# Chapter 1

# Parity Games

## 1.1 Introduction

The parity game is a two player game. The players will be called Odd and Even. It is played on a finite directed graph whose vertices are partitioned into two (one part for player Odd, the other for player Even). The edges of the graph are labelled by a number called *priority*. Formally,

**Definition 1.1** (board of parity game)**.** *The board of the parity game is given by $\mathcal{B} = (V, V_1, V_2, E, P, \gamma)$ where $V$ is a finite set of vertices, $V_1$ and $V_2$ partition $V$, $E \subseteq V \times V$ is the set of edges, $P$ is a set of priorities, $\gamma : E \to P$ is a mapping of edges to priorities.*

*We also assume that there is no sink state in the graph. That is, all vertices have outgoing edges.*

In the definition, the set of vertices $V_1$ are owned by player Odd and $V_2$ are owned by Even. A *parity game* is played on a board from a start state.

**Definition 1.2** (parity game)**.** *A parity game consists of a board as well as a start state. That is, a parity game, $\mathcal{G} = (\mathcal{B}, s)$ where $\mathcal{B}$ is a board and $s \in V$ is the start state.*

The game is played as follows. A *token* is placed on the start state. The player who owns the start state $s$, moves the token to one of its neighbouring vertex $u_1$ (through some outgoing edge with priority $p_0$). The next move is played by the owner of $u_1$ who moves the token to one of its neighbouring vertex through some outgoing edge with priority $p_1$. Note that, it can happen that $s$ and $u_1$ are owned by the same player. The game continues like this. At any point in time, the token is in some vertex $u_i \in V$. The owner of $u_i$ moves the token to a next state through an outgoing edge with priority $p_i$.

The game continues like this *forever*. Notice that, because there are no sink states, there is always a next move. An infinite sequence of moves is called a run. Formally,

**Definition 1.3** (run)**.** *A run is an $\omega$-word (called omega word) over the alphabet $V \times P$. For example*

$$(s, p_0), (u_1, p_1), \ldots, (u_i, p_i), \ldots$$

*Runs will be denoted by the symbol $\pi$. That is $\pi \in (V \times P)^\omega$.*

**Notes on $\omega$ words**: An omega word over the alphabet $\Sigma$ can be thought of as follows. Take the natural numbers, $1, 2, 3, \ldots$. Imagine that every point in the natural number is assigned some letter from $\Sigma$. A trivial example will be every point in the natural number is assigned to the letter $a \in \Sigma$. This gives the omega word denoted by $a^\omega$. Here is another example. The word $(ab)^\omega$ is got by assigning all odd numbers to $a$ and all even numbers to $b$. There is also a notion of regular languages over omega words (these are those accepted by Büchi automata). A study can be found here [13, 2]. Both the above examples we saw were regular words. Here is a non-regular word. Let all primes numbers be assigned letter $a$ and all composite numbers assigned letter $b$.

Omega words, apart from the fact that they are beautiful mathematical objects, are an abstraction which adds great value to simplify mathematical arguments (similar to say, complex numbers). In many situations, rather than arguing there is a "very long finite word", we can argue about an $\omega$ word. For example, the arguments in this lecture notes can always be reworked by saying there are finite words which are greater than some function of the game graph size. Working with $\omega$ words simplify such type of arguments.

As a side track, there are also *countable* words which are words got by assigning every point in rationals to some letter from the alphabet. More about countable words can be found here [4, 5].

Who wins a run? We say that Even wins a run $\pi$ if the maximum among the set of all priorities which appear infinitely often in $\pi$ is even. Otherwise Odd wins the run.

**Definition 1.4** (winner of a run)**.** *Let $\pi \in (V \times P)^\omega$ be a run. We denote by* FIN$(\pi)$ *to be the set of all letters which occur finitely often (which also*

*includes letters which do not occur) and* $\mathrm{INF}(\pi)$ *the set of all letters which occur infinitely often. That is,*

$$\mathrm{FIN}(\pi) = \{(v,p) \in V \times P \mid \exists j \geq 1 \ \forall k > j \ \pi[k] \neq (v,p)\}$$
$$\mathrm{INF}(\pi) = (V \times P) \setminus \mathrm{FIN}(\pi)$$

