $G \sim H$ and $F \sim G \otimes H$.

Independent Implementability: Independent refinement of subsystems.



Interface Theory

Incremental Design: Composition only requires knowledge about the parts being composed.

If $F \sim G$ and $F \otimes G \sim H$, then $G \sim H$ and $F \sim G \otimes H$.

Independent Implementability: Independent refinement of subsystems.

If $F \sim G$ and $F' \preceq F$, then $F' \sim G$ and $F' \otimes G \preceq F \otimes G$.

Information-flow Interfaces

Ezio Bartocci, Thomas Ferrère, Thomas Henzinger, Dejan Nickovic, D., and Ana O. da Costa.
Information-flow interfaces. (2022)

Security policies abstracted as information-flow constraints.

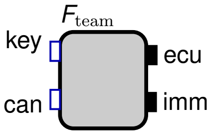
Information-flow Interfaces

Ezio Bartocci, Thomas Ferrère, Thomas Henzinger, Dejan Nickovic, D., and Ana O. da Costa.
Information-flow interfaces. (2022)

Security policies abstracted as information-flow constraints.

Interfaces specify:

- disjoint sets of input and output variables, $X \cap Y = \emptyset$;



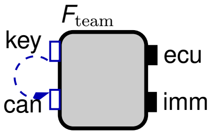
Information-flow Interfaces

Ezio Bartocci, Thomas Ferrère, Thomas Henzinger, Dejan Nickovic, D., and Ana O. da Costa.
Information-flow interfaces. (2022)

Security policies abstracted as information-flow constraints.

Interfaces specify:

- disjoint sets of input and output variables, $X \cap Y = \emptyset$;
- no-flow constraints on the environment as *assumptions*;



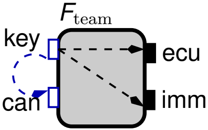
Information-flow Interfaces

Ezio Bartocci, Thomas Ferrère, Thomas Henzinger, Dejan Nickovic, D., and Ana O. da Costa.
Information-flow interfaces. (2022)

Security policies abstracted as information-flow constraints.

Interfaces specify:

- disjoint sets of input and output variables, $X \cap Y = \emptyset$;
- no-flow constraints on the environment as *assumptions*;
- no-flow requirements on implementations as *open-guarantees*;



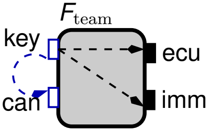
Information-flow Interfaces

Ezio Bartocci, Thomas Ferrère, Thomas Henzinger, Dejan Nickovic, D., and Ana O. da Costa.
Information-flow interfaces. (2022)

Security policies abstracted as information-flow constraints.

Interfaces specify:

- disjoint sets of input and output variables, $X \cap Y = \emptyset$;
- no-flow constraints on the environment as *assumptions*;
- no-flow requirements on implementations as *open-guarantees*;
- no-flow requirements on the closed-system as *closed-guarantees*.



What is next?

- Explore formalisms to specify *what is* an information flow.
- Dive into real-world use cases.
- Explore the limits of interface theory for the design of secure systems.

Finding counterexamples to $\forall\exists$-safety hyperproperties

. . . and other forays into incorrectness

Tobias Nießen

TU Wien

October 9, 2023

∀∃-safety hyperproperties

Definition (informal, intuition)

"For each trace τ there exists a trace τ' such that τ and τ' do not interact badly."

$\forall\exists$-safety hyperproperties

Definition (informal, intuition)

"For each trace τ there exists a trace τ' such that τ and τ' do not interact badly."

Example (Refinement)

$$\forall^{\mathtt{P}}\tau\,\exists^{\mathtt{Q}}\tau'\left(in_{\tau} = in_{\tau'} \land out_{\tau} = out_{\tau'}\right)$$

∀∃-safety hyperproperties

Definition (informal, intuition)

"For each trace τ there exists a trace τ' such that τ and τ' do not interact badly."

Example (Refinement)

$$\forall^{\mathtt{P}} \tau \, \exists^{\mathtt{Q}} \tau' \left(in_{\tau} = in_{\tau'} \wedge out_{\tau} = out_{\tau'} \right)$$

Hint: $\underbrace{y := x * \mathrm{random}(\mathbb{N})}_{\mathtt{P}}$ refines $\underbrace{y := x * \mathrm{random}(\mathbb{Z})}_{\mathtt{Q}}$, but not vice versa

Verification of ∀∃ hyperproperties – unsurprisingly difficult

Undecidability of trace properties

\+ quantification over multiple traces

\+ quantifier alternation

Verification of ∀∃ hyperproperties – unsurprisingly difficult

Undecidability of trace properties

+ quantification over multiple traces

+ quantifier alternation

	Loops	Infinite states	Complete	Counterexamples
Strategy-based approaches	✓	✓	✗	✗
Automata-based approaches	✓	✗	✓	✗
Relational Hoare-style logic	✗	✓	✓	✓

∀∃-safety hyperproperties – our approach to finding counterexamples

Goal: find model for negation of ∀∃-safety property

∀∃-safety hyperproperties – our approach to finding counterexamples

Goal: find model for negation of ∀∃-safety property

Combine underapproximate methods to find counterexamples

- symbolic execution for universally quantified traces
- bounded model checking for existentially quantified traces
- lift both algorithms to an SMT solver for infinite variable domains
- typically requires many iterations to exclude spurious refutations

$\forall\exists$-safety hyperproperties – our approach to finding counterexamples

Goal: find model for negation of $\forall\exists$-safety property

Combine underapproximate methods to find counterexamples

- symbolic execution for universally quantified traces
- bounded model checking for existentially quantified traces
- lift both algorithms to an SMT solver for infinite variable domains
- typically requires many iterations to exclude spurious refutations

Does this terminate? Sometimes. Maybe. It depends...

Runtime Monitoring Neural Certificates

Emily Yu

Klosterneuburg, Austria
October 9, 2023

Institute of Science and Technology Austria

Dynamical Systems

$f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$


[forbes.com]

Learning Certificate Functions

Requirements

- ◇ Stability: Lynapunov function $V : \mathcal{X} \to \mathbb{R}$

  $\longrightarrow$ certifies stability around a fixed point

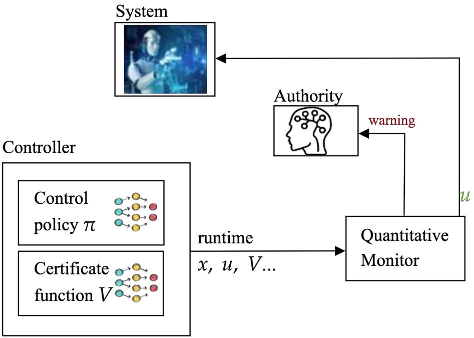
- ◇ Safety: Barrier function $h : \mathcal{X} \to \mathbb{R}$

  $\longrightarrow$ certifies invariance of a region

Verifying Certificates faces challenges

- ◇ Generalization error bounds: [Liu+'20, Boffi+'21, ChangGao'21]
- ◇ Lipschitz arguments : [Richards+'18, BobitiLazar'18]
- ◇ Learner-verifier: [Chang+'19, Peruffo+'21, Chatterjee+'23] etc

Monitoring Certificate Functions



- Validating certificate at runtime

- Chang, Ya-Chien, and Sicun Gao. "Stabilizing neural control using self-learned almost lyapunov critics." 2021 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2021.
- Boffi, Nicholas, et al. "Learning stability certificates from data." Conference on Robot Learning. PMLR, 2021.
- Liu, Shenyu, Daniel Liberzon, and Vadim Zharnitsky. "Almost Lyapunov functions for nonlinear systems." Automatica 113 (2020): 108758.
- Richards, Spencer M., Felix Berkenkamp, and Andreas Krause. "The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems." Conference on Robot Learning. PMLR, 2018.
- Bobiti, Ruxandra, and Mircea Lazar. "Automated-sampling-based stability verification and DOA estimation for nonlinear systems." IEEE Transactions on Automatic Control 63.11 (2018): 3659-3674.
- Chatterjee, Krishnendu, et al. "A Learner-Verifier Framework for Neural Network Controllers and Certificates of Stochastic Systems." International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Cham: Springer Nature Switzerland, 2023.

References II

Chang, Ya-Chien, Nima Roohi, and Sicun Gao. "Neural lyapunov control." Advances in neural information processing systems 32 (2019).

Peruffo, Andrea, Daniele Ahmed, and Alessandro Abate. "Automated and formal synthesis of neural barrier certificates for dynamical models." International conference on tools and algorithms for the construction and analysis of systems. Cham: Springer International Publishing, 2021.

https://www.forbes.com/sites/forbestechcouncil/2022/07/27/ai-from-drug-discovery-to-robotics/?sh=37eef0c53d7f

Credits

Diagrams have been designed using images from Flaticon.com.

Quantitative Safety and Liveness of Quantitative Automata

2023 – Klosterneuburg Austria

Udi Boker [†]
Thomas A. Henzinger [‡]
Nicolas Mazzocchi [‡]
N. Ege Saraç [‡]

† Reichman University, Israel
‡ Institute of Science and Technology, Austria

Boolean Properties

Definition

A Boolean property $\Phi \subseteq \Sigma^\omega$ or equivalently $\Phi \colon \Sigma^\omega \to \{0, 1\}$, is a language

Safety

Requests Not Duplicated

Liveness

All Requests Granted

Boolean Properties

Definition

A Boolean property $\Phi \subseteq \Sigma^\omega$ or equivalently $\Phi \colon \Sigma^\omega \to \{0,1\}$, is a language

Safety

Requests Not Duplicated

Liveness

All Requests Granted

Theorem: Decomposition of Boolean properties[1]

All property Φ can be expressed by: $\Phi = \Phi_{safe} \cap \Phi_{live}$

- Φ_{safe} *is safe*
- Φ_{live} *is live*

[1] Alpern, Schneider. *Defining liveness*. 1985

Boolean Properties

Definition

A Boolean property $\Phi \subseteq \Sigma^\omega$ or equivalently $\Phi \colon \Sigma^\omega \to \{0,1\}$, is a language

Safety

Requests Not Duplicated

Safety closure

smaller enlargement
to get a safe language

Liveness

All Requests Granted

Theorem: Decomposition of Boolean properties[1]

All property Φ can be expressed by:

$$\Phi = \Phi_{safe} \cap \Phi_{live}$$

- Φ_{safe} *is safe*
- Φ_{live} *is live*

[1] Alpern, Schneider. *Defining liveness*. 1985

Quantitative Properties

Definition[2]

A quantitative property $\Phi \colon \Sigma^\omega \to \mathbb{D}$ is a quantitative language where $\mathbb{D}$ is a complete lattice

Safety

Minimal Response Time

Liveness

Average Response Time

[2] Chatterjee, Doyen, Henzinger. *Quantitative Languages*. 2010

Quantitative Properties

Definition

A quantitative property $\Phi \colon \Sigma^\omega \to \mathbb{D}$ is a quantitative language where $\mathbb{D}$ is a complete lattice

Safety	Safety closure	Liveness
Minimal Response Time	the least safety property that bounds the original from above	Average Response Time

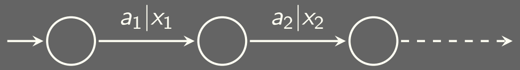
Theorem: Decomposition of quantitative properties[3]

All property Φ can be expressed by: $\Phi(w) = \min\{\Phi_{\text{safe}}(w), \Phi_{\text{live}}(w)\}$ for all $w \in \Sigma^\omega$

- Φ_{safe} is safe
- Φ_{live} is live

[3] Henzinger, Mazzocchi, Saraç. *Quantitative Safety and Liveness*. 2023

Quantitative Automata



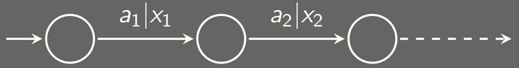
Word: $w = a_1 a_2 \dots$ Run value: $x = f(x_1 x_2 \dots)$

Value functions

Inf, Sup, LimInf, LimSup

LimInfAvg, LimSupAvg, DSum

Quantitative Automata



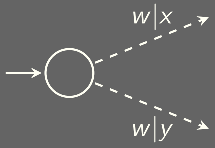
Word: $w = a_1 a_2 \ldots$ Run value: $x = f(x_1 x_2 \ldots)$

Value functions

Inf, Sup, LimInf, LimSup

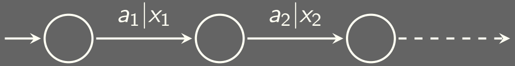
LimInfAvg, LimSupAvg, DSum

Non-determinism



$\mathcal{A}(w) = \sup\{\text{values of } w\text{'s runs}\}$

Quantitative Automata



Word: $w = a_1 a_2 \ldots$ Run value: $x = f(x_1 x_2 \ldots)$

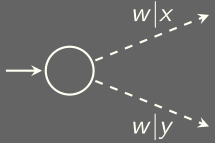
Value functions

Inf, Sup, LimInf, LimSup
LimInfAvg, LimSupAvg, DSum

Theorem[4]

The set $\{w \in \Sigma^\omega \mid \mathcal{A}(w) = \top\}$ is dense if and only if the automaton $\mathcal{A}$ is live

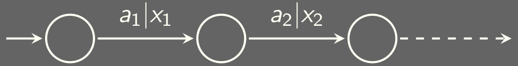
Non-determinism



$\mathcal{A}(w) = \sup\{\text{values of } w\text{'s runs}\}$

[4] Boker, Henzinger, Mazzocchi, Saraç. *Safety and Liveness of Quantitative Automata*. 2023

Quantitative Automata



Word: $w = a_1 a_2 \ldots$ Run value: $x = f(x_1 x_2 \ldots)$

Theorem[4]

The set $\{w \in \Sigma^\omega \mid \mathcal{A}(w) = \top\}$ is dense if and only if the automaton $\mathcal{A}$ is live

Theorem[4]

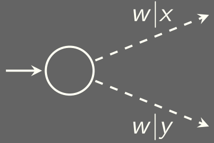
An automaton is live if and only if its safety closure is the constant $\top$

Value functions

Inf, Sup, LimInf, LimSup

LimInfAvg, LimSupAvg, DSum

Non-determinism



$\mathcal{A}(w) = \sup\{\text{values of } w\text{'s runs}\}$

[4] Boker, Henzinger, Mazzocchi, Saraç. *Safety and Liveness of Quantitative Automata*. 2023

Take away message

	Inf	Sup, LimInf, LimSup	LimInfAvg, LimSupAvg	DSum
Is it safe? i.e., $\mathcal{A}^\star = \mathcal{A}$	$O(1)$	PSPACE-complete	EXPSPACE PSPACE-hard	$O(1)$
Is it live? i.e., $\mathcal{A}^\star = \top$	PSPACE-complete			
Decomposition $\mathcal{A} = \min \mathcal{A}_{\text{safe}}\ \mathcal{A}_{\text{live}}$	$O(1)$	PTIME if deterministic	Open	$O(1)$

$\mathcal{A}^\star$ is the Safety closure of $\mathcal{A}$

Take away message

	Inf	Sup, LimInf, LimSup	LimInfAvg, LimSupAvg	DSum
Is it safe? i.e., $\mathcal{A}^\star = \mathcal{A}$	$O(1)$	PSPACE-complete	EXPSPACE PSPACE-hard	$O(1)$
Is it live? i.e., $\mathcal{A}^\star = \top$	PSPACE-complete			
Decomposition $\mathcal{A} = \min \mathcal{A}_{\text{safe}}\ \mathcal{A}_{\text{live}}$	$O(1)$	PTIME if deterministic	Open	$O(1)$

$\mathcal{A}^\star$ is the Safety closure of $\mathcal{A}$

1. T. A. Henzinger, N. Mazzocchi and N. E. Saraç
   Quantitative Safety and Liveness
   In *FOSSACS* proceedings 2023

2. U. Boker, T. A. Henzinger, N. Mazzocchi and N. E. Saraç
   Safety and Liveness of Quantitative Automata
   In *CONCUR* proceedings 2023

Thank you

Solving Parity and Rabin Games

K. S. Thejaswini

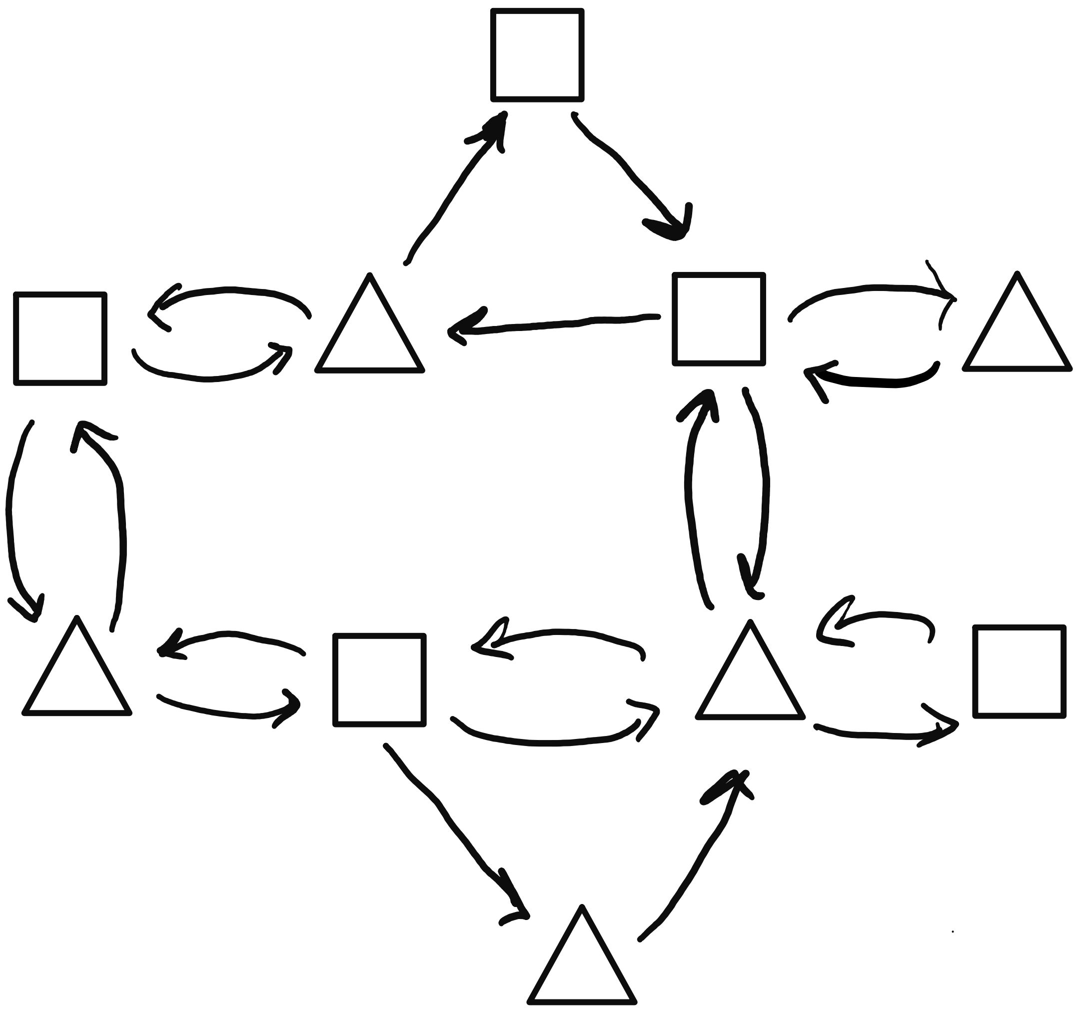
Laure Daviaud Marcin Jurdziński Pierre Ohlmann
Rupak Majumdar Rémi Morvan Irmak Sağlam

