# 1 Basics

## 1.1 List Of Games

- Büchi
- Staiger-Wagner
- weak Parity
- Reachability (E-condition)
- Safety (A-condition)
- Muller
- Parity
- Rabin
- Streett

## 1.2 List Of Properties

- Determined
  For every node $v$, either player has a winning strategy.
- Positionally Determined
  For every node $v$, either player has a positional winning strategy.
- Uniform determined
  There are disjoint sets $W_0 \cup W_1 = V$ and strategies $\sigma_0$ and $\sigma_1$ for player 0 and 1 respectively, such that $\sigma_0$ is winning from all $v \in W_0$ and $\sigma_1$ is winning from all $v \in W_1$.
- Prefix Independent
  $\forall x \in C^*, \alpha \in C^\omega : \alpha \in \text{Win} \leftrightarrow x\alpha \in \text{Win}$

## 1.3 Definitions

**Definition 1.** *A **game graph / arena** is a tuple $G = (V_0, V_1, E, c)$ where $V_0 \cap V_1 = \emptyset$, $E \subseteq V \times V$ where $V = V_0 \cup V_1$, and $c : V \to C$ for a finite set of colors $C$.*

*A **game** is a pair $\mathcal{G} = (G, Win)$ where $G$ is an arena and $Win \subseteq C^\omega$.*

*A **strategy** for player $i$ is a function $\sigma : V^*V_i \to V$ with $(u, v) \in E$ for all $\sigma(xu) = v$. $\sigma$ is a **winning strategy** from $v \in V$, if all plays from $v$ that are according to $\sigma$ are won by player $i$. $\sigma$ is **positional** if for all $x, y \in V^*, v \in V$: $\sigma(xv) = \sigma(yv)$.*

## 2 Memory & Reductions

**Definition 2.** *A **strategy automaton** for player 0 in a game $\mathcal{G}$ is a tuple $\mathcal{A} = (M, C, m_{in}, \sigma^u, \sigma^n)$ with $\sigma^n : M \times V_0 \to V$ and $\sigma^u : M \times C \to M$. The automaton defines a strategy $\sigma_{\mathcal{A}}(xv) = \sigma^n(m, v)$ where $m = (\sigma^u)^*(m_{in}, x)$.*

**Definition 3.** *Let $\mathcal{G}$ and $\mathcal{G}'$ be games. **$\mathcal{G}$ reduces to $\mathcal{G}'$ with memory $m$** if there is an $f_{in} : V \to V'$ such that a player wins from $v \in V$ iff that player wins from $f_{in}(v) \in V'$. For a winning strategy with memory $n$ from $f_{in}(v)$, one can compute a winning strategy with memory $n \cdot m$ from $v$.*

**Definition 4.** *Let $\mathcal{G} = (V_0, V_1, E, c, Win)$ be a game and let $\mathcal{A} = (Q, C, q_0, \delta, Acc)$ be a finite automaton with $L(\mathcal{A}) = Win$. The **product game** is defined as $\mathcal{G} \times \mathcal{A} = (V_0', V_1', E', c', Acc)$ with*

- $V_0' = V_0 \times Q$
- $V_1' = V_1 \times Q$
- $E' = \{((u,p),(v,q) \in (V \times Q)^2 \mid (u,v) \in E \text{ and } q = \delta(p, c(u))\}$
- $c'(v,q) = q$

**Theorem 1.** *$\mathcal{G}$ reduces to $\mathcal{G} \times \mathcal{A}$ with memory $|Q|$.*

**Example** Let $\mathcal{A} = (Q, C, q_0, \delta, F)$ be a DFA and let $\mathcal{G} = (G, C^* L(\mathcal{A}) C^\omega)$. Then $\mathcal{G}$ is a reachability game. Hence, $\mathcal{G} \times \mathcal{A}$ is determined with memory size $|Q|$.

# 3 Prefix Dependent Games

## 3.1 Reachability & Safety

$F \subseteq C$ and $\text{Win} = C^* F C^\omega$ (reachability) or $\text{Win} = (C \setminus F)^\omega$ (safety)

**Theorem 2.** *Reachability games and safety games are positionally determined. The winning regions and winning strategies can be computed in $\mathcal{O}(|G|)$.*

*Proof.* Let $\mathcal{G} = (G, F)$ be a reachability or safety game. The **attractor** for player 0 is defined as follows:

- $\text{Pre}_0(X) = \{v \in V_0 \mid \exists x \in X : (v, x) \in E\} \cup \{v \in V_1 \mid \neg\exists x \in X^{\complement} : (v, x) \in E\}$
  The set of vertices from which player 0 can force a direct move to a vertex in $X$.
- $\text{Attr}_0^0(X) = X$
  $\text{Attr}_0^{i+1}(X) = \text{Attr}_0^i(X) \cup \text{Pre}_0(\text{Attr}_0^i(X))$
  The set of vertices from which player 0 can force to reach $X$ in at most $i$ steps.
- $\text{Attr}_0(X) = \bigcup_{i \in \mathbb{N}} \text{Attr}_0^i$
  The set of vertices from which player 0 can force a play to reach $X$.