*We say that* Even *wins the run* $\pi$ *if the maximum of the set of all priorities occurring in* $\mathrm{INF}(\pi)$ *is even. That is,* Even *wins if the maximum of the following set is even.*

$$\{p \mid (v,p) \in \mathrm{INF}(\pi)\}$$

*We say that* Odd *wins* $\pi$ *if* Even *does not win* $\pi$*. In other words,* Odd *wins if the maximum of the above set is odd. Note that, because* $V$ *and* $P$ *are finite,* $\mathrm{INF}(\pi)$ *is not empty and therefore the maximum is defined.*

Let us look at the example given in Figure 1.1. The run $((a,4)(b,1))^\omega$ is won by Even because 4 is the maximum among the priorities occuring infinitely often (which is $\{1,4\}$). The run $(a,5)(c,7)((b,1)(a,4))^\omega$ is also won by Even for the same reason. On the other hand, the run $\pi = (a,5)((c,2)(d,3))^\omega$ is won by Odd because 3 is the maximum among the priorities of $\mathrm{INF}(\pi)$.

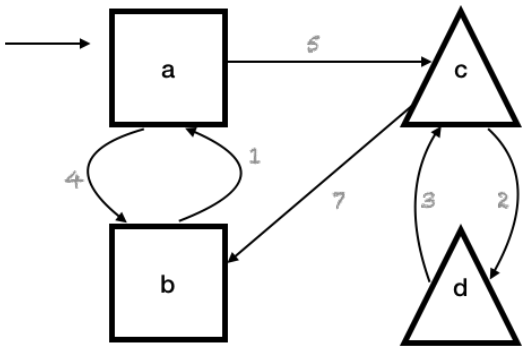


Figure 1.1: A parity game. The start state is marked with an incoming arrow. The vertices owned by Even are shown as squares (even side) and Odd by triangles. The edges are labeled by priorities.

## 1.2 Winners and strategies

We will now look at who can win a game. Informally, a *strategy* of a player is a mapping from every situation to a next move.

**Definition 1.5** (Strategy)**.** *A strategy for* Even *is a function which maps every finite word which ends in an even state to a next move. That is, it is a mapping* $\sigma : (V \times P)^*(V_2 \times P) \to V$*. The strategy for* Odd *is defined similarly. We say that* $\sigma$ *is a* winning strategy *for* Even *if he wins all the runs beginning at the start state, where he follows the strategy* $\sigma$*.*

We say that Even wins a game, if he has a winning strategy. We say that Odd wins the game, if she has a winning strategy. First thing to observe is that given a game, both together cannot have a winning strategy. The interesting observation by Büchi and Landweber [9] is that one of them has a winning strategy. In literature this is often said as parity games are *determined*.

**Theorem 1.1** (determinacy of parity games)**.** *In any parity game, either* Even *has a winning strategy or* Odd *has a winning strategy.*

A stronger version of this theorem is proved in Section 1.3. The *winner* of a game is the one who has a winning strategy. Before reading further it will be good to look at the example in Figure 1.2 and identify who has a winning strategy for the different start states.

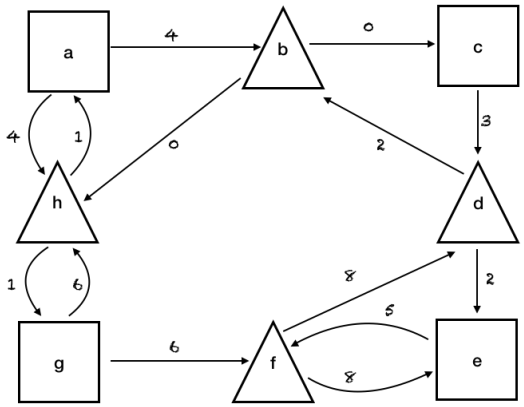


Figure 1.2: The figure represents a board for a parity game. Identify who wins for each of the parity games got by starting the game from different vertices. This figure is from wikipedia.

You would observe that Even wins if the game starts from $a$. His winning strategy is to move to $h$ if he is in $a$ or $g$. Let us analyze the strategy. When the game starts, Even will move to $h$ according to his strategy. Odd can now choose to move to either $a$ or $g$. Let us assume she moves to $g$. Then Even will again move to $h$. Now if Odd moves to $a$, Even will again move to $h$. Thus, no matter what, either priority 4 appears infinitely often or priority 6 appears infinitely often or both appears infinitely often. The *winning region* of Even (respectively Odd) are the set of all vertices from which Even (resp. Odd) has a winning strategy. Thus, the winning region for Even is $\{a, h, g\}$ and Odd is $\{b, c, d, e, f\}$. If you had worked this out yourself, you would have observed the following. The winning strategy you came up with depends only on the state the token is in and not on how the token reached that state. This is called a *positional strategy*.