Solving Parity and Rabin Games

K. S. Thejaswini

Henzinger Group

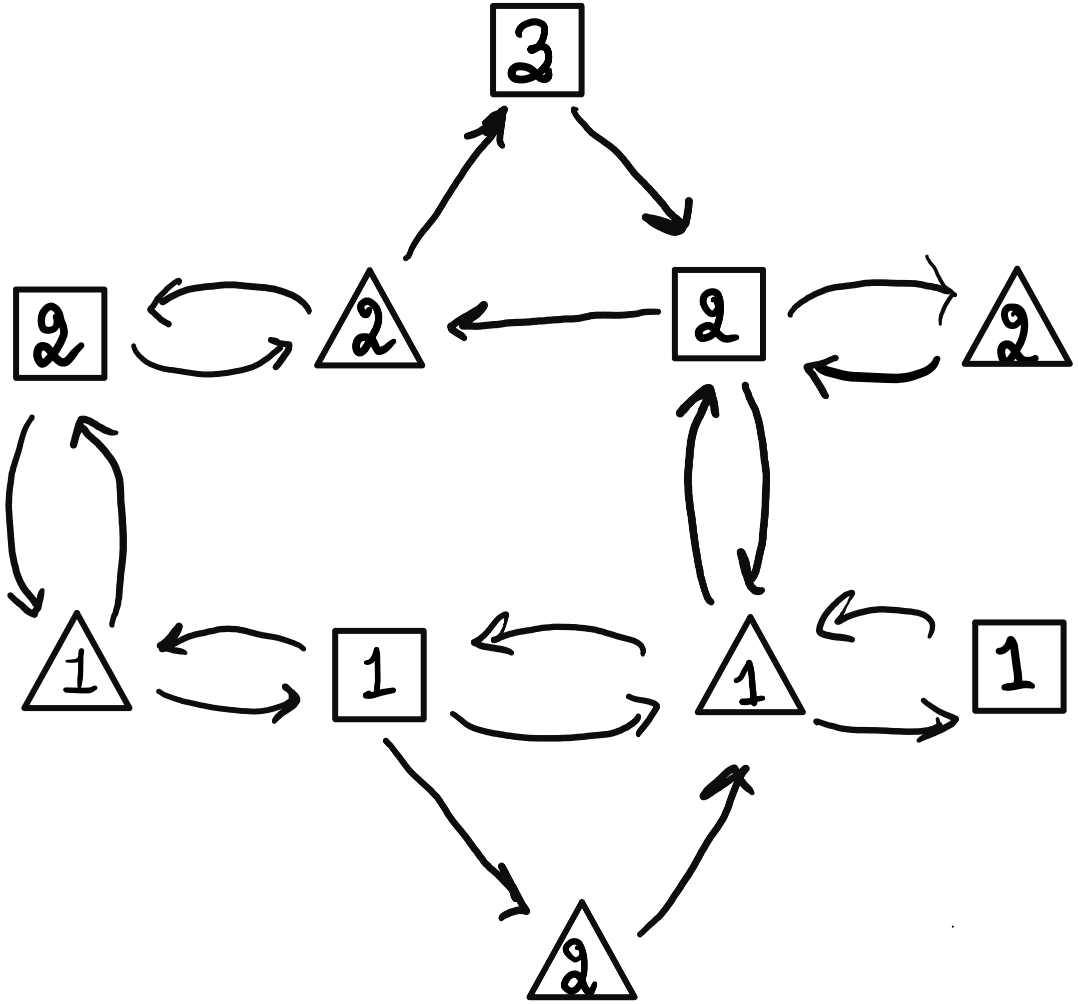
Parity Games



Steven □

Audrey △

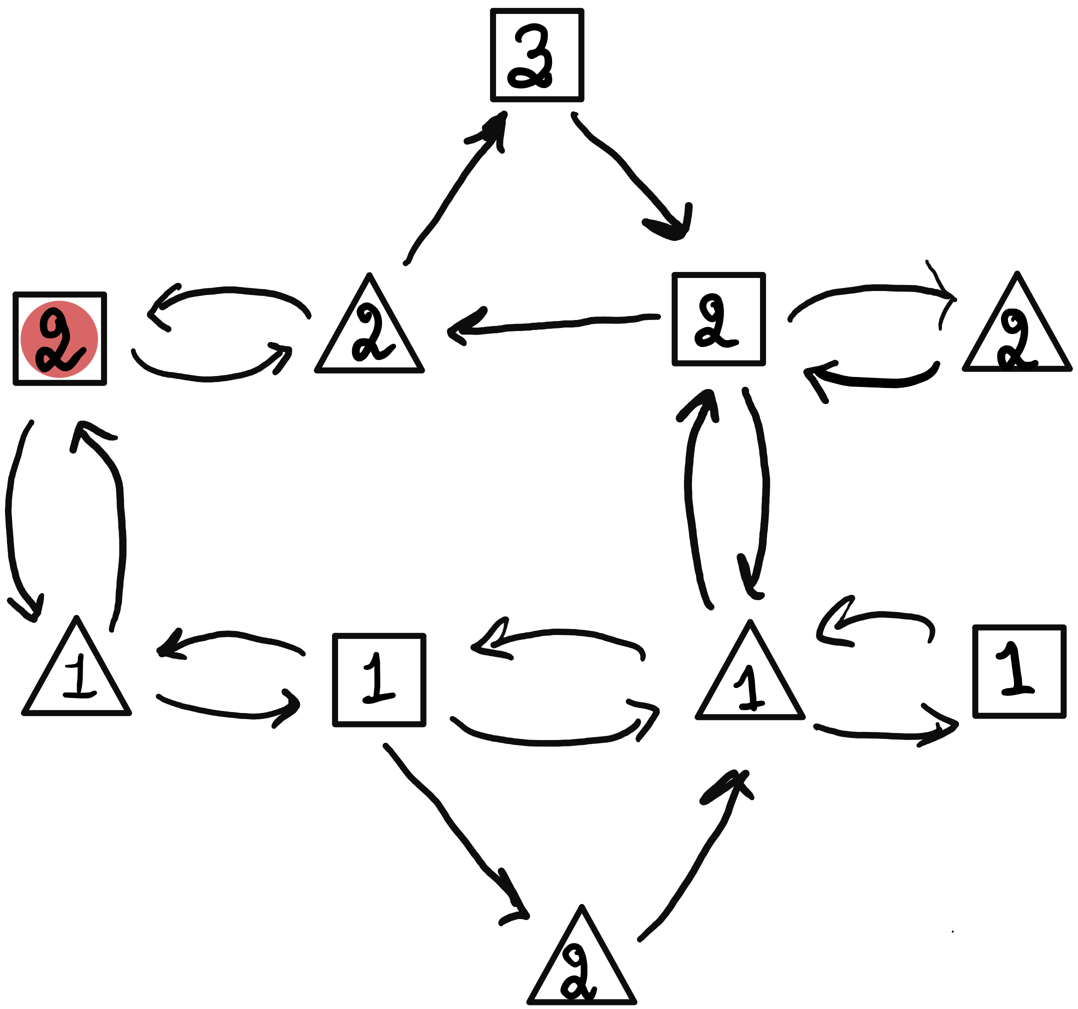
Parity Games



Steven □

Audrey △

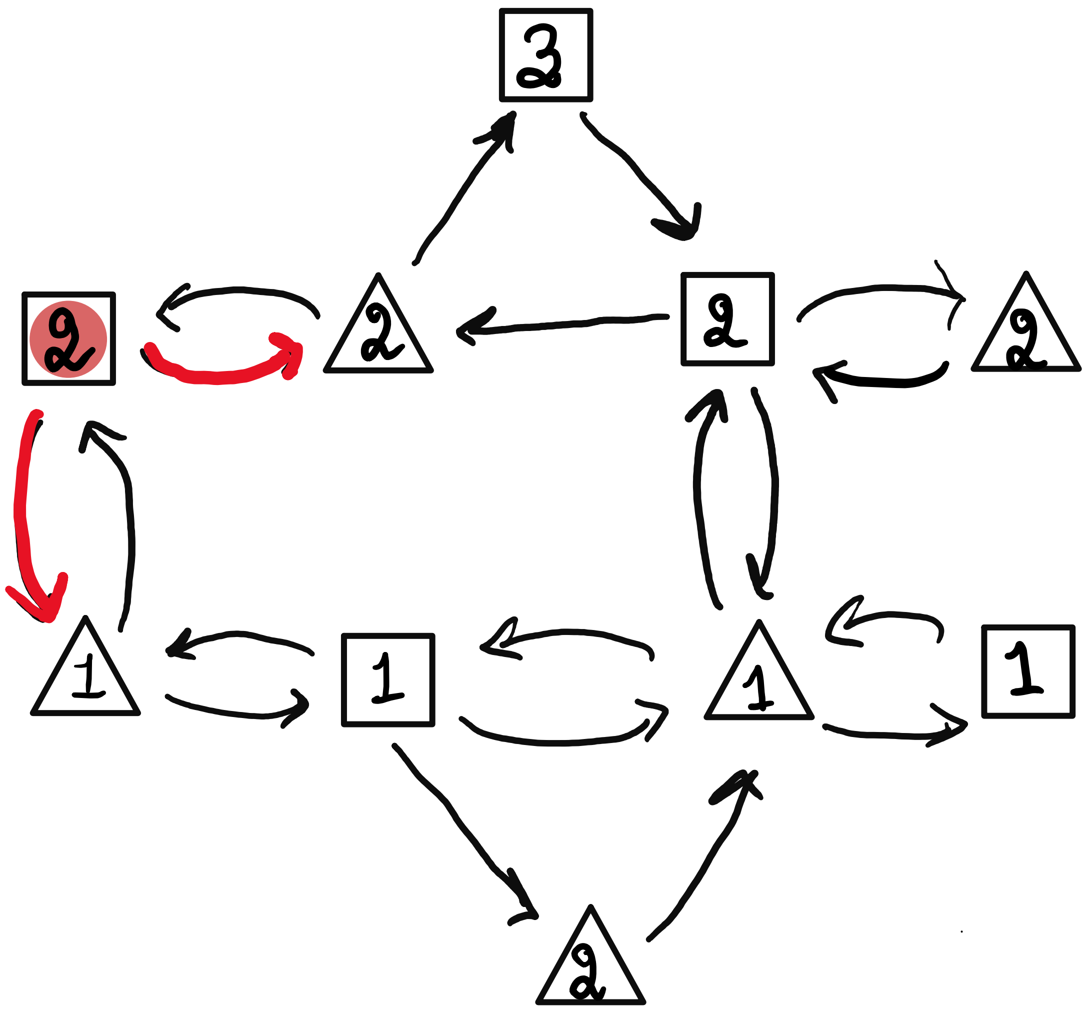
Parity Games



Steven □

Audrey △

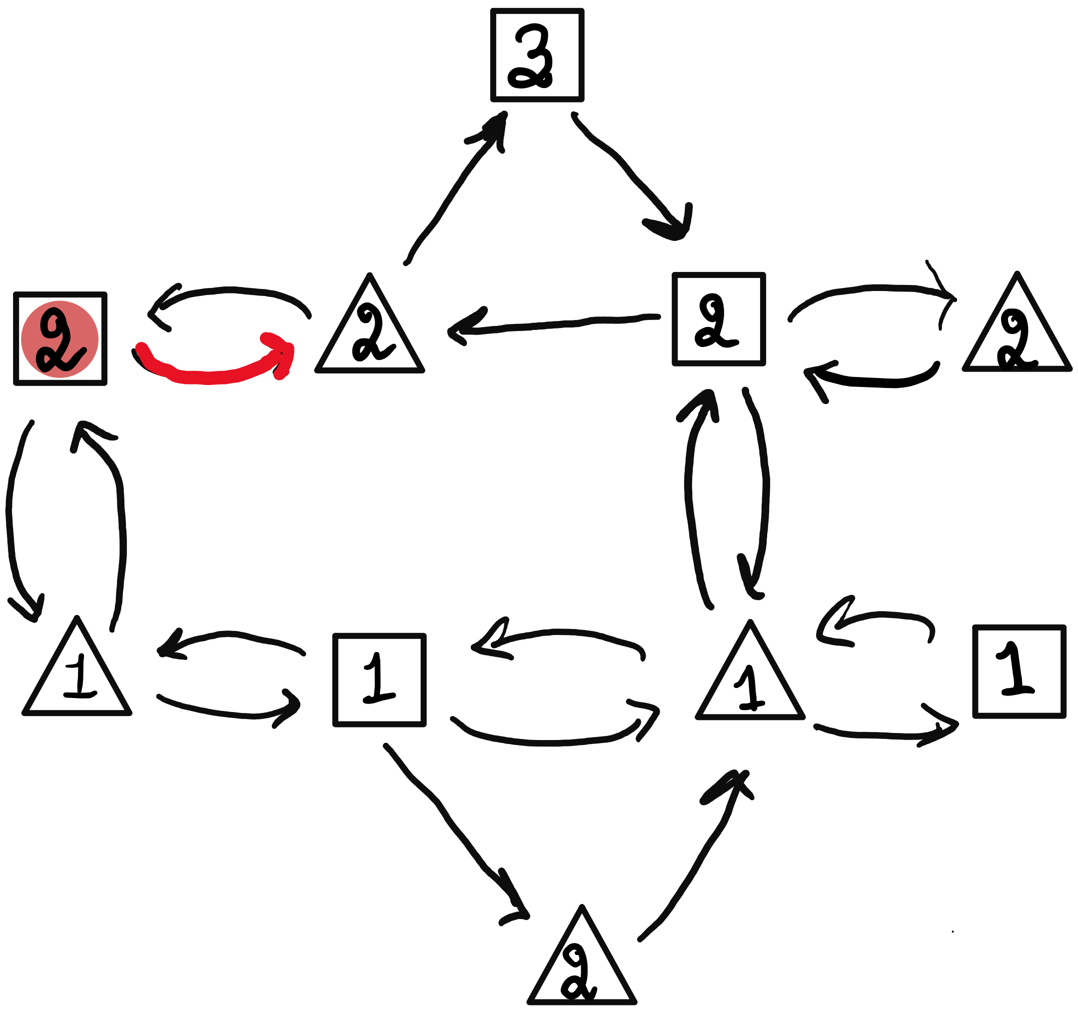
Parity Games

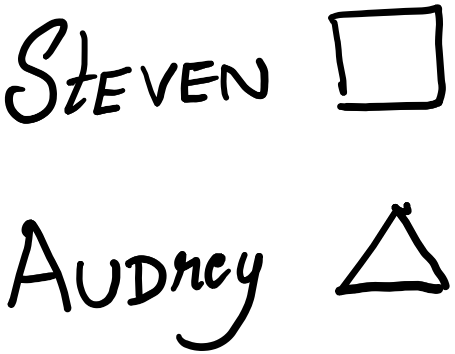


Steven □

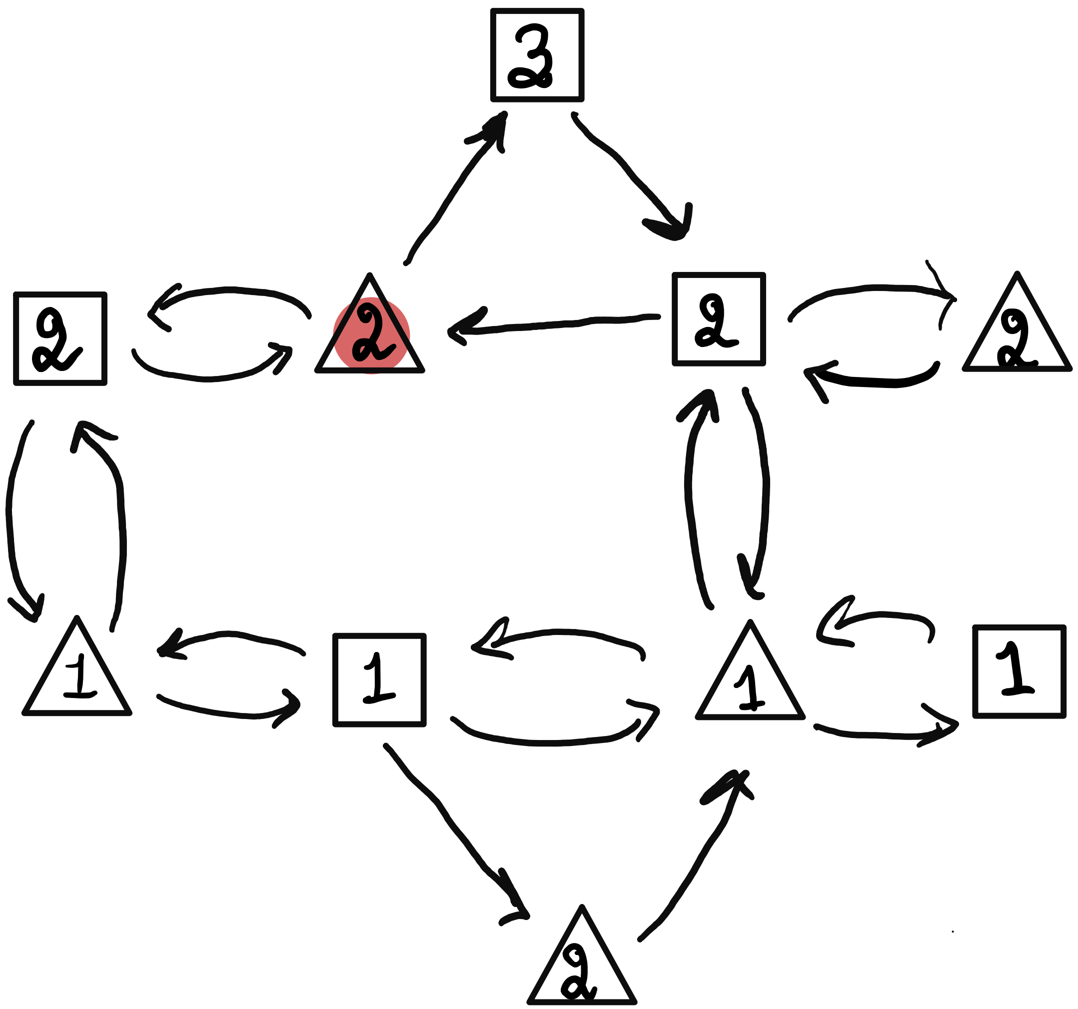
Audrey △

Parity Games





Parity Games



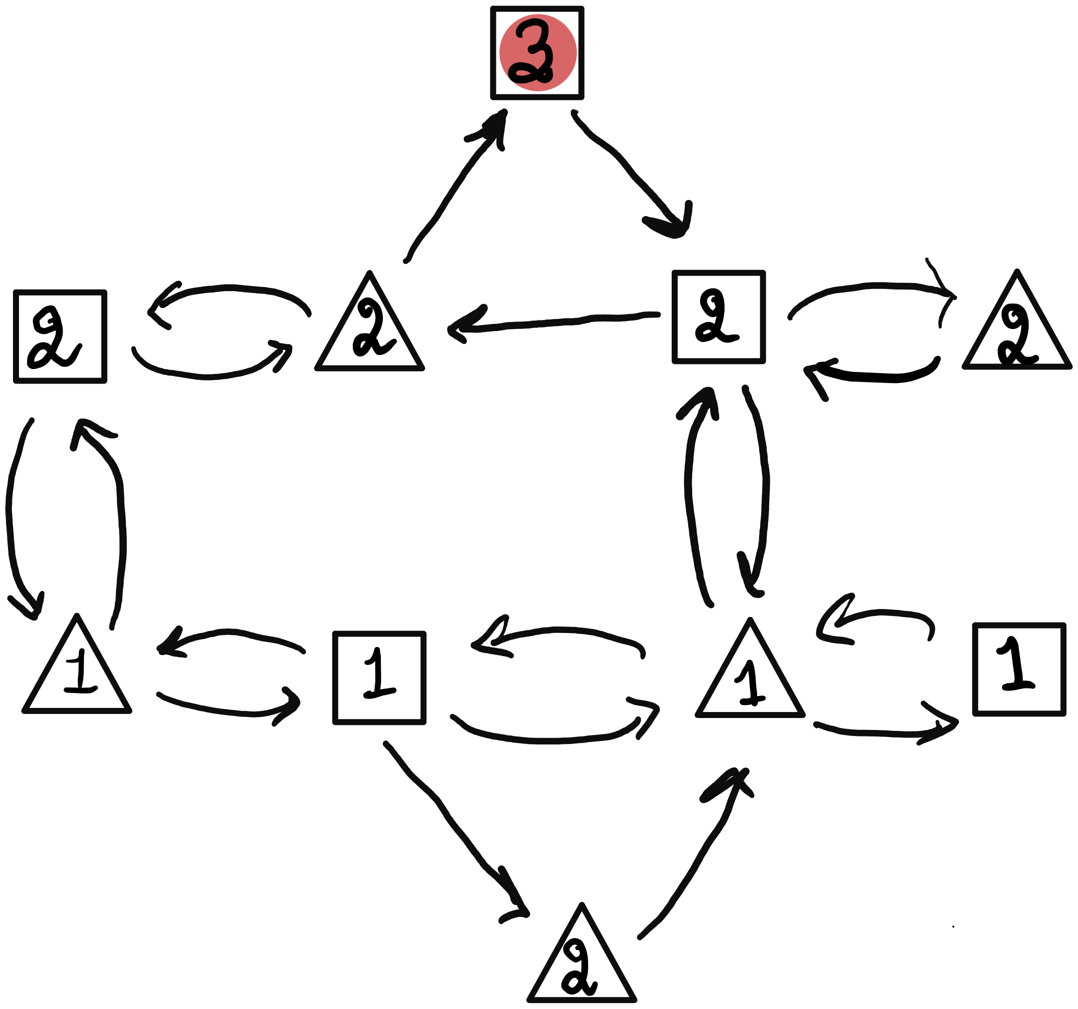
Play

2, 2

Steven □

Audrey △

Parity Games



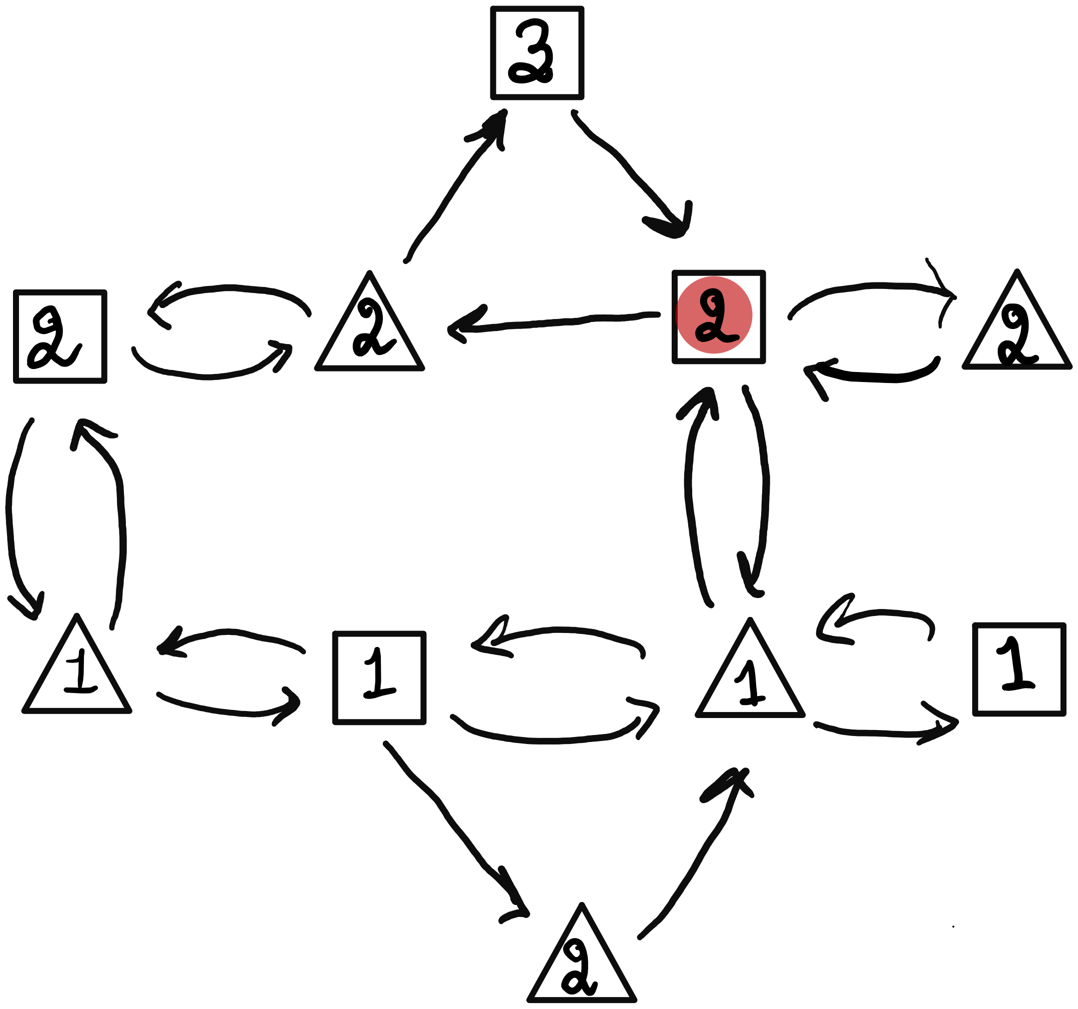
Play

2, 2, 3

Steven □

Audrey △

Parity Games



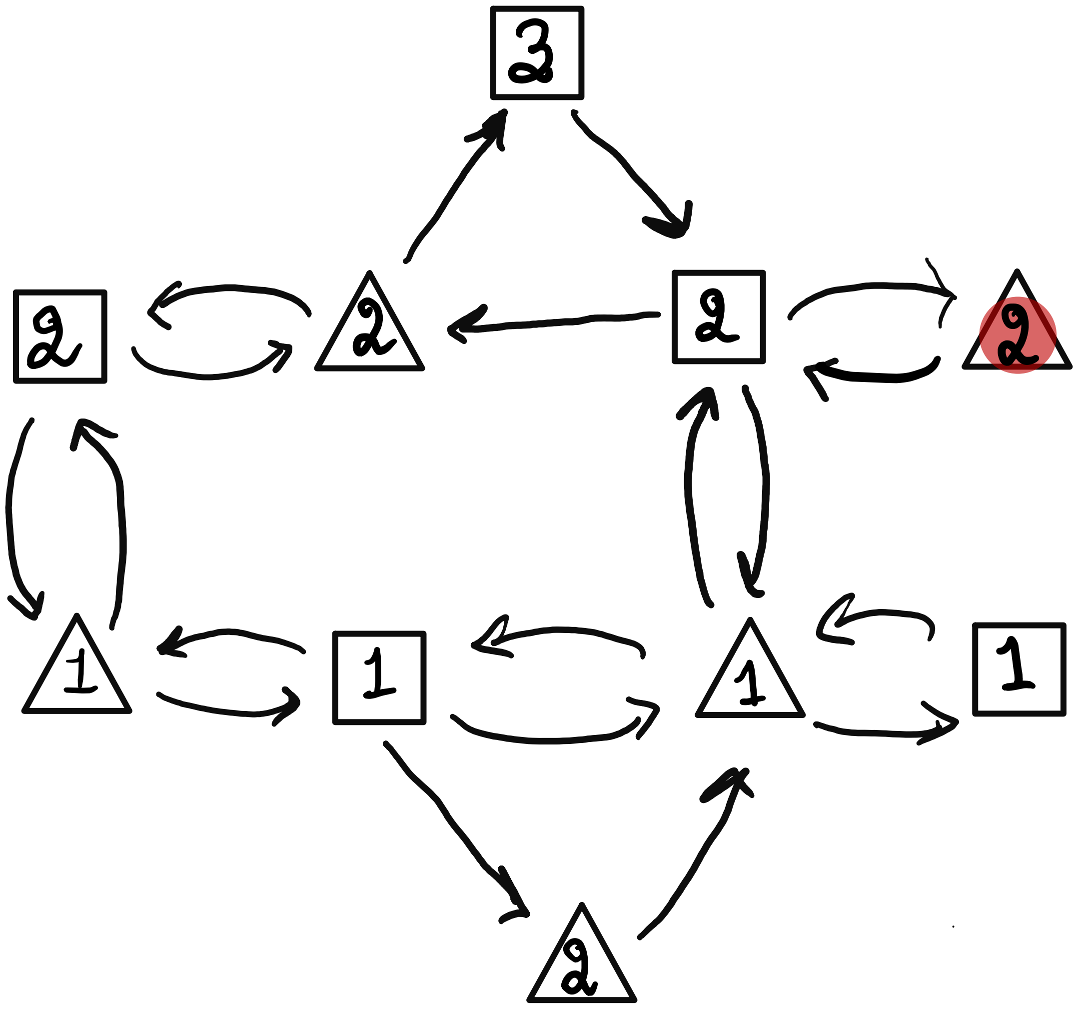
Play

2, 2, 3, 2

Steven □

Audrey △

Parity Games



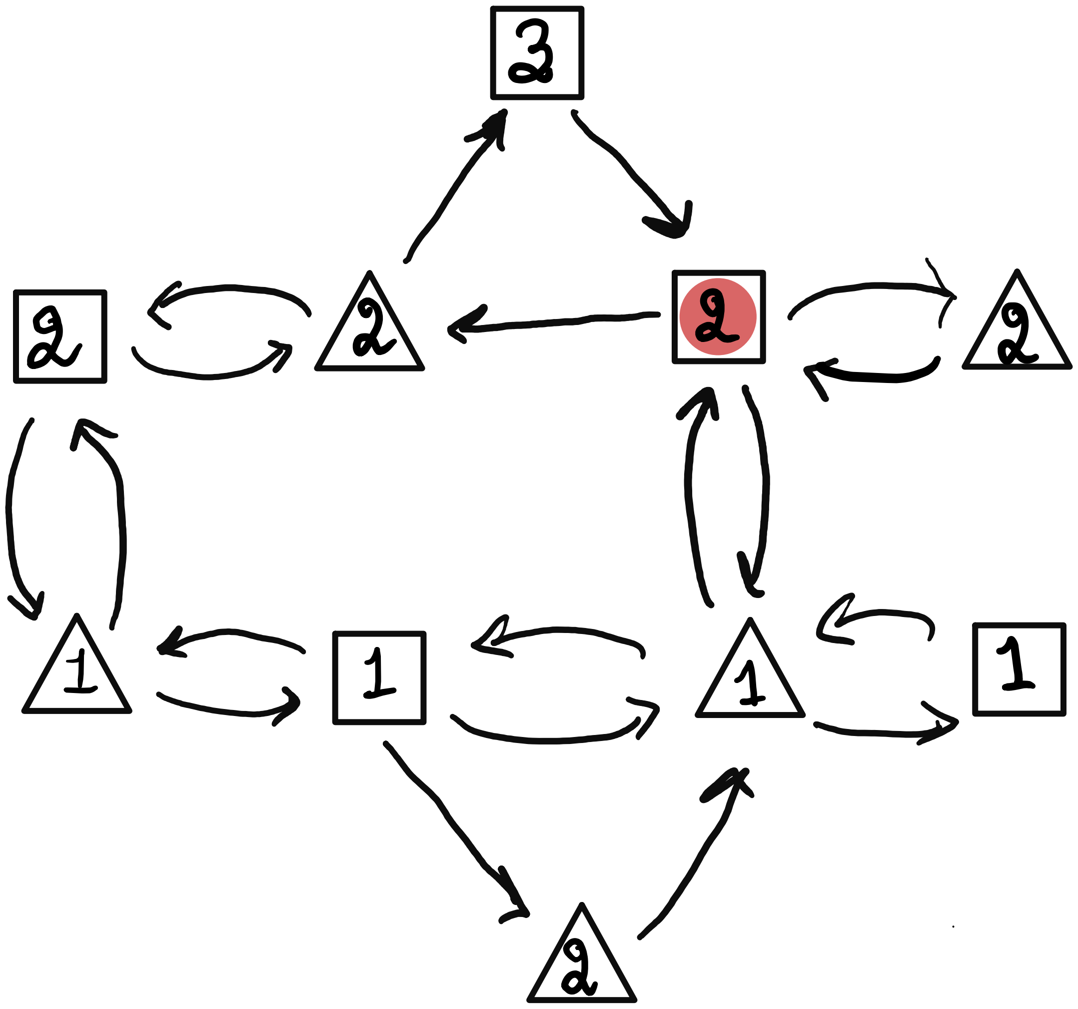
Play

2, 2, 3, 2, 2

Steven □

Audrey △

Parity Games



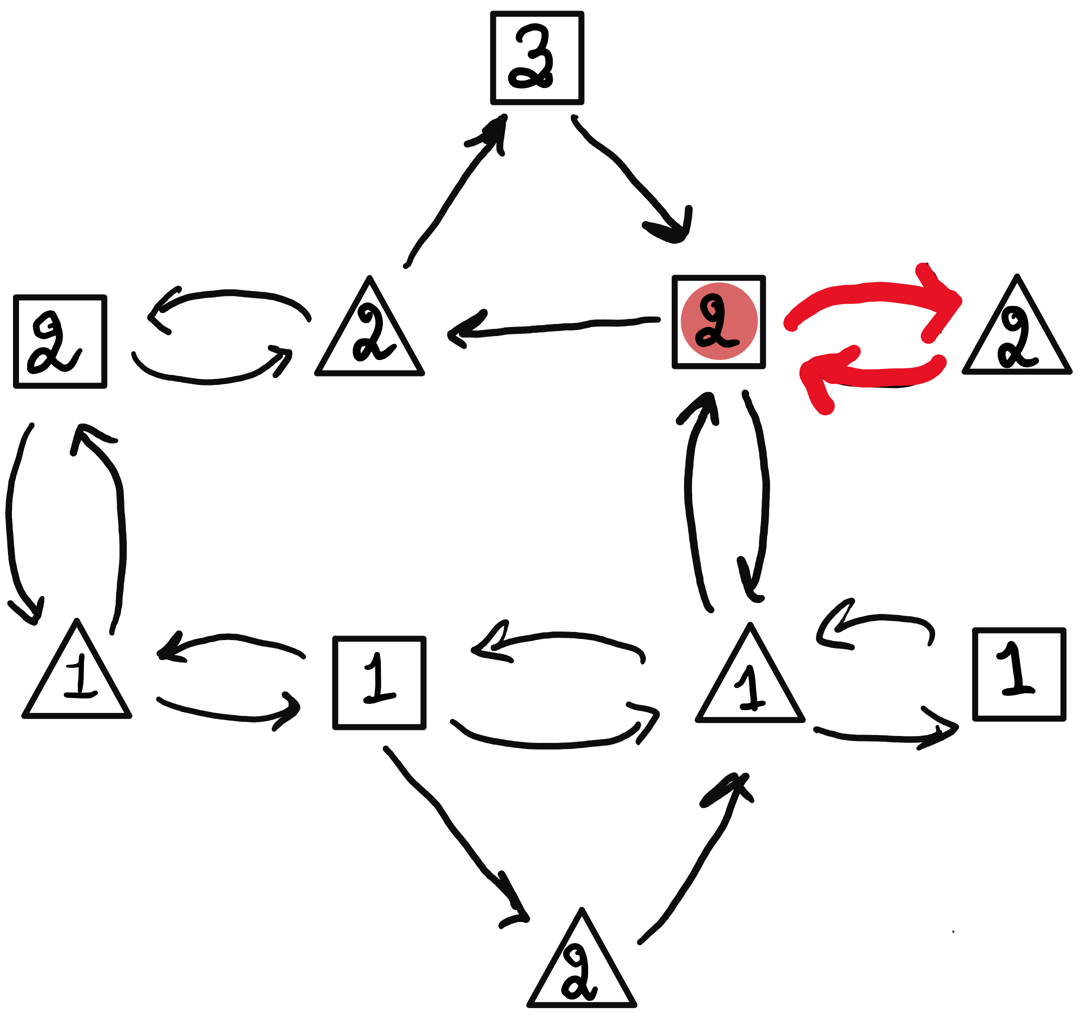
Play

2, 2, 3, 2, 2, 2

Steven □

Audrey △

Parity Games



Play

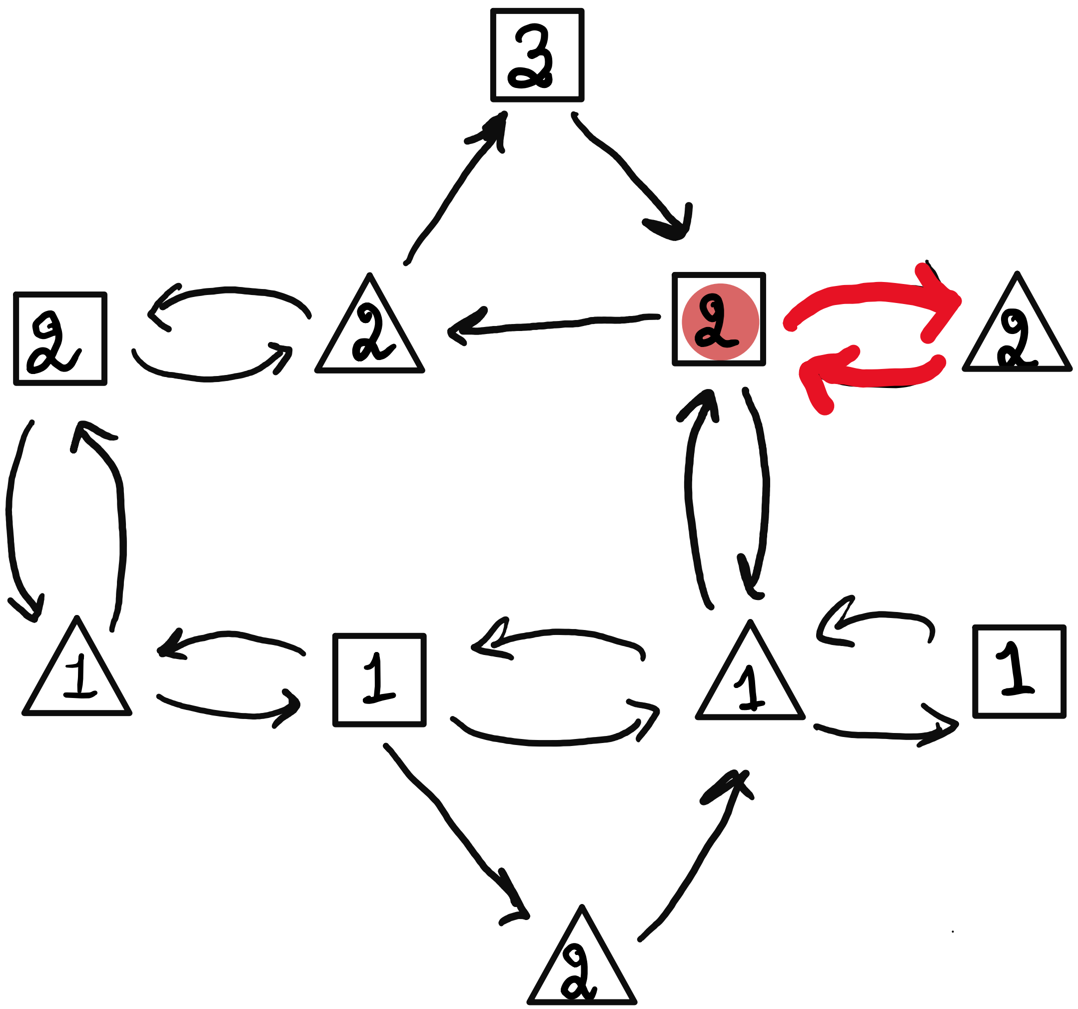
2, 2, 3, 2, 2, 2

... 2, 2, ...

Steven □

Audrey △

Parity Games



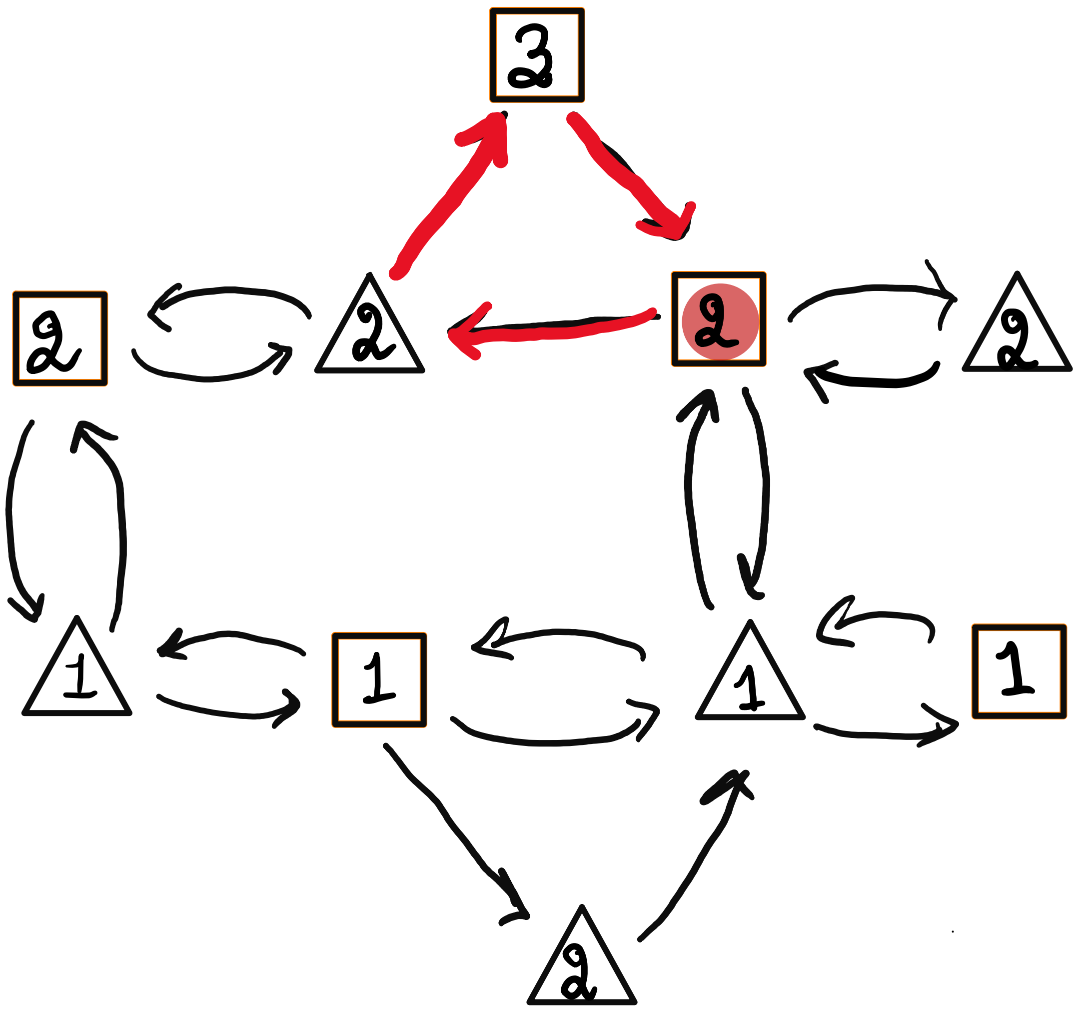
Winner :
parity of limsup

2, 2, 3, 2, 2, 2
... 2, 2, ...
- Steven Wins

Steven □

Audrey △

Parity Games



Winner: parity of limsup

2, 2, 3, 2, 2, 2
... 2, 2, ...

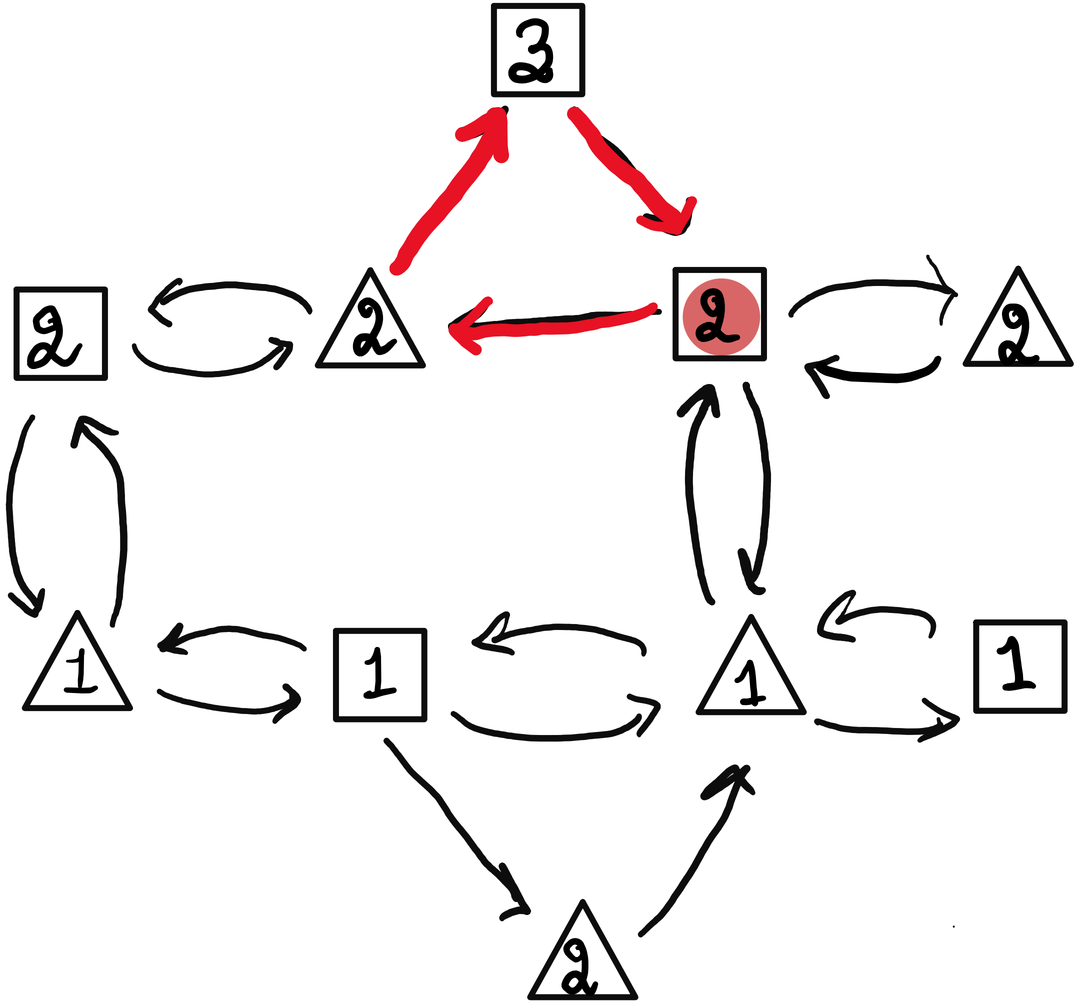
– Steven Wins

2, 2, 3, 2, 2, 3, 2,
... , 2, 3, 2, ...

Steven □

Audrey △

Parity Games



Steven □

Audrey △

Winner: parity of limsup

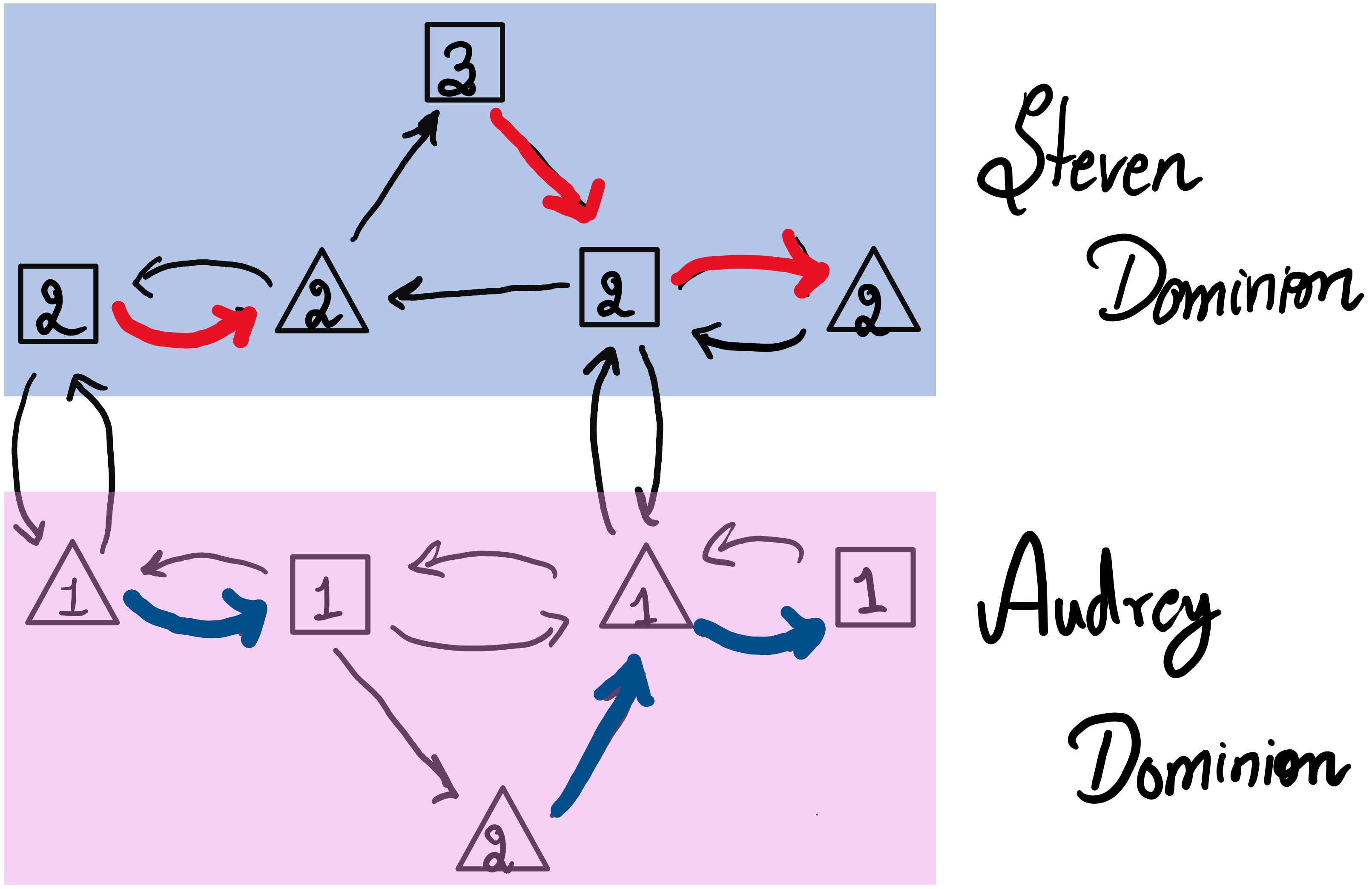
2, 2, 3, 2, 2, 2 ... 2, 2, ...

– Steven Wins

2, 2, 3, 2, 2, 3, 2, ... , 2, 3, 2, ...

