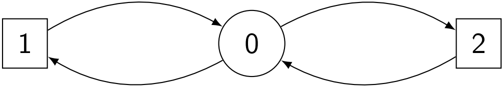
Strategy Machines and their Complexity

Marcus Gelderie

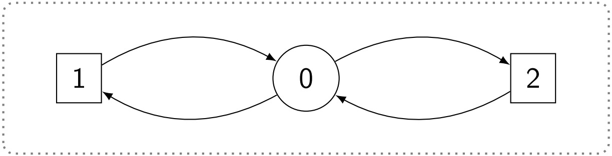
Lehrstuhl für Informatik 7
RWTH Aachen University

GAMES
September 12, 2012

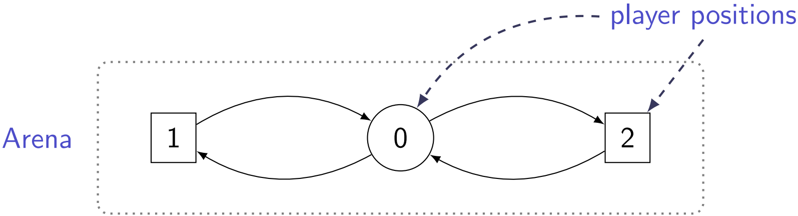
Infinite Games



Infinite Games

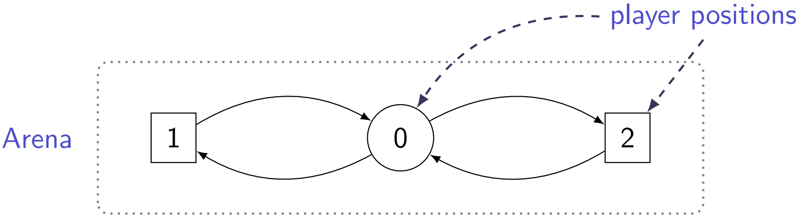
Arena 

Infinite Games



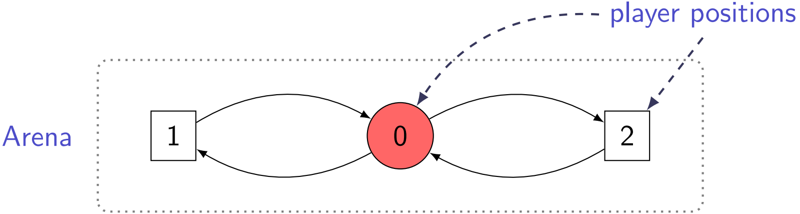
Infinite Games

Winning Condition: Visit 1 and 2 infinitely often



Infinite Games

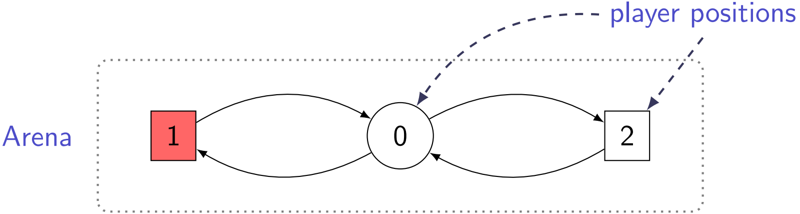
Winning Condition: Visit 1 and 2 infinitely often



play $\pi = 0$

Infinite Games

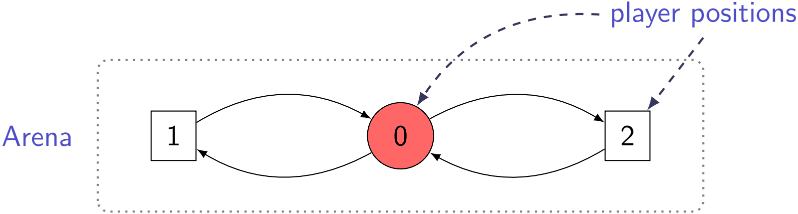
Winning Condition: Visit 1 and 2 infinitely often