**Definition 1.6** (positional strategy)**.** *A strategy $\sigma$ is a positional strategy if it depends only on the current vertex and nothing else. That is, a positional*

*strategy for* Even *is a mapping* $\sigma : V_2 \to V$ *from the set of all vertices owned by even to* $V$*. Similarly, a positional strategy for* Odd *is a mapping from the set of all vertices owned by* Odd *to* $V$*.*

The following theorem says that if a player has a winning strategy then he/she has a positional winning strategy.

**Theorem 1.2** (positional winning strategy)**.** *In any parity game, the winner has a positional winning strategy.*

In Section 1.3, we prove this theorem along with the fact that parity games are determined. Here is another interesting property which will be used in the proof of above theorem. Let $u$ and $v$ be in the winning region for Even. Then there is one positional winning strategy, $\sigma$ with which Even can win the game from both $u$ and $v$. The below theorem generalizes this to one strategy for all vertices in the winning region.

**Theorem 1.3** (common strategy)**.** *For any parity game, there is a positional winning strategy* $\sigma$*, such that if* Even *follows* $\sigma$*, it can win the parity game from any vertex* $v$ *in the winning region of* Even*.*

We leave the proof of this to the reader. Figure 1.3 gives a positional winning strategy for Even and Odd for the game given in Figure 1.2.

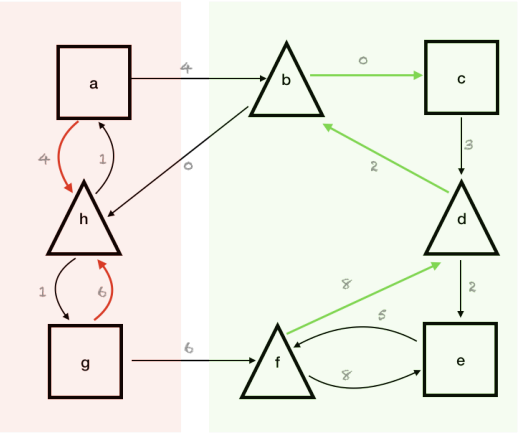


Figure 1.3: The winning strategy for Even is given in red and Odd is given in green. The winning region of Even is marked by light red and Odd is marked by light green.

**Exercise 1.1.** *Prove Theorem 1.3. For any parity game, there is a positional winning strategy* $\sigma$*, such that if* Even *follows* $\sigma$*, it can win the parity game from any vertex* $v$ *in the winning region of* Even*.*

## 1.3 Positional determinacy of parity games

In this section, we prove that parity games are positionally determined. Those who are not interested in the proof can skip this section. Our proof closely follows the proof in [2]. First we give a claim whose proof is left as an exercise.

**Lemma 1.4.** *Let the parity game starting from vertex $v$ be winning for* Even*. Then for any vertex $u$ reachable in the graph from $v$ by following the winning strategy of* Even*, is also in the winning region of* Even*.*

We now show that parity games are positionally determined.

**Theorem 1.5.** *For any parity board $G$, the wining regions of* Even *and* Odd *partition the set of vertices. Moreover, there is one positional strategy for* Even *(respectively* Odd*) following which* Even *(resp.* Odd*) can win the game starting at any vertex in its winning region.*

*Proof.* The proof is by induction on the number of priorities. It is easy to see that the theorem holds for the base case where there is exactly one priority. So, let us consider the game with $d$ priorities. Without loss of generality we assume $d$ to be odd. Let $\mathcal{B} = (V, V_1, V_2, E, P, \gamma)$ be the board of the game. Let $W_2$ be the set of vertices from which Even has a positional winning strategy and $W_1$ be the set of vertices from which Odd has a positional winning strategy. We are done if $W_1 \cup W_2 = V$. Let $U = V \setminus (W_1 \cup W_2)$ (see Figure 1.4). We show by contradiction that $U$ is empty. Let $U$ be non-empty. Here is a property of this partition.