– Audrey Wins

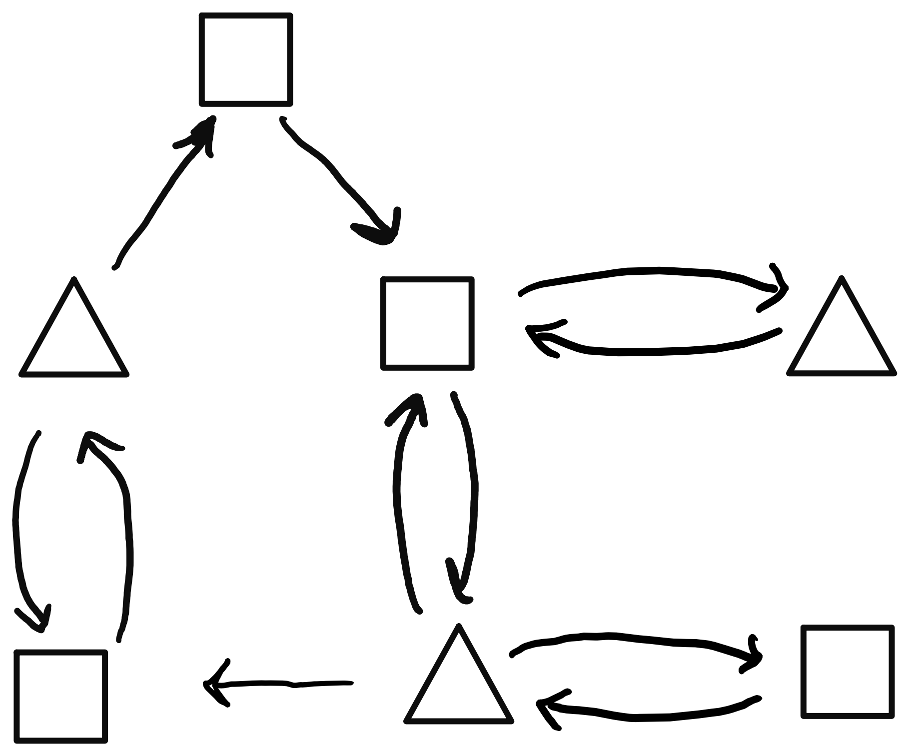
Parity Games

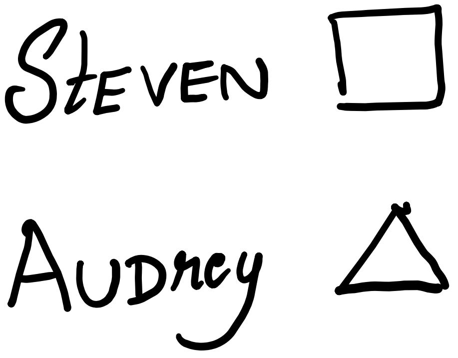


Steven □

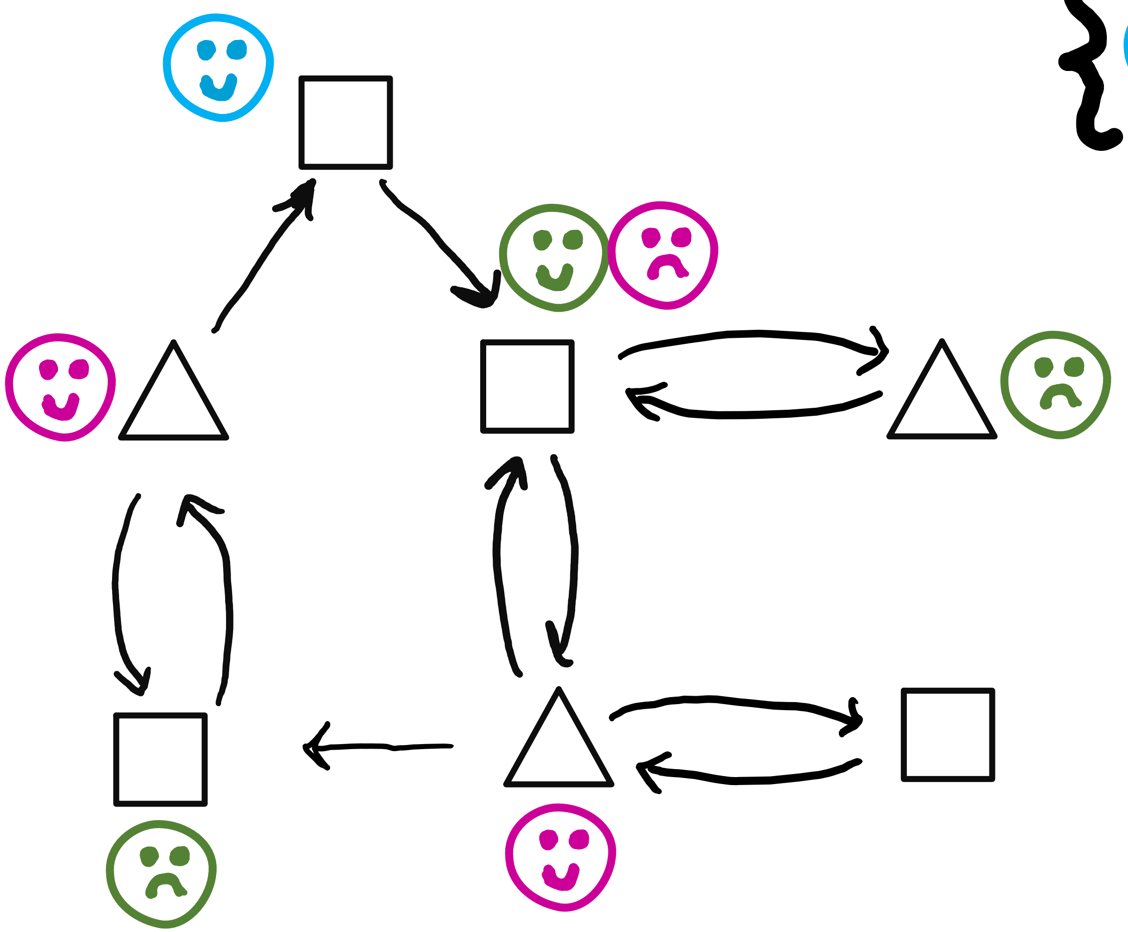
Audrey △

Rabin Games





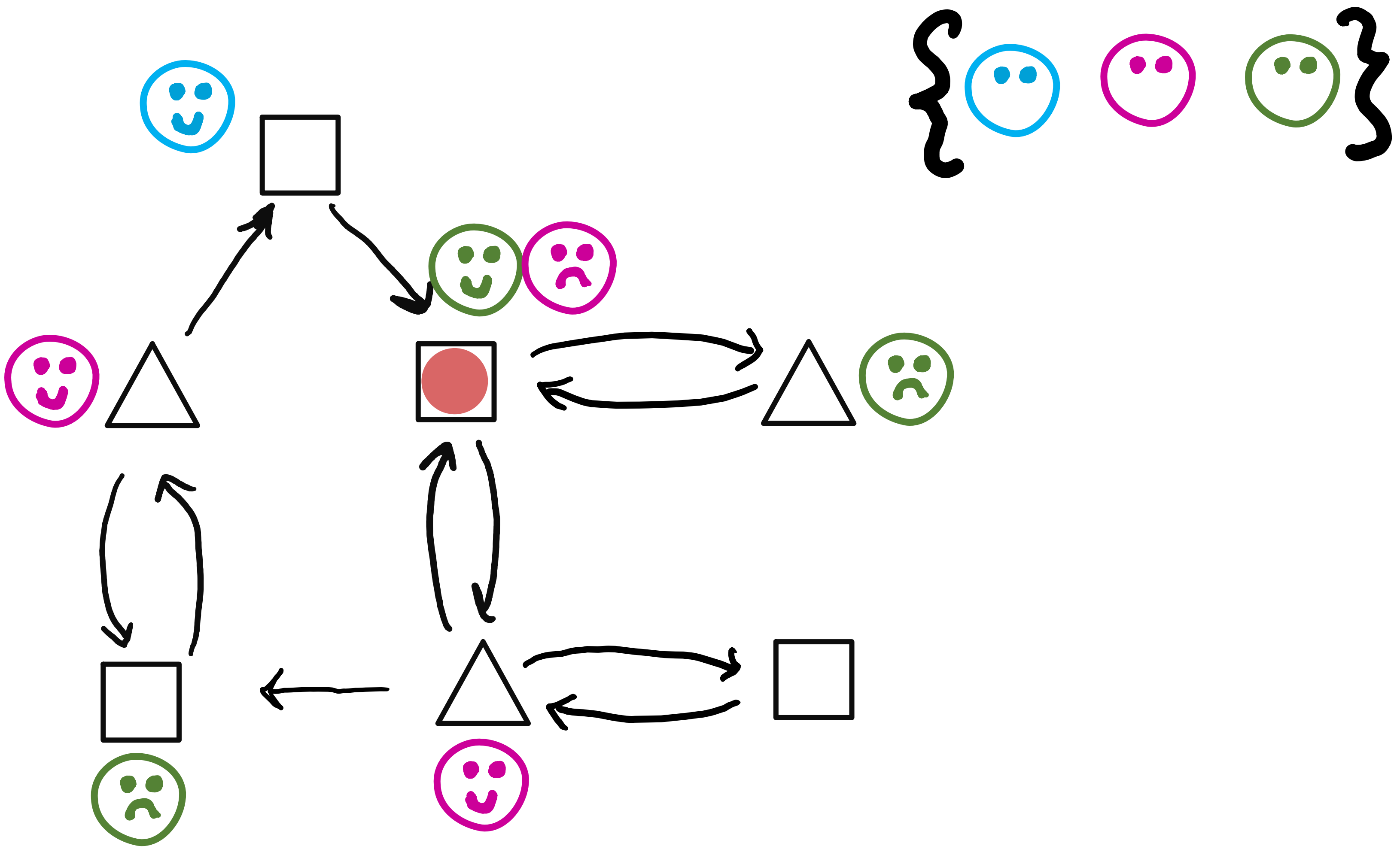
Rabin Games



Steven □

Audrey △

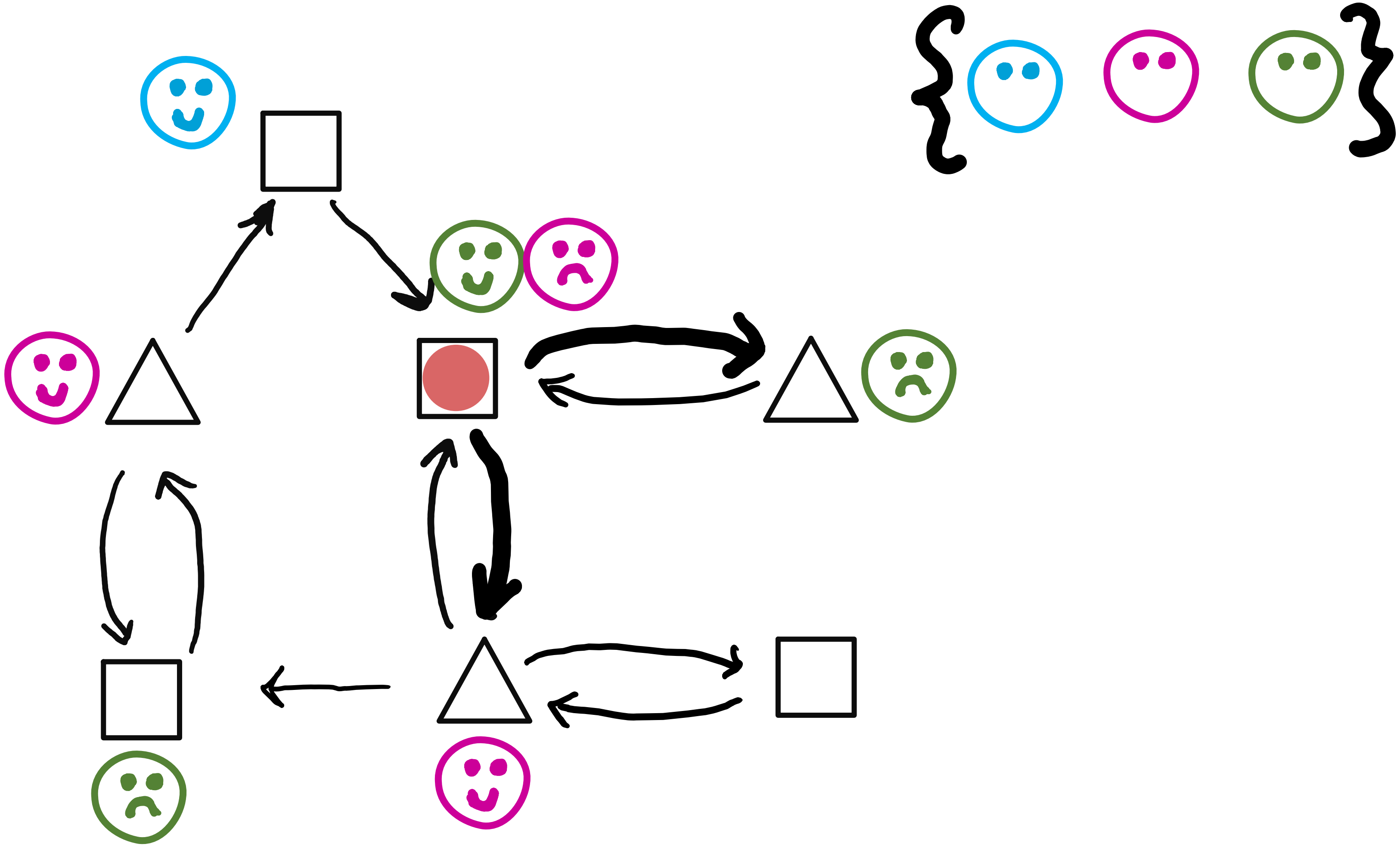
Rabin Games



Steven □

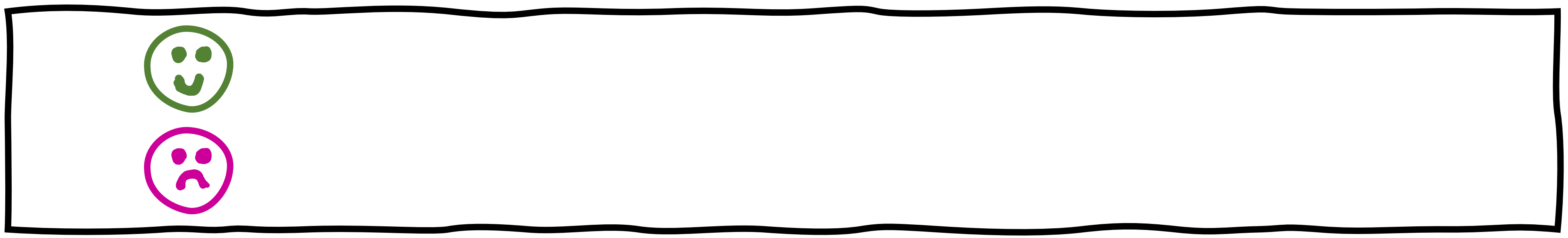
Audrey △

Rabin Games

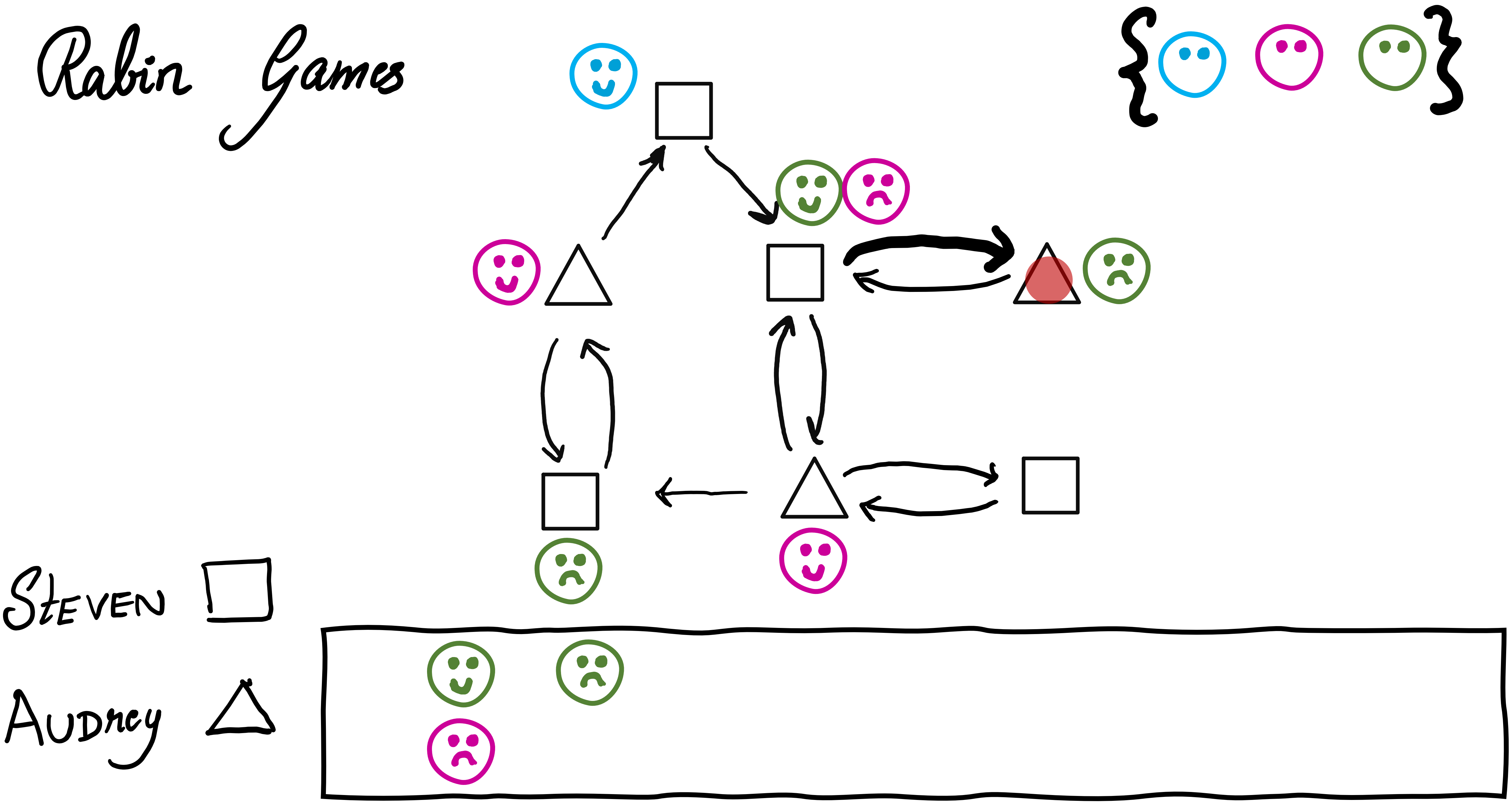


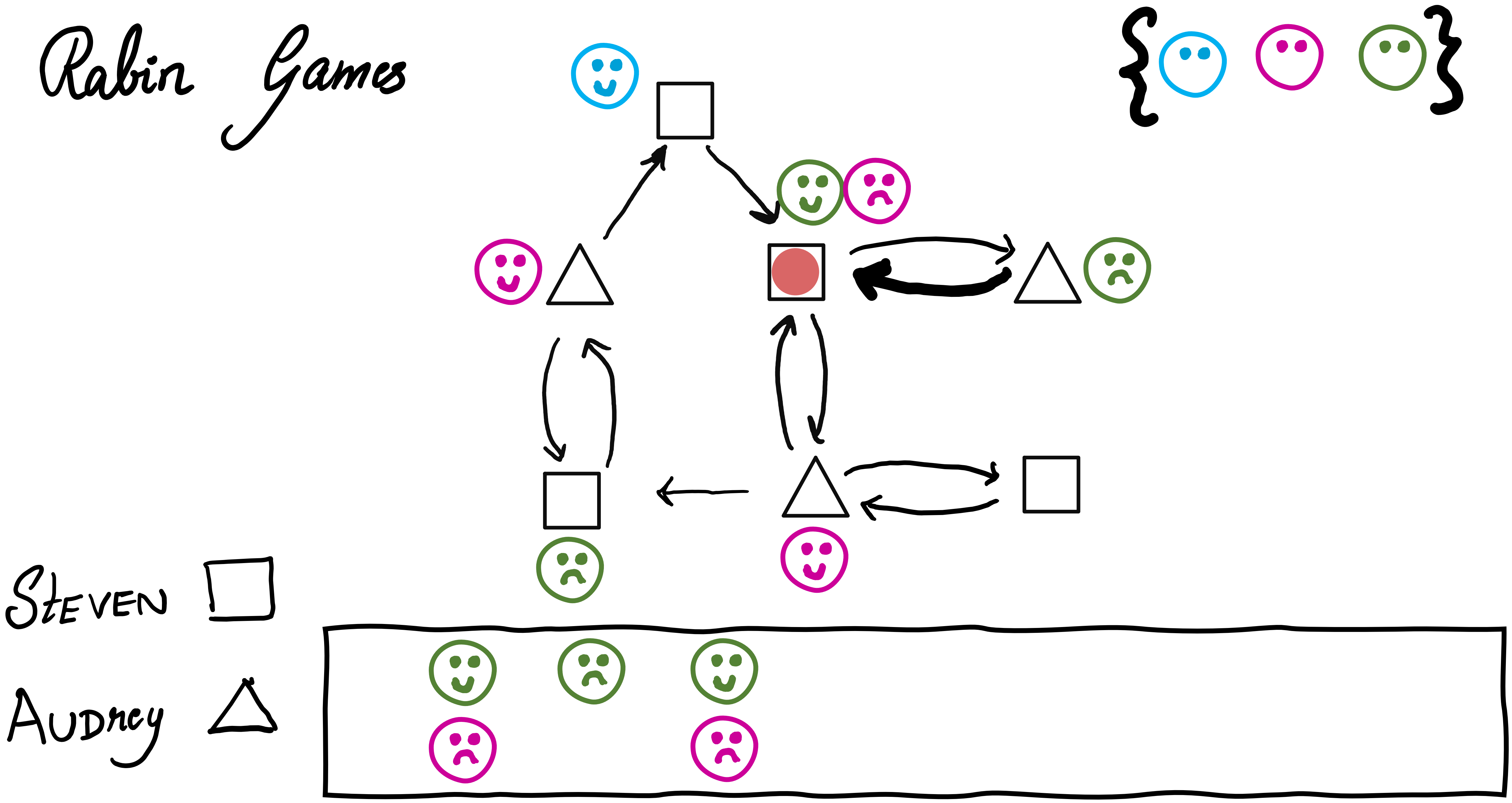
Steven □

Audrey △

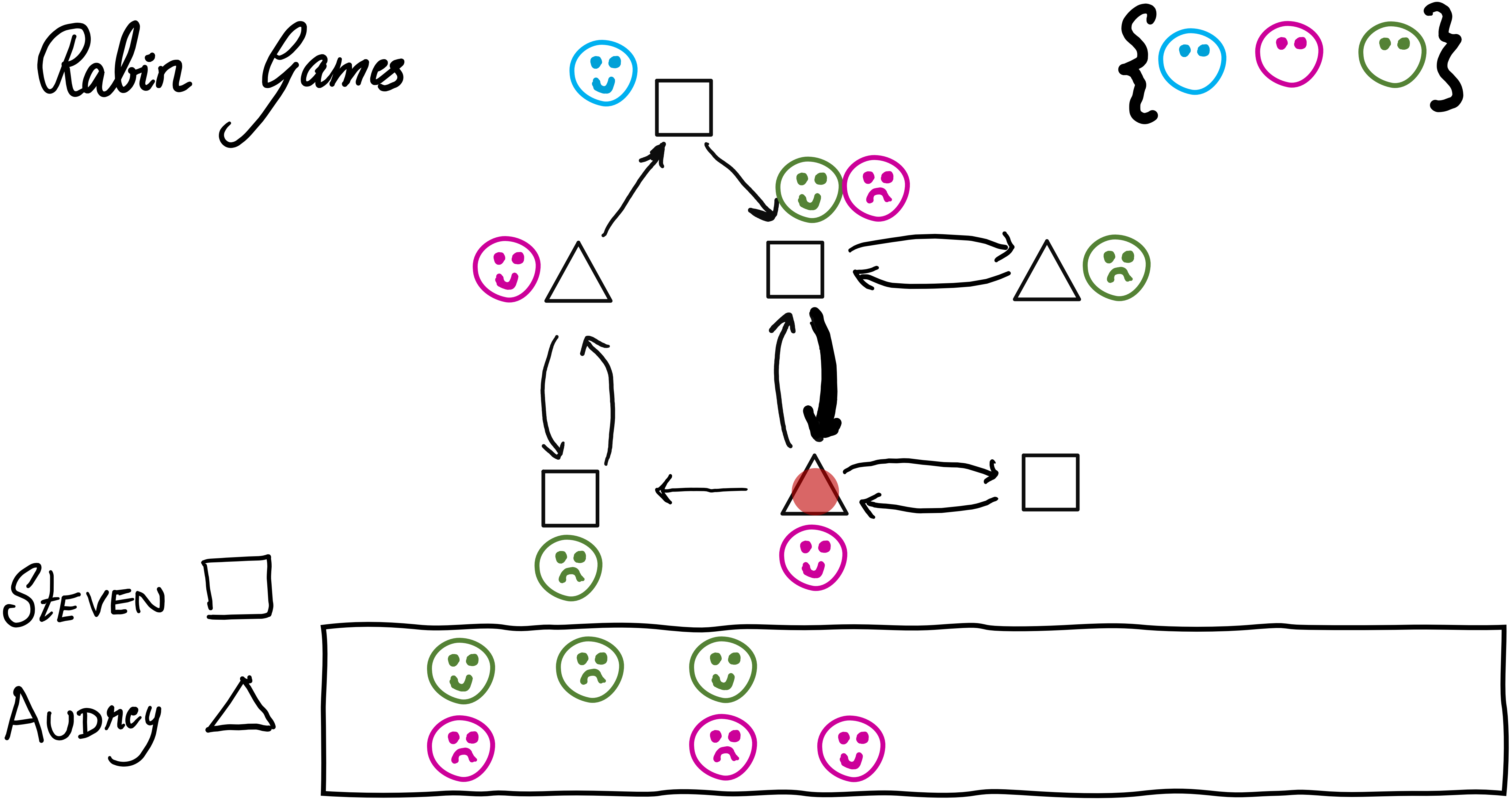


Rabin Games

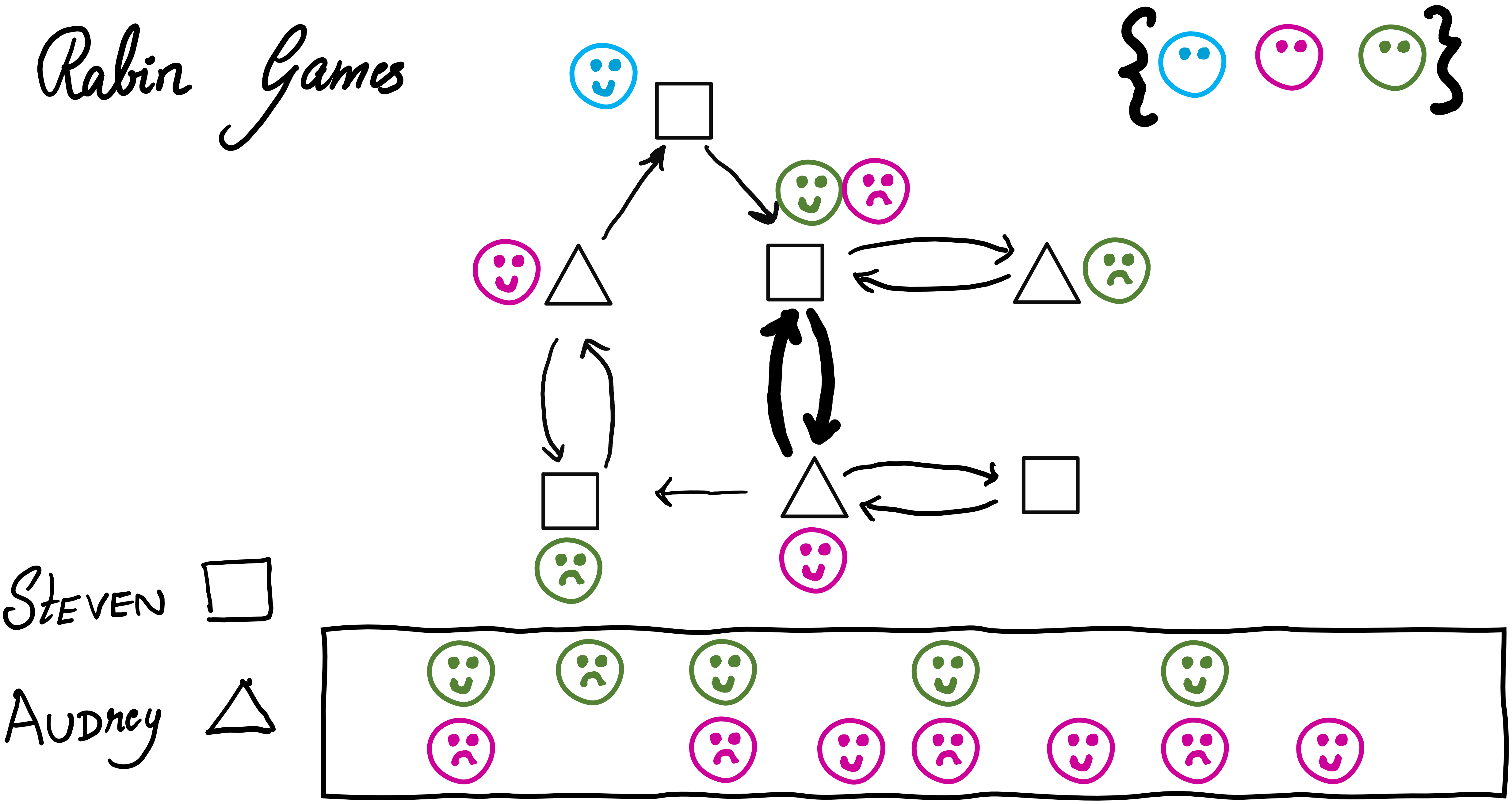



Rabin Games
Steven
Audrey

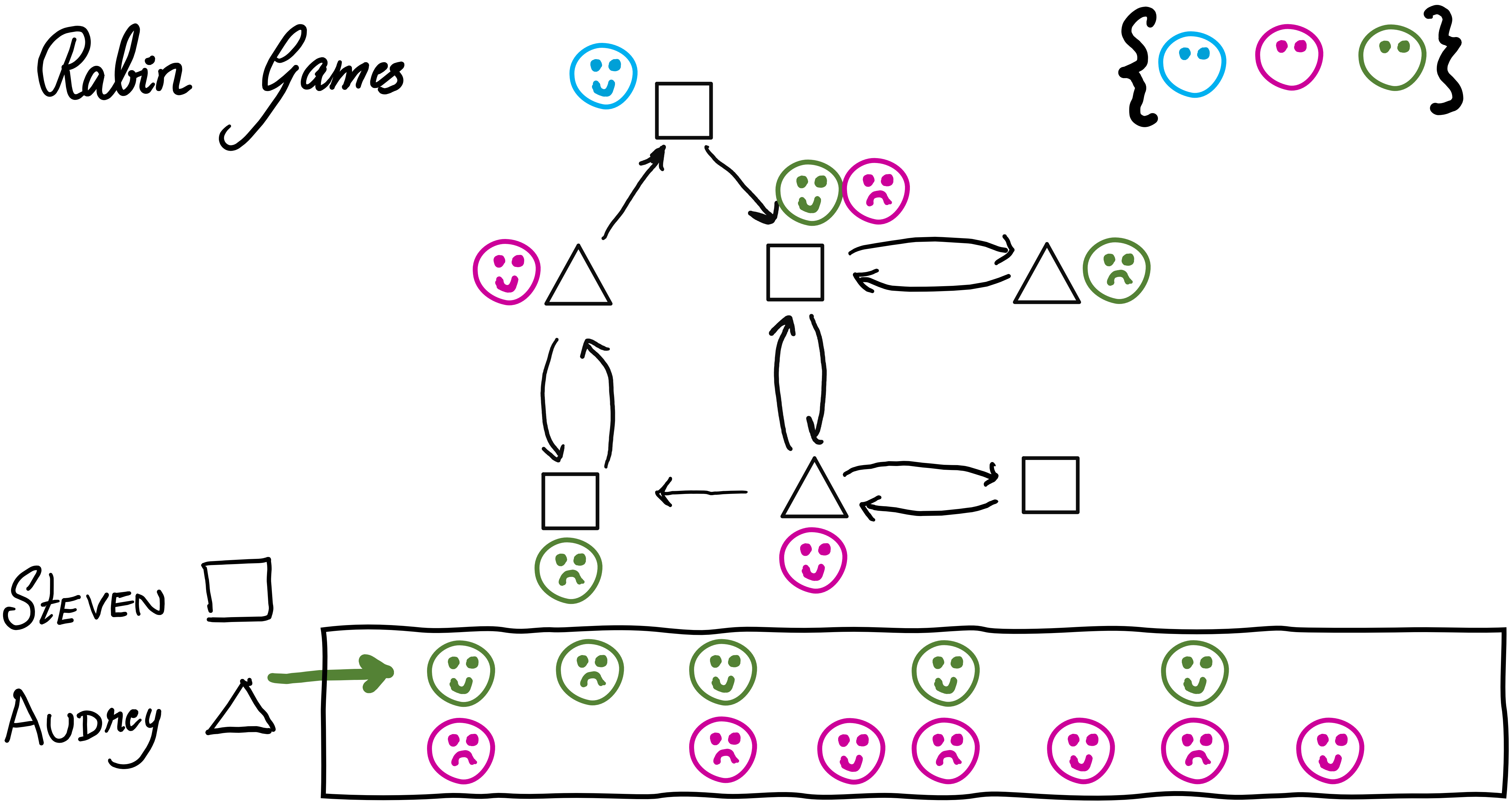
Rabin Games



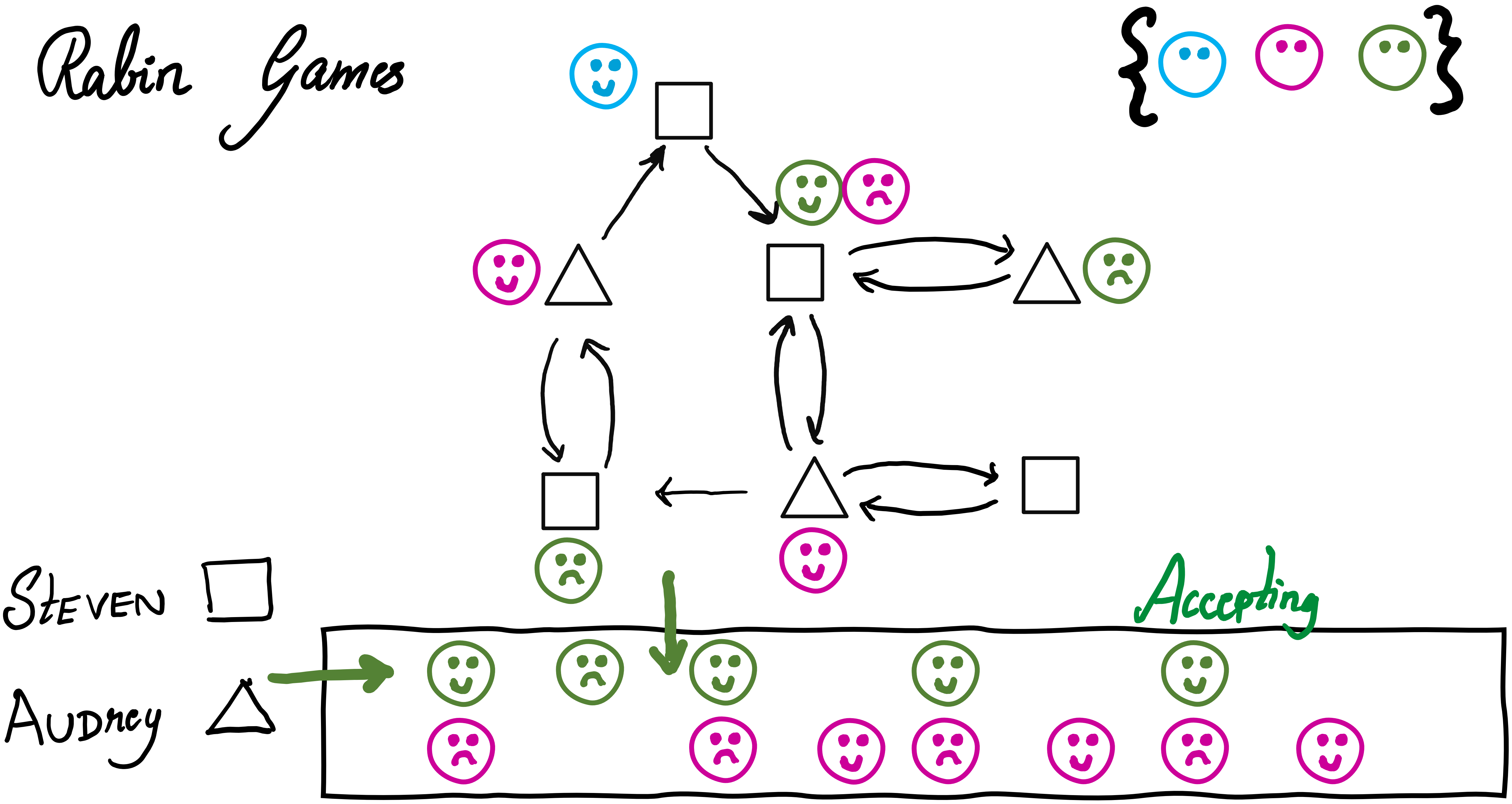
Rabin Games



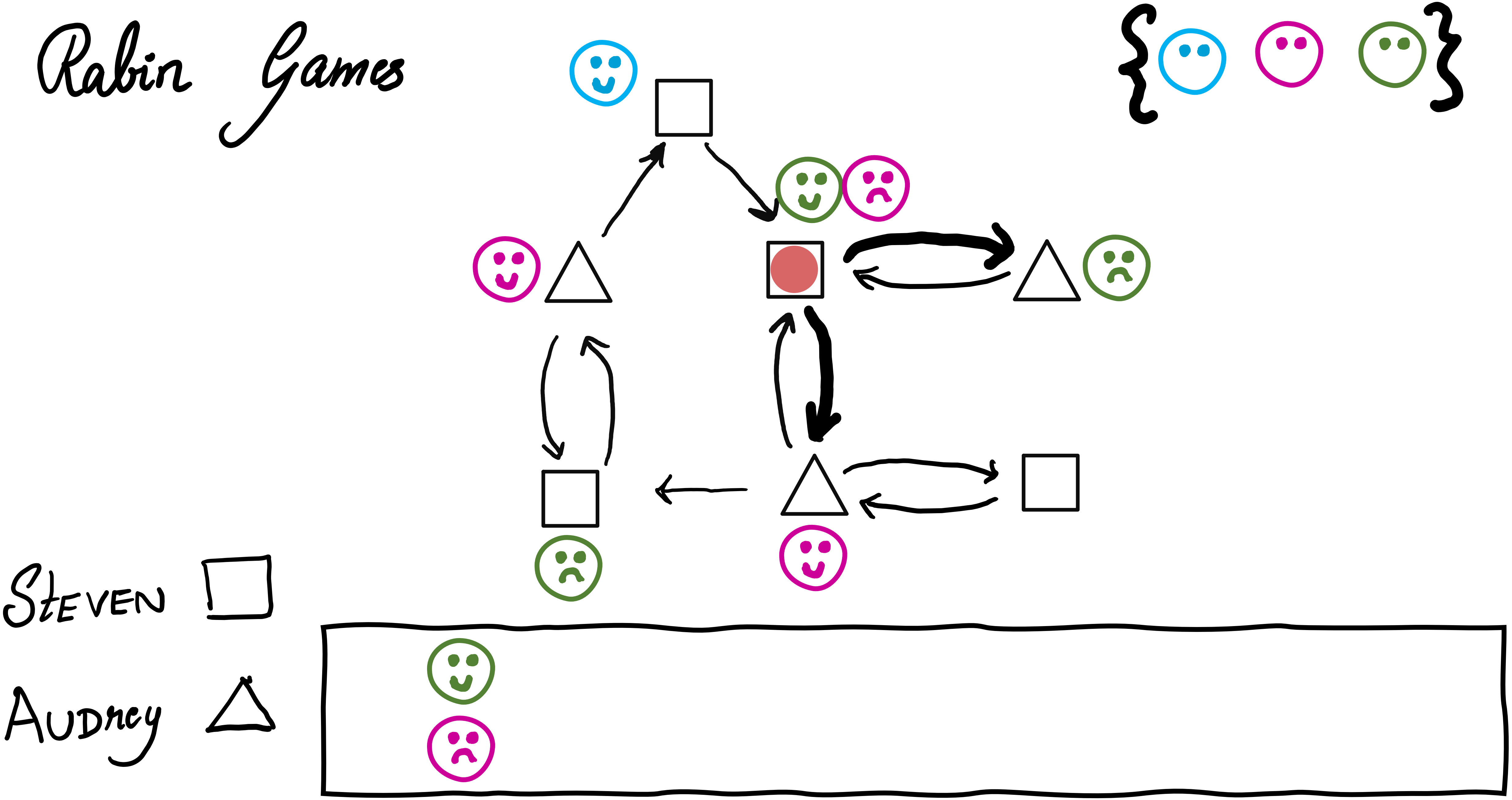
Rabin Games



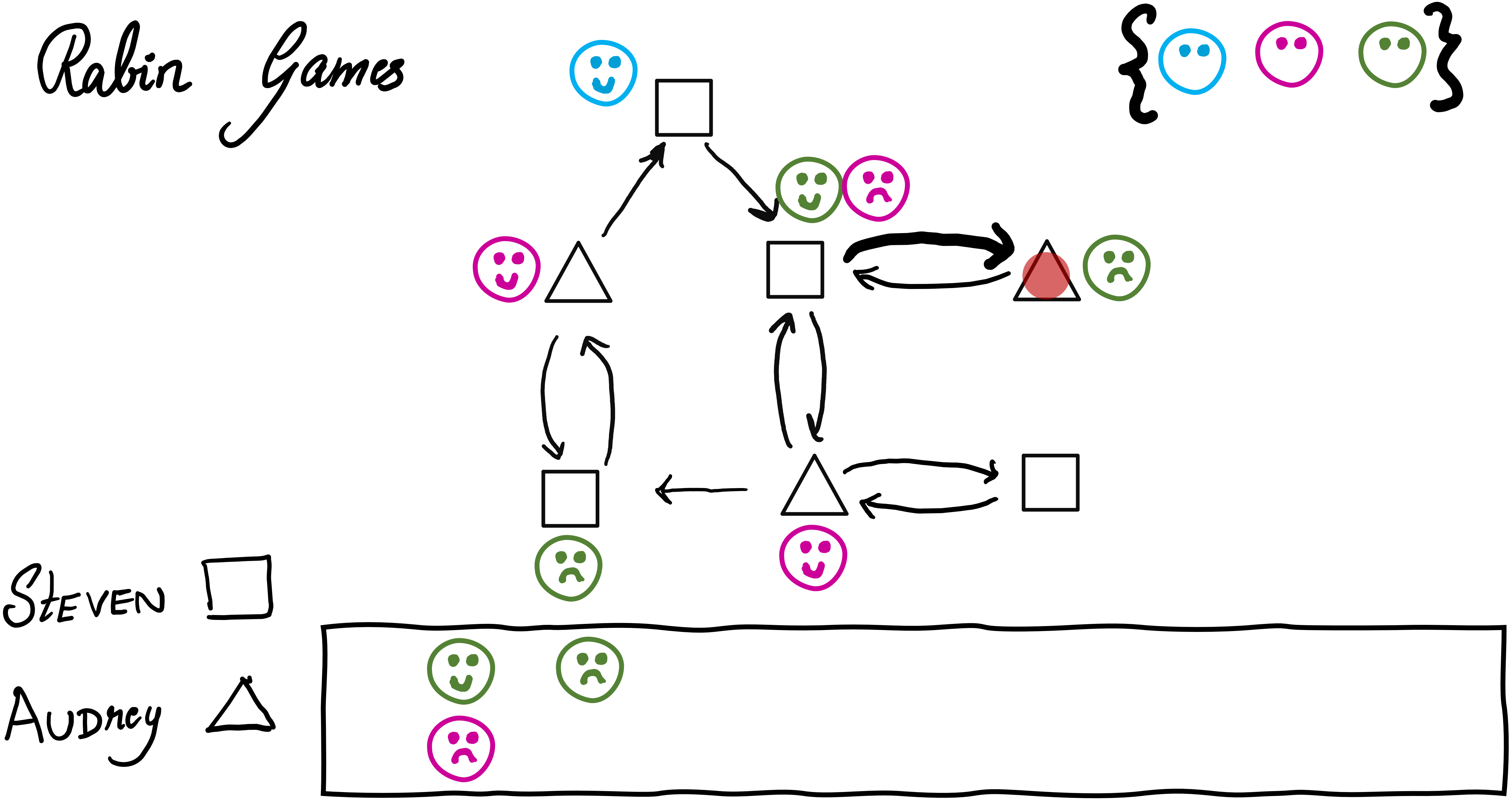
Rabin Games



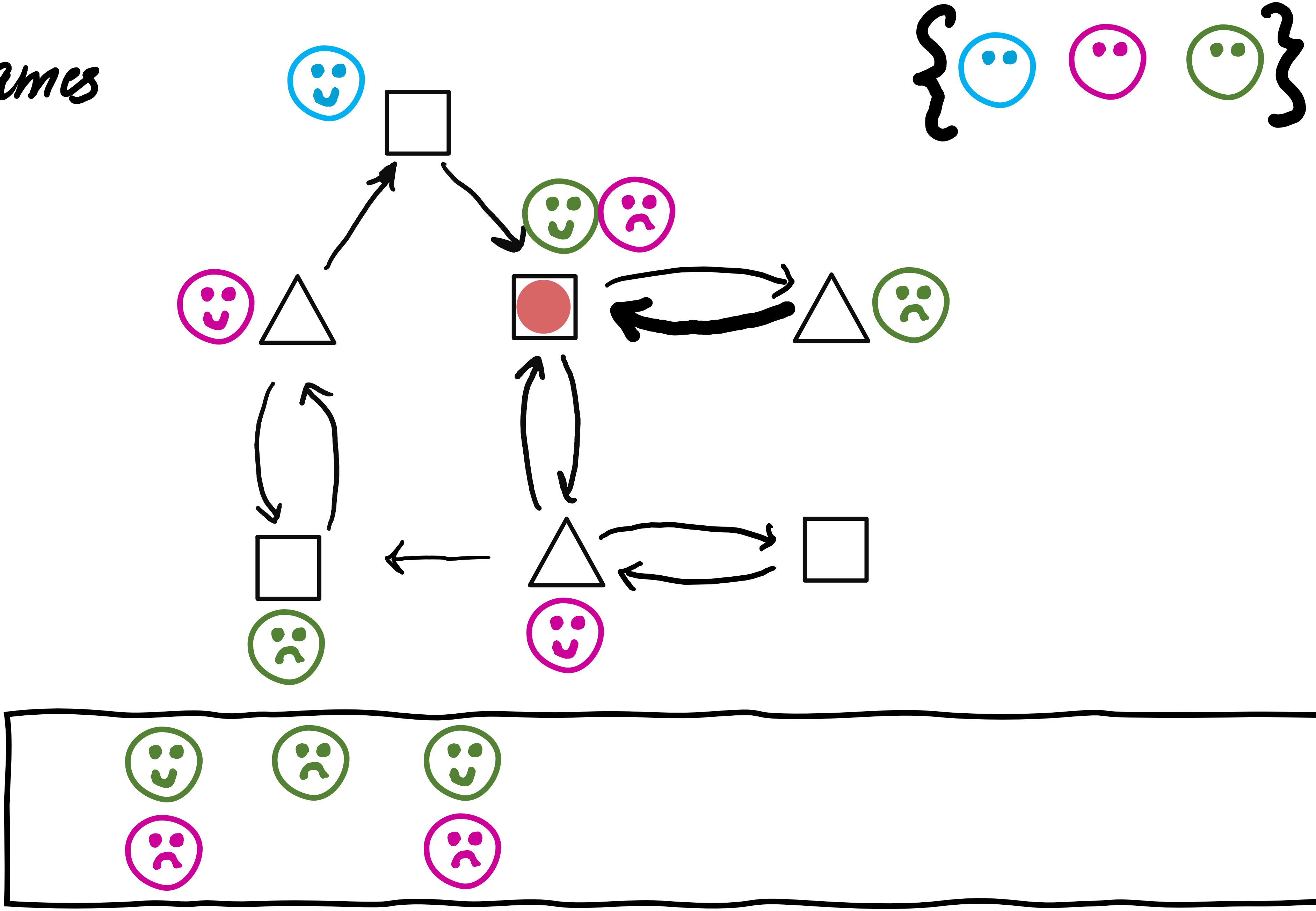
Rabin Games



Rabin Games



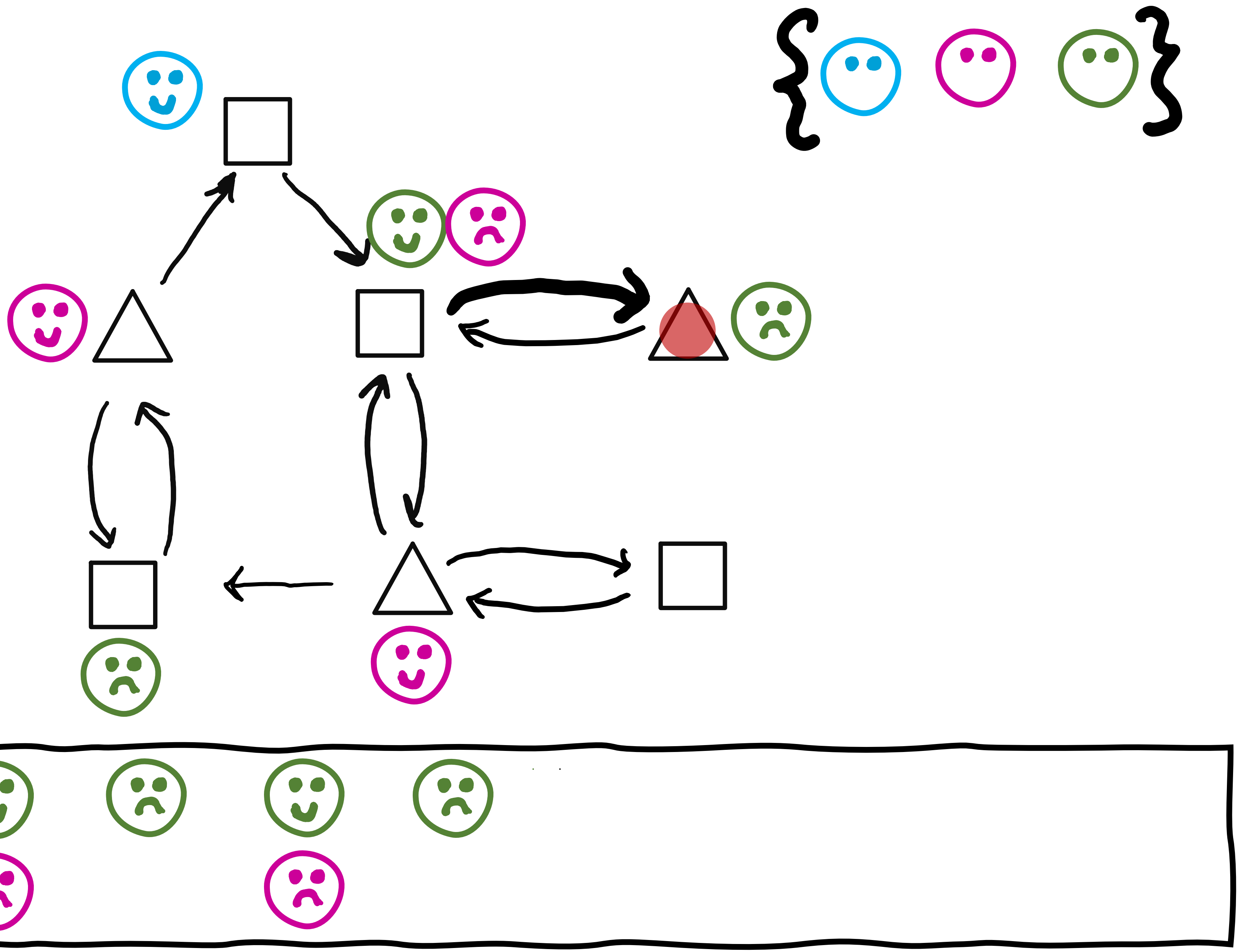
Rabin Games



Steven □

Audrey △

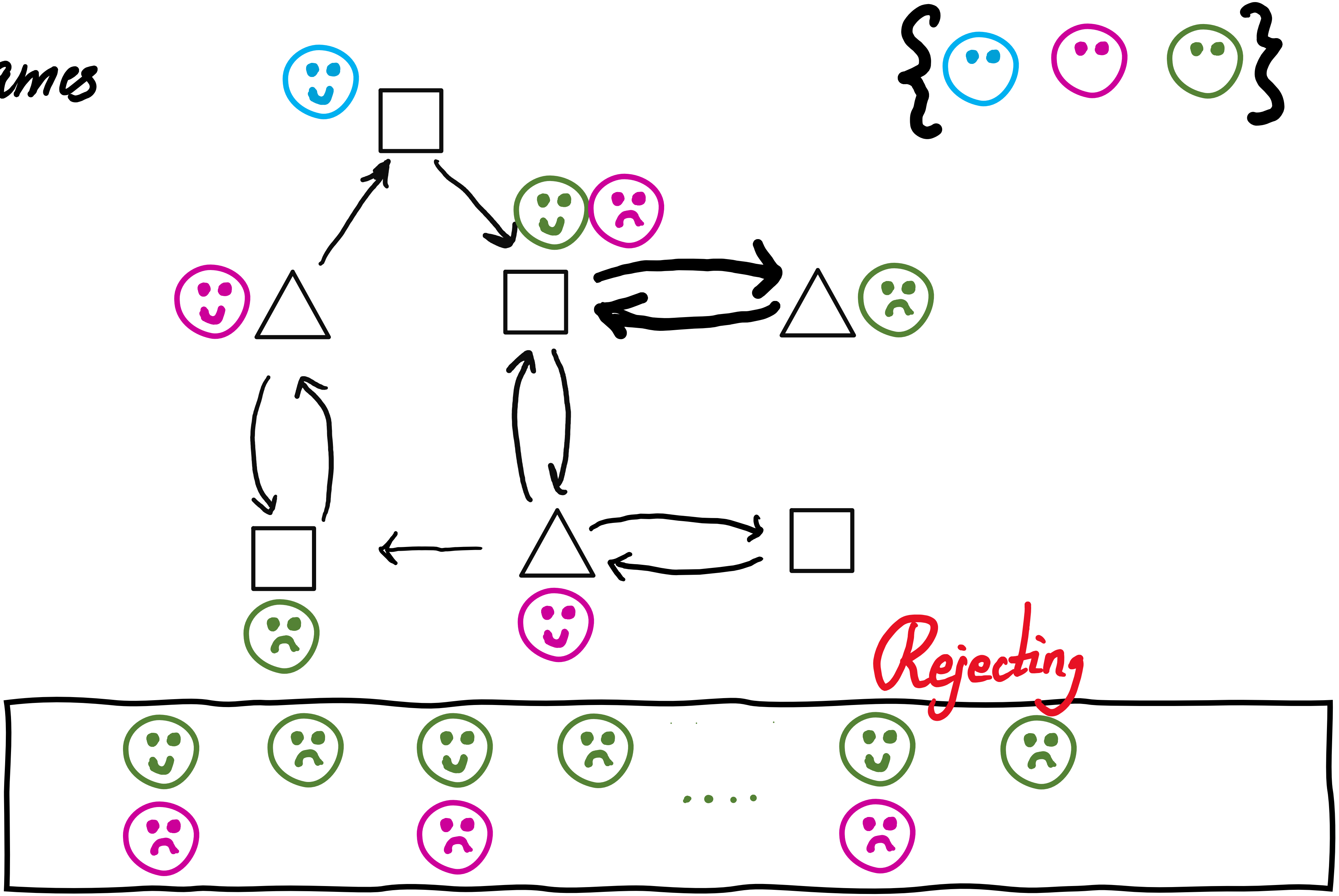
Rabin Games



Steven □

Audrey △

Rabin Games



Steven □

Audrey △

Does Steven win from a given vertex?