play $\pi = 01$

Infinite Games

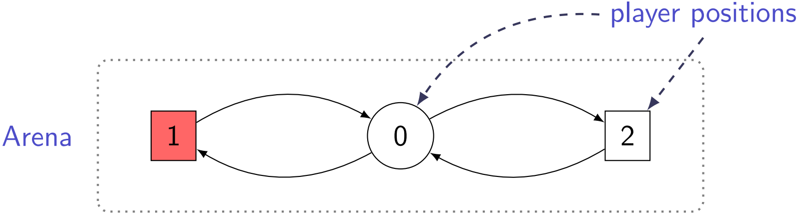
Winning Condition: Visit 1 and 2 infinitely often



play $\pi = 010$

Infinite Games

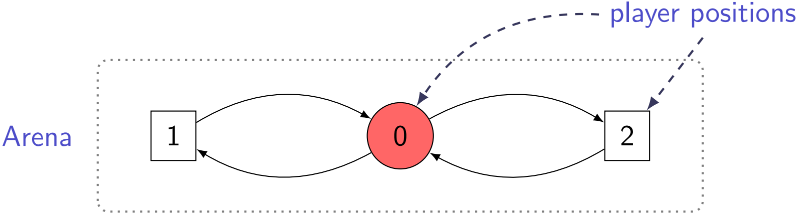
Winning Condition: Visit 1 and 2 infinitely often



play $\pi = 0101$

Infinite Games

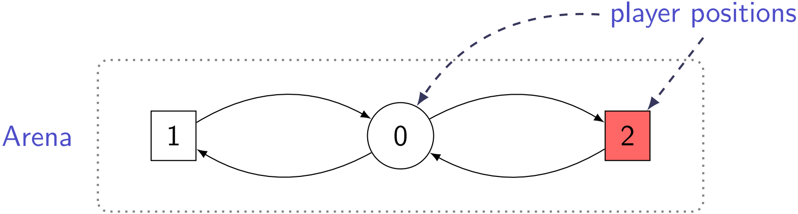
Winning Condition: Visit 1 and 2 infinitely often