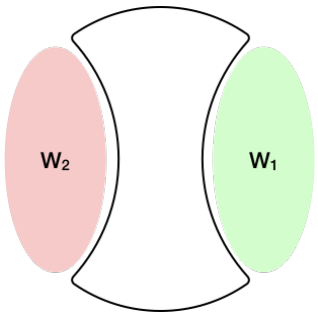


Figure 1.4: The winning regions for Even and Odd are represented by the sets $W_2$ and $W_1$ respectively.

*P*1 If vertex $u \in U$ is owned by Even, then $u$ has no edge which takes it to $W_2$ (otherwise $u$ will also be in $W_2$). Similarly, if $u$ is owned by Odd, then $u$ has no edge which takes it to $W_1$.

We first identify a set of vertices $H \subseteq U$ from which player Odd can force the game to take an edge of priority $d$ in $U$. For example, let $u \in U$ be a vertex owned by Odd and it has an edge labelled $d$. Then $u \in H$. Here is another example: if $u$ is owned by Even and all its edges has priority $d$, then $u \in H$.

Now consider the induced graph generated by the vertices $U \setminus H$. See Figure 1.5. The following property holds in this subgraph.

*P*2 If edge $(u, v)$ has priority $d$ in the induced subgraph $U \setminus H$, then $u$ is owned by Even.

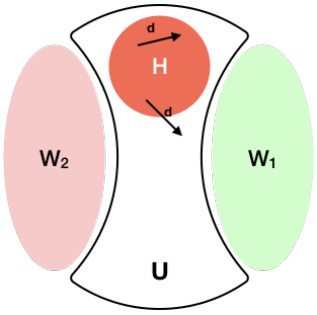


Figure 1.5: The set $H$ consists of all vertices from which Odd can force the game to take an edge of priority $d$ in $U$.

Let $G$ be the board obtained by the induced subgraph of $U \setminus H$ and then removing the edges whose priority is $d$. The following property holds in this subgraph.

*P*3 The board $G$ has atmost $d - 1$ priorities and therefore the induction hypothesis holds. That is, $G$ can be partitioned into winning regions $A_1$ and $A_2$ for Odd and Even respectively. Moreover, there is a positional strategy for Even and Odd.

We now show that $U$ is empty as follows

*P*4 The winning region $A_2$ is empty: Assume not. Consider the following positional strategy for Even in the original graph. In $W_2$, Even will play according to its strategy in $\mathcal{B}$ and in $A_2$ it plays according to the strategy in $G$. We show that this strategy is a win for Even starting the game from the vertices in $A_2$ in the original board $\mathcal{B}$. So, let us assume there is a vertex in $A_2$ from which Even cannot win. This can only happen if there is a vertex $u \in A_2$ owned by Odd from which an edge $(u, v)$ such that $v$ is in $W_1 \cup A_1 \cup H$. We first note that $v \notin W_1$ because otherwise $u$ would have been in $W_1$. If $v \in A_1 \cup H$, then the edge $(u, v)$ should have the priority $d$ (which was only the kind of edges which were removed). But from property *P*2, we know that this cannot

happen. Hence, the strategy is a win for vertices in $A_2$ and therefore these vertices should have been part of $W_2$.

$P5$ The set $A_1 \cup H$ is also empty: Consider the following positional strategy for Odd. In $W_1$ she plays according to her strategy in $\mathcal{B}$. In $A_1$ she plays according to her strategy in $G$ and in $H$ she plays the forcing strategy which takes the edge with priority $d$. We prove by contradiction that, this is a winning strategy for her from $A_1 \cup H$. If not, there is a vertex $u \in A_1 \cup H$ such that $(a)$ it is owned by Even and there is an edge $(u, v)$ and $v \in W_2$, or $(b)$ it is owned by Odd and for all edges $(u, v)$, we have $v \in W_2$. Note that in both the case, $u$ belongs to $W_2$ (see property $P1$). Hence the strategy is a win for Odd and therefore the vertices in $A_1 \cup H$ should be in $W_1$.

Thus we have that $U$ is empty and therefore the board $B$ is partitioned into positional winning regions for Even and Odd.

□

# Chapter 2

# Parity games problem

The most important algorithmic question in parity games is to identify the winning region for Even. Emerson, Jutla and Sistla [7] first showed that Parity games is in NP and in co-NP. In the next section we show the same. Marcin Jurdzinski [10] showed that this could be improved to UP ∩ co-UP. But, we do not yet have a polynomial time algorithm. Infact, we do not even know of a randomized polynomial time algorithm. It is easy to show an exponential time algorithm. Various exponential algorithms and subexponential algorithms has been proposed in literature (see [12, 11]). In a recent breakthrough, Calude et.al [3] gave an $O(n^{\log d})$ algorithm where $n$ is the graph size and $d$ is the number of priorities. Soon other researchers came up with different algorithms which give similar running time (see [6]).