Parity Games

UP ∩ co-UP

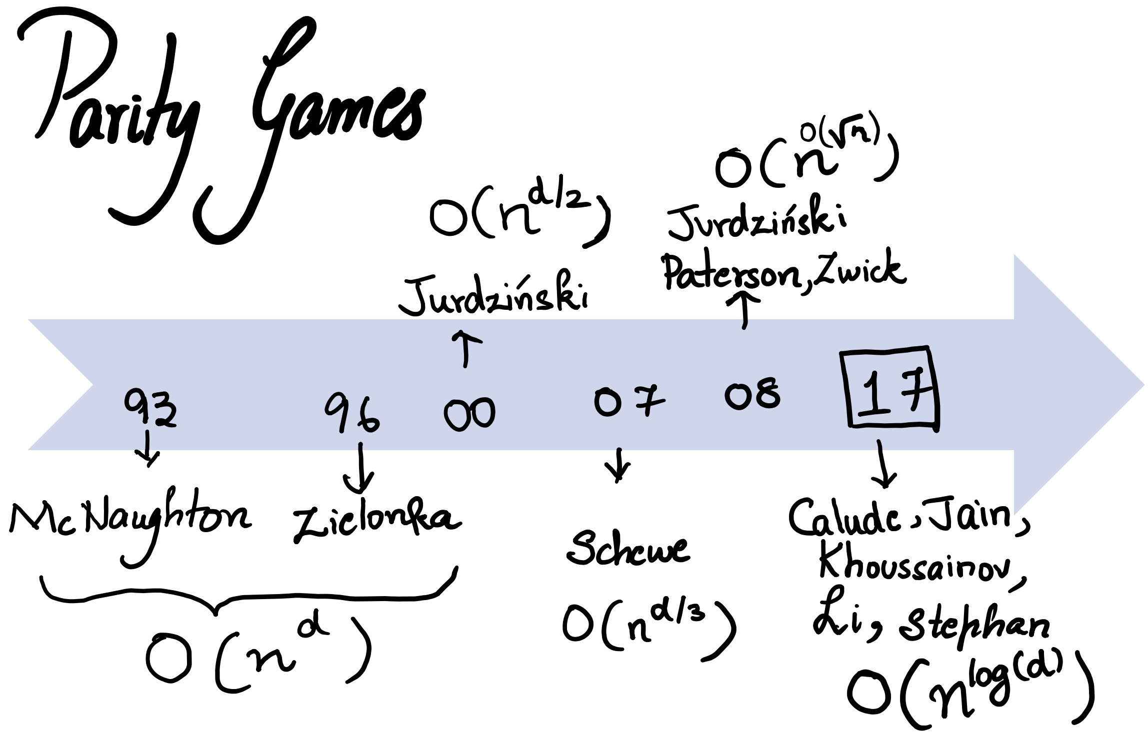
Quasi-polynomial time

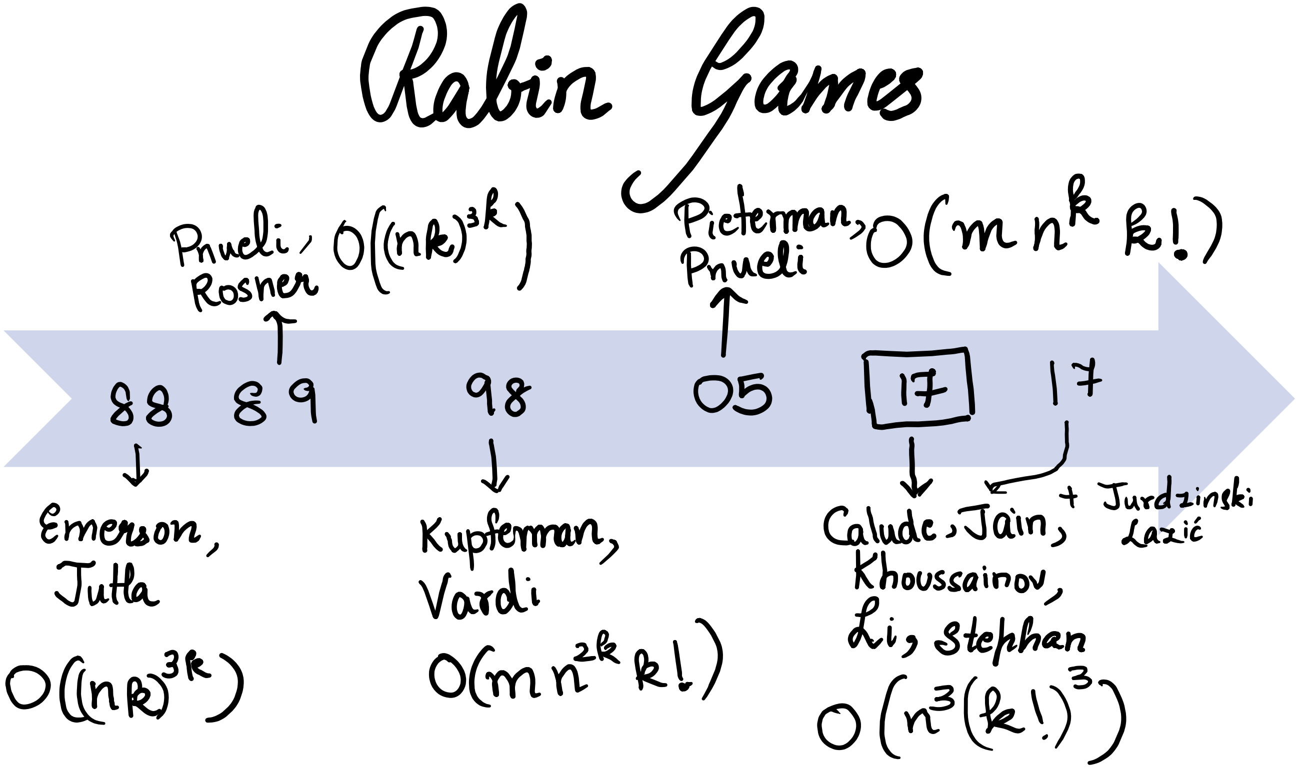
$O(n^{\log(d)+O(1)})$

Rabin Games

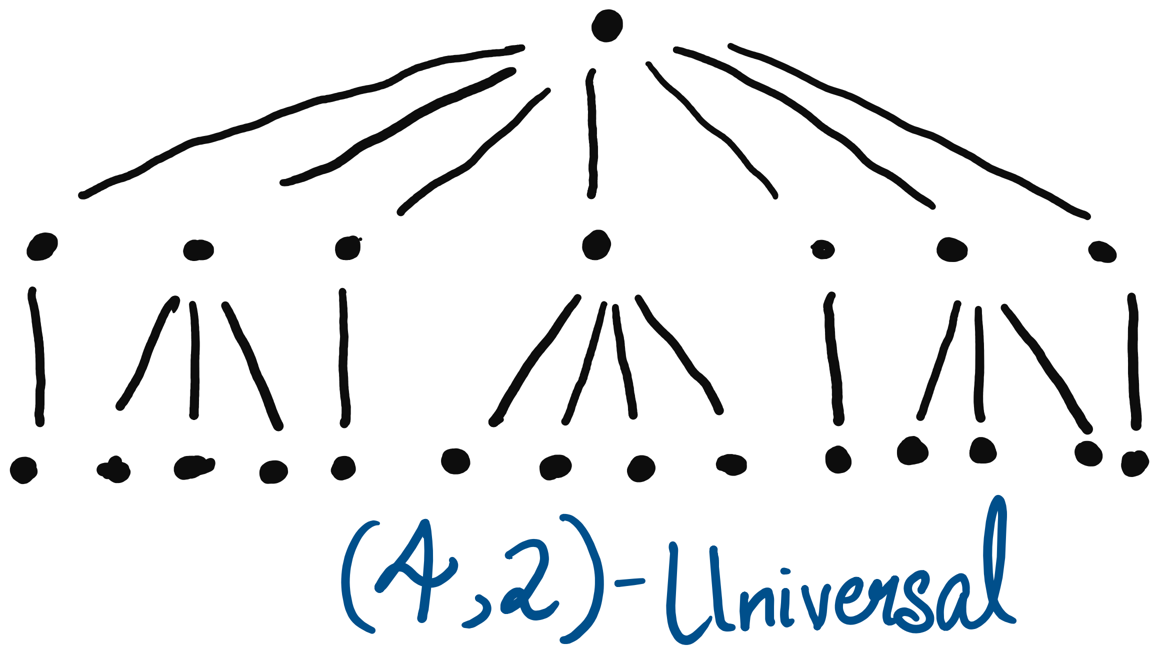
NP-complete

Does Steven win from a given vertex?



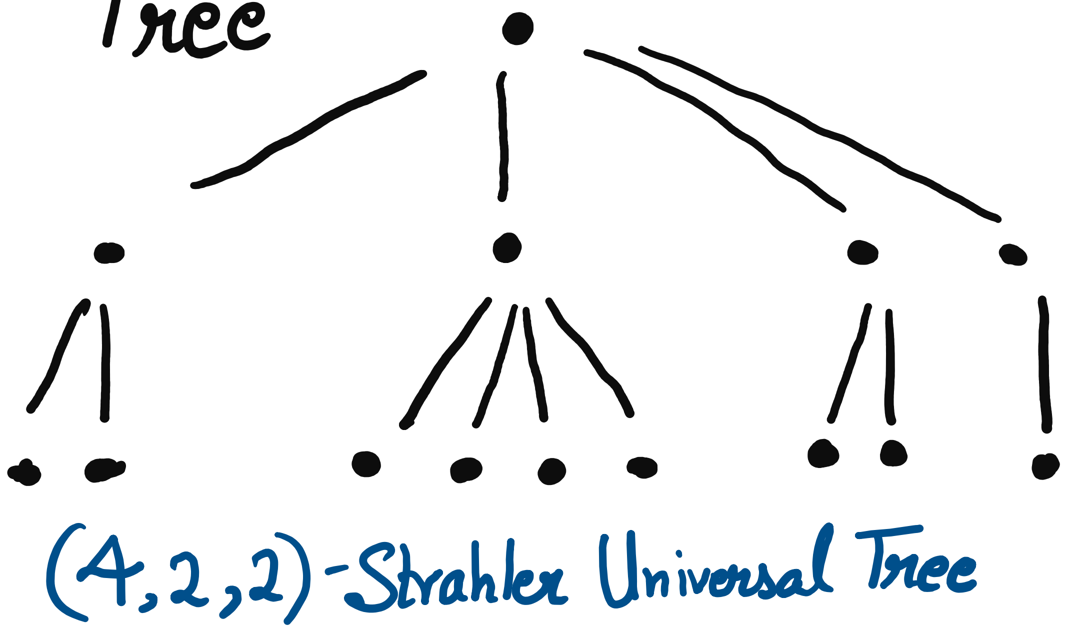


(n,h) Universal Tree



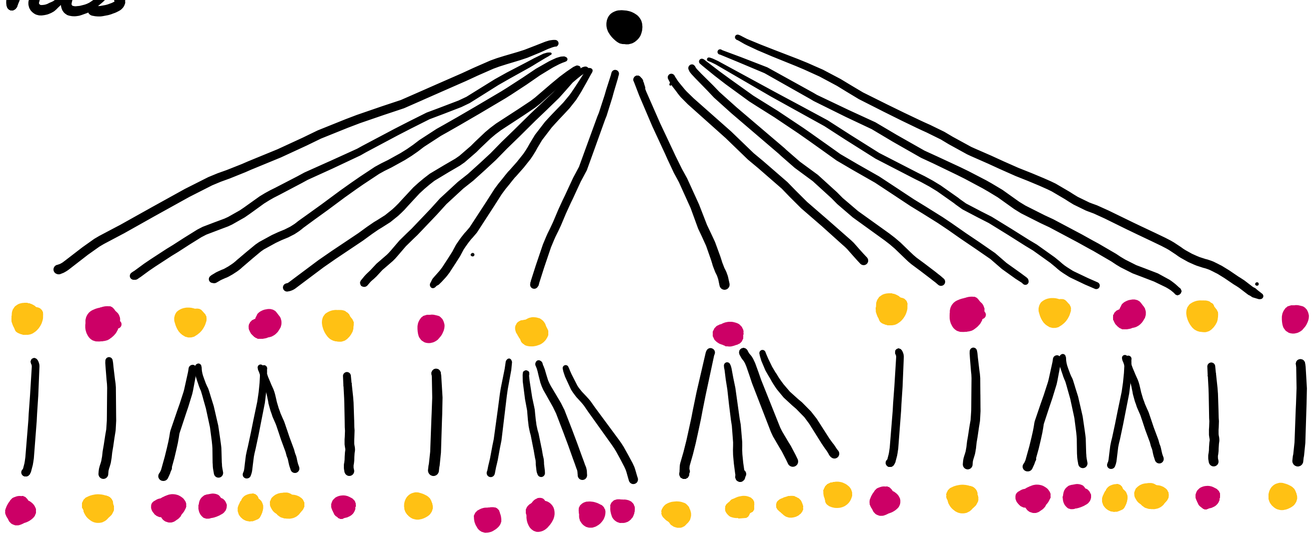
There are small (n, h)-universal trees: $O(n^{\log h})$

(n,h,s)-Strahler Universal Tree



There are (n,h,s)-Strahler Universal Trees of size $O\left((h/s)^s \cdot \mathrm{poly}(n)\right)$

Colourful Universal Trees

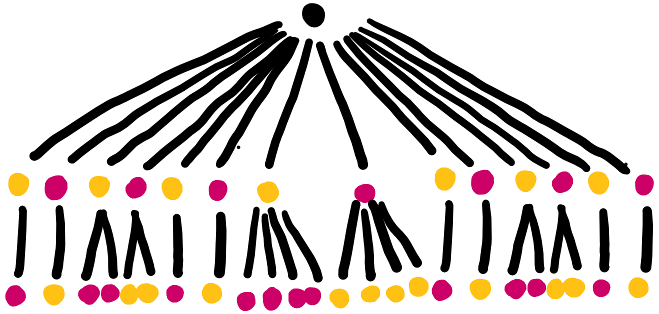


There are C-colourful trees of size $(|C|!)^{1+\varepsilon} \cdot \text{poly}(n)$

Thank you!

Universal symmetric attr. algorithm

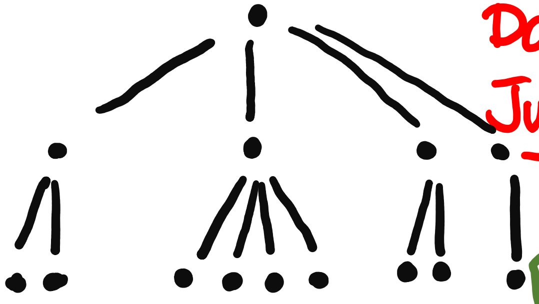
Jurdziński Morvan, Thejaswini



$O(n^2 \cdot k!^{1+o(1)})$

Majumdar Sağlam, Thejaswini

20 → 22 → 22 → 23



Daviaud Jurdziński Thejaswini

Reg. # = str #

Thejaswini, Ohlmann Jurdzinski

Reducing runtime of symmetric attractor based algos

$O(n^2 k!^{1+o(1)})$

Almost-sure winning-stochastic Rabin games

PolySAT
A Word-level Solver for Large Bitvectors

Jakob Rath

TU Wien

Joint work with Clemens Eisenhofer, Daniela Kaufmann,
Nikolaj Bjørner, Laura Kovács

PolySAT: a Word-level Solver for Large Bitvectors

Bitvectors?

1. Sequence of bits, e.g., `01011`
2. Fixed-width machine integers, e.g., `uint32_t`, `int64_t`
3. Modular arithmetic: $\mathbb{Z}/2^k\mathbb{Z}$

PolySAT: a Word-level Solver for Large Bitvectors

Bitvectors?

1. Sequence of bits, e.g., `01011`
2. Fixed-width machine integers, e.g., `uint32_t`, `int64_t`
3. Modular arithmetic: $\mathbb{Z}/2^k\mathbb{Z}$

Examples:

- $2x^2y + z = 3$
- $x + 3 \leq x + y$
- $\neg\Omega^*(x, y), \quad z = x \,\&\, y, \quad x[3{:}0] = 0, \quad \ldots$
- Negation, disjunction of constraints

PolySAT: a Word-level Solver for Large Bitvectors

Bitvectors?

1. Sequence of bits, e.g., `01011`
2. Fixed-width machine integers, e.g., `uint32_t`, `int64_t`
3. Modular arithmetic: $\mathbb{Z}/2^k\mathbb{Z}$

Examples:

- $2x^2y + z = 3$
- $x + 3 \leq x + y$
- $\neg\Omega^*(x, y), \quad z = x \,\&\, y, \quad x[3{:}0] = 0, \quad \ldots$
- Negation, disjunction of constraints

Existing approaches: bit-blasting, translation to integers

Example

$x + 3 \leq x + y \mod 2^3$

- For $x = 0$: $3 \leq y \iff y \in \{3, 4, 5, 6, 7\}$
- For $x = 2$: $5 \leq 2 + y \iff y \in \{3, 4, 5\}$

Example

$x + 3 \leq x + y \mod 2^3$

- For $x = 0$: $3 \leq y \iff y \in \{3, 4, 5, 6, 7\}$
- For $x = 2$: $5 \leq 2 + y \iff y \in \{3, 4, 5\}$
- $x + 3 \leq -y + 2 \mod 2^3$

$$p \leq q$$
$$p \leq p - q - 1$$
$$q - p \leq q$$
$$q - p \leq -p - 1$$
$$-q - 1 \leq -p - 1$$
$$-q - 1 \leq p - q - 1$$

Example

$x + 3 \le x + y \mod 2^3$

- For $x = 0$: $\quad 3 \le y \iff y \in \{3, 4, 5, 6, 7\}$
- For $x = 2$: $\quad 5 \le 2 + y \iff y \in \{3, 4, 5\}$
- $x + 3 \le -y + 2 \mod 2^3$

$$
\begin{aligned}
&p \le q \\
&p \le p - q - 1 \\
&q - p \le q \\
&q - p \le -p - 1 \\
&-q - 1 \le -p - 1 \\
&-q - 1 \le p - q - 1
\end{aligned}
$$

PolySAT is a theory solver for bitvector arithmetic:

- Search for a model of the input formula
- Incrementally assign bitvector variables (e.g., $x := 2$)
- Propagate feasible sets, e.g.:

$$x := 2 \land x + 3 \le x + y \implies y \in \{3, 4, 5\} \pmod{2^3}$$

- Add lemmas on demand, e.g.:

$$px < qx \land \neg\Omega^*(p, x) \implies p < q$$

From loops, to program synthesis, and beyond!

Daneshvar Amrollahi

TU Wien

Joint work with P. Hozzová, L. Kovács, M. Moosbrugger, etc.

October 9, 2023

Loops

A major challenge in formal verification

Loops

A major challenge in formal verification

- Loop invariants
  - Capture loop behavior as a logical formula: $x + 3y^2 = 2z^3$
  - Used in program verification
  - Automated invariant generation techniques based on symbolic computation, algebraic recurrence equations, static analysis, etc.

Loops

A major challenge in formal verification

- Loop invariants
  - Capture loop behavior as a logical formula: $x + 3y^2 = 2z^3$
  - Used in program verification
  - Automated invariant generation techniques based on symbolic computation, algebraic recurrence equations, static analysis, etc.
- Loop synthesis
  - Synthesizing a program (loop) given a specification
  - Program correctness by construction
  - Specification: a polynomial loop invariant
  - Applications in compiler optimization: single path loops, linear updates

Program Synthesis

- A framework based on saturation-based theorem proving.
- Specification: $\forall \bar{x}.\exists y.F[\bar{x}, y]$
- Framework output:
  - A program with `if-then-else` statements
  - A proof that the spec. holds (using Vampire)

Beyond

Something around SMT with Clark Barrett at Stanford

AUTOSARD

Matthias Hetzenberger

supervised by Florian Zuleger

AUTOSARD

Automated **S**ublinear **A**mortised **R**esource **A**nalysis of Data **S**tructures

Matthias Hetzenberger

supervised by Florian Zuleger

- Goal: develop automated reasoning techniques w.r.t. amortised cost analysis of (probabilistic) functional data structures

- Goal: develop automated reasoning techniques w.r.t. amortised cost analysis of (probabilistic) functional data structures

- Extend pilot project ATLAS based on type-and-effect system and potential functions [Leutgeb, Moser, and Zuleger 2022]

- Goal: develop automated reasoning techniques w.r.t. amortised cost analysis of (probabilistic) functional data structures

- Extend pilot project ATLAS based on type-and-effect system and potential functions [Leutgeb, Moser, and Zuleger 2022]

- Current focus *Zip Trees* [Tarjan, Levy, and Timmel 2021]

Leutgeb, Lorenz, Georg Moser, and Florian Zuleger (2022). "Automated Expected Amortised Cost Analysis of Probabilistic Data Structures". In: *Computer Aided Verification*. Springer International Publishing, pp. 70–91. DOI: 10.1007/978-3-031-13188-2_4. URL: https://doi.org/10.1007/978-3-031-13188-2_4.

Tarjan, Robert E., Caleb Levy, and Stephen Timmel (Oct. 2021). "Zip Trees". In: *ACM Transactions on Algorithms* 17.4, pp. 1–12. DOI: 10.1145/3476830. URL: https://doi.org/10.1145/3476830.

IC3

Islam Hamada

TU Wien

2023

Overview

- Prominent model checking algorithm.

Overview

- Prominent model checking algorithm.
- builds multiple successive overapproximations of reachable states simultaneously.

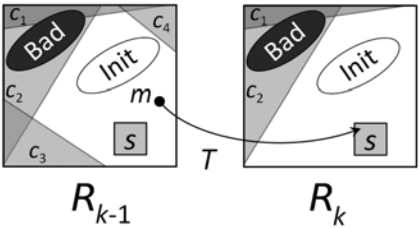
Overview

- Prominent model checking algorithm.
- builds multiple successive overapproximations of reachable states simultaneously.
- looks for a proof of correctness by finding an inductive invariant that is safe, otherwise gives a counter example.

Overview

- Prominent model checking algorithm.
- builds multiple successive overapproximations of reachable states simultaneously.
- looks for a proof of correctness by finding an inductive invariant that is safe, otherwise gives a counter example.
- Building the invariant is guided by **CTIs**.

$$R_i \wedge T \wedge \neg P'$$



Aspects To Investigate

- The used heuristic for generalizing clauses

Aspects To Investigate

- The used heuristic for generalizing clauses
- Save and reuse CTIs

Aspects To Investigate

- The used heuristic for generalizing clauses
- Save and reuse CTIs
- Avoiding duplicate clauses.

Aspects To Investigate

- The used heuristic for generalizing clauses
- Save and reuse CTIs
- Avoiding duplicate clauses.
- Global clauses

Aspects To Investigate

- The used heuristic for generalizing clauses
- Save and reuse CTIs
- Avoiding duplicate clauses.
- Global clauses
- Generalizing the CTIs further

Incremental IC3

- Two related transition relations, T and T_c such that $T_c \subseteq T$.

Incremental IC3

- Two related transition relations, T and T_c such that $T_c \subseteq T$.
- Reusing clauses directly

Incremental IC3

- Two related transition relations, T and T_c such that $T_c \subseteq T$.
- Reusing clauses directly
- Reusing CTIs and lifting them further

Incremental IC3

- Two related transition relations, T and T_c such that $T_c \subseteq T$.
- Reusing clauses directly
- Reusing CTIs and lifting them further
- Reusing the invariant

Learn to be Dynamical

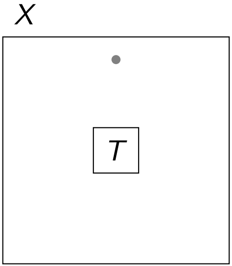
Mahyar Karimi

ISTA

October 9, 2023

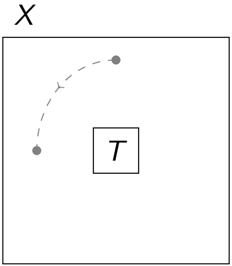
All about Dynamical Systems

- Jumping particle:



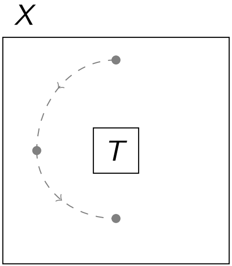
All about Dynamical Systems

- Jumping particle:



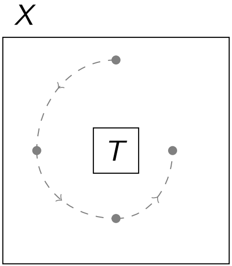
All about Dynamical Systems

- Jumping particle:



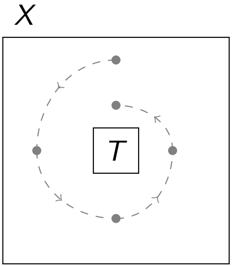
All about Dynamical Systems

- Jumping particle:



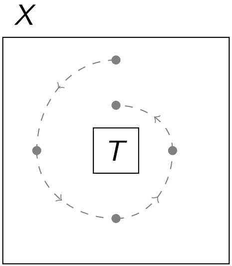
All about Dynamical Systems

- Jumping particle:



All about Dynamical Systems

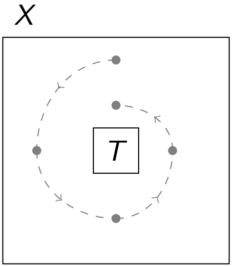
- Jumping particle:



- Transitions: $x_{t+1} = f(x_t)$.

All about Dynamical Systems

- Jumping particle:



- Transitions: $x_{t+1} = f(x_t)$.
- Can we reach T?

Lyapunov Functions

Can we have a function V that

1. is non-negative: $V(x) \geq 0$
2. decreases with every transition: $V(x) > V(f(x))$?

Lyapunov Functions

Can we have a function V that

1. is non-negative: $V(x) \geq 0$
2. decreases with every transition: $V(x) > V(f(x))$?

- For nonlinear systems, V is not easy to find.

Lyapunov Functions

Can we have a function V that

1. is non-negative: $V(x) \geq 0$
2. decreases with every transition: $V(x) > V(f(x))$?

- For nonlinear systems, V is not easy to find.
- SMT for finding V? Precise, but slow.

Lyapunov Functions

Can we have a function V that

1. is non-negative: $V(x) \geq 0$
2. decreases with every transition: $V(x) > V(f(x))$?

- For nonlinear systems, V is not easy to find.
- SMT for finding V? Precise, but slow.
- Guided search for V?

Neural Lyapunov Functions

Let's use a neural network to find V!

- Learning $V \Longleftarrow$ Loss Function + Gradient Descent
- Loss should *capture* V.

Neural Lyapunov Functions

Let's use a neural network to find V!

- Learning $V \Longleftarrow$ Loss Function + Gradient Descent
- Loss should *capture* V.

Catch! No guarantee for generalization.

Neural Lyapunov Functions

Let's use a neural network to find V!

- Learning $V \impliedby$ Loss Function + Gradient Descent
- Loss should *capture* V.

Catch! No guarantee for generalization.
Good news; we can use SMT solving.

Is V All We Can Learn?

Is V All We Can Learn?

No.

Is V All We Can Learn?

No.

- Replacing f with a neural network.

Is V All We Can Learn?

No.

- Replacing f with a neural network.
  Benefit; NN instead of mathematical object.

Is V All We Can Learn?

No.

- Replacing f with a neural network.
  Benefit; NN instead of mathematical object.
  Catch! 2 generalization queries instead of 1.

Is V All We Can Learn?

No.

- Replacing f with a neural network.
  Benefit; NN instead of mathematical object.
  Catch! 2 generalization queries instead of 1.
- More can be learned: partitioning X, error bounds, ...

TECHNISCHE UNIVERSITÄT WIEN

Separation Logic for Program Analysis

Florian Sextl

2023-10-09

Central Ideas

Goals

Central Ideas

Goals

- Verify memory safety even in unsafe programs (e.g. C/`unsafe` in Rust)

Central Ideas

Goals

- Verify memory safety even in unsafe programs (e.g. C/`unsafe` in Rust)
- Make it usable (fully automatic, acceptable runtime, strong guarantees)

Central Ideas

Goals

- Verify memory safety even in unsafe programs (e.g. C/`unsafe` in Rust)
- Make it usable (fully automatic, acceptable runtime, strong guarantees)

Approach

Central Ideas

Goals

- Verify memory safety even in unsafe programs (e.g. C/`unsafe` in Rust)
- Make it usable (fully automatic, acceptable runtime, strong guarantees)

Approach

- Based on strong but manageable separation logic

Central Ideas

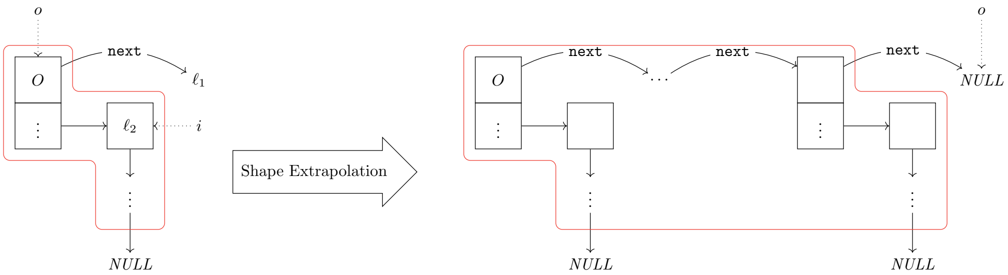
Goals

- Verify memory safety even in unsafe programs (e.g. C/`unsafe` in Rust)
- Make it usable (fully automatic, acceptable runtime, strong guarantees)

Approach

- Based on strong but manageable separation logic
- Symbolic execution with bi-abduction

Previously: Sound Bi-abduction-based Shape Analysis



Program Synthesis via {Saturation, SMT solving}

Petra Hozzová

supervised by Laura Kovács,
and working with Andrei Voronkov, Nikolaj Bjørner, Daneshvar Amrollahi, . . .

Synthesis in saturation

Synthesize a program computing y for any $\overline{x}$ such that $F(\overline{x}, y)$ holds using a saturation-based prover proving $\forall \overline{x}. \exists y. F(\overline{x}, y)$ using induction.

Synthesis in saturation

first-order formula,
$\overline{x}$ are inputs and y is the output

Synthesize a program computing y for any $\overline{x}$ such that $F(\overline{x}, y)$ holds using a saturation-based prover proving $\forall \overline{x}. \exists y. F(\overline{x}, y)$ using induction.

Synthesis in saturation

term, possibly using $\texttt{if}-\texttt{then}-\texttt{else}$,
recursively defined functions,
and only containing computable symbols

first-order formula,
$\overline{x}$ are inputs and y is the output

Synthesize a program computing y for any $\overline{x}$ such that $F(\overline{x}, y)$ holds

using a saturation-based prover proving $\forall \overline{x}. \exists y. F(\overline{x}, y)$ using induction.

Synthesis in saturation

term, possibly using $\texttt{if}-\texttt{then}-\texttt{else}$,
recursively defined functions,
and only containing computable symbols

first-order formula,
$\overline{x}$ are inputs and y is the output

Synthesize a program computing y for any $\overline{x}$ such that $F(\overline{x}, y)$ holds

using a saturation-based prover proving $\forall \overline{x}. \exists y. F(\overline{x}, y)$ using induction.

using answer literals,
supporting derivation of clauses $C \vee \texttt{ans}(r)$ where C is computable,
expressing "if $\neg C$, then r is the program"

Synthesis with SMT-solving

Synthesize a program computing the function f such that $F(\overline{x}, f)$ holds

using quantifier elimination games for $\exists f.\forall \overline{x}.F(\overline{x}, f)$.*

Synthesis with SMT-solving

first-order formula, f's arguments are terms dependent on $\overline{x}$

Synthesize a program computing the function f such that $F(\overline{x}, f)$ holds using quantifier elimination games for $\exists f. \forall \overline{x}. F(\overline{x}, f)$.*

Synthesis with SMT-solving

term, possibly using `if-then-else`, and only containing computable symbols

first-order formula, f's arguments are terms dependent on $\overline{x}$

Synthesize a program computing the function f such that $F(\overline{x}, f)$ holds

using quantifier elimination games for $\exists f. \forall \overline{x}. F(\overline{x}, f)$.*

Synthesis with SMT-solving

term, possibly using if − then − else,
and only containing computable symbols

first-order formula, f's arguments
are terms dependent on $\overline{x}$

Synthesize a program computing the function f such that $F(\overline{x}, f)$ holds

using quantifier elimination games for $\exists f.\forall \overline{x}.F(\overline{x}, f)$.*

Using an interplay of two procedures, that in turns find interpretations of f and $\overline{x}$.
If the final interpretation satisfies the formula, we learn a case in the program.
Otherwise we either learn a lemma or conclude the synthesis.

Krishnendu Chatterjee, Thomas Henzinger, **Stefanie Muroya Lei**

Quantum Information Markov Decision Processes for Robust Quantum Programs Synthesis

ISTA

Quantum Algorithms Workflow

QUANTUM STATE IN A WELL DEFINED STATE

→

APPLY QUANTUM GATES AND MEASUREMENTS

→

A PROBABILITY DISTRIBUTION OVER CLASSICAL STATES

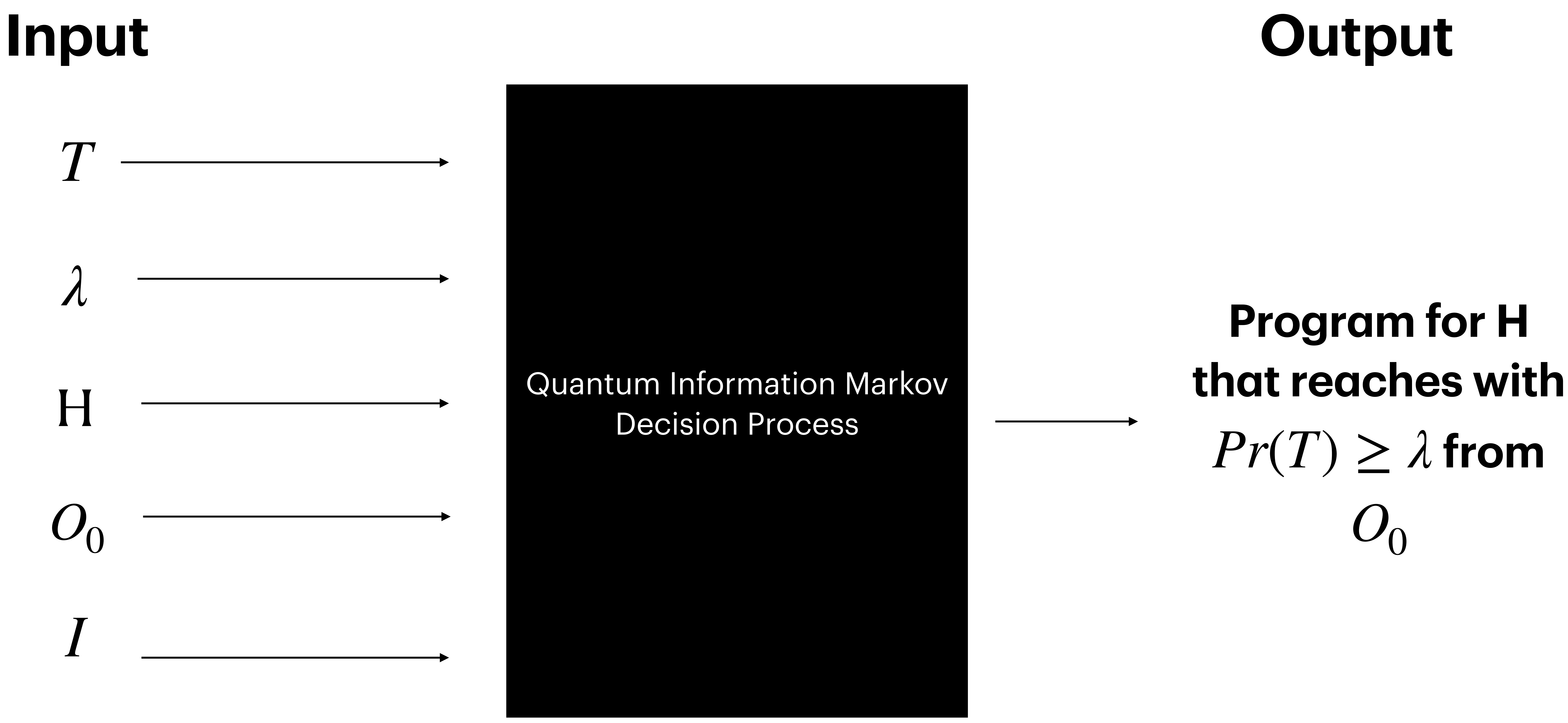
Challenges

- Quantum Computers are very noisy

- The no-cloning theorem

- We cannot directly observe quantum states

- Quantum algorithms are hard to engineer



T: set of target states

λ: threshold

H: hardware spec.

O_0: distribution over states

I: set of instructions

Partially Observable Markov Decision Processes (POMDP)

A POMDP is a tuple $\langle S, A, \mathcal{O}, \Delta, \gamma_1 \rangle$ *where:*

- S *is a set of states*
- A *is a set of actions*
- $\mathcal{O}$ *is a set of observations*
- $\Delta : S \times A \times S \rightarrow [0,1]$ *is a probabilistic transition function*
- $\gamma_1 : S \rightarrow \mathcal{O}$

Quantum Information Markov Decision Processes (QIMDP)

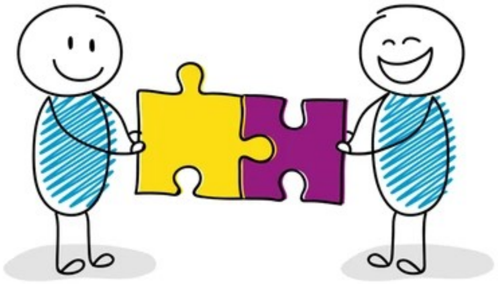
A QIMDP is a tuple $\langle M, I, C, \rightarrow_H, \gamma_2 \rangle$ *where:*

- M *is a set of hybrid states*
- I *is a set of instructions*
- C *is a set of classical states*
- $\rightarrow_H: M \times I \times M \rightarrow [0,1]$ *is a probabilistic transition function*
- $\gamma_2 : M \rightarrow C$

CALGSAT

Combining Computer Algebra with SAT Solving

Daniela Kaufmann



Computer ALGebra

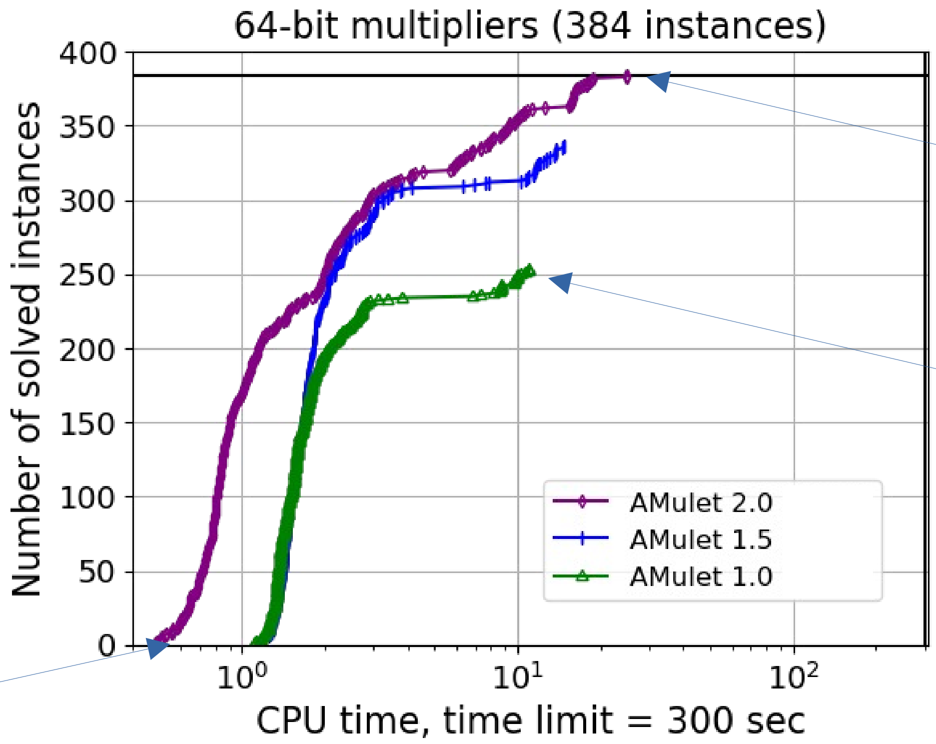
SAT Solving

Computer ALGebra		SAT Solving
Polynomial System $P \subseteq \mathbb{K}[X]$ $\{x^2 + y = 0, -4y + xz = 0, yz + 3 = 0\}$	Model	Propositional Logic Formula $(x \vee y) \wedge (\bar{x} \vee z) \wedge (x \vee \bar{z}) \wedge (\bar{y} \vee \bar{z})$
Computer Algebra System	Reasoning Engine	SAT Solver
System with all solutions $\{z^3 - 48 = 0, 16y + z^2 = 0, 4x + z = 0\}$	Solution	Single assignments $\{x = \top, y = \bot, z = \top\}$

- Recent success in formal verification
- word-level and bit-level models
- general purpose solvers
- returns all solutions

- Over 50 years of research → “Killer application”
- bit-level models
- dedicated heuristics and solving engines
- single assignments

Circuit Verification



Computer algebra + SAT solves 384/384

Computer algebra solves 254/384

SAT solves 0/384

[1] Kaufmann, Biere, Kauers. *Verifying Large Multipliers by Combining SAT and Computer Algebra*. FMCAD 2019: 28-36

$\mathbb{C}$omputer $\mathbb{ALG}$ebra

$$P \subseteq \mathbb{Z}[X], X \in \mathbb{B}$$

Pseudo-Boolean Integer Polynomials

- Hardware verification

  Variables represent signals in circuits
  Integer coefficients for word-level specification

$$P \subseteq \mathbb{Z}/2^w\mathbb{Z}[X], X \in \mathbb{Z}/2^w\mathbb{Z}[X]$$
$$P \subseteq \mathbb{F}_q[X], X \in \mathbb{F}_q$$

Polynomials in finite domains

- Verification of cryptosystems

  Variables and coefficients are used to represent states of the system

Theory Reasoning in Saturation Theorem Proving

Johannes Schoisswohl

Theory Reasoning in Saturation Theorem Proving

Johannes Schoisswohl

Theory Reasoning in Saturation Theorem Proving

Johannes Schoisswohl

Theory Reasoning in Saturation Theorem Proving

- Saturation Algorithms

Theory Reasoning in Saturation Theorem Proving

- Saturation Algorithms
    - Assume $\neg\phi$

Theory Reasoning in Saturation Theorem Proving

- Saturation Algorithms
  - Assume $\neg\phi$
  - Apply a set of rules exhaustively

Theory Reasoning in Saturation Theorem Proving

- Saturation Algorithms
  - Assume $\neg\phi$
  - Apply a set of rules exhaustively
  - Until contradiction found or no rules applicable

Theory Reasoning in Saturation Theorem Proving

- Saturation Algorithms
  - Assume $\neg\phi$
  - Apply a set of rules exhaustively
  - Until contradiction found or no rules applicable
  - **Mainly for Uninterpreted Symbols**

Theory Reasoning in Saturation Theorem Proving

- Saturation Algorithms
  - Assume $\neg\phi$
  - Apply a set of rules exhaustively
  - Until contradiction found or no rules applicable
  - **Mainly for Uninterpreted Symbols**
- Theory Reasoning

Theory Reasoning in Saturation Theorem Proving

- Saturation Algorithms
  - Assume $\neg\phi$
  - Apply a set of rules exhaustively
  - Until contradiction found or no rules applicable
  - **Mainly for Uninterpreted Symbols**
- Theory Reasoning
  - Symbols have predefined meaning (e.g. $+$, $<$)

Theory Reasoning in Saturation Theorem Proving

- Saturation Algorithms
  - Assume $\neg\phi$
  - Apply a set of rules exhaustively
  - Until contradiction found or no rules applicable
  - **Mainly for Uninterpreted Symbols**
- Theory Reasoning
  - Symbols have predefined meaning (e.g. $+$, $<$)
  - Naively handled with axioms (e.g. $x < y \land y < z \rightarrow x < z$)

Theory Reasoning in Saturation Theorem Proving

- Saturation Algorithms
  - Assume $\neg\phi$
  - Apply a set of rules exhaustively
  - Until contradiction found or no rules applicable
  - **Mainly for Uninterpreted Symbols**
- Theory Reasoning
  - Symbols have predefined meaning (e.g. $+$, $<$)
  - Naively handled with axioms (e.g. $x < y \wedge y < z \rightarrow x < z$)
  - **Problem: Very explosive!**

Theory Reasoning in Saturation Theorem Proving

- Saturation Algorithms
  - Assume $\neg\phi$
  - Apply a set of rules exhaustively
  - Until contradiction found or no rules applicable
  - **Mainly for Uninterpreted Symbols**
- Theory Reasoning
  - Symbols have predefined meaning (e.g. $+$, $<$)
  - Naively handled with axioms (e.g. $x < y \wedge y < z \rightarrow x < z$)
  - **Problem: Very explosive!**

$$x_0 < x_1 \wedge x_1 < x_2 \rightarrow x_0 < x_2$$

Theory Reasoning in Saturation Theorem Proving

- Saturation Algorithms
  - Assume $\neg\phi$
  - Apply a set of rules exhaustively
  - Until contradiction found or no rules applicable
  - **Mainly for Uninterpreted Symbols**
- Theory Reasoning
  - Symbols have predefined meaning (e.g. $+$, $<$)
  - Naively handled with axioms (e.g. $x < y \wedge y < z \rightarrow x < z$)
  - **Problem: Very explosive!**

$$x_0 < x_1 \wedge x_1 < x_2 \rightarrow x_0 < x_2$$
$$x_0 < x_1 \wedge x_1 < x_2 \wedge x_2 < x_3 \rightarrow x_0 < x_3$$