Then $W_0 = \text{Attr}_0(F)$ for reachability games and $W_0 = V \setminus \text{Attr}_1(F^{\complement})$ for safety games. The winning strategy for player 0 is to play from each $\text{Attr}_0^i$ to $\text{Attr}_0^{i-1}$ until $F$ is reached. The strategy for player 1 is to play to arbitrary nodes not in $W_0$.

A simple algorithm can compute the attractor in $\mathcal{O}(|G|^2)$ but linear time is also possible. □

## 3.2 Weak Parity

$C \subseteq \mathbb{N}$ and $\text{Win} = \{\alpha \in C^\omega \mid \max \text{Occ}(\alpha) \text{ is even}\}$.

**Theorem 3.** *Weak parity games are positionally determined. The winning regions and winning strategies can be computed in $\mathcal{O}(|C| \cdot |G|)$.*

*Proof.* If the arena is empty, then $W_0 = W_1 = \emptyset$. Otherwise let $k = \max c(V)$. Due to symmetry we can assume that $k$ is even. Let $A_k = \text{Attr}_0(c^{-1}(k))$. Compute the winning regions $(W_0, W_1)$ in the game with vertices $A_k$ removed. Then the winning regions in $\mathcal{G}$ are $(W_0 \cup A_k, W_1)$.

The winning strategy for player 0 in $A_k$ is to move to that set and then play arbitrarily. From all other nodes, the induction strategy suffices.

In each iteration, one color is removed and one attractor is computed, which results in the described runtime. □

## 3.3 Staiger-Wagner

$\mathcal{F} \subseteq 2^C$ and $\text{Win} = \{\alpha \in C^\omega \mid \text{Occ}(\alpha) \in \mathcal{F}\}$.

**Theorem 4.** *Staiger-Wagner games can be reduced to weak parity games with memory $2^{|C|}$.*

*Proof.* Build a weak parity automaton with $2^{|C|}$ states that accepts Win with SW condition $\mathcal{F} \subseteq 2^C$. For that, the PA collects a set of all seen colors as its state. For such a state $P \subseteq C$, the assigned color is $c(P) = \begin{cases} 2 \cdot |P| & \text{if } P \in \mathcal{F} \\ 2 \cdot |P| - 1 & \text{else} \end{cases}$. □

**Theorem 5.** *For every $n > 0$, there is an arena $G_n$ with $|G_n| \in \mathcal{O}(n)$ and a set $\mathcal{F}_n \subseteq 2^C$ with $|\mathcal{F}_n| \in \mathcal{O}(n)$ such that player 0 has a winning strategy in the Staiger-Wagner game $(G_n, \mathcal{F}_n)$ but every winning strategy requires memory of size $2^n$.*

*Proof.* The game uses colors $C_n = \{1, \dots, n, 1', \dots, n', 0\}$ and consists of two phases. First, player 1 selects for each $1 \leq i \leq n$ either $i$ or $i'$. This requires $3n$ vertices. In the second phase, player 0 has the same choices to make as player 1 before. After this, the last vertex is colored 0 and loops to itself. This gives a total of $3n + 1$ vertices. Player 0 wins iff all colors $C_n$ are seen.

Player 0 has to memorize all $2^n$ possible choices that player 1 could make to answer accordingly. Otherwise there would be two different strategies from player 1 that lead to the same memory state of player 0, meaning that one leads to a losing play for player 0. □

# 4 Prefix Independent Games

## 4.1 Büchi Games

$F \subseteq C$ and $\text{Win} = \{\alpha \in C^\omega \mid \text{Inf}(\alpha) \cap F \neq \emptyset\}$.