play $\pi = 01010$

Infinite Games

Winning Condition: Visit 1 and 2 infinitely often

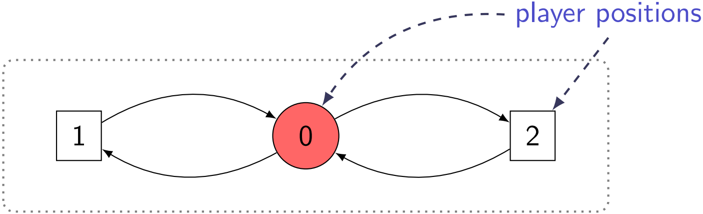


play $\pi = 010102$

Infinite Games

Winning Condition: Visit 1 and 2 infinitely often

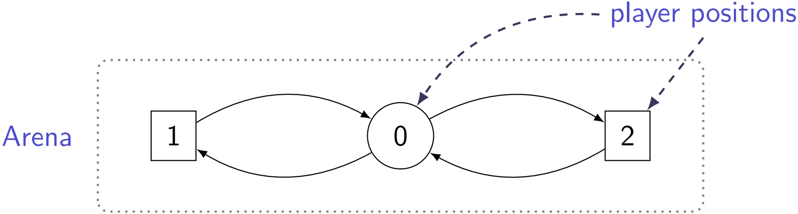
Arena



play $\pi = 0101020$

Infinite Games

Winning Condition: Visit 1 and 2 infinitely often



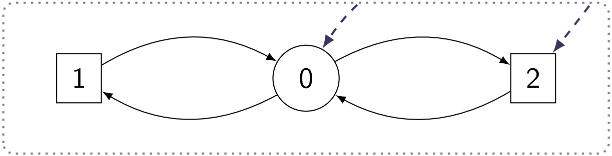
play $\pi = 0101020\cdots$

Infinite Games

Winning Condition: Visit 1 and 2 infinitely often

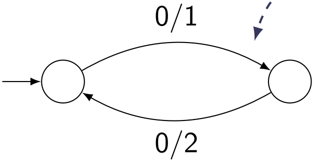
player positions

Arena

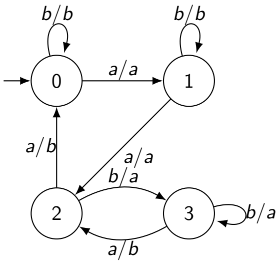


play $\pi = 0101020\cdots$

strategy



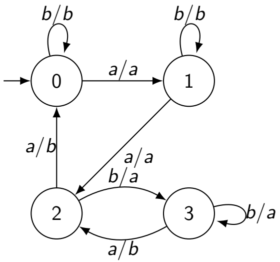
Motivation

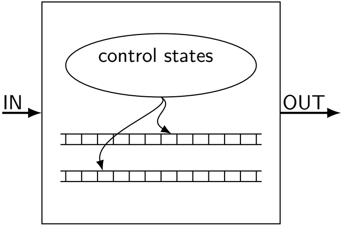


Mealy Machine

- Global view on a machine
- Information about all computations at once
- Common complexity measure: number of states

Motivation





Mealy Machine

- Global view on a machine
- Information about all computations at once
- Common complexity measure: number of states

Turing Machine

- Local view on a machine
- Information only about the current point in the computation
- Common complexity measures:
  - runtime
  - space consumption
  - size of machine

Related Work

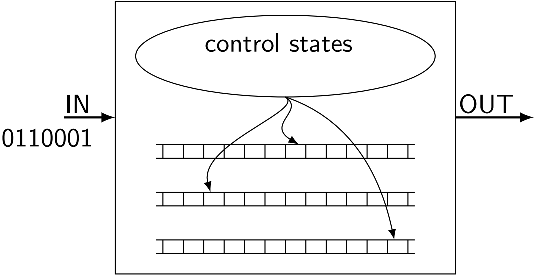
“How much memory is needed to win infinite games”
Dziembowski, Jurdziński, Walukiewicz

Polynomial sized p-automata

“Synthesizing Reactive Programs”
Madhusudan

Synthesis of while-programs over Boolean variables from ω-regular specifications

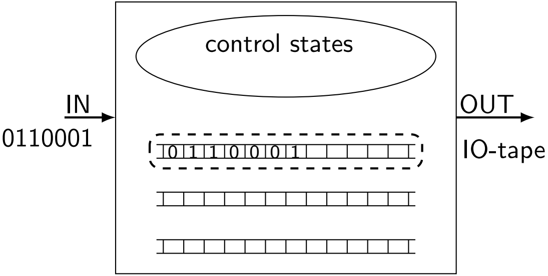
Strategy Machines



Definition (k-Tape Strategy Machine)

$(k+2)$-tape Turing machine with designated input and output states q_I and q_O

Strategy Machines

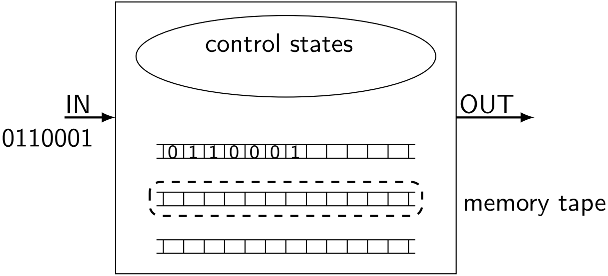


Definition (k-Tape Strategy Machine)

$(k+2)$-tape Turing machine with designated input and output states q_I and q_O

- designated IO-tape

Strategy Machines

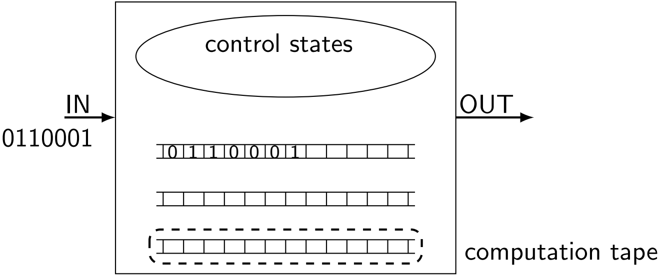


Definition (k-Tape Strategy Machine)

$(k+2)$-tape Turing machine with designated input and output states q_I and q_O

- designated IO-tape
- designated memory tape

Strategy Machines



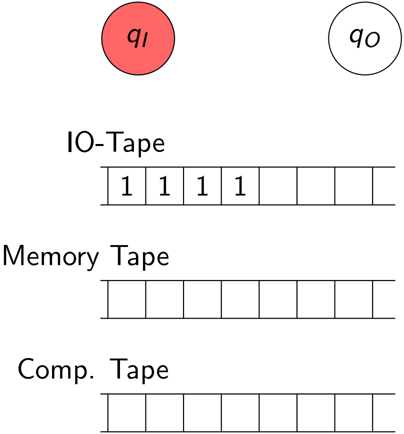
Definition (k-Tape Strategy Machine)