Theory Reasoning in Saturation Theorem Proving

- Saturation Algorithms
  - Assume $\neg\phi$
  - Apply a set of rules exhaustively
  - Until contradiction found or no rules applicable
  - **Mainly for Uninterpreted Symbols**
- Theory Reasoning
  - Symbols have predefined meaning (e.g. $+$, $<$)
  - Naively handled with axioms (e.g. $x < y \wedge y < z \rightarrow x < z$)
  - **Problem: Very explosive!**

$$x_0 < x_1 \wedge x_1 < x_2 \rightarrow x_0 < x_2$$

$$x_0 < x_1 \wedge x_1 < x_2 \wedge x_2 < x_3 \rightarrow x_0 < x_3$$

$$x_0 < x_1 \wedge x_1 < x_2 \wedge x_2 < x_3 \wedge x_3 < x_4 \rightarrow x_0 < x_4$$

$$\ldots$$

Theory Reasoning in Saturation Theorem Proving

- Saturation Algorithms
  - Assume $\neg\phi$
  - Apply a set of rules exhaustively
  - Until contradiction found or no rules applicable
  - **Mainly for Uninterpreted Symbols**
- Theory Reasoning
  - Symbols have predefined meaning (e.g. $+$, $<$)
  - Naively handled with axioms (e.g. $x < y \wedge y < z \rightarrow x < z$)
  - **Problem: Very explosive!**

$$\frac{x_0 < x_1 \qquad x_1 < x_2}{x_0 < x_2}$$

Theory Reasoning in Saturation Theorem Proving

Background Theories $\mathcal{T}$ + Quantifiers

Theory Reasoning in Saturation Theorem Proving

Background Theories $\mathcal{T}$ + Quantifiers

Theory Reasoning in Saturation Theorem Proving

Background Theories $\mathcal{T}$ + Quantifiers

- Naive approach: Axioms

Theory Reasoning in Saturation Theorem Proving

Background Theories $\mathcal{T}$ + Quantifiers

- Naive approach: Axioms
- Better approach: Special Inference Systems

Theory Reasoning in Saturation Theorem Proving

Background Theories $\mathcal{T}$ + Quantifiers

- Naive approach: Axioms
- Better approach: Special Inference Systems
- ALASCA (done)
  - Linear Real Arithmetic + Uninterpreted Functions
  - Beats State of the Art

Theory Reasoning in Saturation Theorem Proving

Background Theories $\mathcal{T}$ + Quantifiers

- Naive approach: Axioms
- Better approach: Special Inference Systems
- ALASCA (done)
  - Linear Real Arithmetic + Uninterpreted Functions
  - Beats State of the Art
- ALASCAI (in progress)
  - ALASCA + Floor Function
  - Allows for integer reasoning