**Theorem 6.** *Büchi games are uniformly positionally determined. The winning regions and winning strategies can be computed in polynomial time in $|G|$.*

*Proof.* Define the set recurrence set for player 0 as follows:

- $\text{Recur}_0^0(X) = X$
  $\text{Recur}_0^{i+1}(X) = R \setminus \text{Pre}_1(V \setminus \text{Attr}_0(R))$ where $R = \text{Recur}_0^i(X)$
  The set of vertices in $X$ from which player 0 can force at leasts $i$ other vists to $X$.
- $\text{Recur}_0(X) := \bigcap_{i \in \mathbb{N}} \text{Recur}_0^i(X)$
  The set of vertices in $X$ from which player 0 can force infinitely many other visits to $X$.

We write $F_i = \text{Recur}_0^i(F)$ for all $i$ and $F_\infty = \text{Recur}_0(F)$. Then $W_0 = \text{Attr}_0(F_\infty)$.

The winning strategy for player 0 is to attract towards $F_\infty$ over and over again. The winning strategy for player 1 is as follows: if the play reaches a node in $V \setminus \text{Attr}_0(F_\infty)$ then that node is in some $V \setminus \text{Attr}_0(F_i)$. The strategy avoids $F_i$. If $F$ is reached at some point, then player 1 can force the next move to go into $V \setminus \text{Attr}_0(F_j)$ for some $j < i$. Hence, $F$ is visited only finitely often.

Note that in each step of Recur, at least one state in $F$ is removed and one attractor is computed, giving runtime $\mathcal{O}(|F| \cdot |G|)$. □

*Proof.* An alternative proof does not compute the explicit sets $\text{Recur}_0^i(F)$ but rather computes $\mu(v) := \max\{i \mid v \in \text{Recur}_0^i(F)\}$. For that, define the bounded addition $\oplus$ on $D = \{0, \dots, |F|, \top\}$ as

$$x \oplus y = \begin{cases} x + y & \text{if } x \neq \top, y \neq \top, x + y \leq |F| \\ \top & \text{else} \end{cases}.$$

Let $f : V \to \mathbb{N}, v \mapsto \begin{cases} 1 & \text{if } v \in F \\ 0 & \text{else} \end{cases}$ and define the functor $M : V^D \to V^D$ as

$$(M(g))(v) = \begin{cases} \max g(vE) \oplus f(v) & \text{if } v \in V_0 \\ \min g(vE) \oplus f(v) & \text{if } v \in V_1 \end{cases}.$$

Then let $\mu_0 = f$ and $\mu_{i+1} = M(\mu_i)$. The desired function is the fixed point of this sequence, $\mu = \mu_k = \mu_{k+1}$. The winning region for player 0 is $W_0 = \mu^{-1}(\top)$. The winning strategy is to play from $v$ to that node $u$ which caused $\mu(v) = \top$. □

## 4.2 Parity Games

$C \subseteq \mathbb{N}$ and $\text{Win} = \{\alpha \in C^\omega \mid \max \text{Inf}(\alpha) \text{ is even}\}$.

**Theorem 7.** *Parity games are uniformly positionally determined. The winning regions and winning strategies can be computed in non-deterministic polynomial time in $|G|$, or in deterministic time $\mathcal{O}\left(|V| \cdot |E| \cdot |C| \cdot (\frac{|V|}{|C|} + 1)^{2|C|}\right)$.*

*Proof.* □

## 4.3 Muller Games

$\mathcal{F} \subseteq 2^C$ and $\text{Win} = \{\alpha \in C^\omega \mid \text{Inf}(\alpha) \in \mathcal{F}\}$.

**Theorem 8.** *Muller games can be reduced to parity games with memory $|C| \cdot |C|!$.*

*Proof.* A Muller automaton can be transformed to a DPA using the LAR construction. □

**Theorem 9.** *For every $n > 0$, there is an arena $G_n$ with $|G_n| \in \mathcal{O}(n)$ and a set $\mathcal{F}_n \subseteq 2^C$ such that player 0 has a winning strategy in the Muller game $(G_n, \mathcal{F}_n)$ but every winning strategy requires memory of size $n!$.*

*Proof.* □

**Theorem 10.** *Let $(G, \mathcal{F})$ be a finite Muller game. Player 0 and player 1 have uniform winning strategies from their respective winning regions of size at most $m^0_\mathcal{F}$ / $m^1_\mathcal{F}$. (the automata use $V$ for the update function instead of $C$)*

*Proof.* □

**Theorem 11.** *For every $\mathcal{F} \subseteq 2^C$, there is an arena $G_\mathcal{F}$ such that player 0 wins $(G_\mathcal{F}, \mathcal{F})$ but every winning strategy requires memory at least $m^0_\mathcal{F}$.*