**Other versions of parity games:** The usual definition for parity game is different from our definition. Usually the vertex is assigned a priority and not the edge. But this version is polynomial time equivalent to our game. See Exercise 2.1.

Here is another variation. In our definition, Even wins a run if the maximum of the infinitely occurring priority is even. We could have thought about a different game, where we define the run to be a win for Even if the *minimum* of the infinitely occurring priority is even. Note that there is a polynomial time reduction from one game to the another.

A natural extension of the two player parity game is a three player game, with the winner being decided based on the maximum priority modulo 3. It turns out that computing the winning regions for the three player game is not fundamentally different from the two player game. See Exercise 2.2.

**Exercise 2.1.** *In the vertex-parity game, each vertex is assigned a priority and not the edge. Show that there is a polynomial time reduction to our parity*

*game which preserves the winning regions. Similarly give a polynomial time reduction from our parity game to the vertex-parity game.*

**Exercise 2.2.** *Show that if we can solve the two player parity game in polynomial time, we can also solve the three player parity game in polynomial time.*

**Exercise 2.3.** *The parity games problem is polynomial time reducible to games where the maximum priority is less than or equal to twice the number of distinct priorities.*

## 2.1 Parity game is in NP ∩ co-NP

In this section we show that parity games is in NP and co-NP. First a few definitions.

**Definition 2.1** (even cycle)**.** *A cycle in a parity graph is called an* even cycle *if the greatest priority in the cycle is even. It is called an* odd cycle *if the greatest priority is odd.*

For example, in Figure 1.3 the cycle (*abha*) is an even cycle whereas (*bcdb*) is an odd cycle . Note that, the definition of even and odd cycle here is different from what is traditionally used in graph theory.

Let us first look at a simpler game. Consider a parity game where all vertices are owned by Even. This game is called a *one player* parity game. The winning condition for Even is exactly like in the two player game. In this restriction, the winning region can be identified in polynomial time.

**Theorem 2.1** (one player game)**.** *The winning region for* Even *in the one player game can be decided in polynomial time.*

*Proof.* We identify the winning region for Even as follows. Let $C$ be all vertices $v$ which are part of an **even** cycle. This can be done in polynomial time (checking whether there is an even cycle from $v$ to $v$ can be done by non-deterministic logspace machine). Let $R$ be all the vertices from which there is a path to some vertex in $C$. This can also be determined in polynomial time. We say that the winning region for Even is $C \cup R$ and leave the correctness to the readers. □

We are now in a position to show that parity games is in NP.

**Theorem 2.2.** *The problem of checking whether* Even *has a winning strategy in a parity game is in NP.*

*Proof.* From Theorem 1.2 we know that if Even has a winning strategy, then he has a positional winning strategy, $\sigma : V_2 \to V$. The NP machine first guess this strategy. It needs to now verify whether this guess is correct. For every even vertex $v \in V_2$, we remove all outgoing edges of $v$ other than $(v, \sigma(v)$. In the resultant graph, every Even vertex has exactly one outgoing edge. We now rename every Even vertex as an Odd vertex. This gives us an one player game which can be decided in polynomial time by Theorem 2.1. □

Note that, by a similar argument is also follows that the question of whether Odd has a winning strategy is also in NP. Because of the symmetrical nature of the problem, we get that parity games is also in co-NP.

**Theorem 2.3.** *The problem of checking whether* Even *has a winning strategy in a parity game is in co-NP.*

*Proof.* We show that the problem of checking whether Even does not have a winning strategy is in NP. From Theorem 1.1, we know that Even does not have a winning strategy if and only if Odd has a winning strategy. The latter problem is in NP due to Theorem 2.2 and hence parity games is also in co-NP. □

## 2.2 Deterministic algorithms for Parity games

In this section we look at deterministic algorithms which solve the parity game problem. We first look at an $O(n^{d/2})$ algorithm followed by the recent $O(n^{\log d})$ algorithm. We first show that the parity games can be solved by a reduction to *alternating graphs* (see Section 2.3). The reduction is carried out by building deterministic finite automatas. You would observe that the reduction is polynomial in the size of the parity game as well as the automata. This means, the question we will be interested in is, how small can the automata be given a parity game. Unfortunately, we do not have automatas of polynomial size. But we show that it is easy to build an automata of $O(n^{d/2})$ size which helps us resolve the parity games problem in same time. With some effort, we can build an automata of size $O(n^{\log d})$, thereby giving us a faster algorithm. Our proofs follows the arguments in [2].