Bidding Games taking *Charge*

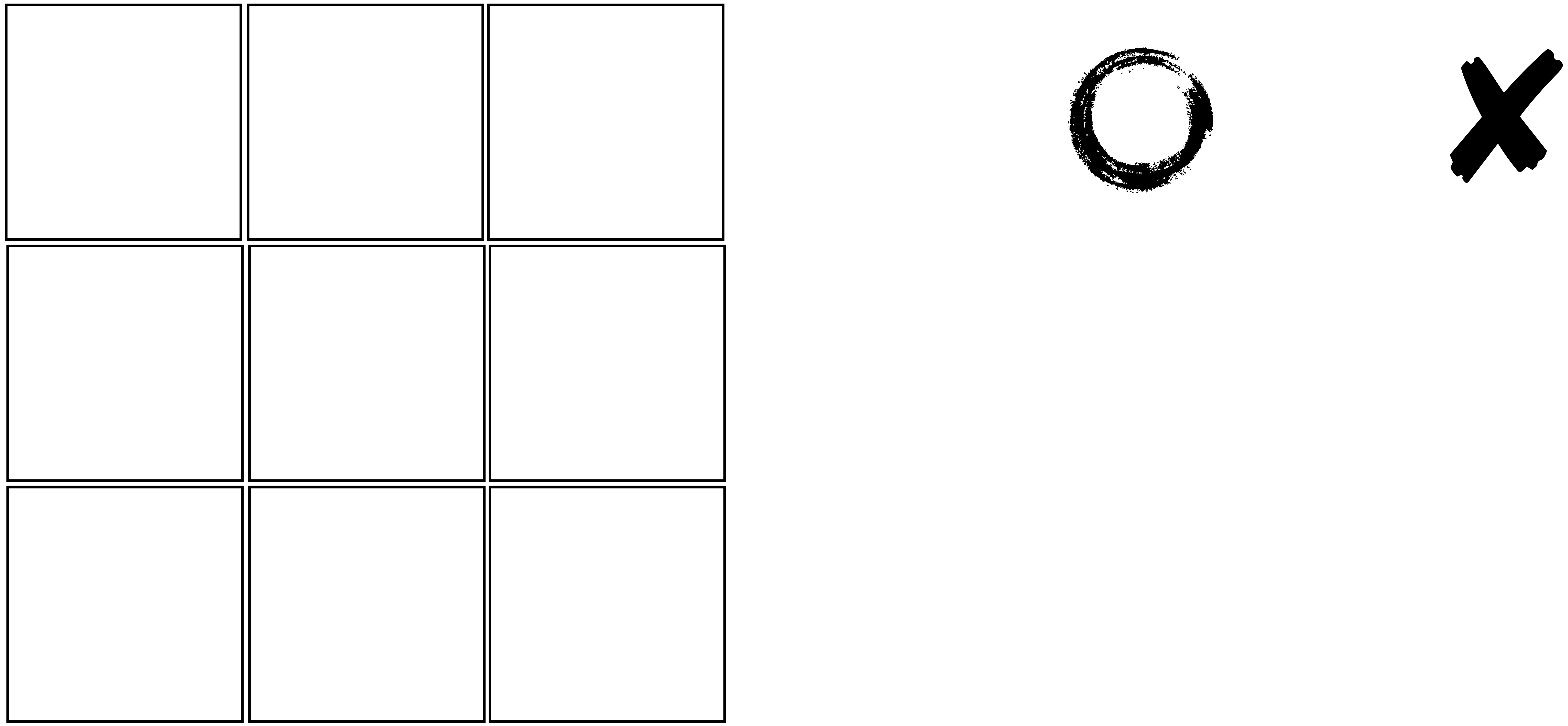
…in *theory* and in *practice*

Kaushik Mallik

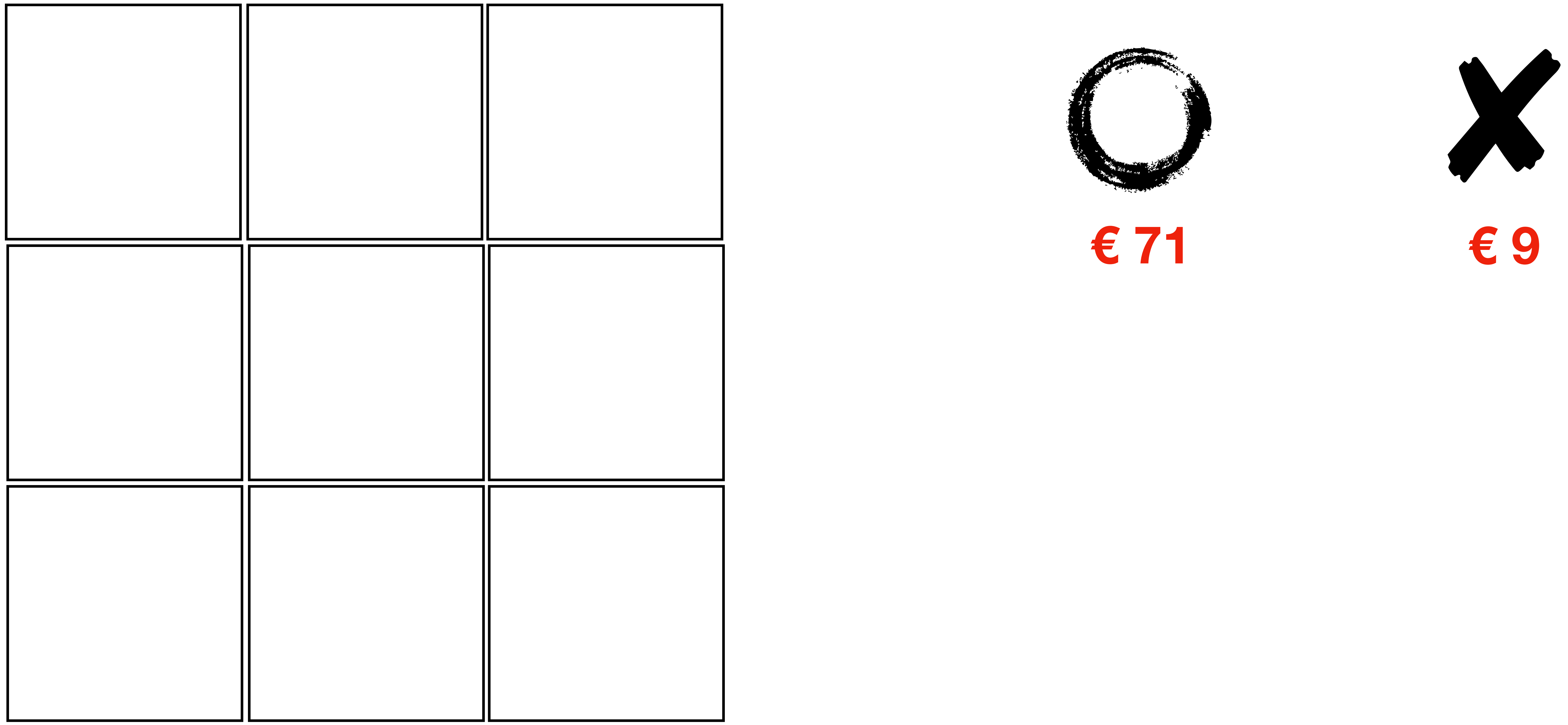
Henzinger Group

Institute of Science and Technology Austria

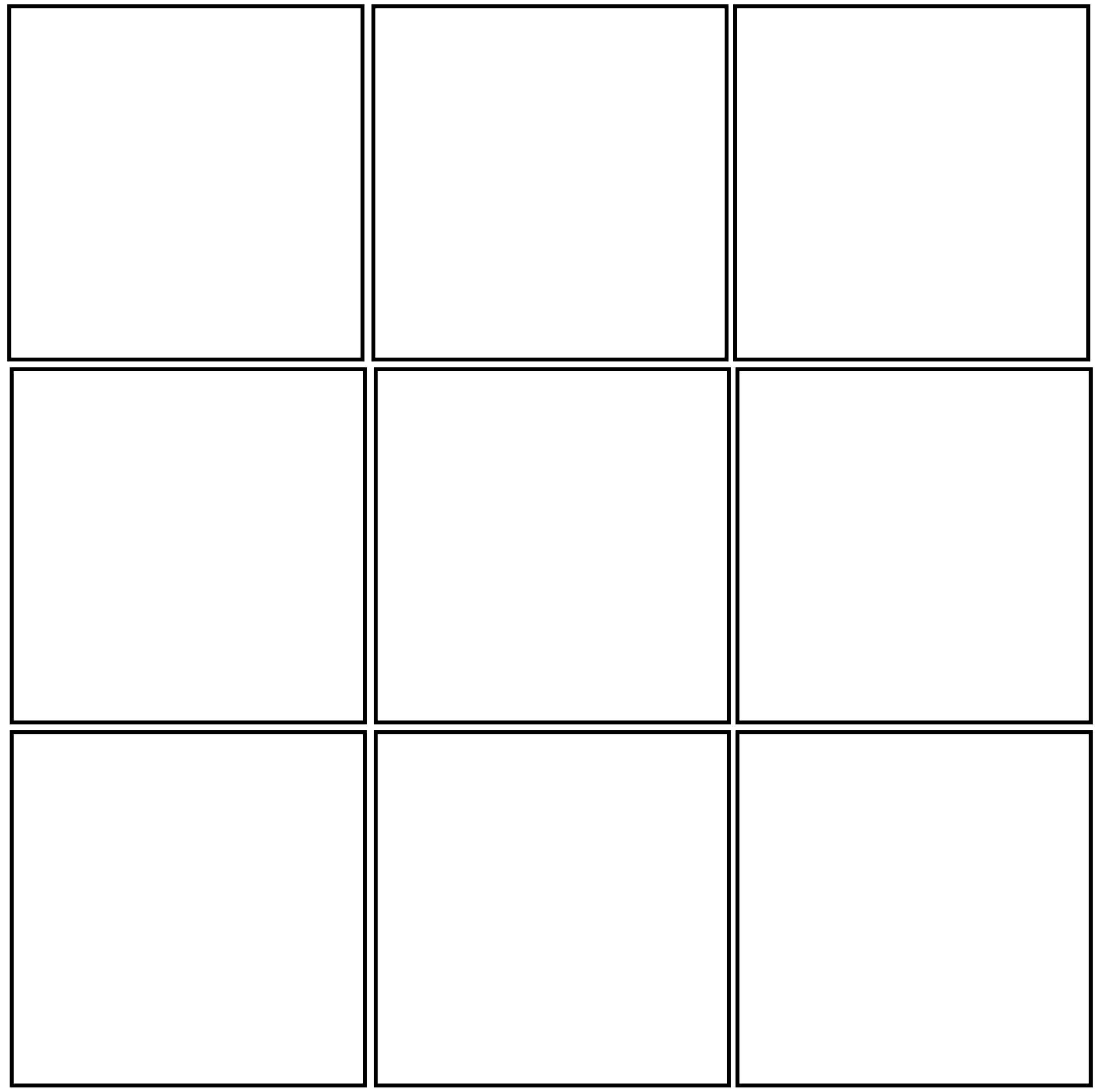
Bid-Tac-Toe



Bid-Tac-Toe

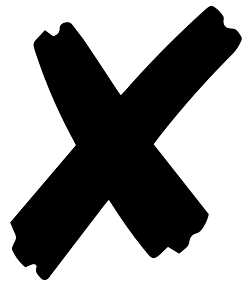


Bid-Tac-Toe



$\frac{7}{8} + \varepsilon$

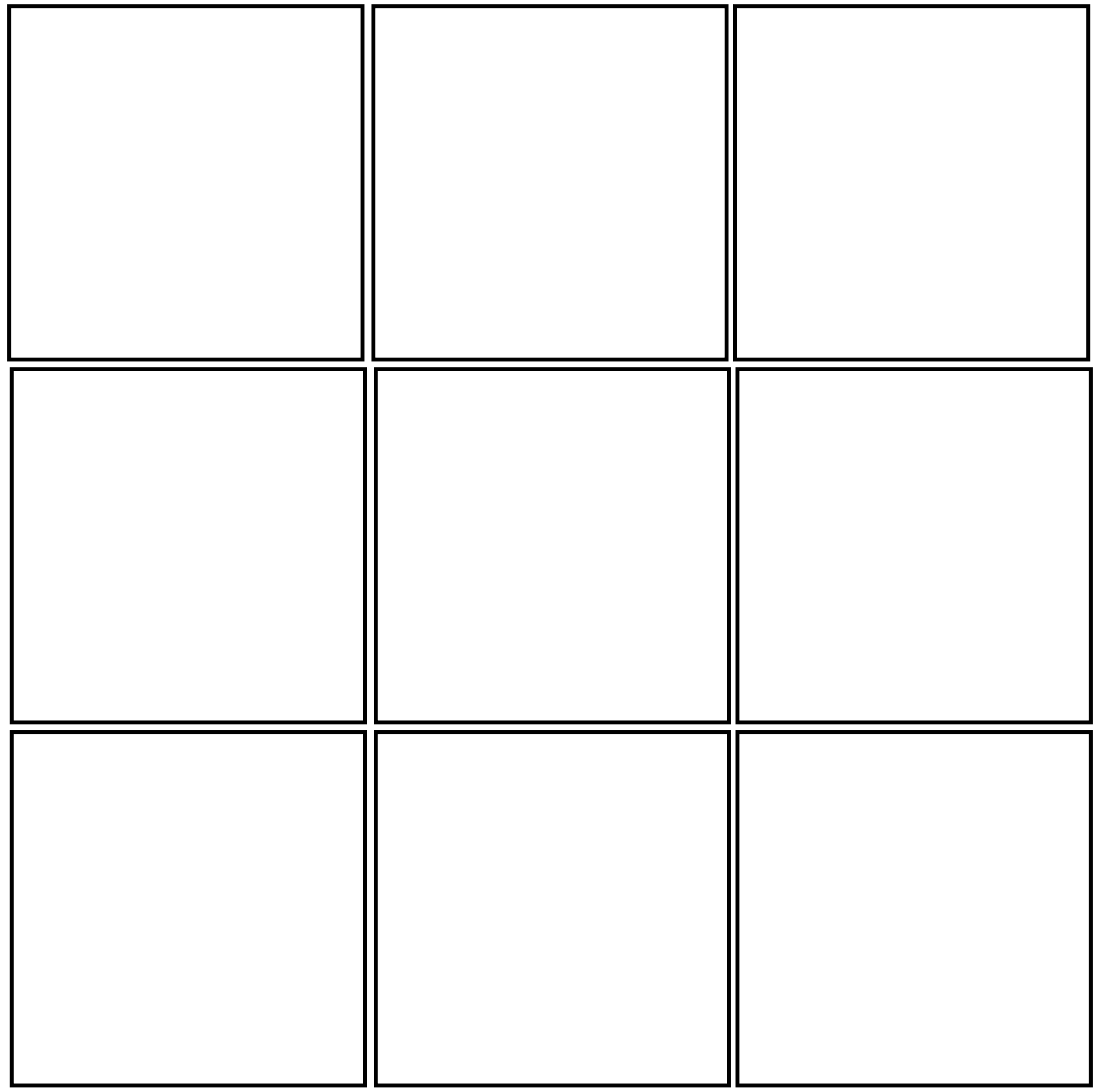
€ 71



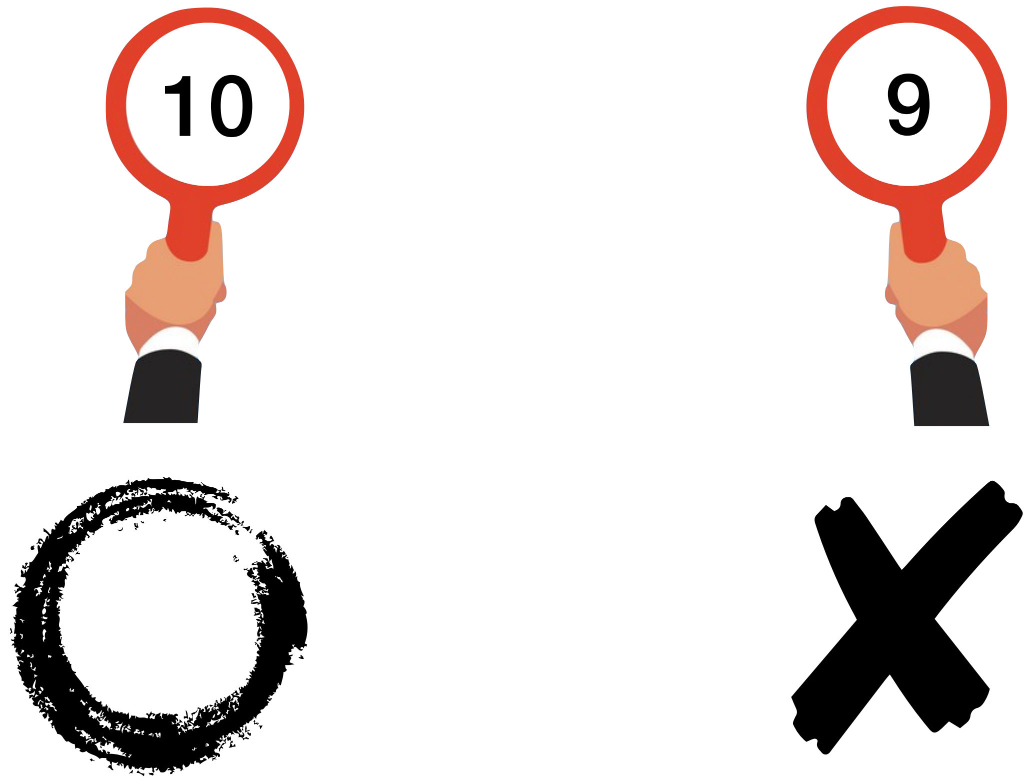
€ 9

$\frac{1}{8} - \varepsilon$

Bid-Tac-Toe



$\frac{7}{8}+\varepsilon$

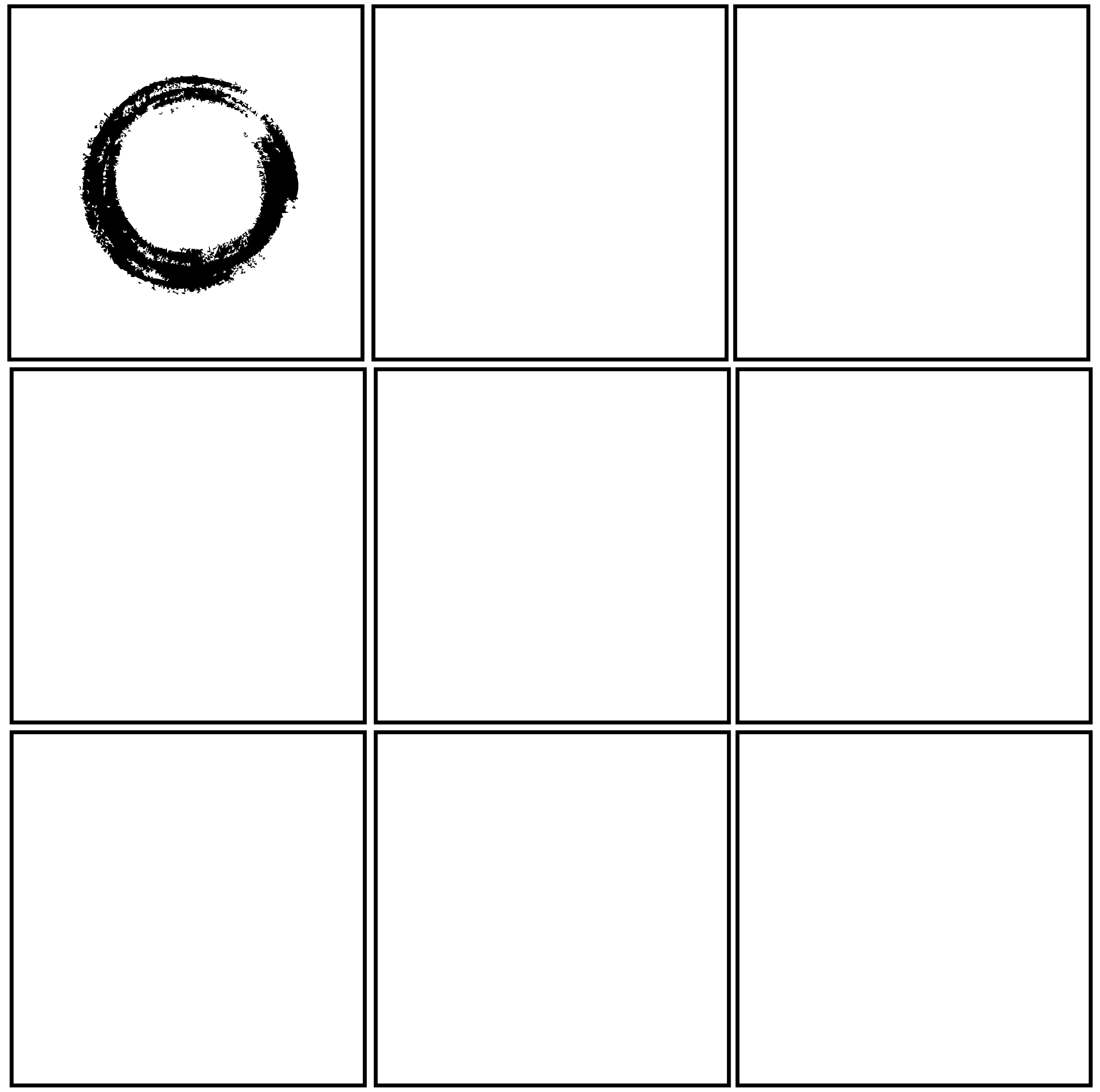


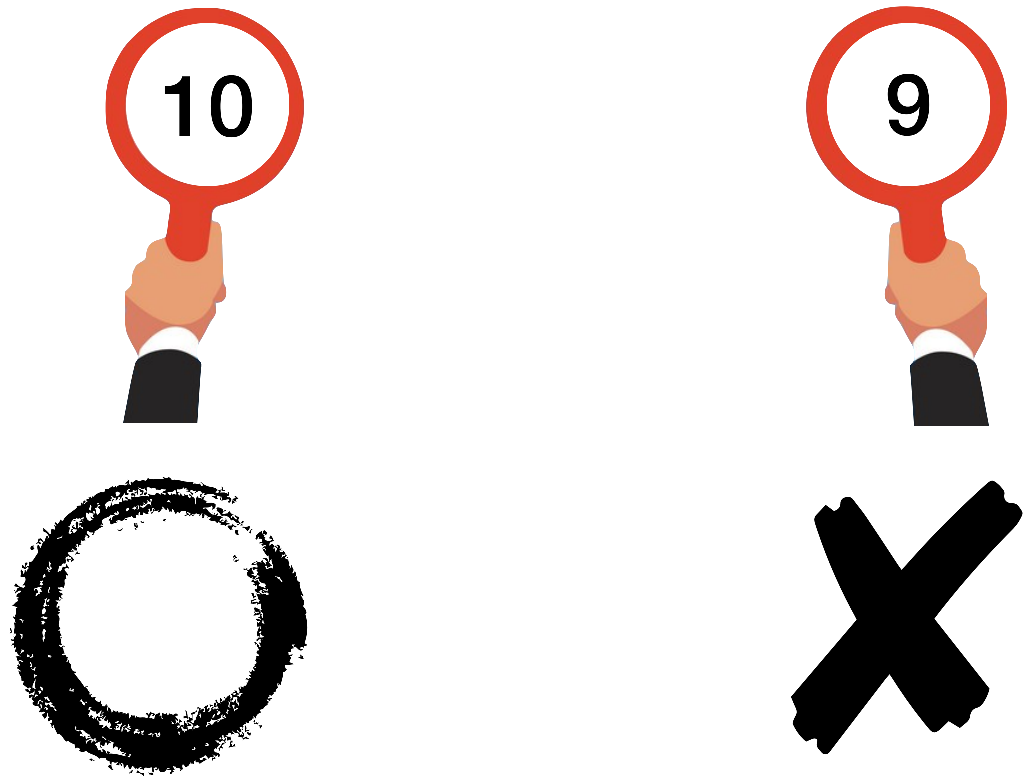
€ 71

€ 9

$\frac{1}{8}-\varepsilon$

Bid-Tac-Toe





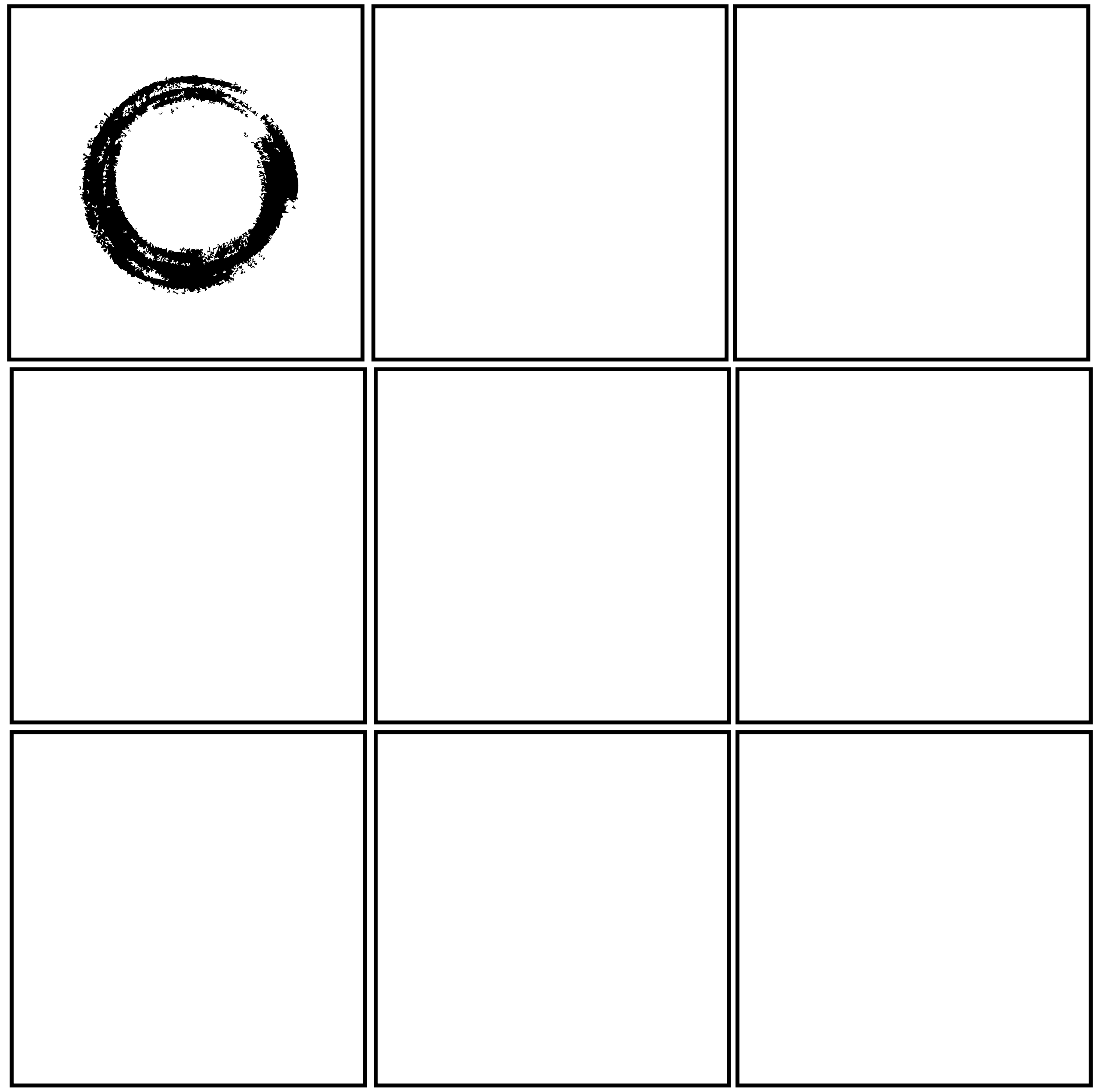
$\frac{7}{8} + \varepsilon$

€ 71

€ 9

$\frac{1}{8} - \varepsilon$

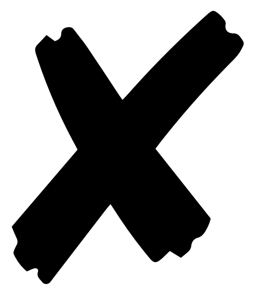
Bid-Tac-Toe



$\frac{7}{8} + \varepsilon$

~~€ 71~~

€ 61

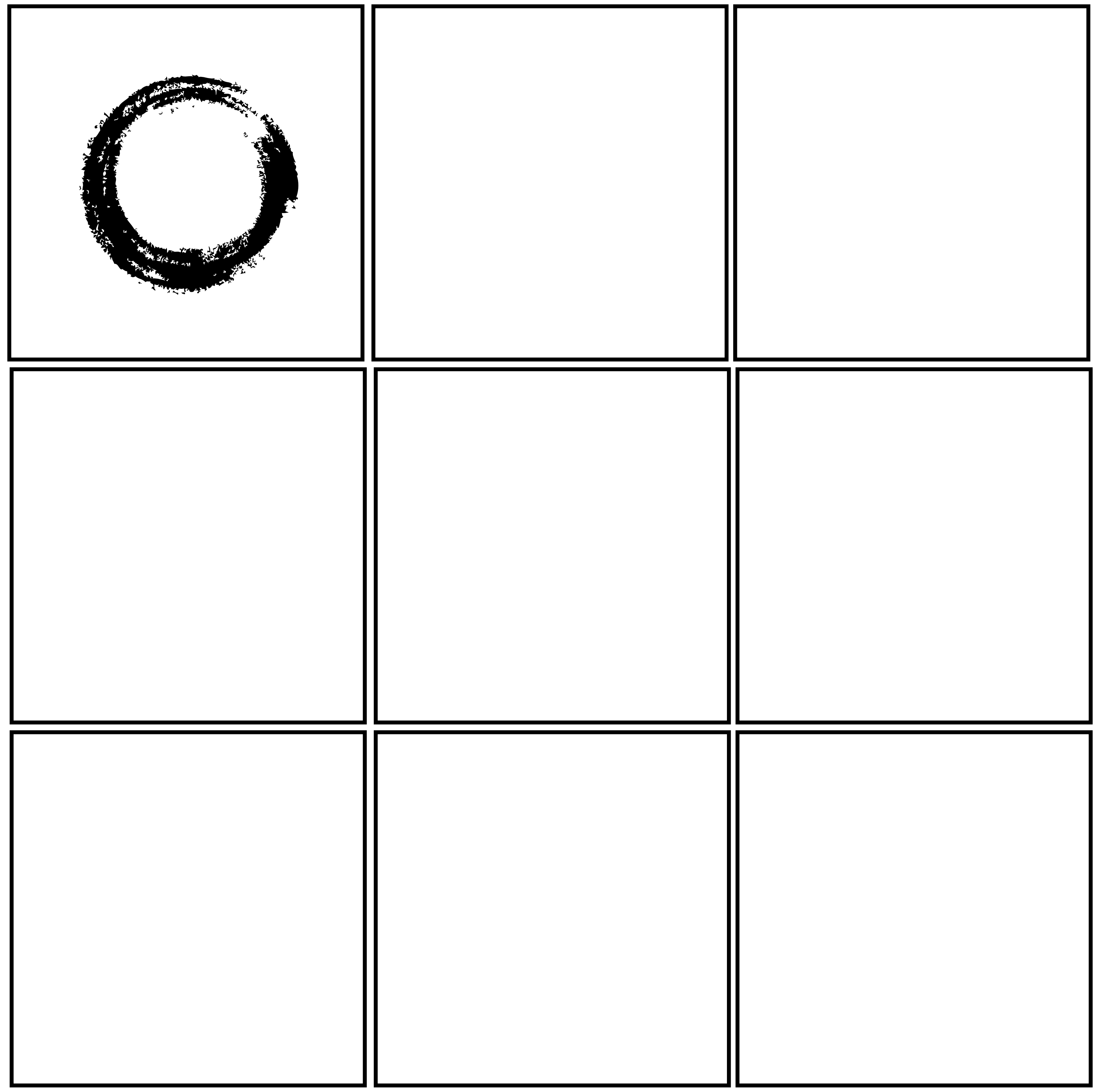


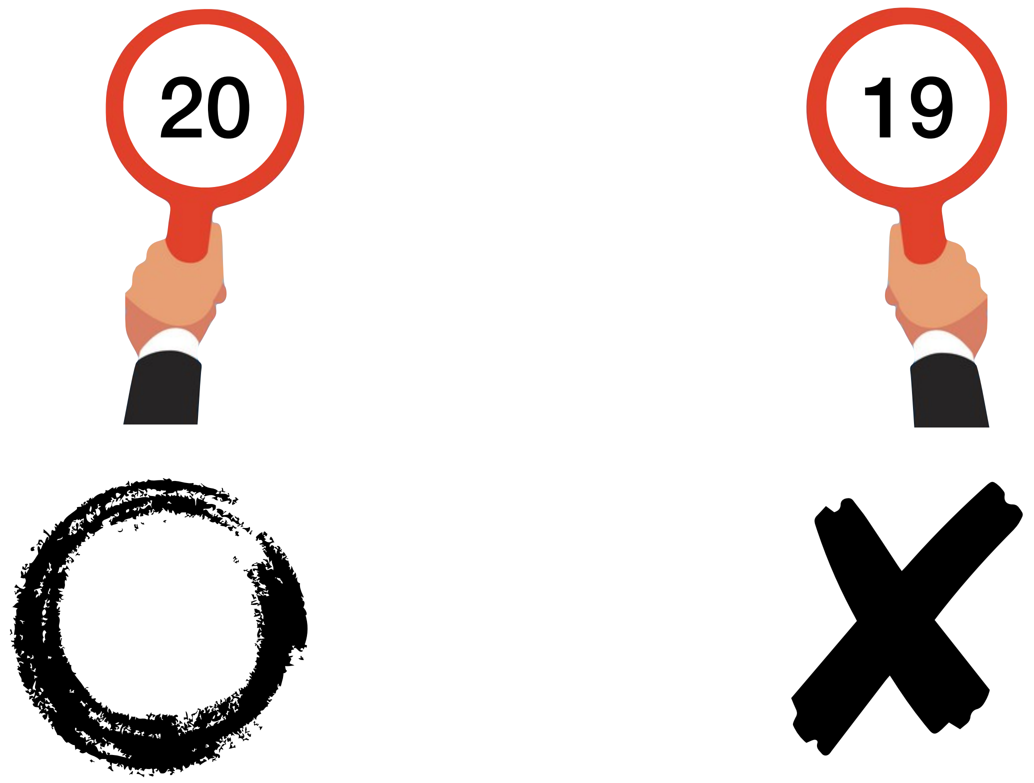
~~€ 9~~

€ 19

$\frac{1}{8} - \varepsilon$

Bid-Tac-Toe





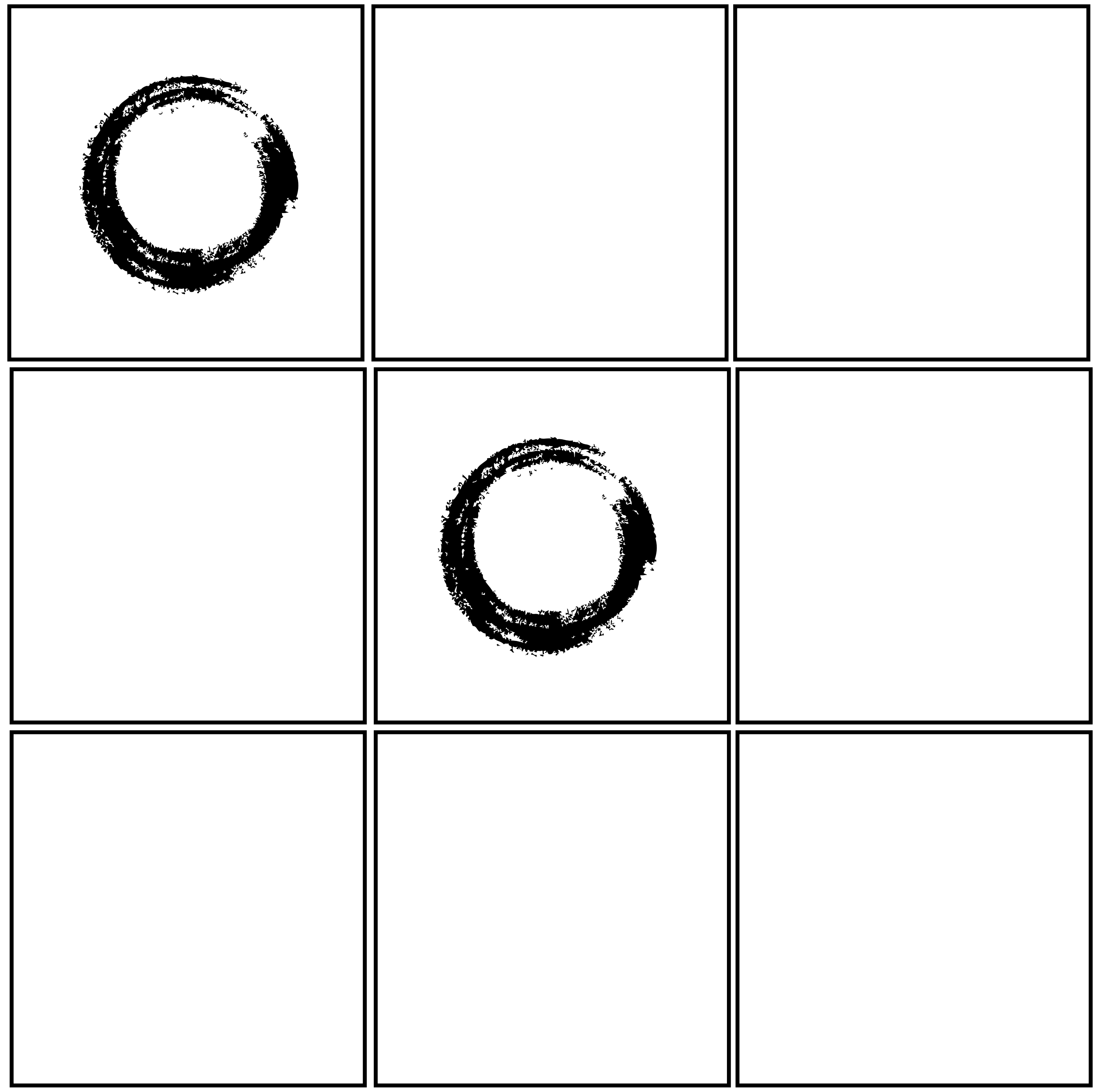
$\frac{7}{8}+\varepsilon$

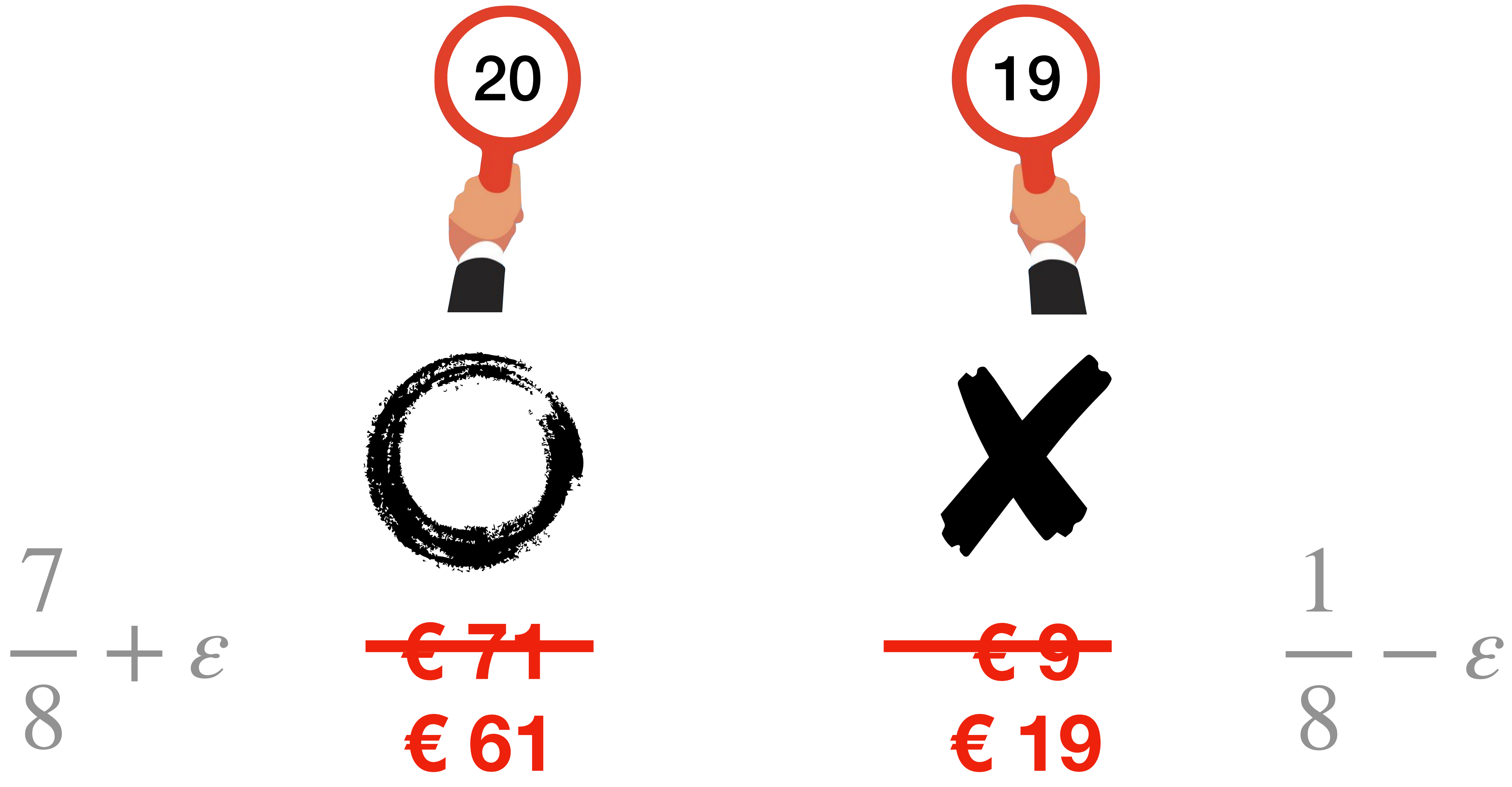
~~€ 71~~
€ 61

~~€ 9~~
€ 19

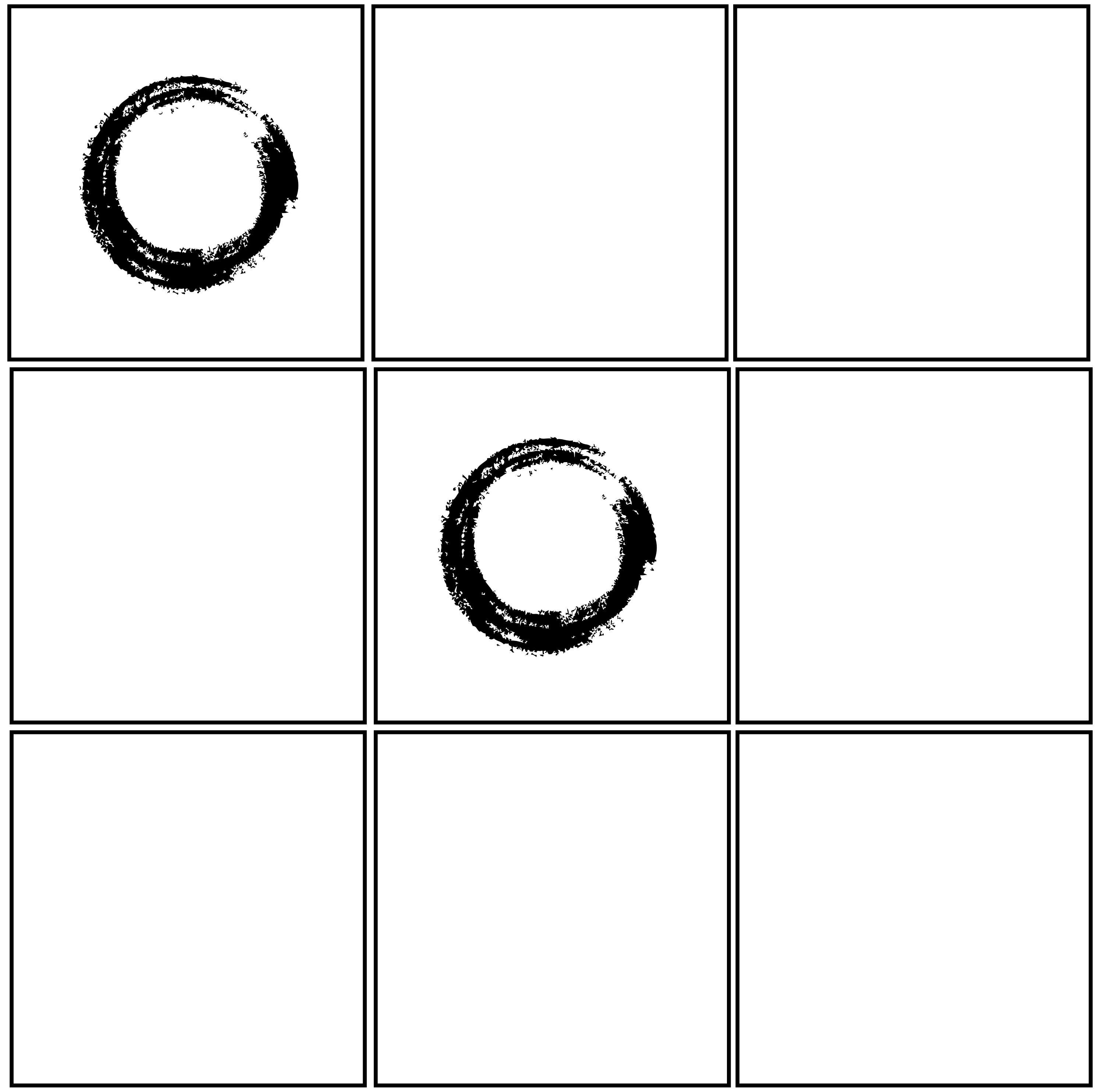
$\frac{1}{8}-\varepsilon$

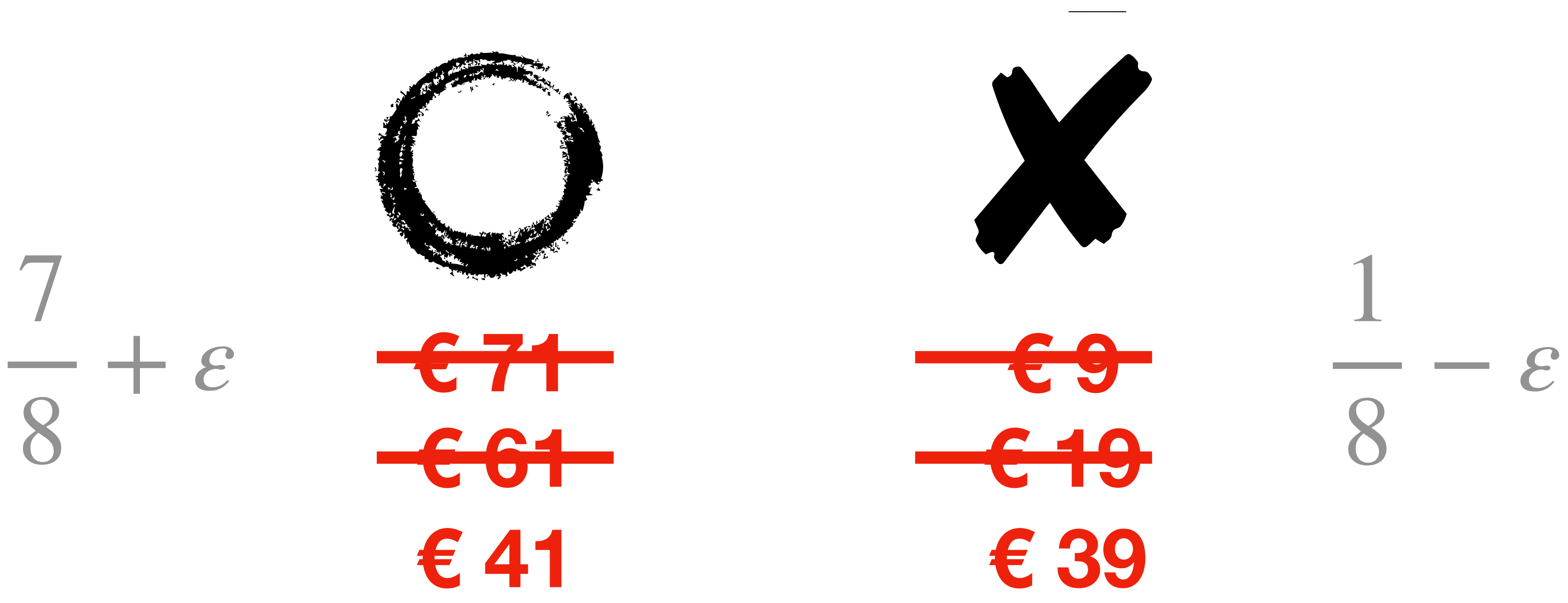
Bid-Tac-Toe



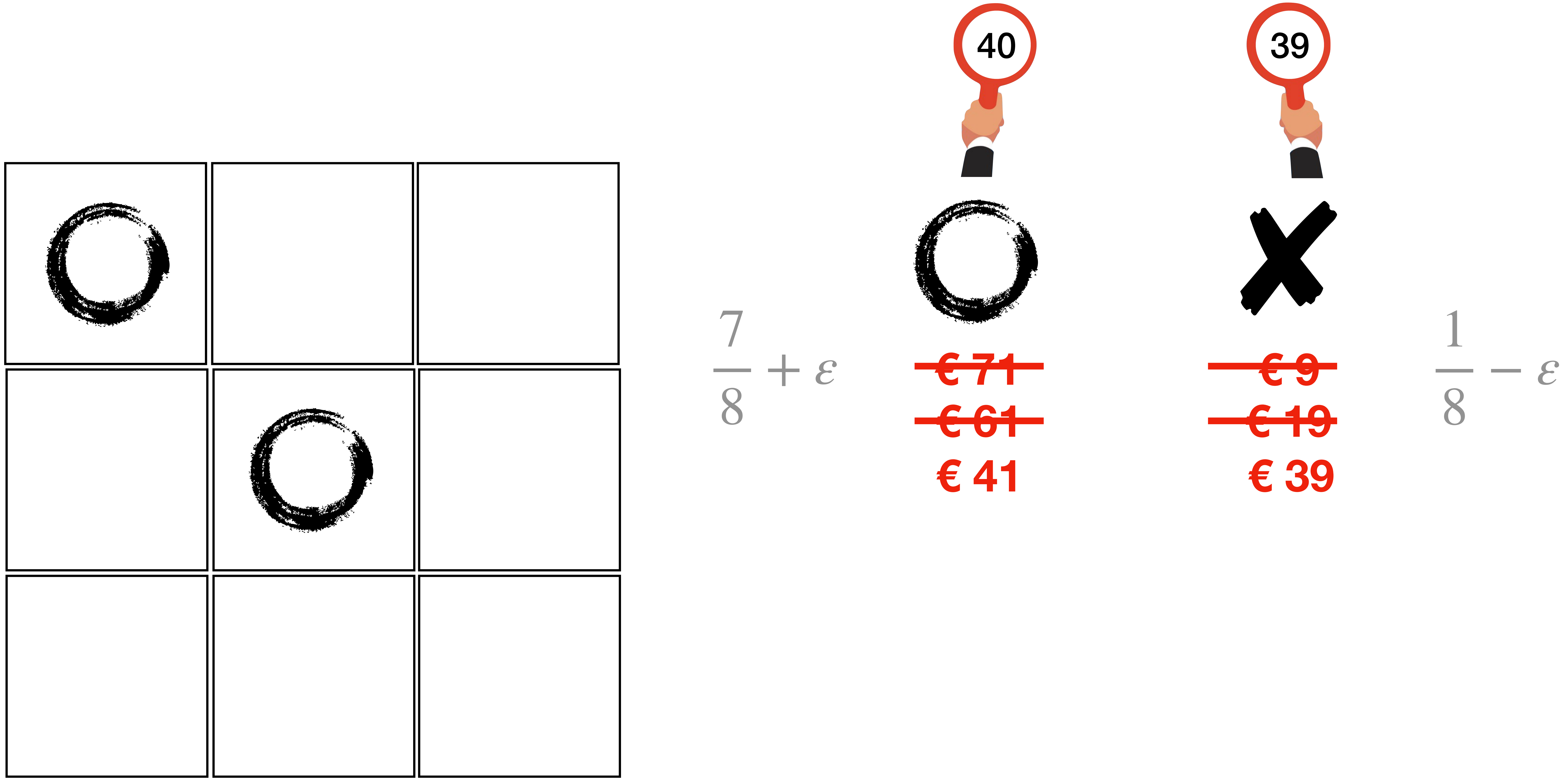


Bid-Tac-Toe

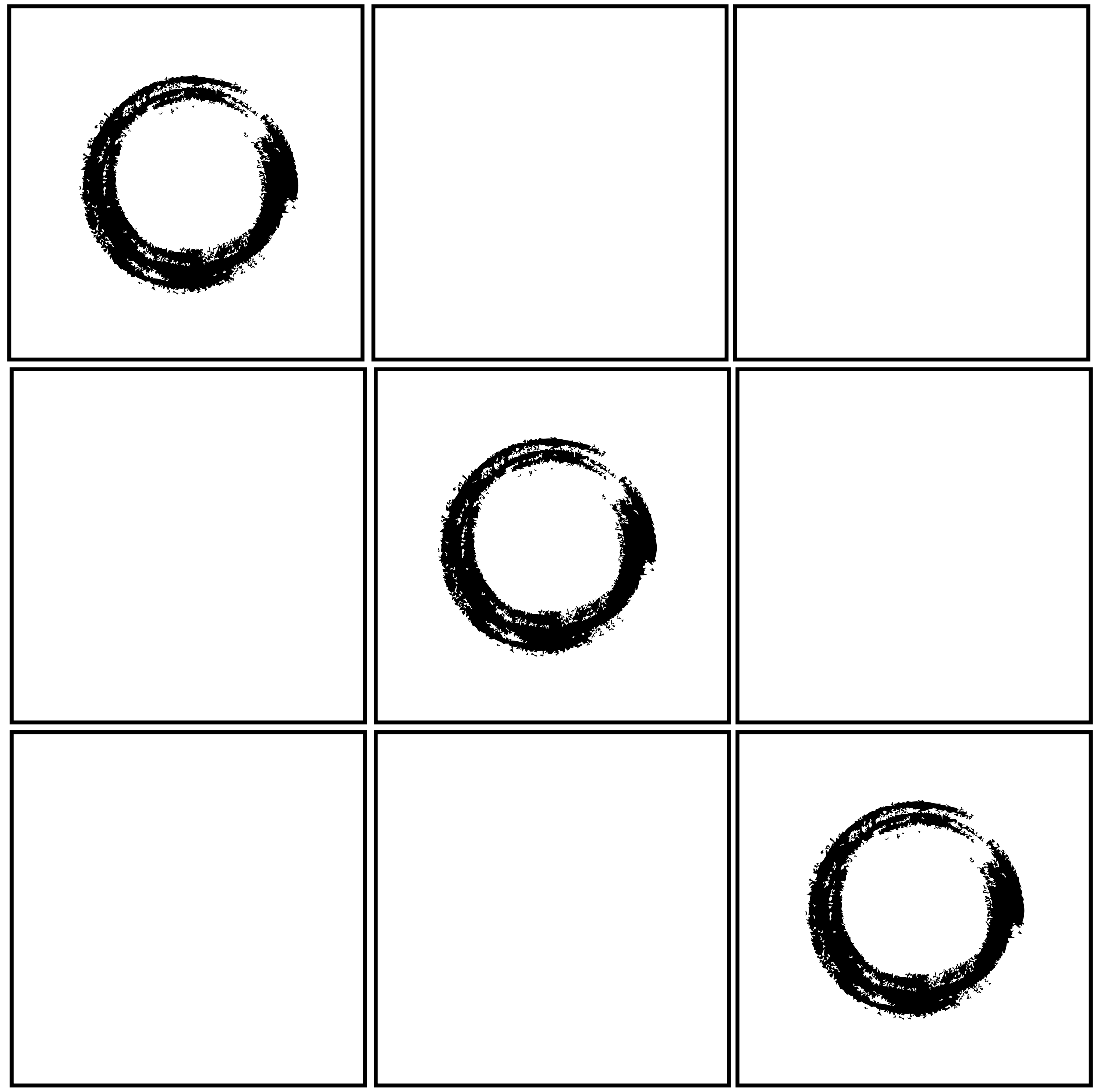


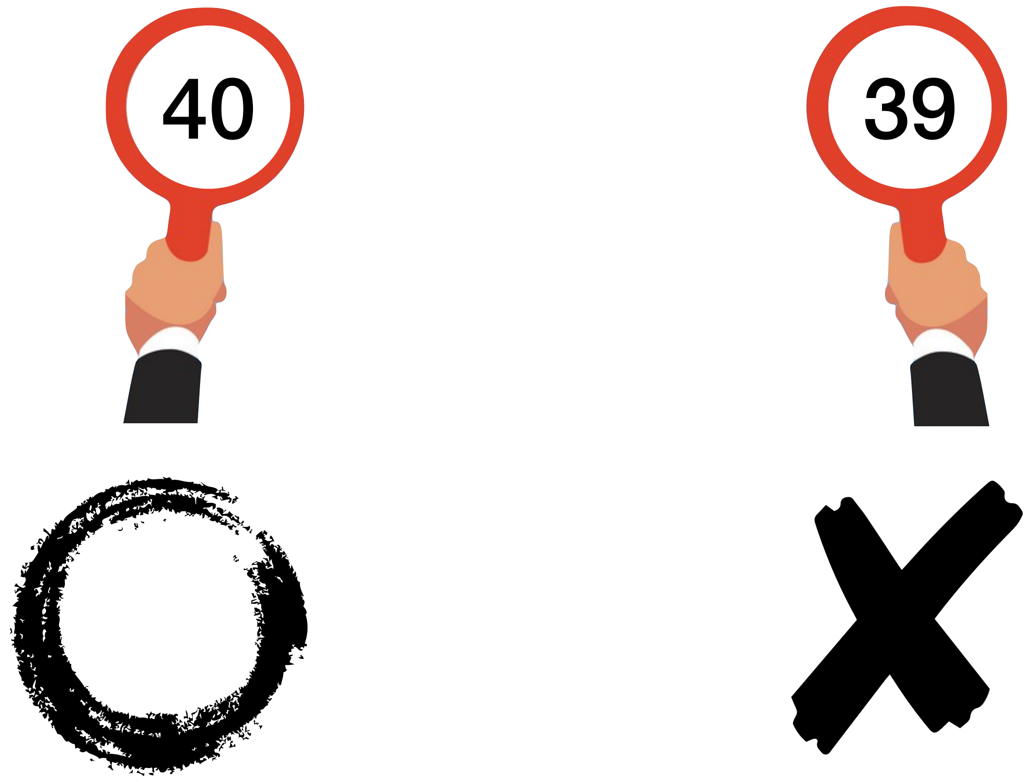


Bid-Tac-Toe



Bid-Tac-Toe





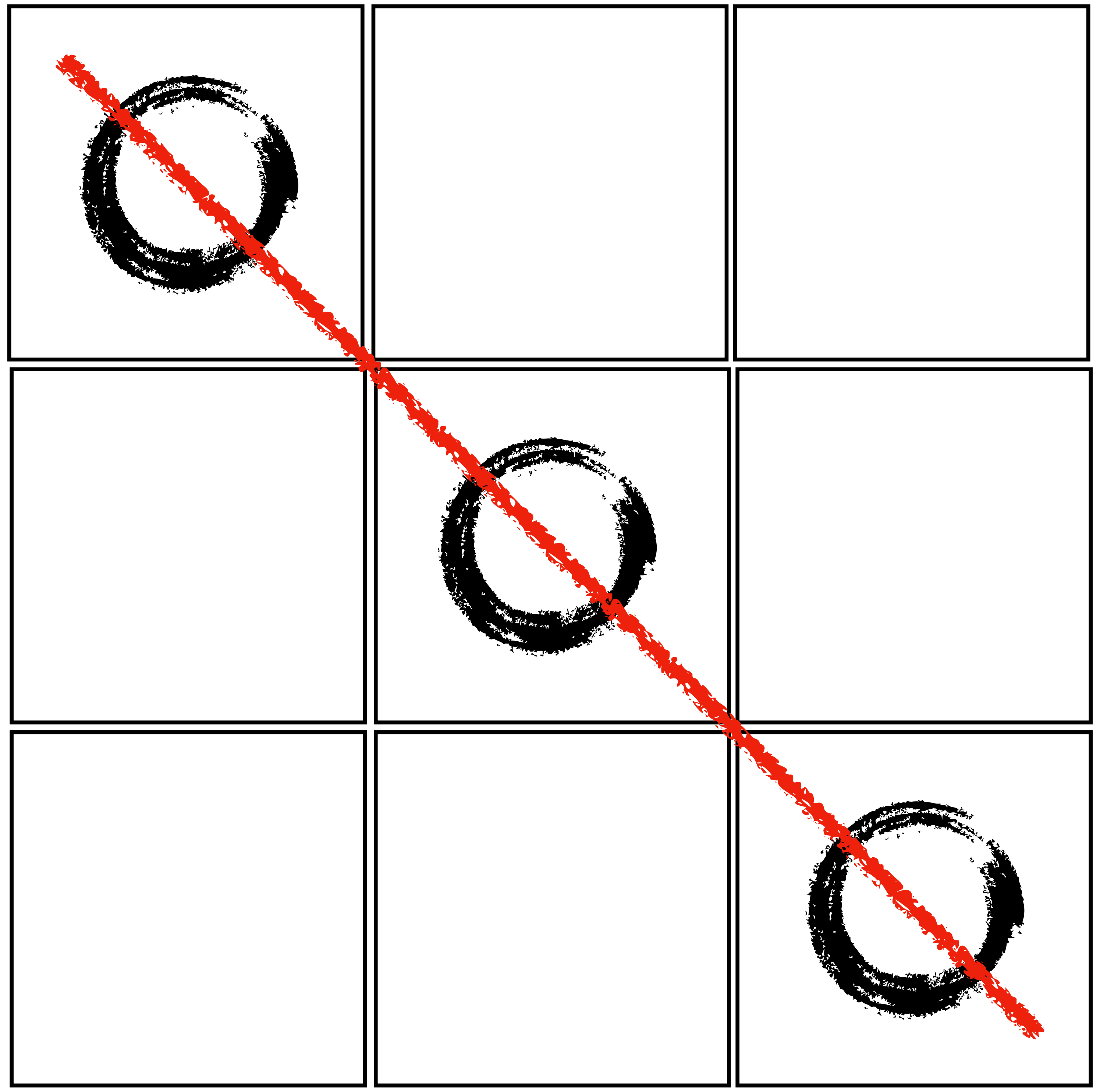
$\frac{7}{8} + \varepsilon$

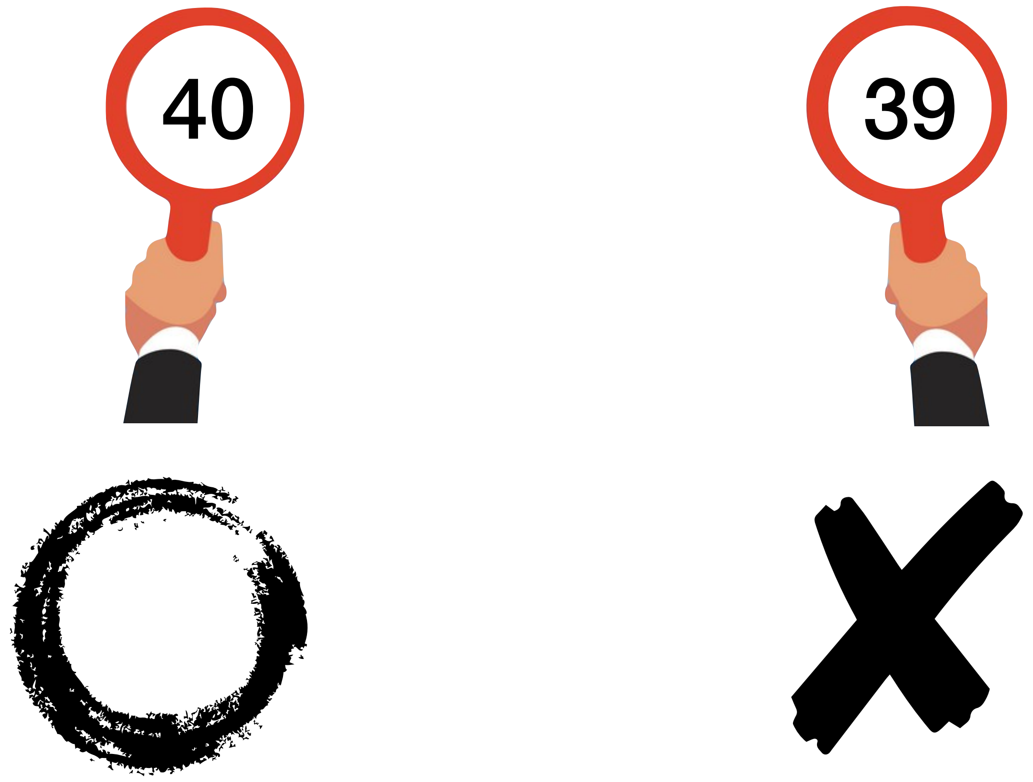
~~€ 71~~
~~€ 61~~
€ 41

~~€ 9~~
~~€ 19~~
€ 39

$\frac{1}{8} - \varepsilon$

Bid-Tac-Toe





$\frac{7}{8}+\varepsilon$

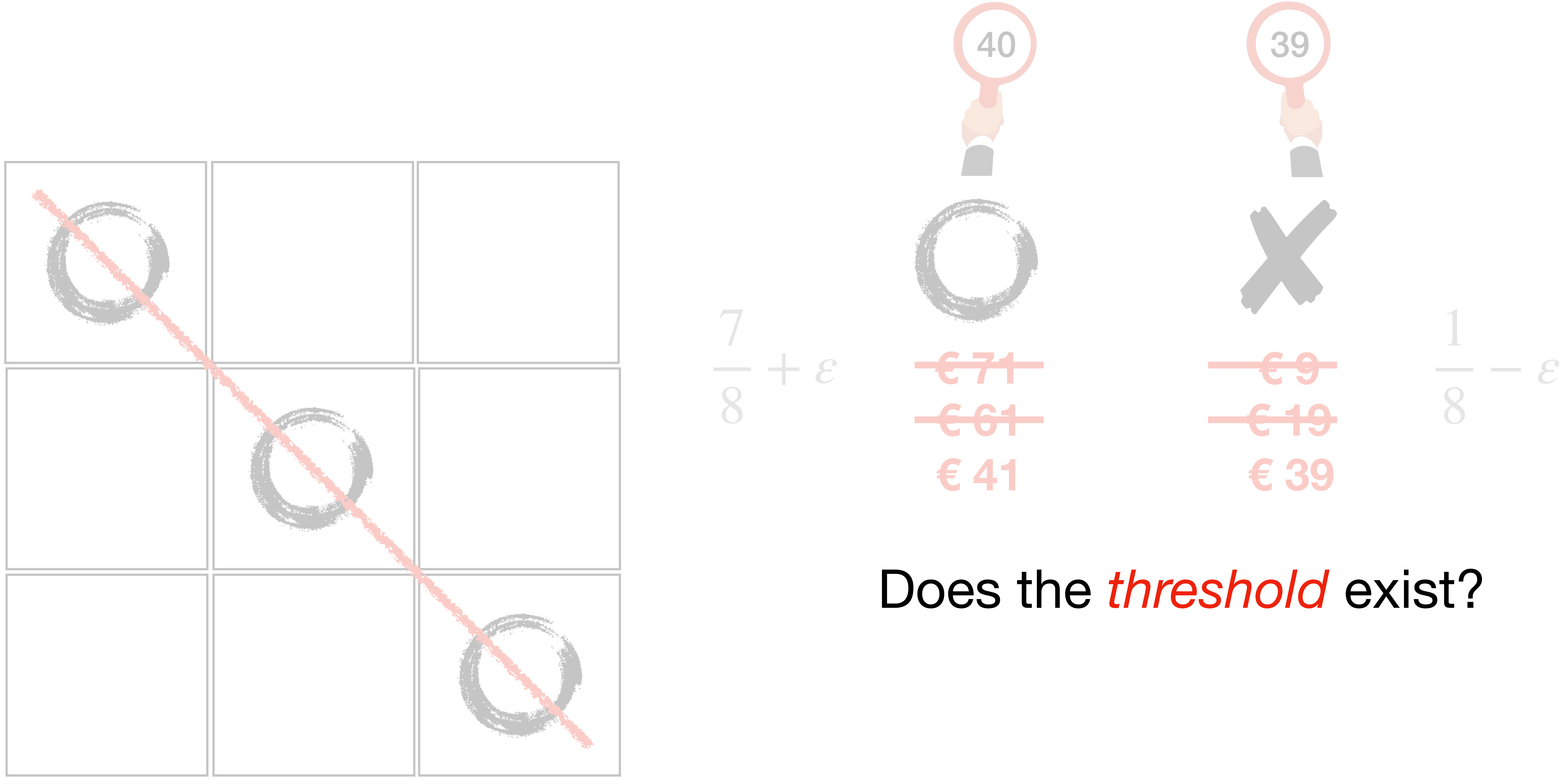
~~€ 71~~
~~€ 61~~
€ 41

~~€ 9~~
~~€ 19~~
€ 39

$\frac{1}{8}-\varepsilon$

[Lazarus et al. ’99, Develin & Payne ’08, Meir et al. ’18, Avni et al. ’19,…]

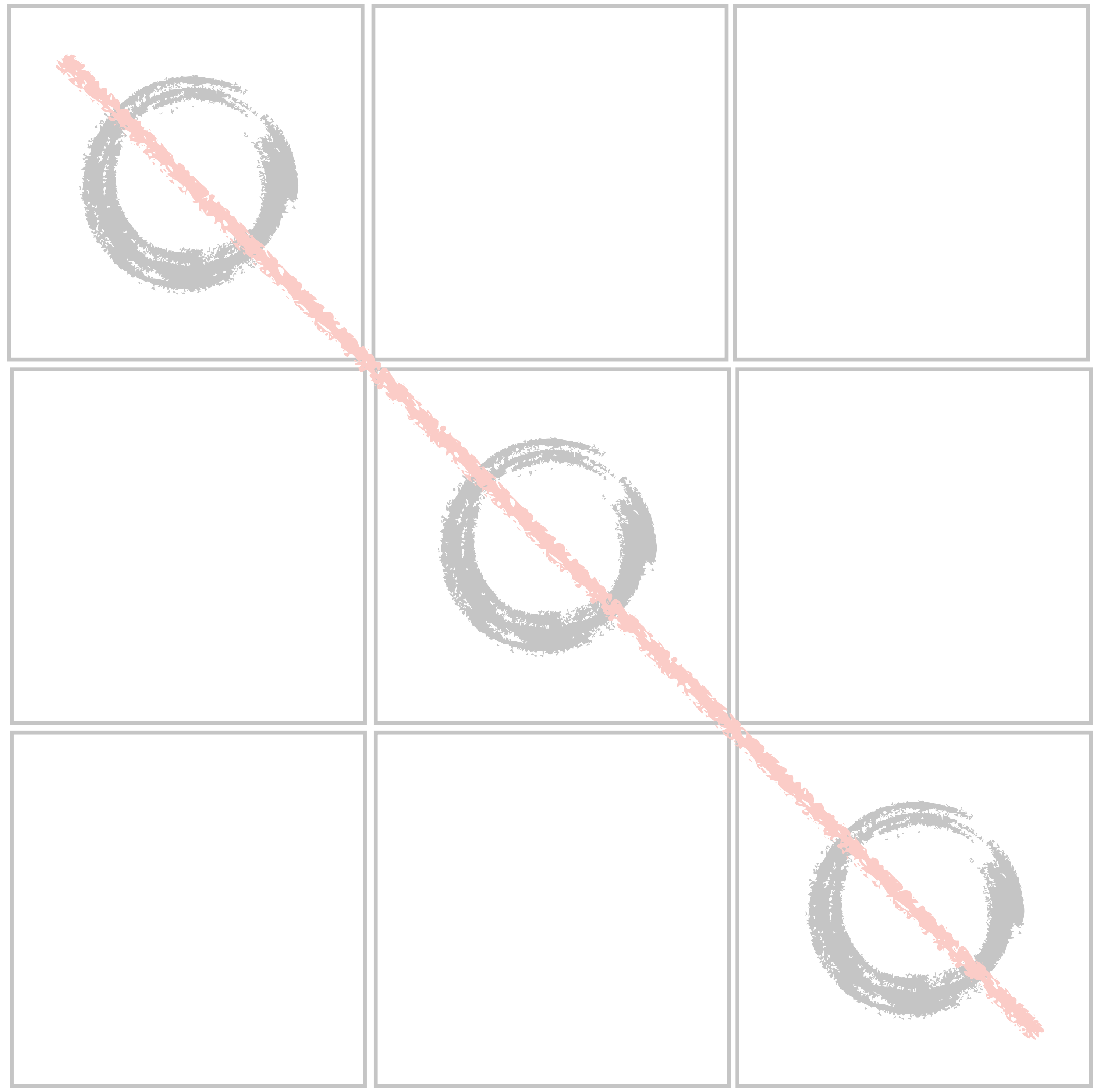
Bid-Tac-Toe

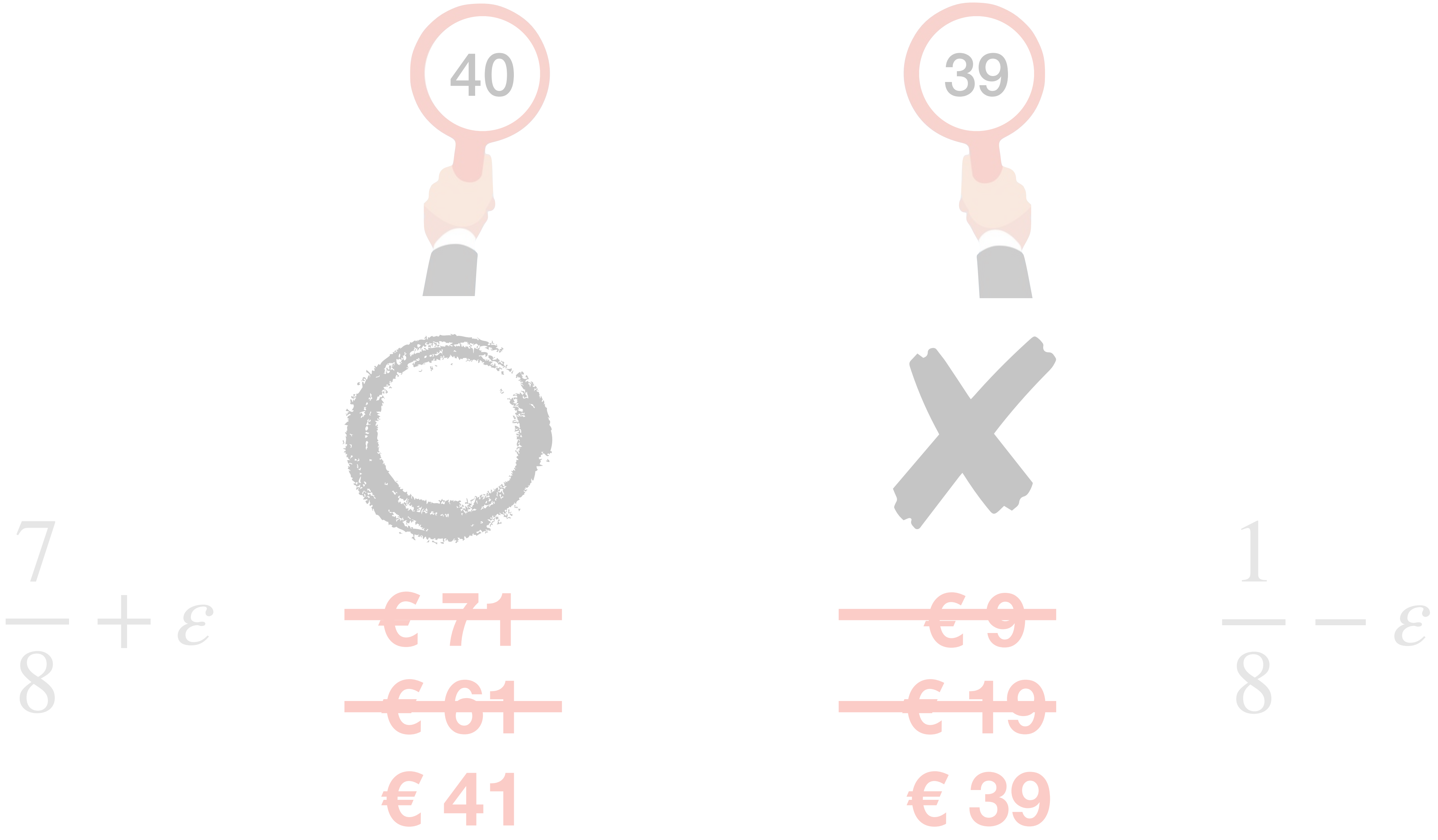


Does the *threshold* exist?

[Lazarus et al. ’99, Develin & Payne ’08, Meir et al. ’18, Avni et al. ’19,…]

Bid-Tac-Toe



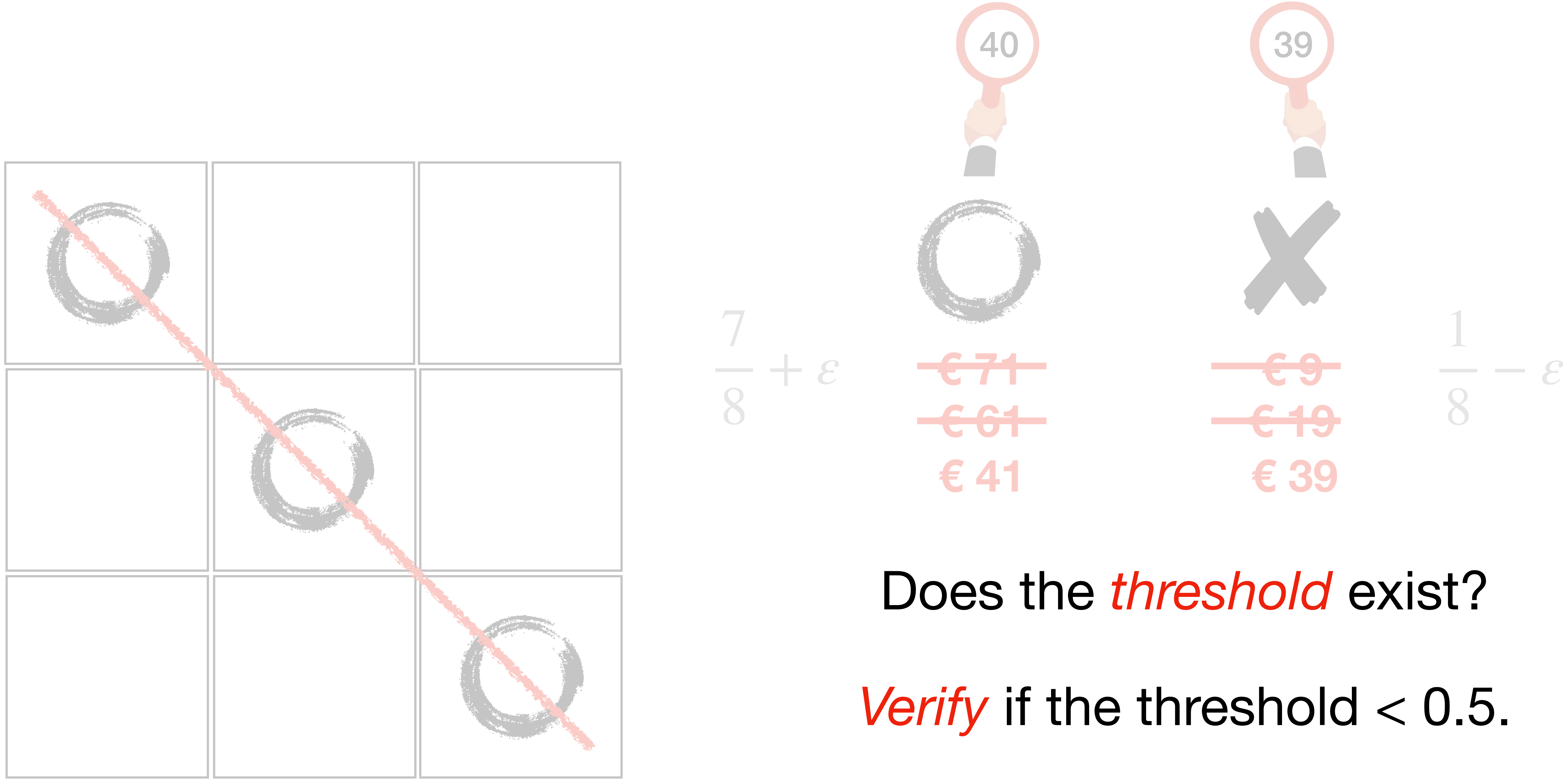


Does the *threshold* exist?

Verify if the threshold < 0.5.

[Lazarus et al. ’99, Develin & Payne ’08, Meir et al. ’18, Avni et al. ’19,…]

Bid-Tac-Toe



Does the *threshold* exist?

Verify if the threshold < 0.5.

Characterize the winning strategies.