$(k+2)$-tape Turing machine with designated input and output states q_I and q_O

- designated IO-tape
- designated memory tape
- k computation tapes

Strategy Machine Semantics $f\colon V^+ \to V$ where $V \subseteq \mathbb{B}^*$

Example: Current input $x_1 \cdots x_k \in \mathbb{B}^k$, previous input $p_1 \cdots p_k \in \mathbb{B}^k$, output $(p_1 \oplus x_1) \cdots (p_k \oplus x_k)$

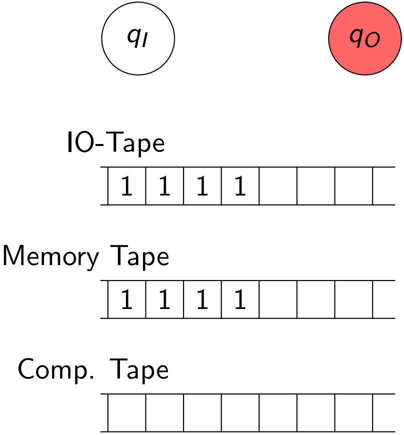


First Iteration

- input 1111

Strategy Machine Semantics $f\colon V^+ \to V$ where $V \subseteq \mathbb{B}^*$

Example: Current input $x_1 \cdots x_k \in \mathbb{B}^k$, previous input $p_1 \cdots p_k \in \mathbb{B}^k$, output $(p_1 \oplus x_1) \cdots (p_k \oplus x_k)$

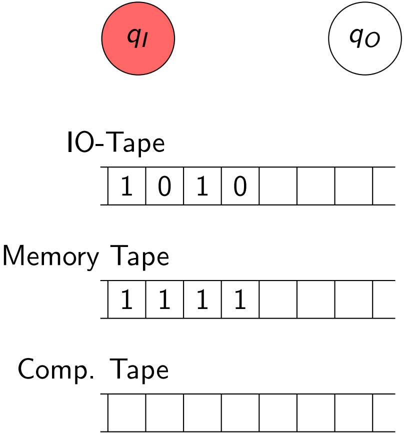


First Iteration

- input 1111
- copy input on memory tape & output 1111

Strategy Machine Semantics $f\colon V^+ \to V$ where $V \subseteq \mathbb{B}^*$

Example: Current input $x_1 \cdots x_k \in \mathbb{B}^k$, previous input $p_1 \cdots p_k \in \mathbb{B}^k$, output $(p_1 \oplus x_1) \cdots (p_k \oplus x_k)$



First Iteration

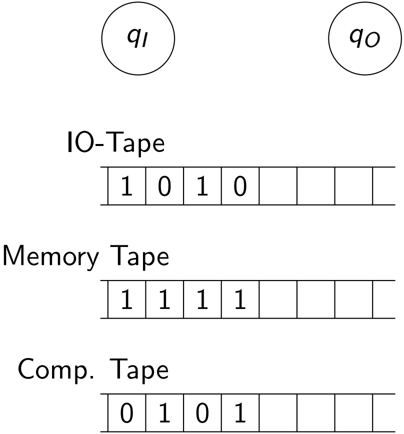
- input 1111
- copy input on memory tape & output 1111

Second Iteration

- input 1010

Strategy Machine Semantics $f\colon V^+ \to V$ where $V \subseteq \mathbb{B}^*$

Example: Current input $x_1 \cdots x_k \in \mathbb{B}^k$, previous input $p_1 \cdots p_k \in \mathbb{B}^k$, output $(p_1 \oplus x_1) \cdots (p_k \oplus x_k)$