First, few definitions for building the automata. We say that an $\omega$ word $w \in (V \times P)^\omega$ contains an even cycle (respectively odd cycle) if $w$ contains a cycle whose highest priority is even (respectively odd). For example

$$(a, 2)(b, 3)(c, 4)(d, 2)(b, 3)(c, 4) \dots$$

contains an even cycle since the highest priority in the factor $(b, 3) \dots (b, 3)$ is even. An $\omega$ word is called an *even word* (respectively *odd word*) if it do

not contain odd cycles (respectively even cycles). In other words, even words contain only even cycles. Figure 2.1 gives a pictorial representation of even and odd words.

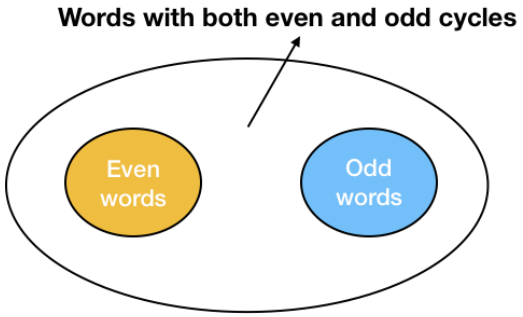


Figure 2.1: The set of all $\omega$ words are partitioned into even words, odd words and words which contain both even and odd cycles.

We now define, *acceptance of $\omega$-words* by deterministic finite automata (DFA). Normally, DFAs are defined for accepting finite words [8]. We modify the acceptance condition here for $\omega$ words. Formally,

**Definition 2.2.** *A DFA is given by $\mathcal{D} = (Q, \Sigma, \delta, s, F)$ where $Q$ is a finite set of states, $\Sigma$ is a finite alphabet, $\delta : Q \times \Sigma \to Q$ is a $\Sigma$-labelled edge, $s$ is the start state and $F$ is a set of final states.*

*An $\omega$-word $w$ is accepted by $\mathcal{D}$ if a prefix of $w$ is accepted. In other words, $w$ is accepted by $\mathcal{D}$ if $w$ can be written as $uv$ where $u \in \Sigma^*$ and $v \in \Sigma^\omega$ and there is a path from $s$ to an $f \in F$ which is labelled by $u$.*

We say that an automata *separates* even words from odd words, if the automata accepts a language $L$ which contains all even words and do not contain any odd word. Note that we do not mind whether words which are neither even or odd are in $L$ or not. Figure 2.2 shows a separating language. We are interested in DFAs which can separate even words from odd words. That is, those DFAs which can accepts all even words and rejects all odd words.

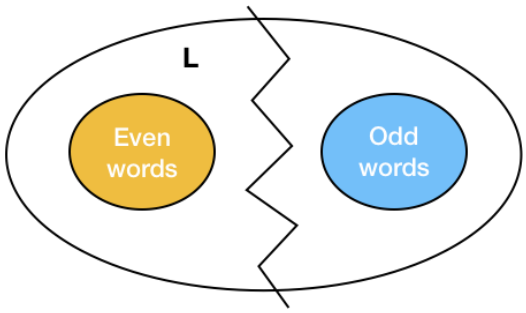


Figure 2.2: The language $L$ separates even words from odd words.

We show that using $D$ we can reduce the parity games problem to reachability in universal graphs.

**Theorem 2.4.** *Let $V \times P$ be an alphabet and $D$ an automata which separates even words from odd words. Then, there is an algorithm to decide parity games in time $O(\text{size of parity game} \times \text{size of } D)$.*

We prove the above theorem in Section 2.3. In Section 2.4 we give an automata which separates even words from odd words in time $O(n^{d/2})$ and in Section 2.5 we give an automata of size $O(n^{\log d})$.

## 2.3 Reduction to Alternating graphs

In this section we prove Theorem 2.4 by reducing the parity games problem to reachability in alternating graphs.