Two Ongoing Projects

Bidding games with *charging*

- State-dependent monetary incentives

Ex.: ✗ earns 50 EUR when O captures 2 corners

- joint work with Guy Avni, Ehsan, and Tom

Two Ongoing Projects

Bidding games with *charging*

- State-dependent monetary incentives

Ex.: ✘ earns 50 EUR when  captures 2 corners

	Reach	Safe	Büchi	Co-Büchi	Rabin	Streett
Threshold	✓	✓	✓	✓		
Verification*	coNP	NP	Π_2^P	Σ_2^P	NP-hard	coNP-hard
Winning strategies	✓	✓	✓	✓		

*for Richman bidding

- joint work with Guy Avni, Ehsan, and Tom

Two Ongoing Projects

Bidding games with *charging*

- State-dependent monetary incentives

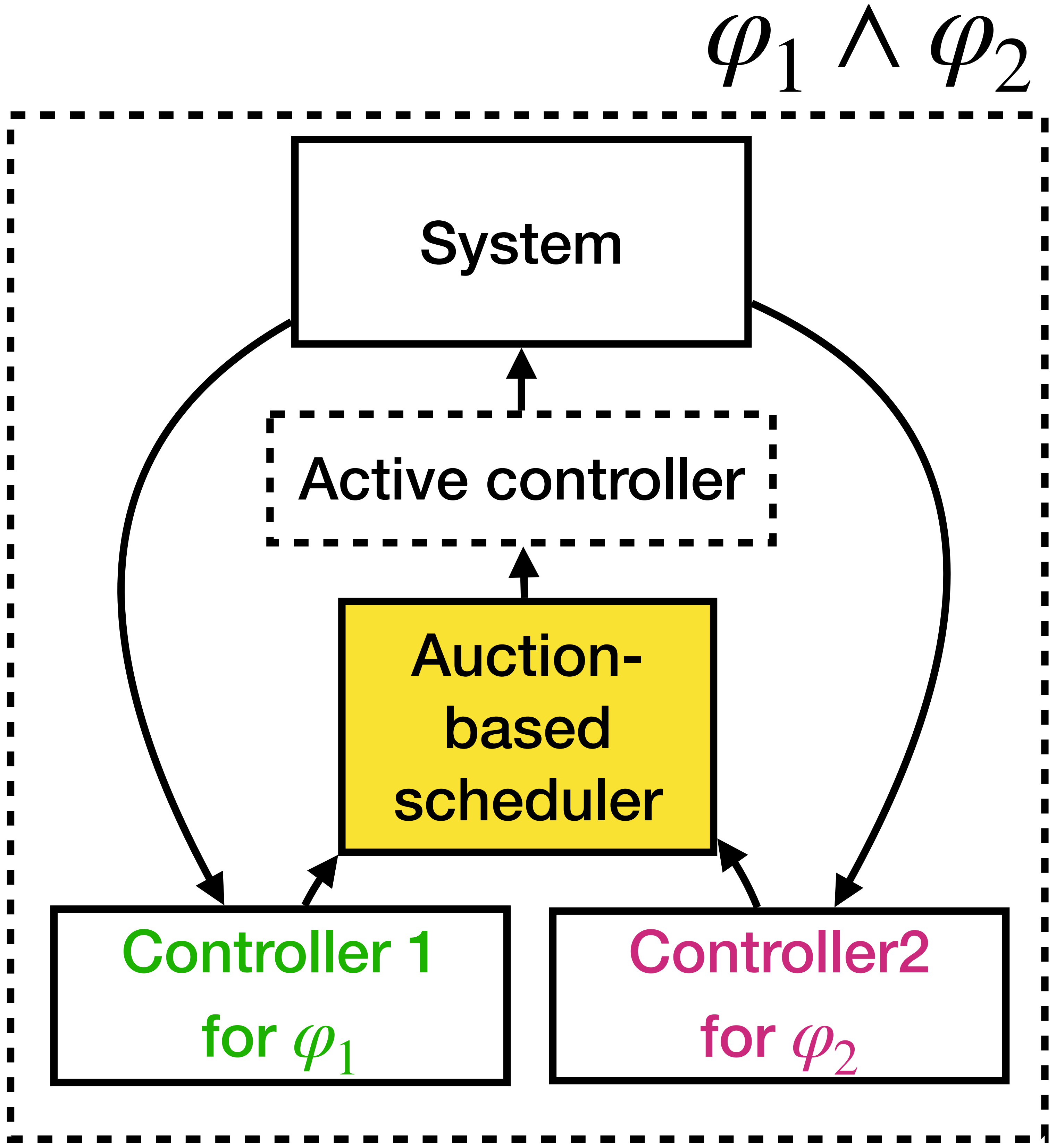
Ex.: ✗ earns 50 EUR when ○ captures 2 corners

	Reach	Safe	Büchi	Co-Büchi	Rabin	Streett
Threshold	✓	✓	✓	✓		
Verification*	coNP	NP	Π_2^P	Σ_2^P	NP-hard	coNP-hard
Winning strategies	✓	✓	✓	✓		

*for Richman bidding

- joint work with Guy Avni, Ehsan, and Tom

Auction-based scheduling



- joint work with Guy Avni and Suman Sadhukhan

Two Ongoing Projects

Bidding games with *charging*

- State-dependent monetary incentives

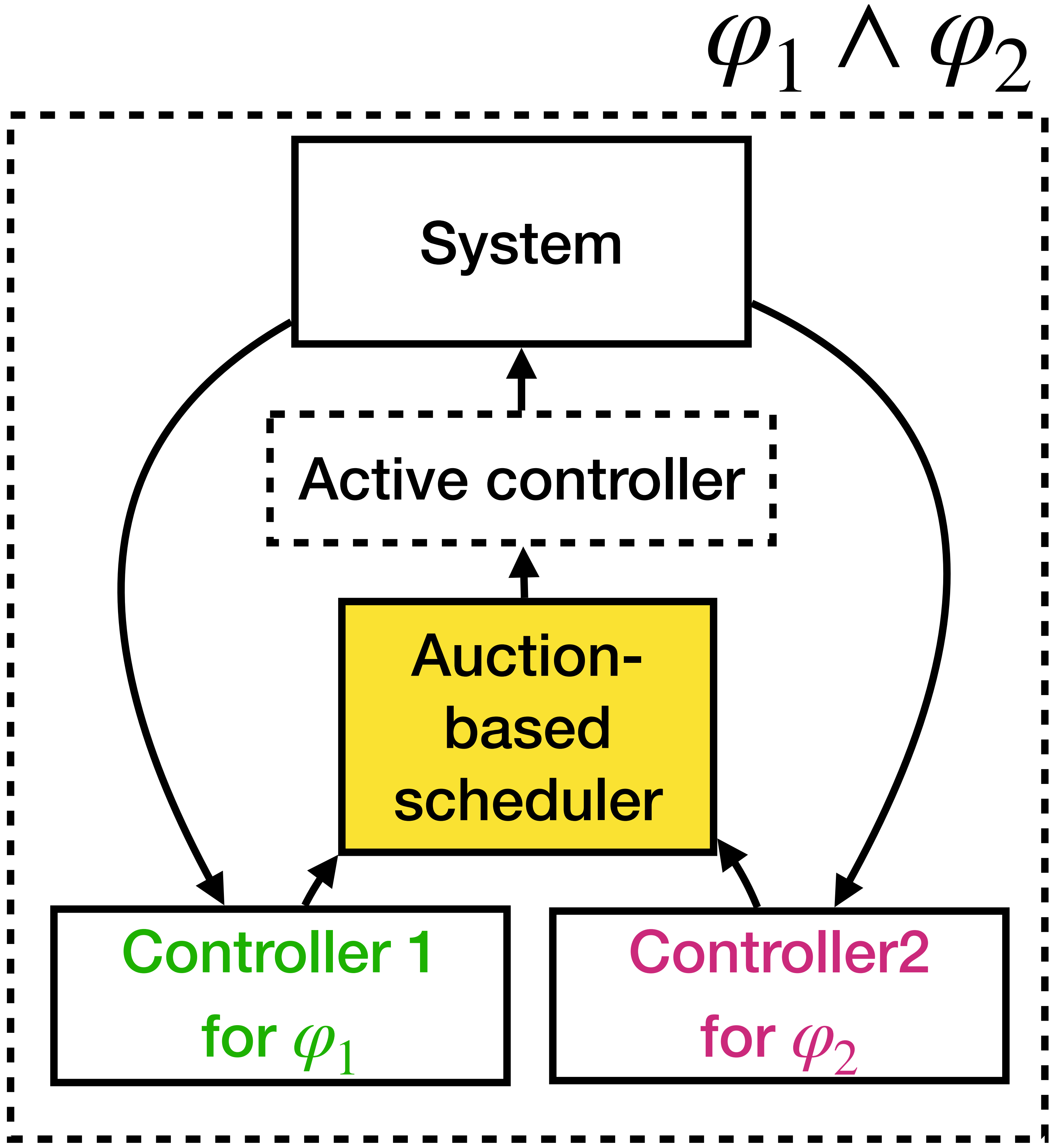
Ex.: ✗ earns 50 EUR when ○ captures 2 corners

	Reach	Safe	Büchi	Co-Büchi	Rabin	Streett
Threshold	✓	✓	✓	✓		
Verification*	coNP	NP	Π_2^P	Σ_2^P	NP-hard	coNP-hard
Winning strategies	✓	✓	✓	✓		

*for Richman bidding

- joint work with Guy Avni, Ehsan, and Tom

Auction-based scheduling



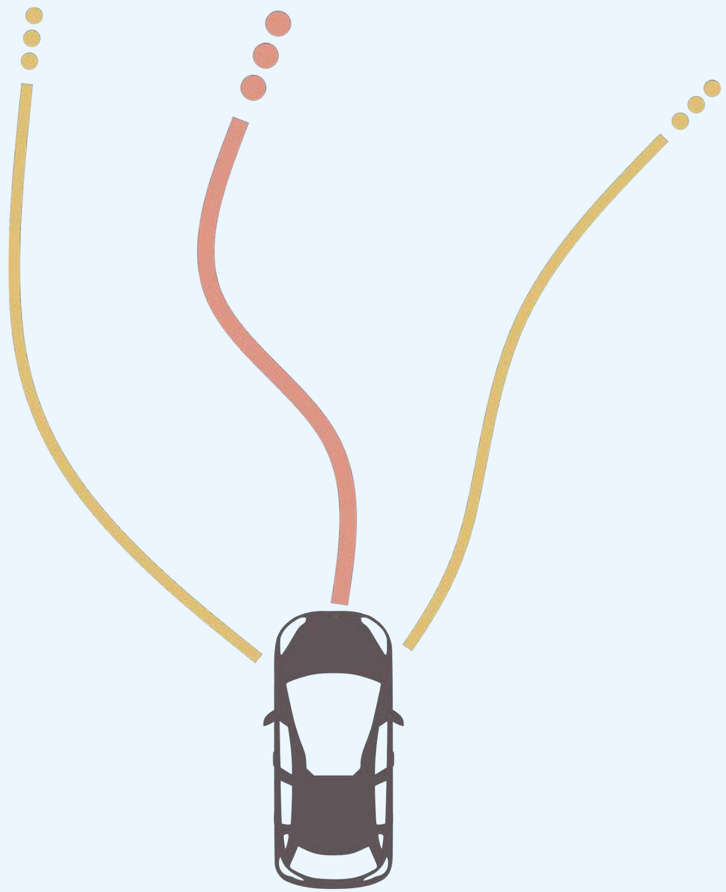
- joint work with Guy Avni and Suman Sadhukhan

Automated Analysis of Probabilistic Loops

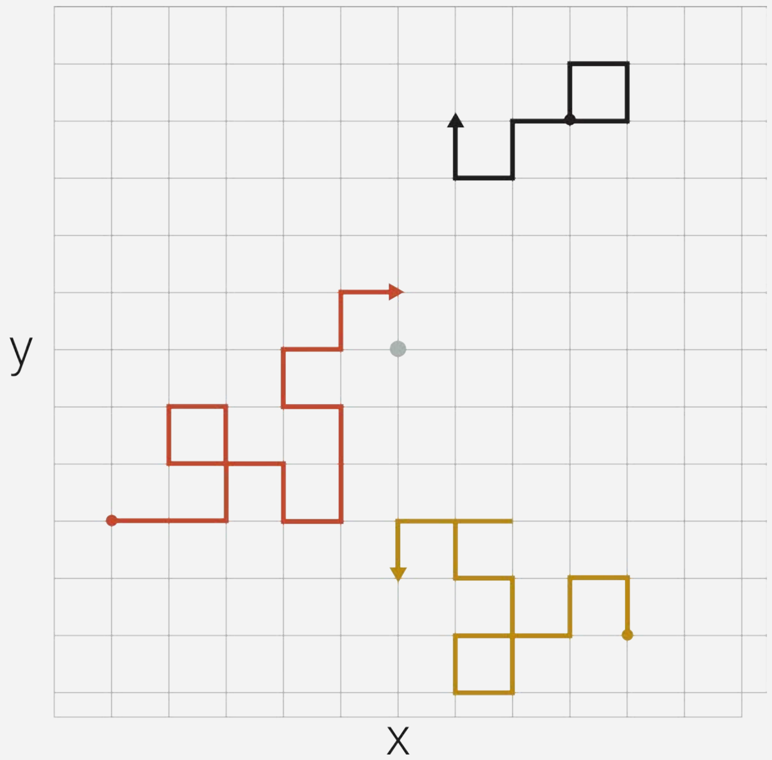
Marcel Moosbrugger

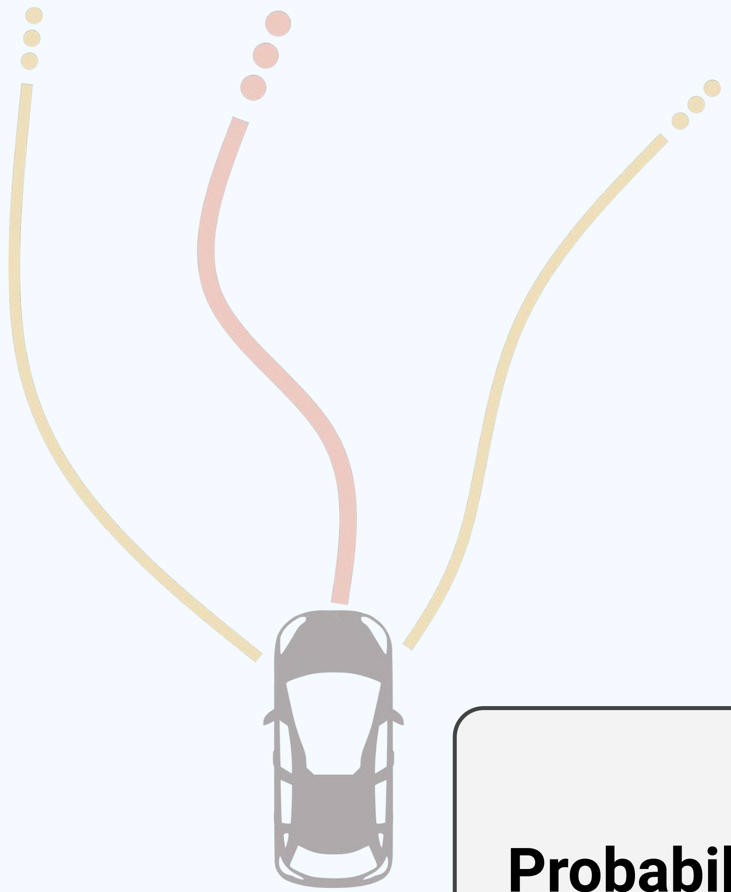
ISTA – October 2023

TU WIEN Informatics

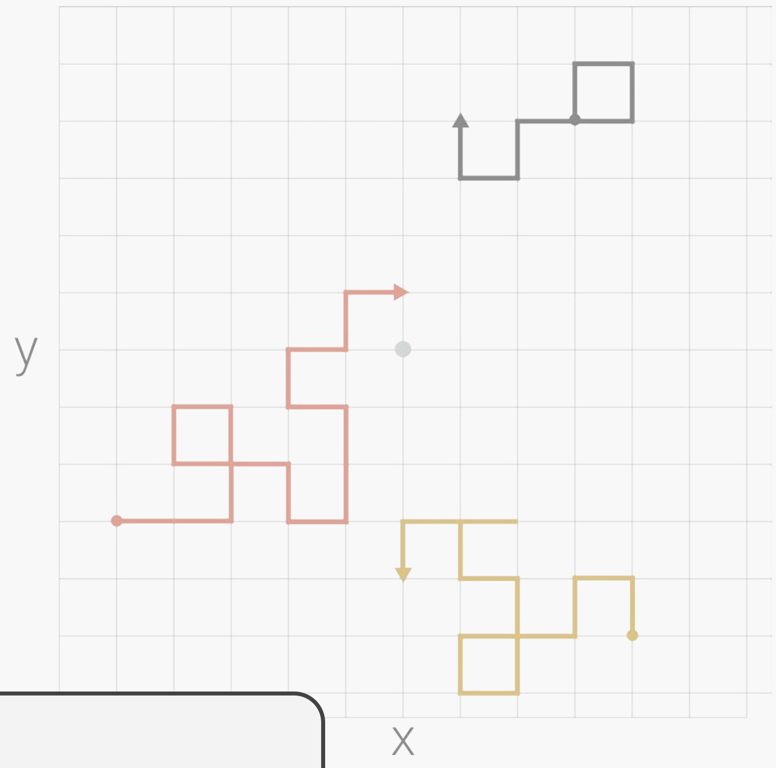


```
stop := 0
y := 1
x := 0
while stop == 0:
    stop := flip_coin()
    y := 2y
    x := x + 1
```





```
stop := 0
y := 1
x := 0
while stop == 0:
    stop := flip_coin()
    y := 2y
    x := x + 1
```



Probabilistic programs/loops as universal models.

```
stop := 0
y := 1
x := 0
while stop == 0:
    stop := flip_coin()
    y := 2y
    x := x + 1
```

→

MY PHD PROJECT

Develop **PL & verification** techniques to analyze **probabilistic loops**

Termination Analysis
[ESOP 2021, FM 2021, FMSD 2022]

Invariant Synthesis
[OOPSLA 2022, SAS 2022, FMSD 2023]

Sensitivity Analysis
[iFM 2023]

Predicting movement of robots under uncertainty
[QEST 2022, TOMACS 2023]

Focus on: automation, exact results (no sampling)

```
stop := 0
y := 1
x := 0
while stop == 0:
    stop := flip_coin()
    y := 2y
    x := x + 1
```

→

MY PHD PROJECT

Develop **PL & verification** techniques to analyze **probabilistic loops**

Polar Tool:
Probabilistic Loop Analyzer

https://github.com/probing-lab/polar

Ongoing Work

Theoretical foundations: Hardness bounds
Stability of control systems with uncertainty

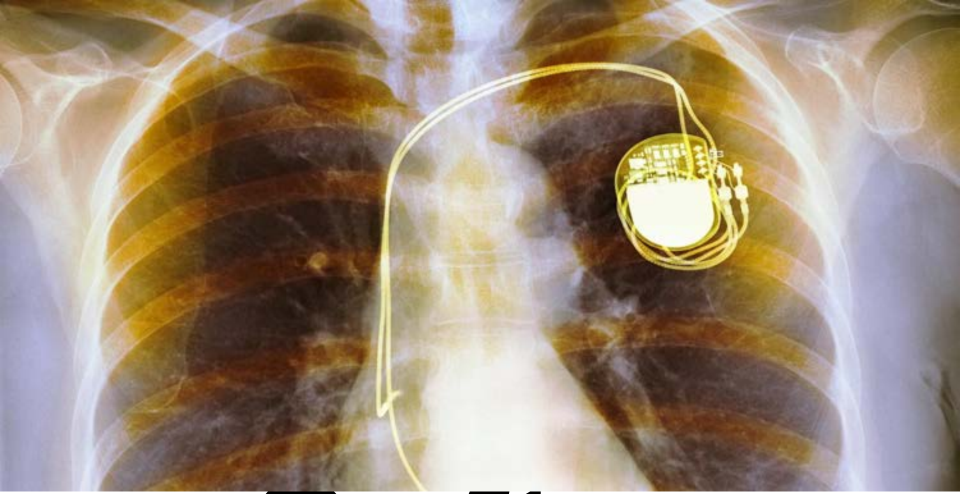
Focus on: automation, exact results (no sampling)

Solving Stochastic Games Reliably

Maximilian Weininger

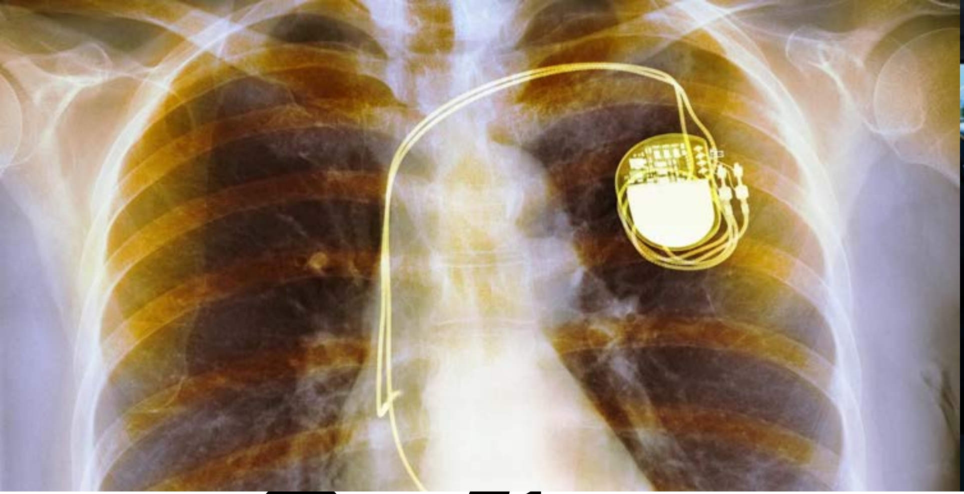
ISTA Seminar
09.10.2023

Software has bugs



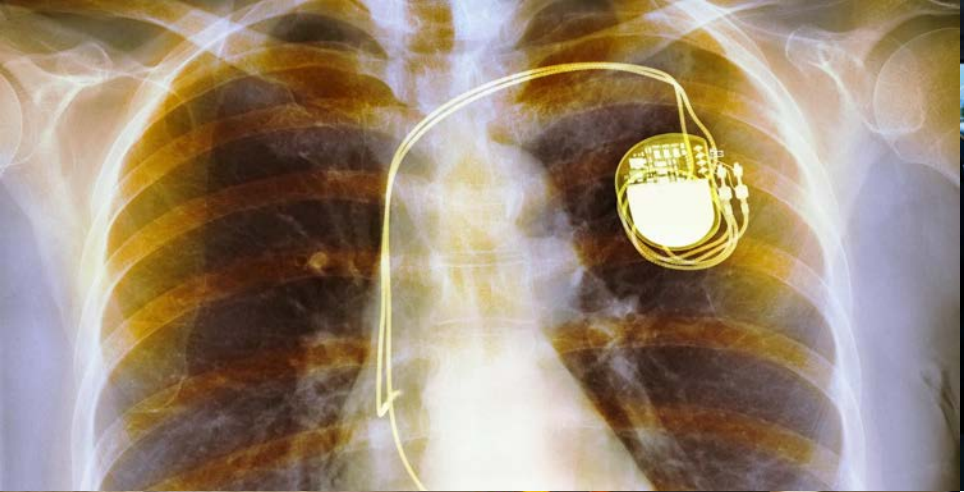
Software has bugs

https://bilder.t-online.de/b/76/51/02/04/id_76510204/tid_da/ein-herzschrittmacher-soll-leben-retten-hacker-koennten-ihn-als-mordwerkzeug-nutzen-.jpg



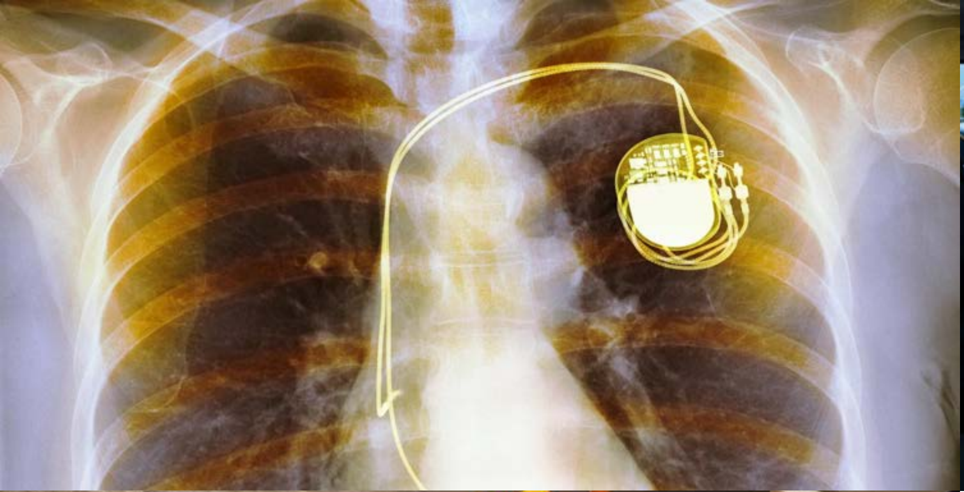
Software has bugs

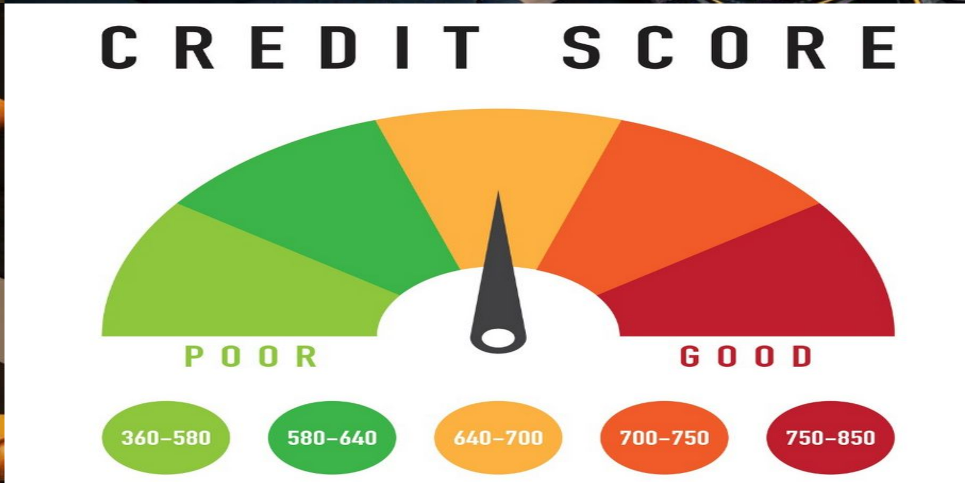
https://bilder.t-online.de/b/76/51/02/04/id_76510204/tid_da/ein-herzschrittmacher-soll-leben-retten-hacker-koennten-ihn-als-mordwerkzeug-nutzen-.jpg
https://miro.medium.com/max/1200/1*GRIl0B9HNJaTr7RVnqDRhg.jpeg



has bugs

https://bilder.t-online.de/b/76/51/02/04/id_76510204/tid_da/ein-herzschrittmacher-soll-leben-retten-hacker-koennten-ihn-als-mordwerkzeug-nutzen-.jpg
https://miro.medium.com/max/1200/1*GRIl0B9HNJaTr7RVnqDRhg.jpeg
https://il3.picdn.net/shutterstock/videos/10426736/thumb/1.jpg





https://bilder.t-online.de/b/76/51/02/04/id_76510204/tid_da/ein-herzschrittmacher-soll-leben-retten-hacker-koennten-ihn-als-mordwerkzeug-nutzen-.jpg
https://miro.medium.com/max/1200/1*GRIl0B9HNJaTr7RVnqDRhg.jpeg
https://il3.picdn.net/shutterstock/videos/10426736/thumb/1.jpg
https://entrepreneursbreak.com/wp-content/uploads/2020/05/Credit-Score.jpg

FORMAL VERIFICATION

Formal verification

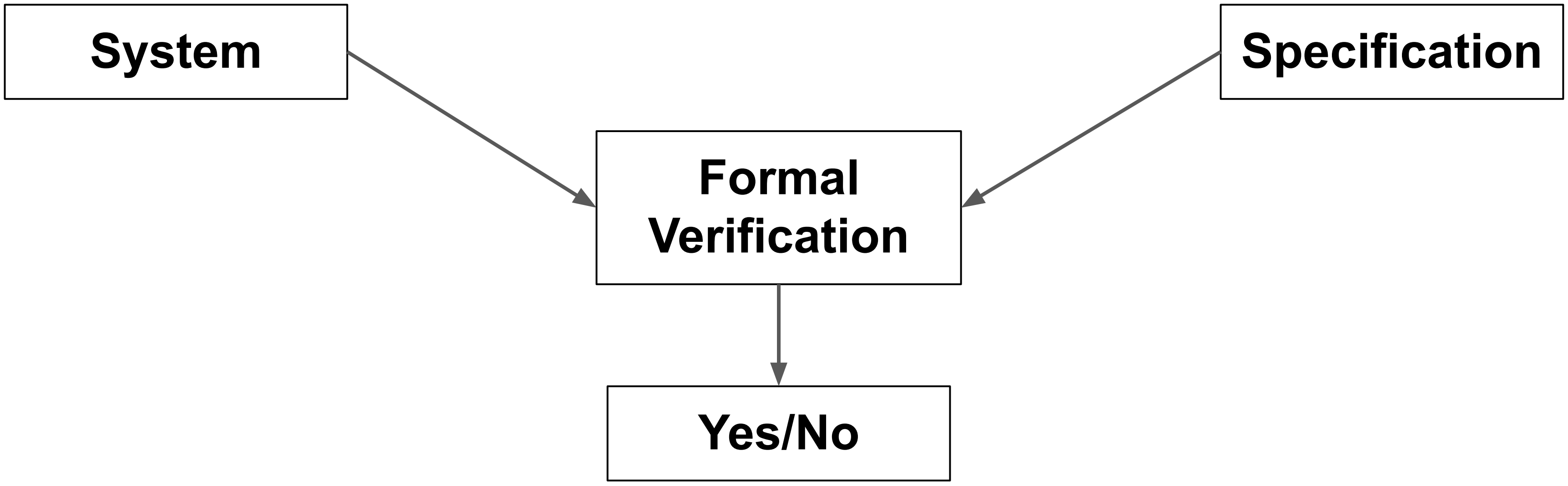
System

Specification

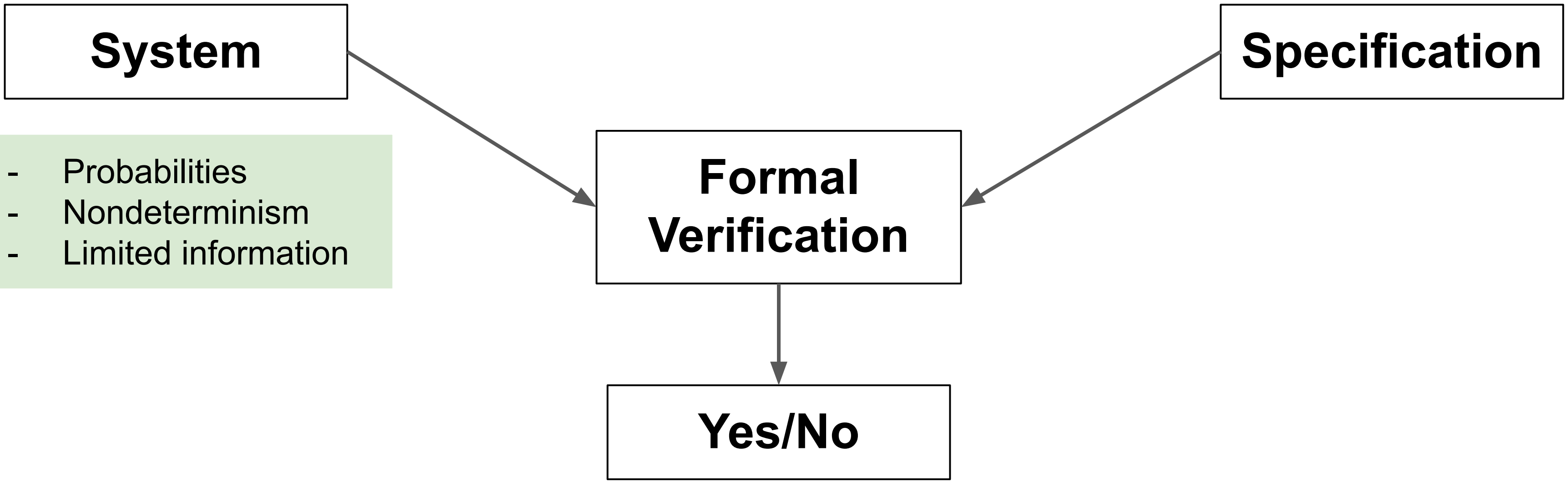
Formal Verification

Yes/No

Formal verification with special effects

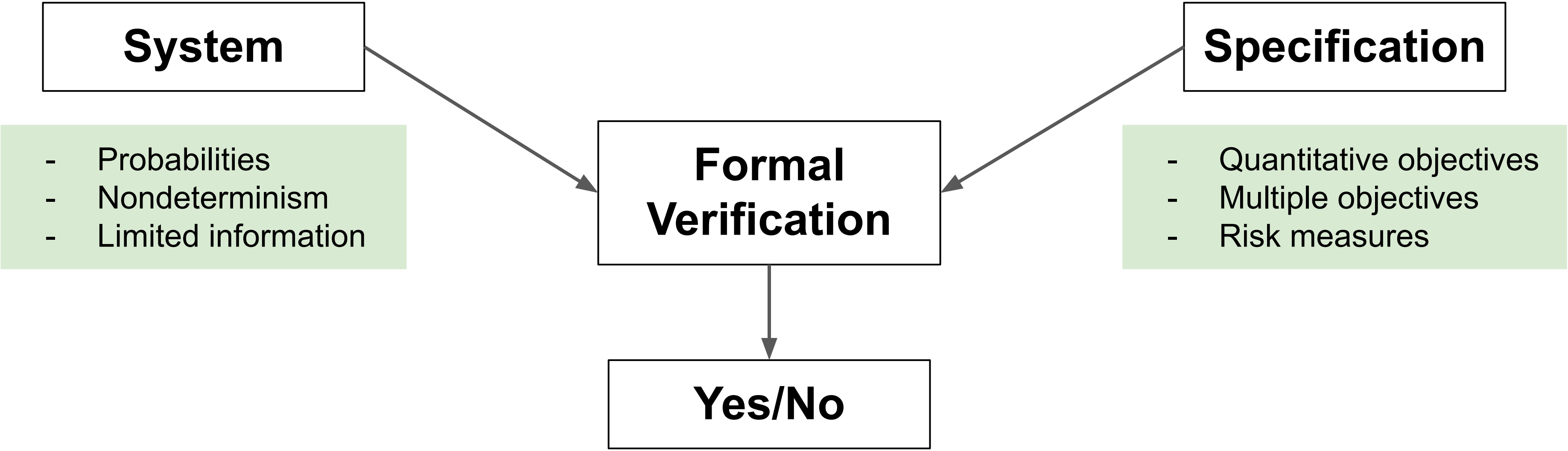


Formal verification with special effects

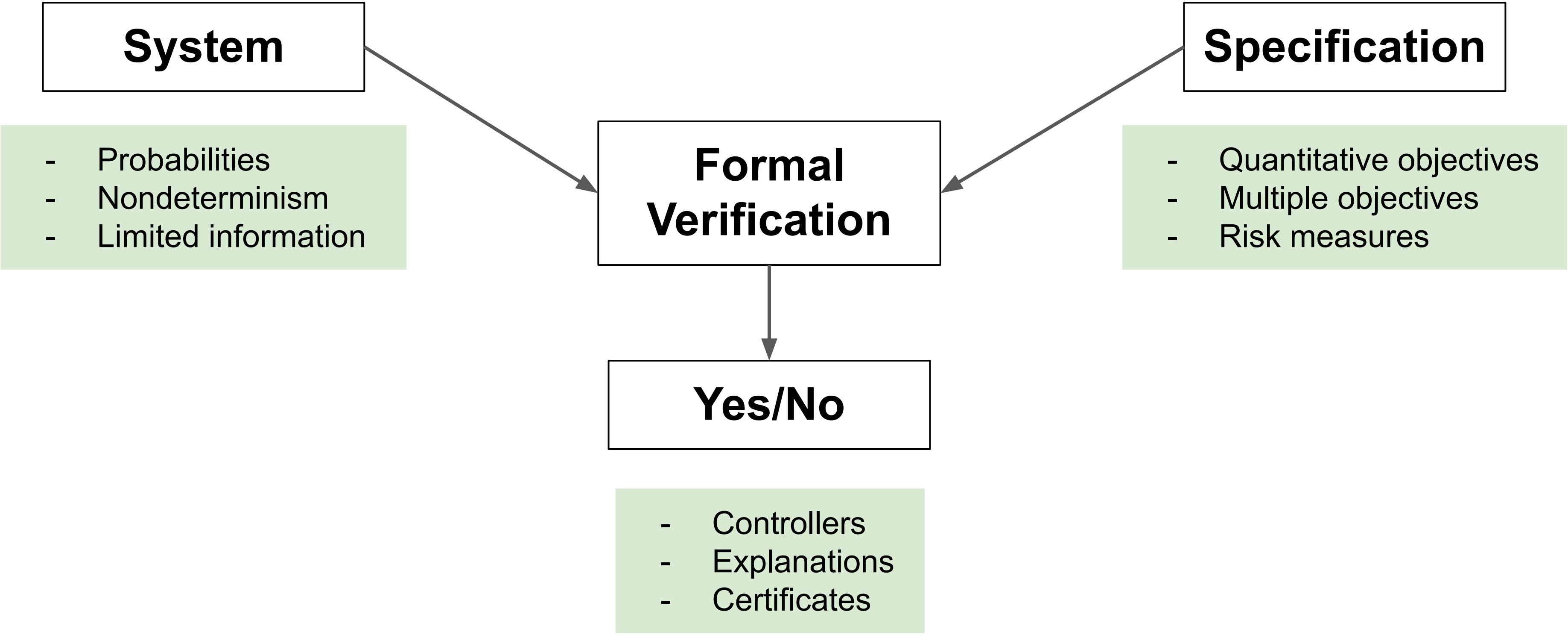


- Probabilities
- Nondeterminism
- Limited information

Formal verification with special effects



Formal verification with special effects



Ground orderedness in superposition

Márton Hajdu

October 4, 2023

The superposition calculus

- The superposition calculus is the state-of-the-art approach for first-order equational logic

The superposition calculus

- The superposition calculus is the state-of-the-art approach for first-order equational logic

$$\frac{s[u] \bowtie t \lor C \qquad l \simeq r \lor D}{(s[r] \bowtie t \lor C \lor D)\theta}$$

where $\theta = mgu(u, l)$, u not a variable, $r\theta \not\succeq l\theta$, $t\theta \not\succeq s[u]\theta$ and $C\theta \not\succeq s[u] \bowtie t\theta$

The superposition calculus

- The superposition calculus is the state-of-the-art approach for first-order equational logic

$$\frac{s[u] \bowtie t \vee C \qquad l \simeq r \vee D}{(s[r] \bowtie t \vee C \vee D)\theta}$$

where $\theta = mgu(u, l)$, u not a variable, $r\theta \not\succeq l\theta$, $t\theta \not\succeq s[u]\theta$ and $C\theta \not\succeq s[u] \bowtie t\theta$

- Strong restrictions on the inferences and redundancy elimination make it efficient

The superposition calculus

- The superposition calculus is the state-of-the-art approach for first-order equational logic

$$\frac{s[u] \bowtie t \vee C \qquad l \simeq r \vee D}{(s[r] \bowtie t \vee C \vee D)\theta}$$

where $\theta = mgu(u, l)$, u not a variable, $r\theta \not\succeq l\theta$, $t\theta \not\succeq s[u]\theta$ and $C\theta \not\succeq s[u] \bowtie t\theta$

- Strong restrictions on the inferences and redundancy elimination make it efficient
- It can also be adapted to arithmetic, induction, HOL, etc.

The superposition calculus

- The superposition calculus is the state-of-the-art approach for first-order equational logic

$$\frac{s[u] \bowtie t \vee C \qquad l \simeq r \vee D}{(s[r] \bowtie t \vee C \vee D)\theta}$$

where $\theta = mgu(u, l)$, u not a variable, $r\theta \not\succeq l\theta$, $t\theta \not\succeq s[u]\theta$ and $C\theta \not\succeq s[u] \bowtie t\theta$

- Strong restrictions on the inferences and redundancy elimination make it efficient
- It can also be adapted to arithmetic, induction, HOL, etc.

The superposition calculus

- The superposition calculus is the state-of-the-art approach for first-order equational logic

$$\frac{s[u] \bowtie t \vee C \qquad l \simeq r \vee D}{(s[r] \bowtie t \vee C \vee D)\theta}$$

where $\theta = mgu(u, l)$, u not a variable, $r\theta \not\succeq l\theta$, $t\theta \not\succeq s[u]\theta$ and $C\theta \not\succeq s[u] \bowtie t\theta$
- Strong restrictions on the inferences and redundancy elimination make it efficient
- It can also be adapted to arithmetic, induction, HOL, etc.

Example

Given $f > a > b > c$

$$\frac{P(f(f(a,x),c)) \qquad f(f(y,b),z) \simeq f(y,f(b,z))}{P(f(a,f(b,c)))} \; \theta = \begin{Bmatrix} x \mapsto b, \\ y \mapsto a, \\ z \mapsto c \end{Bmatrix}$$

The orderedness redundancy criteria

Given $f > a > b > c$ and clause $f(x, y) \simeq f(y, x)$, this inference is redundant:

$$\frac{P(f(f(a,x),c)) \qquad f(f(y,b),z) \simeq f(y,f(b,z))}{P(f(a,f(b,c)))} \; \theta = \begin{Bmatrix} x \mapsto b, \\ y \mapsto a, \\ z \mapsto c \end{Bmatrix}$$

The orderedness redundancy criteria

Given $f > a > b > c$ and clause $f(x, y) \simeq f(y, x)$, this inference is redundant:

$$f(a, b) \simeq f(b, a)$$

reduces $\rightarrow P(f(f(a, b), c))$

smaller than $\rightarrow f(f(a, b), c) \simeq f(a, f(b, c))$

$$\frac{P(f(f(a, x), c)) \qquad f(f(y, b), z) \simeq f(y, f(b, z))}{P(f(a, f(b, c)))} \; \theta = \begin{Bmatrix} x \mapsto b, \\ y \mapsto a, \\ z \mapsto c \end{Bmatrix}$$

The orderedness redundancy criteria

Given $f > a > b > c$ and clause $f(x, y) \simeq f(y, x)$, this inference is redundant:

$$f(a, b) \simeq f(b, a)$$

reduces ↙ ↘ smaller than

$$P(f(f(a, b), c)) \qquad f(f(a, b), c) \simeq f(a, f(b, c))$$

$$\frac{P(f(f(a, x), c)) \qquad f(f(y, b), z) \simeq f(y, f(b, z))}{P(f(a, f(b, c)))} \; \theta = \left\{ \begin{array}{c} x \mapsto b, \\ y \mapsto a, \\ z \mapsto c \end{array} \right\}$$

Orderedness is a generalization of *compositeness* from completion-based theorem proving.

Ground orderedness

Given clauses $\{f(x, y) \simeq f(y, x), f(x, x) \simeq x\}$, consider the inference:

$$\frac{Q(f(f(x, y), z), f(y, x)) \qquad f(f(x, y), z) \simeq f(x, f(y, z))}{Q(f(x, f(y, z)), f(y, x))}$$

Ground orderedness

Given clauses $\{f(x,y) \simeq f(y,x), f(x,x) \simeq x\}$, consider the inference:

$f(x,y) \simeq f(y,x)$

assuming $x > y$ or $x < y$

reduces

smaller than

$Q(f(f(x,y),z), f(y,x))$ $\quad$ $f(f(x,y),z) \simeq f(x,f(y,z))$

$$\frac{Q(f(f(x,y),z), f(y,x)) \qquad f(f(x,y),z) \simeq f(x,f(y,z))}{Q(f(x,f(y,z)), f(y,x))}$$

Ground orderedness

Given clauses $\{f(x,y) \simeq f(y,x), f(x,x) \simeq x\}$, consider the inference:

$$f(x,x) \simeq x$$

assuming $x \sim y$

reduces

smaller than

$$Q(f(f(x,x),z),f(x,x)) \qquad f(f(x,x),z) \simeq f(x,f(x,z))$$

$$\frac{Q(f(f(x,y),z),f(y,x)) \qquad f(f(x,y),z) \simeq f(x,f(y,z))}{Q(f(x,f(y,z)),f(y,x))}$$

Ground orderedness

Given clauses $\{f(x,y) \simeq f(y,x), f(x,x) \simeq x\}$, consider the inference:

$f(x,x) \simeq x$

assuming $x \sim y$

reduces $\rightarrow$ $Q(f(f(x,x),z), f(x,x))$

smaller than $\rightarrow$ $f(f(x,x),z) \simeq f(x, f(x,z))$

$$\frac{Q(f(f(x,y),z), f(y,x)) \qquad f(f(x,y),z) \simeq f(x, f(y,z))}{Q(f(x, f(y,z)), f(y,x))}$$

The inference is redundant w.r.t. ground orderedness!