First Iteration

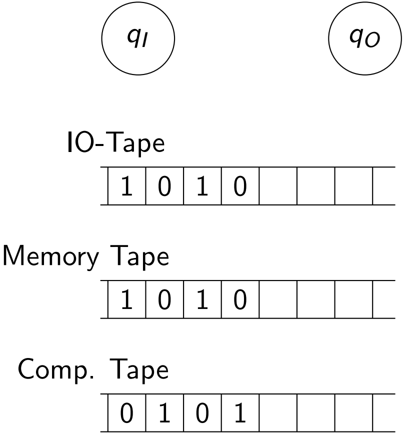
- input 1111
- copy input on memory tape & output 1111

Second Iteration

- input 1010
- XOR input with memory tape content

Strategy Machine Semantics $f\colon V^+ \to V$ where $V \subseteq \mathbb{B}^*$

Example: Current input $x_1 \cdots x_k \in \mathbb{B}^k$, previous input $p_1 \cdots p_k \in \mathbb{B}^k$, output $(p_1 \oplus x_1) \cdots (p_k \oplus x_k)$



First Iteration

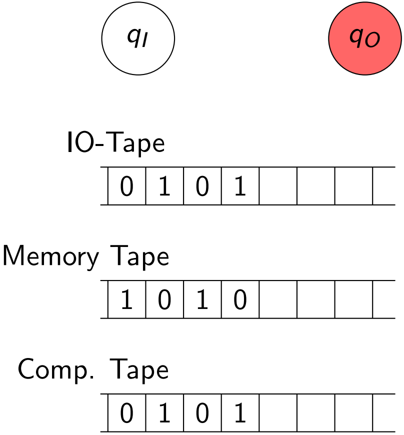
- input 1111
- copy input on memory tape & output 1111

Second Iteration

- input 1010
- XOR input with memory tape content
- copy input onto memory tape

Strategy Machine Semantics $f\colon V^+ \to V$ where $V \subseteq \mathbb{B}^*$

Example: Current input $x_1 \cdots x_k \in \mathbb{B}^k$, previous input $p_1 \cdots p_k \in \mathbb{B}^k$,
output $(p_1 \oplus x_1) \cdots (p_k \oplus x_k)$



First Iteration

- input 1111
- copy input on memory tape & output 1111

Second Iteration

- input 1010
- XOR input with memory tape content
- copy input onto memory tape
- write output back on tape

Strategy Machines – Continued

(1) receive input onto IO-tape (overwriting previous content)

- state is q_I
- computation tape is empty
- memory tape unchanged

Strategy Machines – Continued

(1) receive input onto IO-tape (overwriting previous content)

- state is q_I
- computation tape is empty
- memory tape unchanged

(2) compute an output

- using the computation tape
- using the memory tape

Strategy Machines – Continued

(1) receive input onto IO-tape (overwriting previous content)
   - state is q_I
   - computation tape is empty
   - memory tape unchanged

(2) compute an output
   - using the computation tape
   - using the memory tape

(3) write output onto IO-tape and move to state q_O

Strategy Machines – Continued

(1) receive input onto IO-tape (overwriting previous content)
  - state is q_I
  - computation tape is empty
  - memory tape unchanged

(2) compute an output
  - using the computation tape
  - using the memory tape

(3) write output onto IO-tape and move to state q_O

This sequence of steps above is an iteration

Strategy Machines – Continued

(1) receive input onto IO-tape (overwriting previous content)
- state is q_I
- computation tape is empty
- memory tape unchanged

(2) compute an output
- using the computation tape
- using the memory tape

(3) write output onto IO-tape and move to state q_O

This sequence of steps above is an iteration

Definition (Complexity Parameters)

- Latency: Time between leaving q_I and entering q_O
- Space requirement: #memory-tape-cells visited during any previous iteration
- Size: Number of control states

Relation with Mealy Machines

Proposition

Given Mealy machine $\mathfrak{M}$ with m states and alphabet V.
One can construct an equivalent strategy machine $\mathcal{M}_{\mathfrak{M}}$

- *of size* $m \cdot |V|$
- *with latency* $\log_2(m) + \log_2(|V|)$
- *and space requirement* $\log_2(m)$.

Lower Bounds

Let $\mathcal{C}$ be a class of games.