Alternating graphs (see Alternating turing machines [1]) are defined as:

**Definition 2.3.** *An alternating graph, $G = (V, V_\exists, V_\forall, E)$ is a graph with finite number of vertices denoted by $V$ and edges $E$. The vertices are partitioned into existential ($V_\exists$) and universal ($V_\forall$) vertices. We say that vertex $t$ is reachable from an existential state $s$ if there is an outgoing neighbour of $s$ from which $t$ is reachable. On the other hand, $t$ is reachable from an universal state $s$ if $t$ is reachable from all outgoing neighbours of $s$.*

We are interested in the question of reachability of a vertex $t$ from another vertex $s$ in an alternating graph. It can be shown that this is in deterministic polynomial time. We leave the proof of this theorem which follows from some standard results in complexity theory.

**Theorem 2.5** (alternating reachability [1])**.** *Reachability in alternating graphs is in deterministic polynomial time.*

Those who are comfortable with complexity theory can observe the following. Without the for all states, reachability is in non-deterministic log space (and therefore in deterministic polynomial time). With the for all states, reachability can be done in alternating log space. This complexity class is equivalent to deterministic polynomial time.

We now prove Theorem 2.4 by reducing parity games to reachability in alternating graphs. Let us restate the theorem.

**Theorem 2.4.** *Let $V \times P$ be an alphabet and $D$ an automata which separates even words from odd words. Then, there is an algorithm to decide parity games in time $O(\text{size of parity game} \times \text{size of } D)$.*

*Proof.* Let $\mathcal{B} = (V, V_1, V_2, E, P, \gamma, s)$ be the parity game and $D = (Q, V \times P, \delta, s', F)$ be the DFA which separates even words from odd words. Without loss of generality we assume $s \in V_2$, that is $s$ is owned by Even. Consider the alternating graph $\mathcal{G} = (K, K_\exists, K_\forall, T)$ where $K = V \times Q$, $K_\exists = V_2 \times Q$ and $K_\forall = V_1 \times Q$. The edges are drawn as follows. There is an edge

$$(u, q) \to (v, r)$$

in $T$, if there is an edge

$$u \xrightarrow{p} v$$

in $\mathcal{B}$ and an edge

$$q \xrightarrow{(u,p)} r$$

in $D$. We claim that Even has a winning strategy in $\mathcal{B}$ if and only if one of the states $V \times F$ is reachable from $(s, s')$ in the alternating graph $\mathcal{G}$. The proof has to be filled. □

## 2.4 The $O(n^{d/2})$ separating automata

In this section we give an automata of size $O(n^{d/2})$ which separates even words from odd words over the alphabet $\Sigma = (V \times P)$. Here $P$ is the set of all priorities. We also denote by $n$ the cardinality of $V$ and by $d$ the maximum priority in $P$. Note from Exercise 2.3 that, $d$ can be atmost twice the number of priorities in $P$. Recall that even words are $\omega$ words which contains only even cycles.

We first identify the separating language and then construct a deterministic finite automata which accepts the language. We say that an $\omega$ word $w$ is *nice* if $w$ can be written as $u_1(a, p)u_2(b, p)u_3$ such that it satisfies all the following properties:

1. $p$ is an even priority.
2. $(a, p), (b, p) \in \Sigma$, $u_1, u_2 \in \Sigma^*$ and $u_3 \in \Sigma^\omega$.
3. The factor $u_2$ do not contain any priority greater than $p$.
4. The number of letters with priority $p$ in $u_2$ is $n - 1$.

Note that in a nice word, there is a factor which is an even cycle (for eg. $(a, p)u_2(b, p)$ is an even cycle). But also observe that we have a restriction on the kind of even cycle in a nice word. The even cycle should contain

$n+1$ occurrence of the highest even priority. The language we propose for separation is

$$L = \{w \mid w \text{ is a } \textit{nice} \text{ word}\}$$

Before we give a DFA which accepts $L$, let us show that it separates even words from odd (see also Figure 2.3).