Ground orderedness

Given clauses $\{f(x,y) \simeq f(y,x), f(x,x) \simeq x\}$, consider the inference:

$$f(x,x) \simeq x$$

assuming $x \sim y$

reduces | smaller than

$$\frac{\begin{array}{cc} Q(f(f(x,x),z),f(x,x)) & f(f(x,x),z) \simeq f(x,f(x,z)) \\ Q(f(f(x,y),z),f(y,x)) & f(f(x,y),z) \simeq f(x,f(y,z)) \end{array}}{Q(f(x,f(y,z)),f(y,x))}$$

The inference is redundant w.r.t. ground orderedness!

Both orderedness and ground orderedness are currently being implemented in Vampire

Shorter, more usable proofs in SAT and beyond

Adrián Rebola-Pardo

Vienna University of Technology
Johannes Kepler University

IST Austria
October 9th, 2023

Wait, wasn't that a solved problem?

DRAT proofs have *weird* semantics

Wait, wasn't that a solved problem?

DRAT proofs have *weird* semantics

can derive clauses not implied by the premises

Wait, wasn't that a solved problem?

DRAT proofs have *weird* semantics

can derive clauses not implied by the premises

mutation
semantics

Wait, wasn't that a solved problem?

DRAT proofs have *weird* semantics

can derive clauses not implied by the premises

new SAT proof systems

mutation semantics

Wait, wasn't that a solved problem?

DRAT proofs have *weird* semantics

can derive clauses not implied by the premises

new SAT proof systems

clearer semantics

easier to generate

shorter proofs

smaller unsat cores

mutation semantics

Wait, wasn't that a solved problem?

DRAT proofs have *weird* semantics

can derive clauses not implied by the premises

can we extract interpolants?

new SAT proof systems

clearer semantics

easier to generate

shorter proofs

smaller unsat cores

mutation semantics

Wait, wasn't that a solved problem?

DRAT proofs have *weird* semantics

can derive clauses not implied by the premises

clearer semantics

can we extract interpolants?

easier to generate

new SAT proof systems

shorter proofs

smaller unsat cores

can we unify QBF proof systems?

mutation semantics

extension to QBF solving

Wait, wasn't that a solved problem?

DRAT proofs have *weird* semantics

can derive clauses not implied by the premises

clearer semantics

can we extract interpolants?

easier to generate

new SAT proof systems

shorter proofs

smaller unsat cores

can we unify QBF proof systems?

mutation semantics

extension to QBF solving

can we uniformly sample?

extension to model counting

Recognizing an Owl·Bear in the Forest

Regular Languages of Tree-Width Bounded Graphs

Mark Chimes

October 4, 2023

Finite alphabet **A** of terminal symbols e.g. $\{a, b, c, \ldots, z\}$

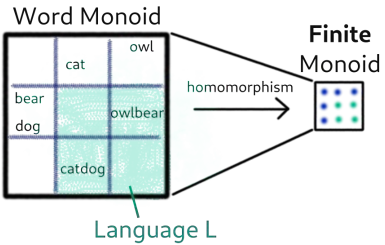
Regular languages

- Regular Expression
- Automaton
- Generated by Regular Grammar
- **Definable:** Monadic Second-Order Logic
- **Recognizable:** Inverse image under homomorphism into a finite monoid

Words

Words form a monoid $\langle \Sigma^*, \epsilon, \cdot \rangle$

$$owl \cdot bear = owlbear$$



Finite alphabet **A** of terminal symbols e.g. $\{a, b, c, \ldots, z\}$

Words

Words form a monoid $\langle \Sigma^*, \epsilon, \cdot \rangle$

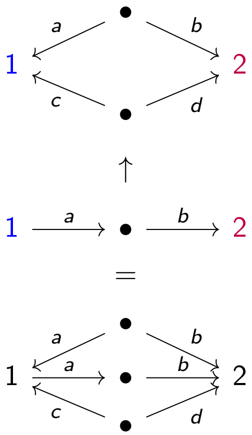
Graphs - Generalize Words

Label edges with symbols in $\mathbb{A}$

- Need to know *how* to combine two graphs
- Vertices are not ordered, but finitely many are numbered
- Graph operations combine graphs along numbers

Graphs form a **Multi-Sorted Magma** - generalizes Monoid.

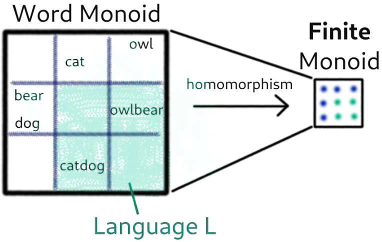
$owl \cdot bear = owlbear$

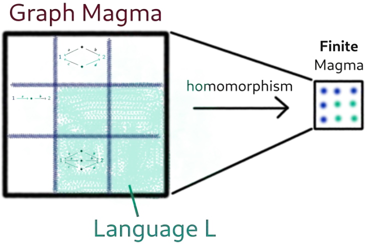


Families of graphs (Languages) with **bounded tree-width**

Regular languages of Graphs

- Regular Expression
- Automaton
- Generated by Regular Grammar
- **Definable:** Monadic Second-Order Logic with counting
- **Recognizable:** Inverse image under homomorphism into a locally-finite multi-sorted Magma





Stability in Matrix Games


K. Chatterjee[1] **R. Saona**[1] M. Oliu-Barton[2]

[1]IST Austria

[2]CEREMADE, CNRS, Université Paris Dauphine, PSL Research Institute

Main idea

Classical settings. Matrix games and Linear Programming (LP).
Classical question. Stability:

How do our objects of interest change upon perturbations?

Observables. Solutions and value of the problems.

How do solutions and value change upon perturbations?

Matrix Games

$$
\begin{array}{cc}
 & j \\
i & \begin{pmatrix} & & \\ & m_{i,j} & \\ & & \end{pmatrix}
\end{array}
$$

$$\operatorname{val} M := \max_{p \in \Delta[m]} \min_{q \in \Delta[n]} p^t M q \,.$$

$$M(\varepsilon) = M_0 + M_1 \varepsilon \,.$$

Derivative of the value function [Mills56]

Define

$$D\mathrm{val}M(0^+) := \lim_{\varepsilon \to 0^+} \frac{\mathrm{val}M(\varepsilon) - \mathrm{val}M(0)}{\varepsilon} \,.$$

Results.

1. Characterization of $D\mathrm{val}M(0^+)$.
2. (Poly-time) algorithm for computing it.

Theorem ([Mills56])

Given $M(\varepsilon) = M_0 + M_1 \varepsilon$,

$$D\mathrm{val}M(0^+) = \mathrm{val}_{P(M_0) \times Q(M_0)} M_1 \,.$$

Our framework

Polynomial matrix games. Matrix games where payoff entries are given by polynomials.

$$M(\varepsilon) = M_0 + M_1 \varepsilon + \ldots + M_K \varepsilon^K \, .$$

Definition (Value-positivity problem)

$\exists \varepsilon_0 > 0$ such that $\forall \varepsilon \in [0, \varepsilon_0] \quad \mathrm{val} M(\varepsilon) \geq \mathrm{val} M(0)$.

Definition (Uniform value-positivity problem)

$\exists p_0 \in \Delta[m] \quad \exists \varepsilon_0 > 0 \quad \forall \varepsilon \in [0, \varepsilon_0] \quad \mathrm{val}(M(\varepsilon); p_0) \geq \mathrm{val} M(0).$

Definition (Functional form problem)

Return the maps $\mathrm{val} M(\cdot)$ and $p^*(\cdot)$, for $\varepsilon \in [0, \varepsilon_0]$.

Polynomial matrix game

Consider $\varepsilon > 0$.

$$M(\varepsilon) = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} + \begin{pmatrix} 1 & -3 \\ 0 & 2 \end{pmatrix} \varepsilon \,.$$

The optimal strategy is given by, for $\varepsilon < 1/2$,

$$p^*_\varepsilon = \left(\frac{1+\varepsilon}{2+3\varepsilon}, \frac{1+2\varepsilon}{2+3\varepsilon} \right)^t \,.$$

Therefore,

$$\mathrm{val} M(\varepsilon) = \frac{\varepsilon^2}{2+3\varepsilon} \,.$$

Polynomial matrix game, negative direction

Consider $\varepsilon > 0$.

$$M(\varepsilon) = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} + \begin{pmatrix} -1 & 3 \\ 0 & -2 \end{pmatrix} \varepsilon \,.$$

The optimal strategy is given by, for $\varepsilon < 2/3$,

$$p_\varepsilon^* = \left(\frac{1-\varepsilon}{2-3\varepsilon}, \frac{1-2\varepsilon}{2-3\varepsilon} \right)^t \,.$$

Therefore,

$$\mathrm{val} M(\varepsilon) = \frac{\varepsilon^2}{2-3\varepsilon} \,.$$

Statistical Monitoring of Stochastic Systems

(with focus on Algorithmic Fairness)

$$f : \Sigma^* \to \mathbb{R}$$

some function

$$\overrightarrow{X} := \left(X_t\right)_{t>0}$$

a stochastic process

$$t \in \mathbb{N}^+$$

at any point in time

$$\vec{x}_t := x_1, \ldots, x_t$$

observe a realisation

$I \subseteq [1;t]$

$$\mathbb{E}(f(\overrightarrow{X_t}) \mid \vec{x}_I)$$

want to compute

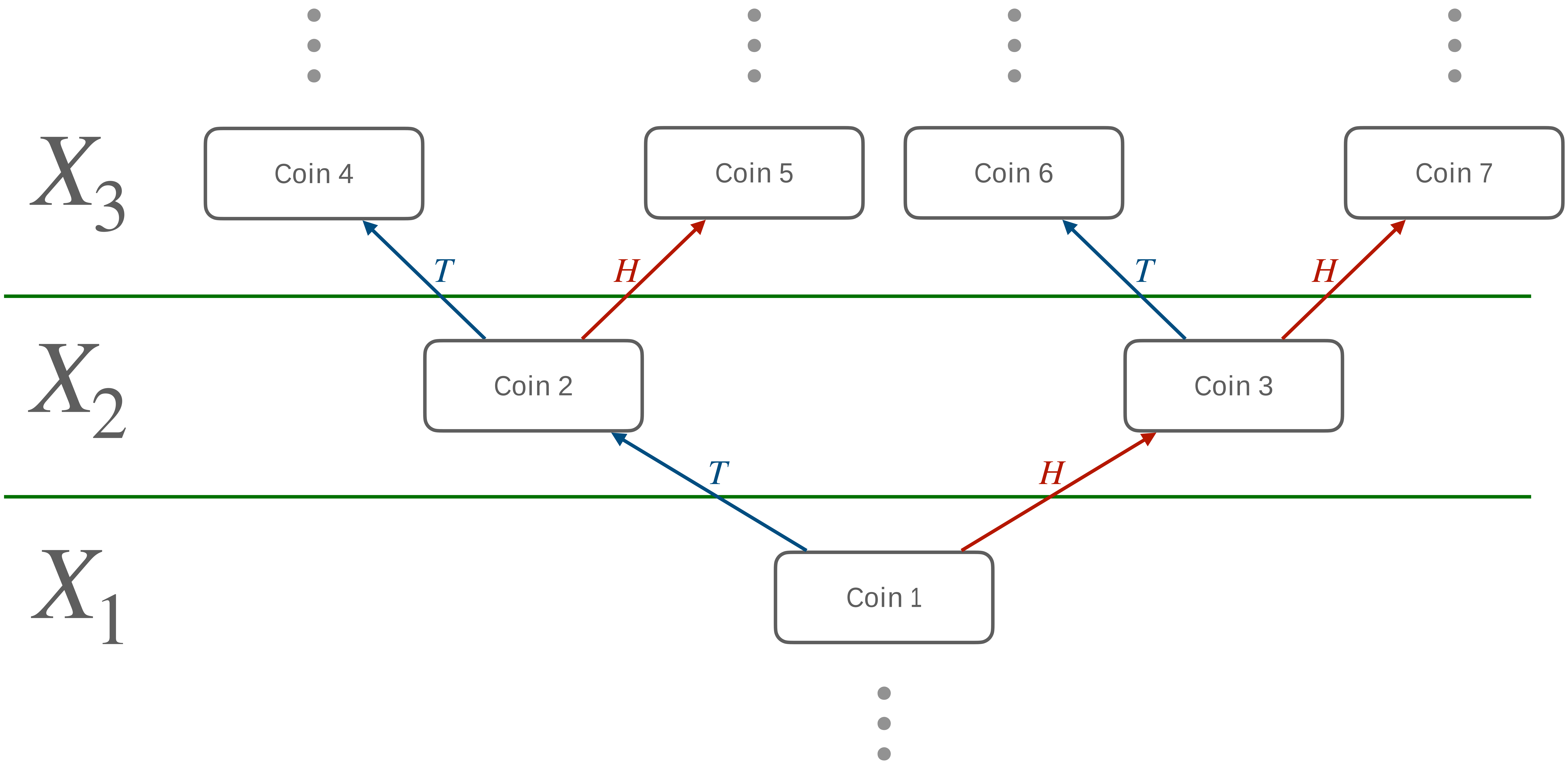
Example.

Too many coins.

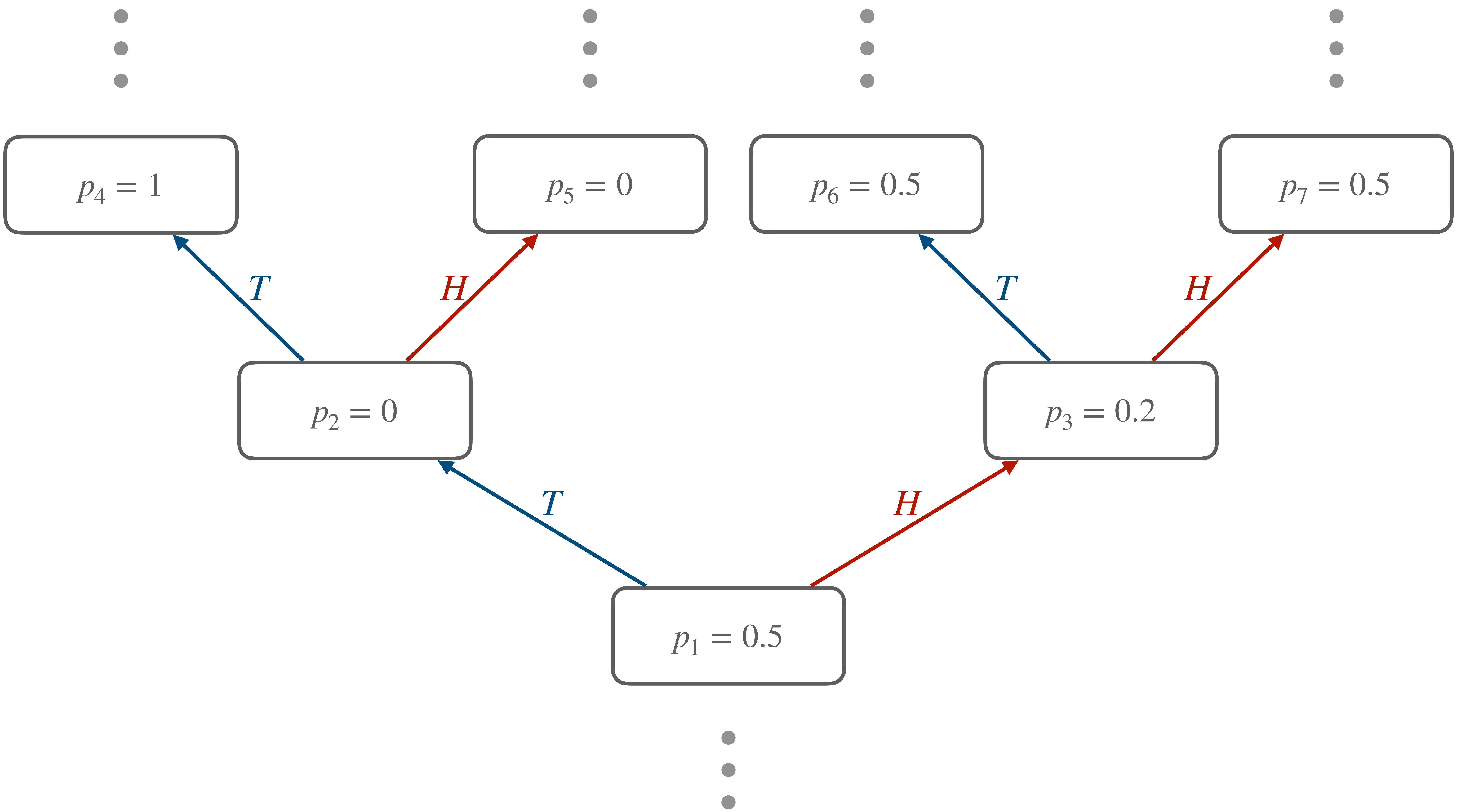
X_3

X_2

X_1



X_3
X_2
X_1
Coin 4
Coin 5
Coin 6
Coin 7
Coin 2
Coin 3
Coin 1
T
H
T
H
T
H



$p_4 = 1$
$p_5 = 0$
$p_6 = 0.5$
$p_7 = 0.5$
T
H
T
H
$p_2 = 0$
$p_3 = 0.2$
T
H
$p_1 = 0.5$

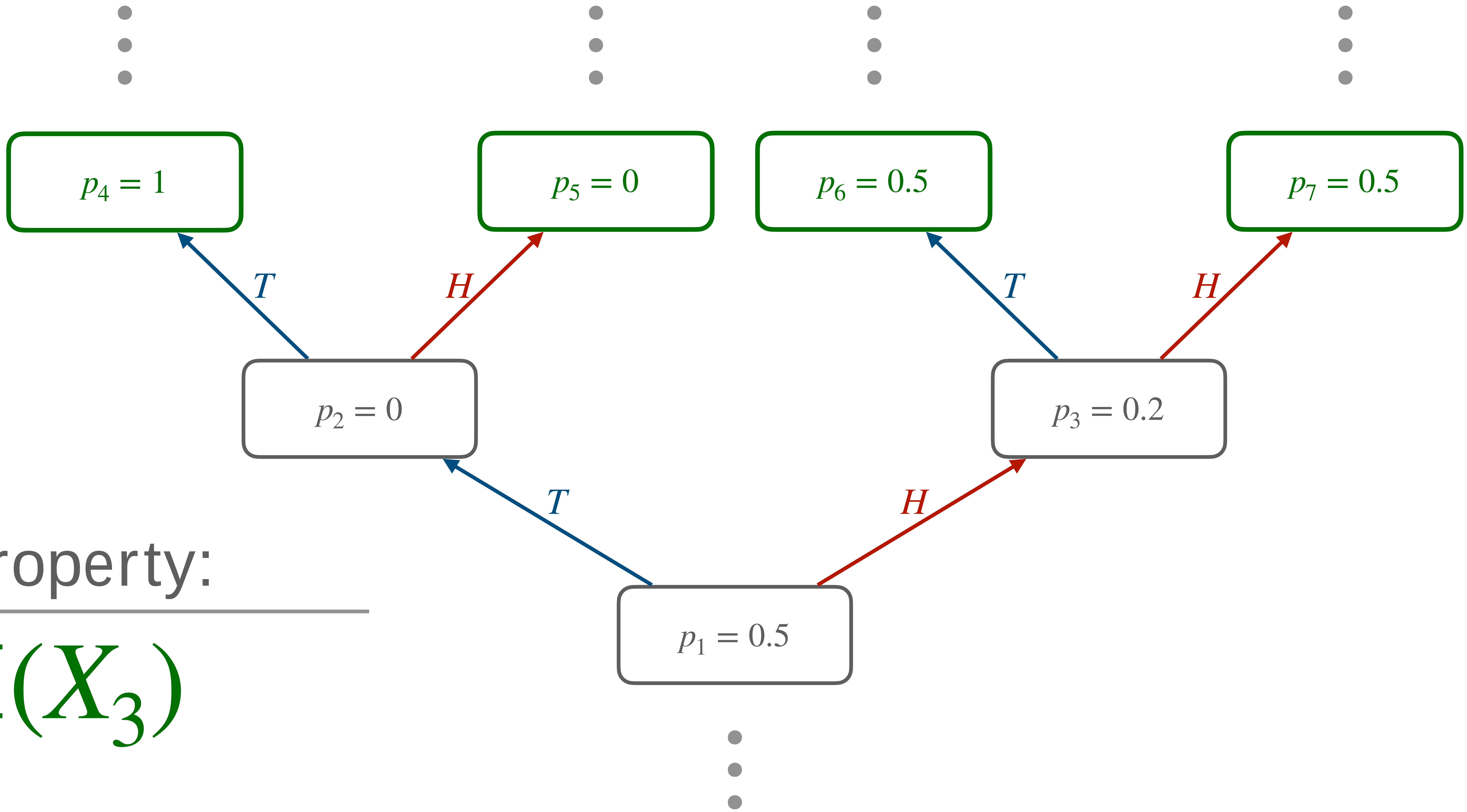
Is this process “fair”

Many different definitions.

$$\mathbb{P}(\mathsf{H}) - \mathbb{P}(\mathsf{T})$$

How fair is it…

…at time t?



Property:

$$\mathbb{E}(X_3)$$

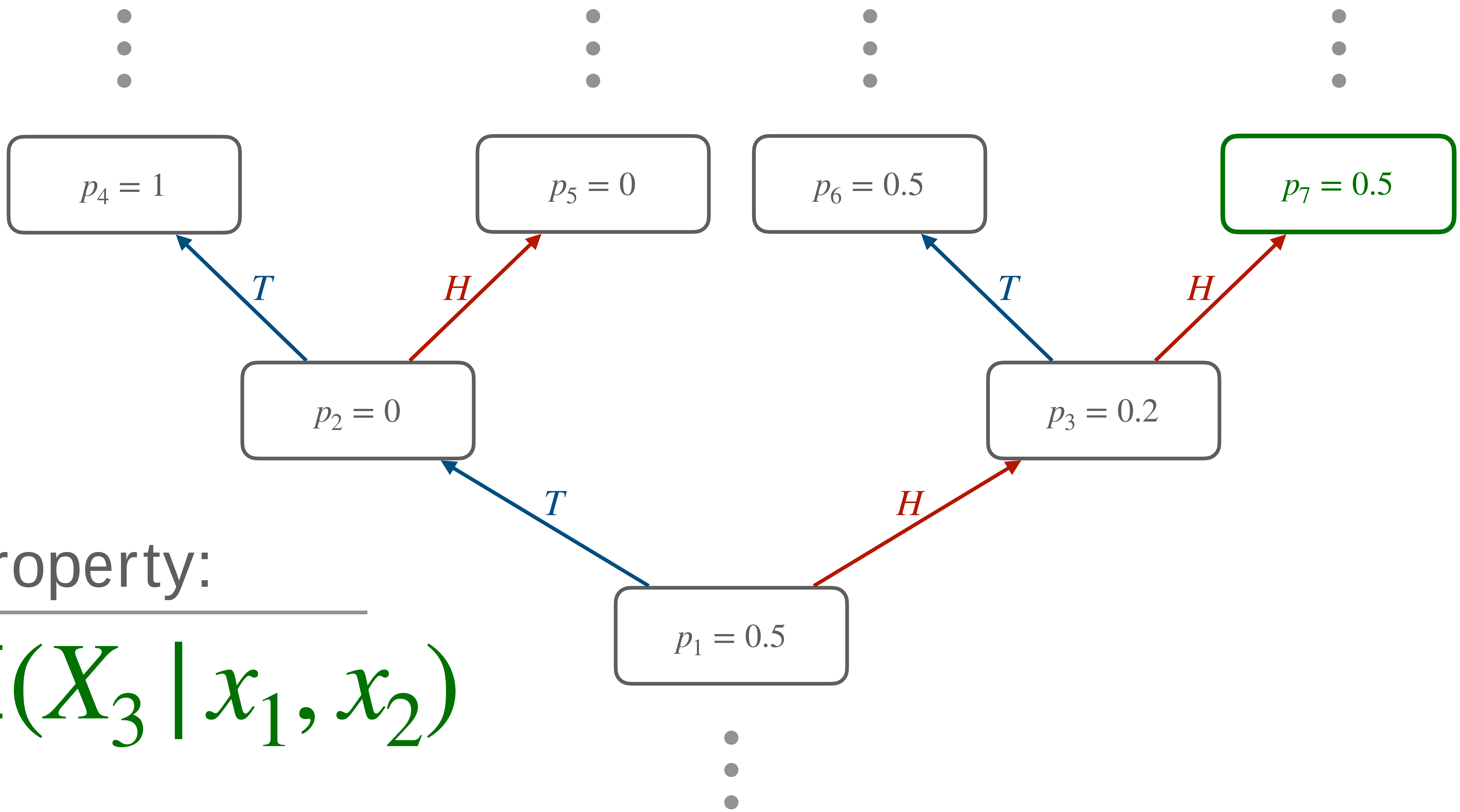
$$x_3 = T$$

$$x_2 = H$$

$$x_1 = H$$

How fair is it…

…at this very moment?



Property:

$$\mathbb{E}(X_3 \,|\, x_1, x_2)$$

The model could be…

… too big.
… wrong.
… hidden.
… mistrusted.

But maybe

you have some…

$$\mathbb{P} \in \mathscr{P}$$

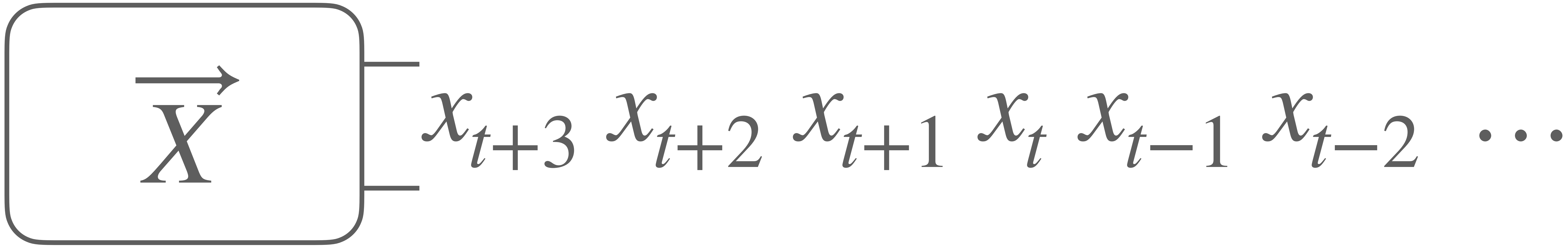
assumptions

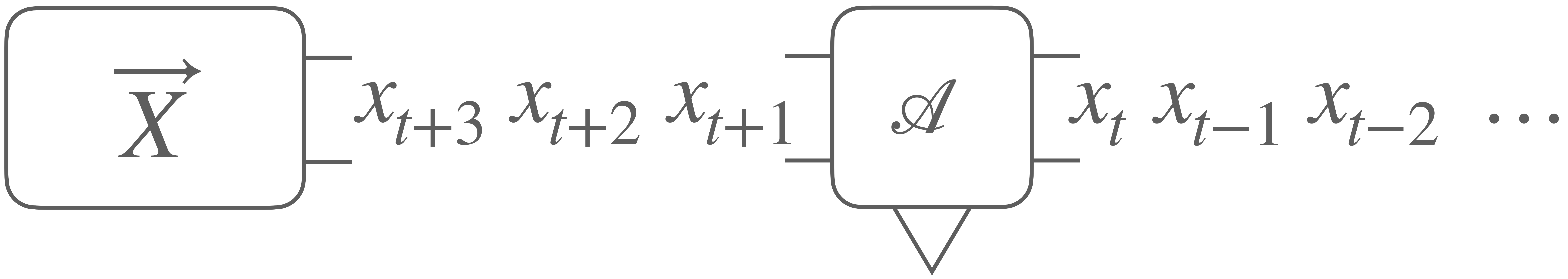
$$\hat{E}_f(\vec{x}_t)$$

you estimate

The Big Picture.

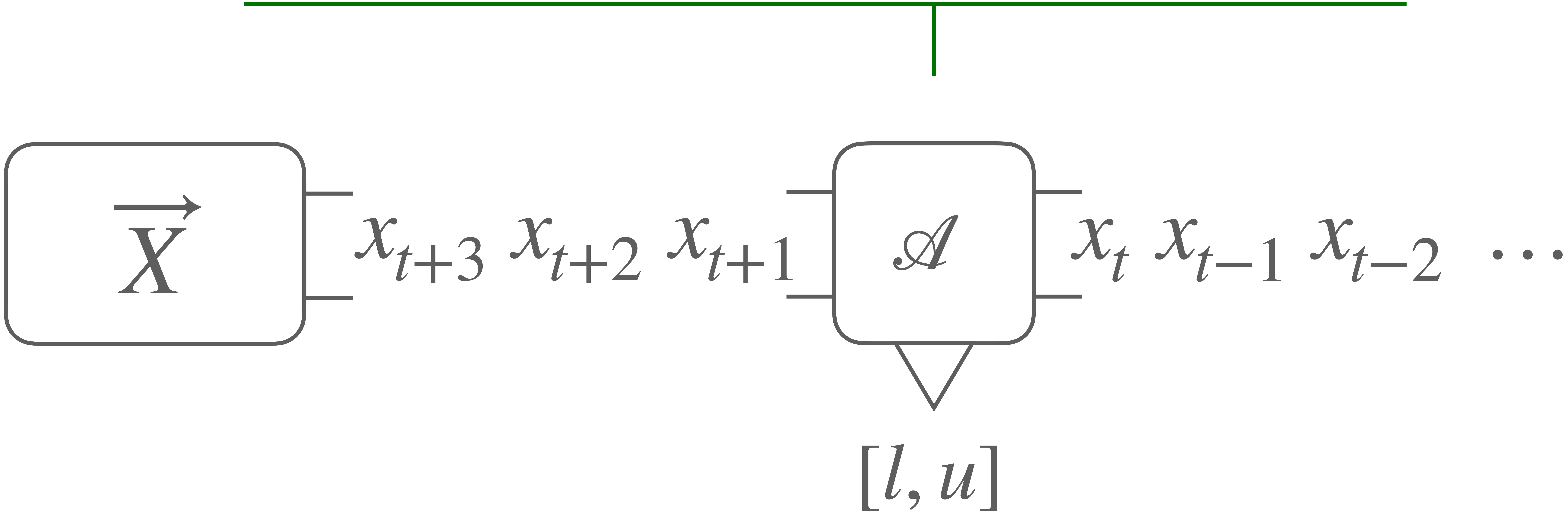
What is the general setting?


$\overrightarrow{X}$
$x_{t+3}\ x_{t+2}\ x_{t+1}\ x_t\ x_{t-1}\ x_{t-2}\ \cdots$



$\overrightarrow{X}$
x_{t+3} x_{t+2} x_{t+1}
$\mathscr{A}$
x_t x_{t-1} x_{t-2} $\cdots$

$\mathbb{E}(f(\overrightarrow{X}) \mid \vec{x}_I) \in \mathscr{A}(\overrightarrow{x_t})$ with probability $1 - \delta$



Previous Work.

A quick overview.

System MCs

Property $\mathbb{P}(r \mid q)$

Henzinger et al. "Monitoring Algorithmic Fairness." CAV 2023.

System some POMCs

Property $\mathbb{E}(f(X_{t:t+n}))$

Henzinger et al. "Monitoring Algorithmic Fairness under Partial Observations." RV 2023.

System $\mathbb{E}(X_{t+1} \mid \vec{x}_t) = \mathbb{E}(X_t \mid \vec{x}_{t-1}) + \Delta(x_t)$

Property $\mathbb{E}(f(X_t) \mid \vec{x}_{t-1})$

Henzinger et al. "Runtime Monitoring of Dynamic Fairness Properties." FAccT 2023.

Summary.

What are we doing?

Interested in monitoring “distributional” properties,
e.g. conditional expectation, of stochastic processes.

Leverage tools from non-asymptotic statistics to
provide valid guarantees for each time step.

We focused on monitoring Algorithmic Fairness,
but those techniques have wide applicability.

Use statistical monitoring to breach
the gap between the model and reality.

On the decidability of algebraic loop analysis

Anton Varonka

2nd year PhD student supervised by Laura Kovács

In my PhD project, I explore the decidability landscape of verification-motivated problems, in particular, those that underlie automated reasoning about program loops.

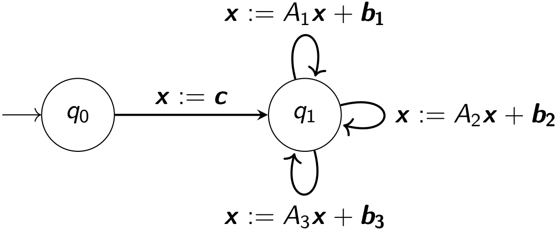
- code fragment $\longleftrightarrow$ behaviours
- model loops as dynamical systems, i.e., algebraic program analysis
- linear vs not

What is it all about

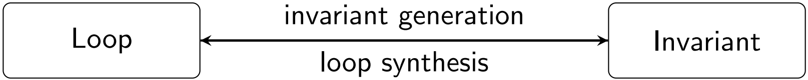
A simple loop acting on a vector $\boldsymbol{x}$ of integer variables.

Program correctness:

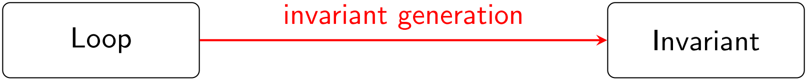
- Termination on all branches
- Finding good invariants



Loops and invariants



Loops and invariants



$(x, y) := (0, 0)$
while $y < N$ **do**
  $x := x + 2y + 1$
  $y := y + 1$

$y = x^2$

Loops and invariants

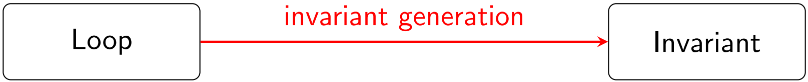
Loop —invariant generation→ Invariant

$(x, y) := (0, 0)$
while $y < N$ **do**
  $x := x + 2y + 1$
  $y := y + 1$

$(0, 0)$

$y = x^2$
holds before

Loops and invariants

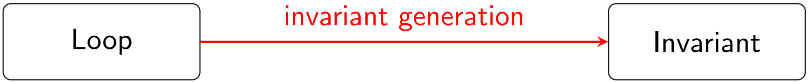


$(x, y) := (0, 0)$
while $y < N$ **do**
  $x := x + 2y + 1$
  $y := y + 1$

$y = x^2$
holds before and after each iteration

$(0,0) \quad (1,1) \quad (2,4) \quad \ldots$

Loops and invariants



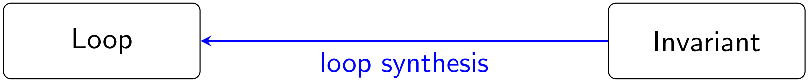
$(x, y) := (0, 0)$
while $y < N$ **do**
  $x := x + 2y + 1$
  $y := y + 1$

$y = x^2$
holds before and after each iteration

$(0, 0) \quad (1, 1) \quad (2, 4) \quad \ldots$

For a loop $\mathcal{L}$, generate all polynomial invariants $p = 0$ which $\mathcal{L}$ preserves.

Loops and invariants



$(x, y) := (0, 0)$
while $y < N$ **do**
  $x := x + 2y + 1$
  $y := y + 1$

$y = x^2$
holds before and after each iteration

$(0, 0) \quad (1, 1) \quad (2, 4) \quad \dots$

For a polynomial invariant $p = 0$, synthesise a partially correct linear loop.

Institute of Science and Technology Austria

Vamos!

Presenter: *Marek Chalupa*

October 9, 2023

Previously

Previously...

A long time ago
in a galaxy far, far away

$\approx$ 2 years
Brno (aka. Wien-Nord)

Previously...

A long time ago
in a galaxy far, far away

$\approx$ 2 years
Brno (aka. Wien-Nord)

...I got PhD from Masaryk University.

Previously...

A long time ago
in a galaxy far, far away

$\approx$ 2 years
Brno (aka. Wien-Nord)

...I got PhD from Masaryk University.

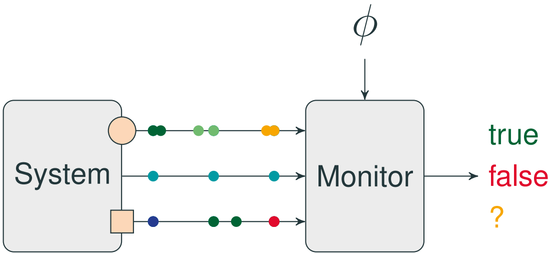
Static verification of software

- forward and backward symbolic execution
- k-induction, invariant generation, ...
- dependency analysis, program slicing

At ISTA

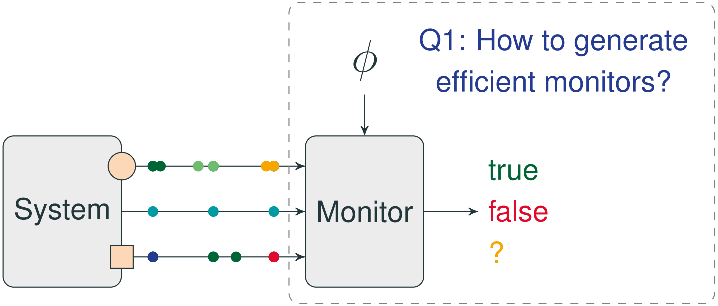
Runtime Verification

Observing a system as it is running and formally verifying properties of the run.



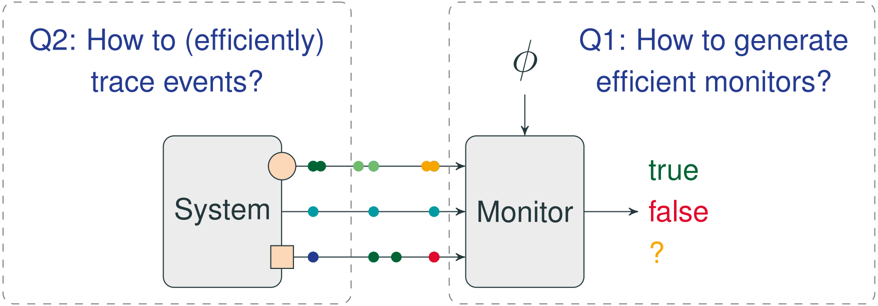
Runtime Verification

Observing a system as it is running and formally verifying properties of the run.



Runtime Verification

Observing a system as it is running and formally verifying properties of the run.



Project #1: VAMOS

VAMOS is a runtime monitoring framework

- written in C, C++, Python, and Rust

VAMOS

VAMOS is a runtime monitoring framework

- written in C, C++, Python, and Rust

Team:

- M., Tom Henzinger, Stefanie M. Lei, Fabian Muehlboeck

VAMOS

Goals of VAMOS are:

- provide basic building blocks for implementations of monitors
  - tracing events and transmitting them to monitors,
  - events and streams pre-processing and transformations

VAMOS

Goals of VAMOS are:

- provide basic building blocks for implementations of monitors
  - tracing events and transmitting them to monitors,
  - events and streams pre-processing and transformations
- support connecting heterogeneous event sources to different monitors (with best-effort and black-box monitoring in mind)

Goals of VAMOS are:

- provide basic building blocks for implementations of monitors
  - tracing events and transmitting them to monitors,
  - events and streams pre-processing and transformations
- support connecting heterogeneous event sources to different monitors (with best-effort and black-box monitoring in mind)
- focus on scenarios with multiple parallel streams of events

Project #2:
Monitoring hyperproperties

Hyperproperties

Properties that relate multiple execution traces.

Hyperproperties

Properties that relate multiple execution traces.

For each trace that contains event A, there exists a different trace with A on the same position.

Monitoring hyperproperties

Setup:

- new traces are announced anytime on runtime
- new events come incrementally to traces

Monitoring hyperproperties

Setup:

- new traces are announced anytime on runtime
- new events come incrementally to traces

We work with:

- Multi-trace prefix transducers
- Hypernode automata and logic

Monitoring hyperproperties

Setup:

- new traces are announced anytime on runtime
- new events come incrementally to traces

We work with:

- Multi-trace prefix transducers
- Hypernode automata and logic

Team:

- M., Ana Costa, Tom Henzinger, Oldouz Neysari

That's it

The presentation raises more questions than answers?

That's it

The presentation raises more questions than answers?

Good – come and talk to me :)

CirVer

Verifying algebraic circuits

Thomas Hader, Daniela Kaufmann

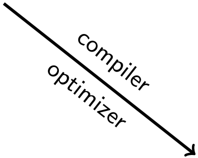
October, 9 2023

zk-SNARKs

zk-Proof: Prover P ensures verifier V that a valid computation of code is known.

zero-knowledge proof code
written in DSL

```
component unit[k - 1];
for (var i = 1; i < k; i++){
    unit[i - 1].a <== a[i] * b[i];
```



Algebraic circuit
(e.g. R1CS, PLONKish)

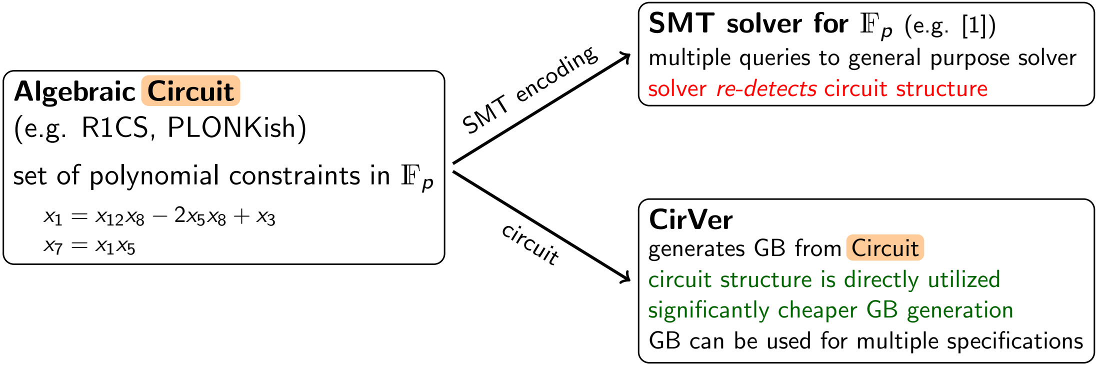
set of polynomial constraints in $\mathbb{F}_p$

$$x_1 = x_{12}x_8 - 2x_5x_8 + x_3$$
$$x_7 = x_1x_5$$

generated to code for
prover P and verifier V

Verifying algebraic circuits

Verification target: Circuit must not be under-constraint (otherwise incorrect execution traces are accepted).



[1] Hader, Kaufmann, Kovács. *SMT Solving over Finite Field Arithmetic.* LPAR 2023