Definition (Hardness of a Class of Games)

Let $f\colon \mathbb{N} \to \mathbb{N}$. $\mathcal{C}$ is f-hard if there exists a family $(\mathbf{G}_n)_{n\geq 0}$ of games $\mathbf{G}_n = ((V_n, E_n), \varphi_n)$ in $\mathcal{C}$ such that

- $|V_n| \in \mathcal{O}(n)$
- φ_n is given in some fixed formalism, such that $\|\varphi_n\| \in \mathcal{O}(n)$
- no Mealy machine with $< f(n)$ states implements a winning strategy for player 0 in $\mathbf{G}_n$.

Lower Bounds

Let $\mathcal{C}$ be a class of games.

Let $\mathcal{S}$ be a mapping assigning a strategy machine $\mathcal{S}(\mathbf{G})$ to every game $\mathbf{G} \in \mathcal{C}$.

$\mathcal{S}$ is a solution scheme for $\mathcal{C}$ if $\mathcal{S}(\mathbf{G})$ implements a winning strategy for player 0 in $\mathbf{G}$ for every game $\mathbf{G}$.

Lower Bounds

Let $\mathcal{C}$ be a class of games.

Let $\mathcal{S}$ be a mapping assigning a strategy machine $\mathcal{S}(\mathbf{G})$ to every game $\mathbf{G} \in \mathcal{C}$.

$\mathcal{S}$ is a solution scheme for $\mathcal{C}$ if $\mathcal{S}(\mathbf{G})$ implements a winning strategy for player 0 in $\mathbf{G}$ for every game $\mathbf{G}$.

Theorem

Let $q \in (0,1)$. Let $\mathcal{C}$ be a $\mathbf{\Omega}(2^n)$-hard class of games. Then there exists no solution scheme which assigns a strategy machine with space requirement in $\mathcal{O}(n^q)$ to every game $\mathbf{G}$ on n vertices.

Lower Bounds

Let $\mathcal{C}$ be a class of games.

Let $\mathcal{S}$ be a mapping assigning a strategy machine $\mathcal{S}(\mathbf{G})$ to every game $\mathbf{G} \in \mathcal{C}$.

$\mathcal{S}$ is a solution scheme for $\mathcal{C}$ if $\mathcal{S}(\mathbf{G})$ implements a winning strategy for player 0 in $\mathbf{G}$ for every game $\mathbf{G}$.

Theorem

Let $q \in (0,1)$. Let $\mathcal{C}$ be a $\boldsymbol{\Omega}(2^n)$-hard class of games. Then there exists no solution scheme which assigns a strategy machine with space requirement in $\mathcal{O}(n^q)$ to every game $\mathbf{G}$ on n vertices.

Corollary

Let $q \in (0,1)$. Let $\mathcal{C}$ be a $\boldsymbol{\Omega}(2^n)$-hard class of games. Then there exists no solution scheme which assigns a strategy machine with latency in $\mathcal{O}(n^q)$ to every game $\mathbf{G}$ on n vertices.

Memory-Adaptive Algorithms

- Strategy machines distinguish between static memory (control states) and dynamic memory (tape content)
- This allows tradeoff between storing information and recomputing it

Memory-Adaptive Algorithms

- Strategy machines distinguish between static memory (control states) and dynamic memory (tape content)
- This allows tradeoff between storing information and recomputing it

Recomputing information allows for new optimization methods:

- Compute necessary information as the play proceeds
- The machine adapts to the play and converges towards a winning strategy

Memory-Adaptive Algorithms

- Strategy machines distinguish between static memory (control states) and dynamic memory (tape content)
- This allows tradeoff between storing information and recomputing it

Recomputing information allows for new optimization methods:

- Compute necessary information as the play proceeds
- The machine adapts to the play and converges towards a winning strategy

Remark

This observation can be used to obtain a small strategy machine for Muller and Streett games.
Moreover, for Streett games, this strategy machine is also fast.