**Theorem 2.6.** *All even words are in $L$ whereas all odd words are not in $L$.*

*Proof.* From the definition of nice word, it contains an even cycle. Therefore odd words are not in $L$. On the other hand, consider an even word $w \in \Sigma^\omega$. Since $\Sigma$ is finite and because $w$ contains only even cycles, there is a letter $(a, 2p)$ which occur infinitely often and hence there is an even cycle with priority $2p$ occurring more than $n$ times. This word is therefore in the language $L$. □

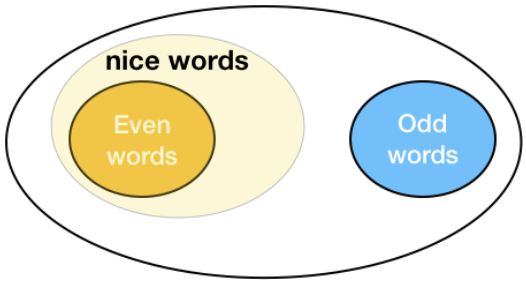


Figure 2.3: The nice words separates the even words from the odd words.

Finally we give an automata which recognize $L$.

**Theorem 2.7.** *There is a deterministic finite state automata over the alphabet $V \times P$ of size $O(n^{\lfloor d/2 \rfloor})$ which recognize the language $L$. Here $n = |V|$ and $d = 2|P|$.*

*Proof.* The automata needs to recognize all nice words. For this, every state in the automata keeps a register for every even priority (that is, every state has $\lfloor \frac{d}{2} \rfloor$ registers). The $i^{th}$ register (where $i$ is even) keeps track of the number of letters with priority $i$ after the last letter with a priority greater than $i$. Once this count in register $i$ reach a value greater than $n$, the automata goes to a unique final state and remain there. We elaborate the transitions of the automata now. Let $k$ be the count of register $i$ at some stage of reading the word. We do a case analysis based on the next letter.

- next letter is $(a, p)$ where $p = i$ and $k = n$: Since we have now identified that the word is nice, the automata moves to a final state and continue to remain there.

- next letter is $(a, p)$ where $p = i$ and $k < n$: The automata increases the register value of $i^{th}$ register to $k + 1$. All registers numbered less than $p$ is reset to zero. No change in other registers.
- next letter is $(a, p)$ where $p < i$: Value of $i^{th}$ register in remains the same. All registers numbered less than $p$ is reset to zero. No change in other registers.
- next letter is $(a, p)$ where $p > i$: The value of $i^{th}$ register is reset to zero. All registers numbered less than $p$ is reset to zero. No change in other registers.

It is clear that the automata recognize the language $L$. We will now analyze the number of states in this automata. Since every even register counts from 0 to $n$, we require $(n + 1)^{\lfloor d/2 \rfloor}$ states other than the final state. Hence the total number of states is

$$(n + 1)^{\lfloor d/2 \rfloor} + 1 = O(n^{\lfloor d/2 \rfloor})$$

□

In the next section we construct another dfa which uses less space.

## 2.5 The $O(n^{\log d})$ separating automata

# Bibliography


[1] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[2] Mikolaj Bojanczyk and Wojciech Czerwinski. *An automata toolbox*.

[3] Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *STOC*, pages 252–263. ACM, 2017.

[4] Olivier Carton, Thomas Colcombet, and Gabriele Puppis. An algebraic approach to MSO-definability on countable linear orderings. *J. Symb. Log.*, 83(3):1147–1189, 2018.

[5] Thomas Colcombet and A. V. Sreejith. Limited set quantifiers over countable linear orderings. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Proceedings, Part II*, pages 146–158, 2015.

[6] Laure Daviaud, Marcin Jurdzinski, and Ranko Lazic. A pseudo-quasi-polynomial algorithm for mean-payoff parity games. In Anuj Dawar and Erich Grädel, editors, *LICS*, pages 325–334. ACM, 2018.

[7] Emerson, Jutla, and Sistla. On model checking for the mu-calculus and its fragments. *TCS: Theoretical Computer Science*, 258, 2001.

[8] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, Reading, Mass., 1979.

[9] J.R. Büchi and L.H. Landweber. Solving sequential conditions finite-state strategies. *Trans. Ameri. Math. Soc.*, 138:295–311, 1969.

[10] Marcin Jurdzinski. Deciding the winner in parity games is in up and co-up. *Inf. Process. Lett*, 68(3):119–124, 1998.

[11] Marcin Jurdzinski. Small progress measures for solving parity games. *Lecture Notes in Computer Science*, 1770:290–??, 2000.

[12] Marcin Jurdzinski, Mike Paterson, and Uri Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. Comput*, 38(4):1519–1532, 2008.

[13] Wolfgang Thomas. Handbook of formal languages, vol. 3. chapter Languages, Automata, and Logic, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA, 1997.