*Proof.* □

**Theorem 12.** *Muller games can be reduced to parity games with memory $l_\mathcal{F}$.*

### 4.3.1 Split Trees

**Definition 5.** *Let $\mathcal{F} \subseteq 2^C$. We write $\mathcal{F}|_D = \mathcal{F} \cap 2^D$ for all $D \subseteq C$. The **split tree** of $\mathcal{F}$ is called $\mathcal{S}_\mathcal{F}$ and is defined as follows:*

- *Nodes in the tree are labeled by $2^C \times \{0, 1\}$.*
- *If $C \in \mathcal{F}$, the root is labeled $(C, 0)$. Otherwise, the root is labeled $(C, 1)$.*
- *For every $\subseteq$-maximal set $D$ with $D \notin \mathcal{F}$, the root has the subtree $\mathcal{S}_{\mathcal{F}|_D}$ as a child.*

**Definition 6.** *Let $\mathcal{F} \subseteq 2^C$. Let $\mathcal{F}_1, \ldots, \mathcal{F}_n \subseteq 2^C$ such that $\mathcal{S}_{\mathcal{F}_1}, \ldots, \mathcal{S}_{\mathcal{F}_n}$ are the direct subtrees of the root in $\mathcal{S}_\mathcal{F}$. We define the* ***memory number***

$$m_i(\mathcal{S}_\mathcal{F}) = \begin{cases} 1 & \text{if } n = 0 \\ \max_j m_i(S_{\mathcal{F}_j}) & \text{if the root is } (C, i) \\ \sum_j m_i(S_{\mathcal{F}_j}) & \text{if the root is } (C, 1-i) \end{cases}.$$

*For a short form, we write $m^i_\mathcal{F} = m_i(\mathcal{S}_\mathcal{F})$.*
*We write $l_\mathcal{F} \in \mathbb{N}$ for the number of leaves in $\mathcal{S}_\mathcal{F}$.*

**Theorem 13.**
- $m^0_\mathcal{F} = m^1_{\mathcal{F}^\complement}$
- $m^i_\mathcal{F} \leq l_\mathcal{F}$
- $l_\mathcal{F} \leq |C|!$

## 4.4 Rabin & Streett Games

$\Omega = \{(E_i, F_i) \mid 1 \leq i \leq n\} \subseteq C \times C$ and
$\text{Win} = \{\alpha \in C^\omega \mid \exists i : \text{Inf}(\alpha) \cap E_i = \emptyset \wedge \text{Inf}(\alpha) \cap F_i \neq \emptyset\}$ (Rabin)
$\text{Win} = \{\alpha \in C^\omega \mid \forall i : \text{Inf}(\alpha) \cap E_i \neq \emptyset \wedge \text{Inf}(\alpha) \cap F_i = \emptyset\}$ (Streett).

**Theorem 14.** *Rabin and Streett games are determined. In a Rabin game, player 0 has a uniform positional winning strategy from their winning region. In a Streett game, player 1 has a uniform positional winning strategy from their winning region.*

*For every $n$, there is a game graph $G_n$ and a condition $\Omega_n$ with $|\Omega_n| = n$ such that the opposite player requires memory $n!$ for a winning strategy from their winning region.*

*Proof.* □

## 4.5 Logic Games

Let $\mathcal{L}$ be a logic and $\varphi \in \mathcal{L}$. Then $\text{Win}_\varphi = \{\alpha \in C^\omega \mid \alpha \models \varphi\}$.

**Theorem 15.** *For $\mathcal{L} = LTL$, logic games are uniformly positionally determined and the winning strategies can be computed in $2^{2^{|\varphi|}}$.*

*Proof.* One can compute an NBA for $\varphi$ in exponential time which can then be transformed to a DPA. □

**Theorem 16.** *For $\mathcal{L} = S1S$, logic games are uniformly positionally determined and the winning strategies can be computed in $2 \uparrow |\varphi|$.*

*Proof.* One can compute an NBA for $\varphi$ in non-elementary time which can then be transformed to a DPA. □

### 4.5.1 Church Synthesis

Goal: given a specification $\varphi(\alpha, \beta)$, construct a function/program $f$ such that $f(\alpha) = \beta$ iff $\models \varphi(\alpha, \beta)$.

Define a game $(G, \text{Win}_\varphi)$ where $G$ defines a game in which player 0 and player 1 alternatingly choose bits 0 or 1. By using the previous results, the game can be solved. A winning strategy for player 0 can be used as a program $f$.