Adaption

Zielonka's algorithm distinguishes two basic cases:

(A) The root of the current subtree is labeled with a player 0 set

(B) The root of the current subtree is labeled with a player 1 set

Adaption

Zielonka's algorithm distinguishes two basic cases:

(A) The root of the current subtree is labeled with a player 0 set

- then play attractor strategies
- cheap to compute $\implies$ do not memorize

(B) The root of the current subtree is labeled with a player 1 set

Adaption

Zielonka's algorithm distinguishes two basic cases:

(A) The root of the current subtree is labeled with a player 0 set

- then play attractor strategies
- cheap to compute $\implies$ do not memorize

(B) The root of the current subtree is labeled with a player 1 set

- iteratively solve subgames
- expensive to compute $\implies$ "spread out" computation across multiple iterations

Adaption

Zielonka's algorithm distinguishes two basic cases:

(A) The root of the current subtree is labeled with a player 0 set

- then play attractor strategies
- cheap to compute $\implies$ do not memorize

(B) The root of the current subtree is labeled with a player 1 set

- iteratively solve subgames
- expensive to compute $\implies$ "spread out" computation across multiple iterations

Difficulties:

- What to store and what to recompute
- Ensure that, ultimately, the strategy is a winning strategy

Theorem

For any Muller game $\mathbf{G} = (\mathcal{A}, \mathfrak{F})$, *where* $\mathcal{A} = (V, E)$ *and* $\mathfrak{F}$ *is given by a propositional formula* ϕ: *There exists a strategy machine* $\mathcal{M}$ *of*

- *size* $\|\mathcal{M}\| \in \mathrm{poly}(|V|, \|\phi\|)$
- *space consumption* $S(\mathcal{M}) \in \mathrm{poly}(|V|, \|\phi\|)$
- *latency* $T(\mathcal{M})$ *polynomial in* $|V| + |E|$ *and linear in* $\max\{|\mathfrak{F}|, |\mathfrak{F}^{\mathscr{C}}|\}$

Theorem

For any Muller game $\mathbf{G} = (\mathcal{A}, \mathfrak{F})$*, where* $\mathcal{A} = (V, E)$ *and* $\mathfrak{F}$ *is given by a propositional formula* ϕ*: There exists a strategy machine* $\mathcal{M}$ *of*

- *size* $\|\mathcal{M}\| \in \mathrm{poly}(|V|, \|\phi\|)$
- *space consumption* $S(\mathcal{M}) \in \mathrm{poly}(|V|, \|\phi\|)$
- *latency* $T(\mathcal{M})$ *polynomial in* $|V| + |E|$ *and linear in* $\max\{|\mathfrak{F}|, |\mathfrak{F}^{\mathscr{C}}|\}$

Remark

- *Muller games are solved via latest appearance records*
- *This yields Mealy machine of size* $|V|! \cdot |V|$
- *Simulation by strategy machine needs size* $|V|! \cdot |V|^2$
- *This means: exponentially smaller size at the price of exponentially longer latency*

Theorem

Let $\mathbf{G} = (\mathcal{A}, \Omega)$ *be a Streett game. Then there exists a strategy machine* $\mathcal{M}$ *of*

- *size* $\|\mathcal{M}\| \in \mathrm{poly}(|V|, |\Omega|)$
- *space consumption* $S(\mathcal{M}) \in \mathrm{poly}(|V|, |\Omega|)$
- *latency* $T(\mathcal{M}) \in \mathrm{poly}(|V|, |\Omega|)$

Theorem

Let $\mathbf{G} = (\mathcal{A}, \Omega)$ *be a Streett game. Then there exists a strategy machine* $\mathcal{M}$ *of*

- *size* $\|\mathcal{M}\| \in \mathrm{poly}(|V|, |\Omega|)$
- *space consumption* $S(\mathcal{M}) \in \mathrm{poly}(|V|, |\Omega|)$
- *latency* $T(\mathcal{M}) \in \mathrm{poly}(|V|, |\Omega|)$

Remark

- *Streett games are solved via index appearance records*
- *This yields Mealy machine of size* $|\Omega|! \cdot |\Omega|^2$
- *Simulation by strategy machine needs size* $|\Omega|! \cdot |\Omega|^3$
- *This means: exponentially smaller size while maintaining polynomial latency*

Conclusion

- Mealy machines are global representations of strategies
- Strategy machines are local representations of strategies
- Local representations offer a broader range of criteria to measure the quality of a strategy
  - Latency
  - Space requirement (dynamic memory)
  - Size (static memory)
- The latency and space requirement can be bounded from below for several classes of games
- For ω-regular winning conditions, based on the winning condition, small and fast controllers can be obtained.